

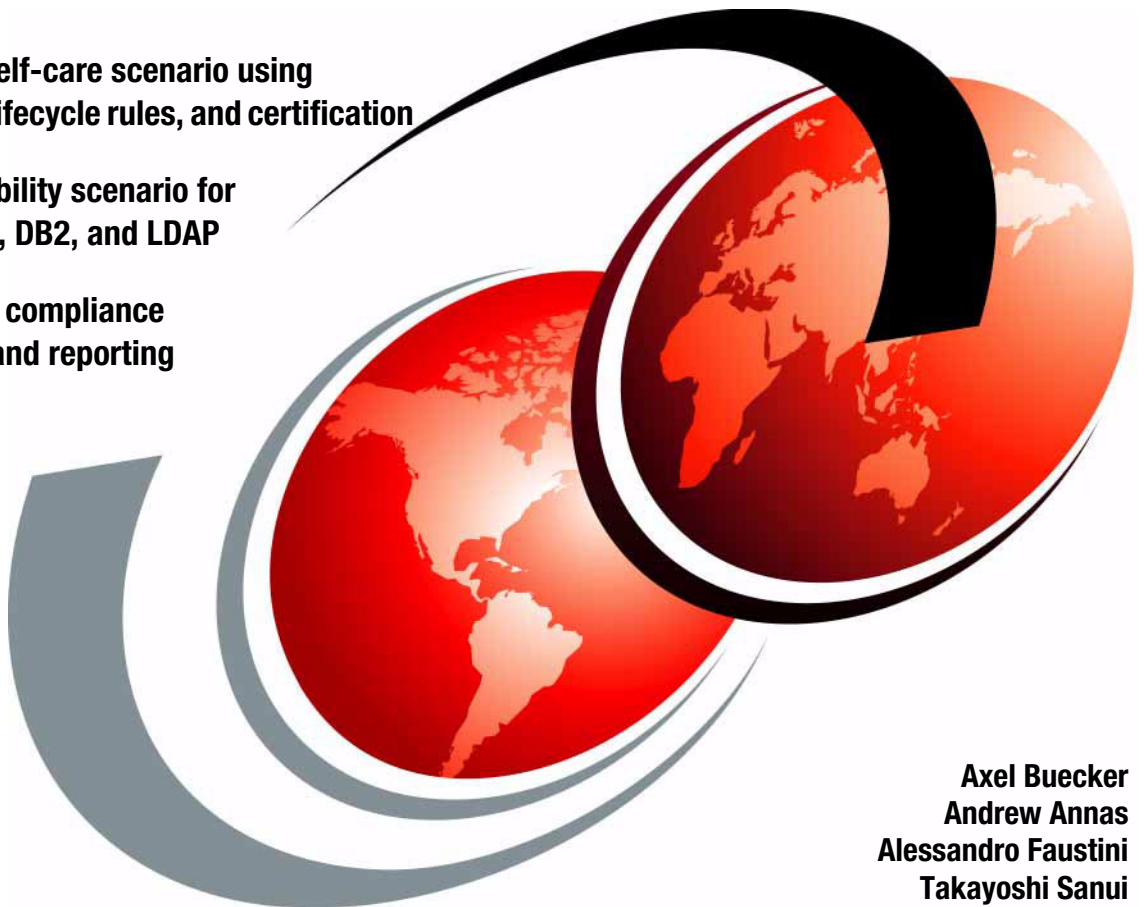
Identity Management Advanced Design

for IBM Tivoli Identity Manager

Complete self-care scenario using
workflow, lifecycle rules, and certification

High availability scenario for
WebSphere, DB2, and LDAP

Addressing compliance
with audit and reporting



Axel Buecker
Andrew Annas
Alessandro Faustini
Takayoshi Sanui



International Technical Support Organization

**Identity Management Advanced Design for IBM
Tivoli Identity Manager**

August 2006

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2006)

This edition applies to IBM Tivoli Identity Manager Version 4.6.

© Copyright International Business Machines Corporation 2006. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii

Preface	ix
The team that wrote this redbook	ix
Become a published author	xi
Comments welcome	xi

Part 1. Advanced Identity Topics	1
---	---

Chapter 1. Advanced design overview	3
1.1 Understanding the system architecture	4
1.1.1 Logical component architecture	4
1.1.2 Web User Interface Layer	5
1.1.3 Applications Layer	7
1.1.4 Services Layer	9
1.1.5 LDAP Directory	11
1.1.6 Database	11
1.1.7 Resource connectivity	12
1.2 Application Programming Interface (API)	13
1.2.1 Application API	15
1.2.2 Authentication API	17
1.2.3 Data Services	19
1.2.4 Logging	39
1.2.5 Mail	40
1.2.6 Policy	41
1.2.7 Password rules	43
1.2.8 Remote Services	46
1.2.9 Workflow	47
1.2.10 FESI extensions	55
1.3 Workflow	55
1.3.1 Script nodes	56
1.3.2 Workflow extensions	56
1.4 Custom service provider	57
1.5 Custom reporting	58
1.6 Conclusion	58

Chapter 2. Architect a high availability solution	61
2.1 Application server	63

2.2 Directory server	67
2.2.1 Manual failover to secondary LDAP	69
2.2.2 Automated failover to secondary LDAP	71
2.3 Relational database.	73
2.3.1 Operating system cluster with DB2 active/standby	74
2.3.2 DB2 mutual takeover multiple partition	74
2.3.3 DB2 High Availability Disaster Recovery (HADR)	75
2.4 Identity Manager adapters.	76
2.4.1 Manual failover to secondary adapter	78
2.4.2 Automated failover to secondary adapter	79
2.4.3 Event notification on an HA adapter configuration	81
2.5 Physical HA component architecture	83
2.5.1 Component configuration and placement	83
2.5.2 Network zones	84
2.6 Security and integrity for high availability	87
2.7 Conclusion.	90

Part 2. Customer Scenario 93

Chapter 3. Tivoli Austin Airlines, Inc. 95

3.1 Company profile	95
3.1.1 Geographic distribution of TAA	96
3.1.2 Organization of TAA	99
3.1.3 HR and personnel procedures	101
3.2 Current IT architecture	102
3.2.1 Overview of the TAA network	102
3.2.2 TAA's e-business initiative	104
3.2.3 Security infrastructure for the e-business initiative	105
3.2.4 Secured e-business initiative architecture.	106
3.2.5 Identity management and emerging issues	109
3.3 Corporate business vision and objectives	110
3.4 Project layout and implementation phases	111

Chapter 4. Project design 113

4.1 Business requirements	113
4.2 Functional requirements	114
4.3 Design approach	117
4.4 Implementation approach	118
4.4.1 Non-functional requirements.	118
4.4.2 Requirement priorities	119
4.4.3 Implementation tasks and efforts	119
4.4.4 Project phases	119

Chapter 5. Technical implementation phase I. 121

5.1 TAA's high availability scenario	121
5.1.1 Requirements	121
5.1.2 TAA's high availability planning	123
5.2 Application server high availability	125
5.2.1 Requirements	125
5.2.2 Design considerations	125
5.2.3 Application server high availability implementation	126
5.3 Relational database high availability	132
5.3.1 Requirements	132
5.3.2 Design considerations	132
5.3.3 Relational database high availability implementation	133
5.4 Directory Server high availability	152
5.4.1 Requirements	153
5.4.2 Design considerations	153
5.4.3 TAA's Directory Server high availability implementation	153
5.5 Conclusion	206
Chapter 6. Technical implementation phase II	209
6.1 Self-care	211
6.1.1 Requirements	211
6.1.2 Design considerations	212
6.1.3 TAA's implementation	212
6.2 Delegated administration	230
6.2.1 Requirements	230
6.2.2 Design considerations	230
6.2.3 TAA's implementation	233
6.3 Advanced custom report design	248
6.3.1 Requirements	249
6.3.2 Design considerations	249
6.3.3 TAA's implementation	251
6.4 Automated operation report delivery	269
6.4.1 Requirements	269
6.4.2 Design considerations	270
6.4.3 TAA's implementation	271
6.5 Recertification process	290
6.5.1 Requirements	290
6.5.2 Design considerations	290
6.5.3 TAA's implementation	291
6.6 Conclusion	309

Part 3. Appendixes	311
-------------------------------------	------------

Appendix A. Corporate policy and standards	313
Standards, practices, and procedures	315

Practical example	315
External standards and certifications	316
Industry specific requirements	317
Product or solution certifications	317
Nationally and internationally recognized standards	318
Legal requirements	318
Summary	319
Appendix B. Source code	321
BulkFeedAdminDomain.java	322
AdminDomainModelExtension.java	327
AbstractExtension.java	330
AbstractExtension.java	333
applicationServlet.java	340
applications.jsp	356
application_sub.jsp	360
todolistServlet.java	363
todolist.jsp	371
mainServlet.java	376
main.jsp	381
Appendix C. Tivoli Directory Server proxy server	385
Appendix D. Additional material	389
Locating the Web material	389
Using the Web material	390
System requirements for downloading the Web material	390
How to use the Web material	390
Glossary	395
Related publications	403
IBM Redbooks	403
Other publications	403
Online resources	404
How to get IBM Redbooks	404
Help from IBM	404
Index	405

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.


COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®

Redbooks (logo) ™

xSeries®

z/OS®

AIX®

DB2 Universal Database™

DB2®

HACMP™

IBM®

Lotus Notes®

Lotus®

Notes®

Redbooks™

RACF®

Tivoli®

WebSphere®

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, JavaBeans, JavaScript, JavaServer, JavaServer Pages, JDBC, JSP, JVM, J2EE, Solaris, Sun, Sun Microsystems, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Active Directory, JScript, Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Mozilla, Firefox, Thunderbird, Mozilla logo, the Firefox logo and the Thunderbird logo are trademarks or registered trademarks of Mozilla in the United States, other countries or both

Other company, product, or service names may be trademarks or service marks of others.

Preface

Identity and user lifecycle management projects are being deployed more and more frequently - and demand is growing. By demonstrating how IBM® Tivoli® Identity Manager can be made resilient and adapted to special functional requirements, this IBM Redbook creates or enhances confidence in the IBM Tivoli Identity Manager-based solution for senior management, architects, and security administrators.

Advanced design topics can start with infrastructure availability for all involved components, Web application and database server clustering as well as LDAP multi-master setups. Advanced care topics can continue with compliance challenges addressing enhanced auditing and reporting, and designing and creating your own self-care and self-registration application environment that embraces external users and business partners, offering fine-tuned workflow options and lifecycle management capabilities.

The powerful features and extensions of IBM Tivoli Identity Manager is opening doors into a world of advanced design and customization for every identity management challenge you might encounter.

The team that wrote this redbook

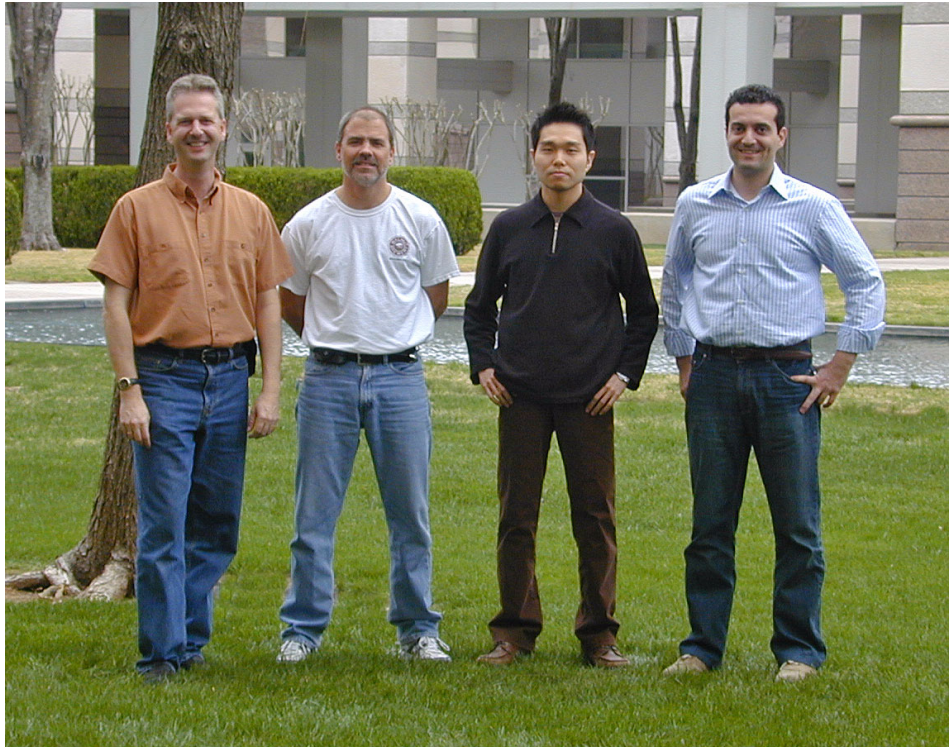
This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Axel Buecker is a Certified Consulting Software IT Specialist at the International Technical Support Organization, Austin Center. He writes extensively and teaches IBM classes worldwide about areas of Software Security Architecture and Network Computing Technologies. He holds a degree in computer science from the University of Bremen, Germany. He has 20 years of experience in a variety of areas related to Workstation and Systems Management, Network Computing, and e-business Solutions. Before joining the ITSO in March 2000, Axel worked for IBM in Germany as a Senior IT Specialist in Software Security Architecture.

Andrew Annas is a Senior IT Specialist with IBM Tivoli Software in the United States. He often teaches classes about deploying and customizing IBM Tivoli Identity Manager. He has five years of experience in identity management and twelve years experience in the deployment of enterprise software applications. He holds a degree in Computer Science from the University of California, Irvine.

Alessandro Faustini is an IT Specialist in IBM Software Group, Italy. He joined IBM in 2000 and has a three-year history in security and systems management solutions. His product experience includes the Tivoli Identity Manager, Tivoli Access Manager, and Tivoli Federated Identity Manager. He holds a degree in Electronic Engineering from University “La Sapienza” in Rome.

Takayoshi Sanui is an IT Specialist in IBM Japan Systems Engineering. He has four years of experience in identity management and IT security. He has also been teaching classes about Tivoli security products in Japan. He holds a degree in Computer Science from the Tokyo Institute of Technology.



From left to right: Axel, Andrew, Takayoshi, and Alessandro

Thanks to the following people for their contributions to this project:

Leslie Parham
International Technical Support Organization, San Jose Center

Alex Amies, Dave Bachmann, Chris Bauserman, Brian Davis, Mark McConaughy, Connie Nelin, Casey Peel, Jeffrey Robke, Thomas Sharp, Jim Sides, James Sun, and Weibo Yuan
IBM US

Gene Kligerman, Dale McInnis, and Vivien Page
IBM Canada

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400



Part 1

Advanced Identity Topics

In this part, we introduce advanced identity management topics, specifically, the Application Programming Interfaces (APIs) of IBM Tivoli Identity Manager 4.6, and what you can achieve by using these APIs as well as high availability scenarios. Identity Manager can handle a multitude of integration aspects and many IT infrastructures, Web portals, and application environments, which we describe in detail throughout this part of the book. After we discuss the advanced topics, Part 2, “Customer Scenario” on page 93 provides a more solution-oriented, scenario-based approach to the topics discussed in this part.



Advanced design overview

This chapter introduces the high-level components and new concepts for the design of advanced identity management solutions.

This chapter provides you with an understanding of the following IBM Tivoli Identity Manager topics:

- ▶ The high-level logical component architecture
- ▶ The various internal modules and subprocesses
- ▶ The high-level physical architecture

1.1 Understanding the system architecture

In order to understand IBM Tivoli Identity Manager system architecture and how to utilize its capabilities, it is important to understand how the architecture is laid out logically. In the following section, we explain the logical components of the Identity Manager architecture.

1.1.1 Logical component architecture

IBM Tivoli Identity Manager can be thought of logically as having two primary areas of functionality: presentation and provisioning. The presentation functionality is represented logically by the Web User Interface, and provisioning can be thought of as the provisioning platform. The provisioning platform can be further logically divided into two more layers of functionality, the applications layer and that of the core services layer. The applications layer acts as an external interface to core services from both the Web User Interface and external applications such as an identity store or other types of applications. In Figure 1-1 on page 5, we depict these three functional areas graphically, showing the Identity Manager API and the two functional areas we have just described.

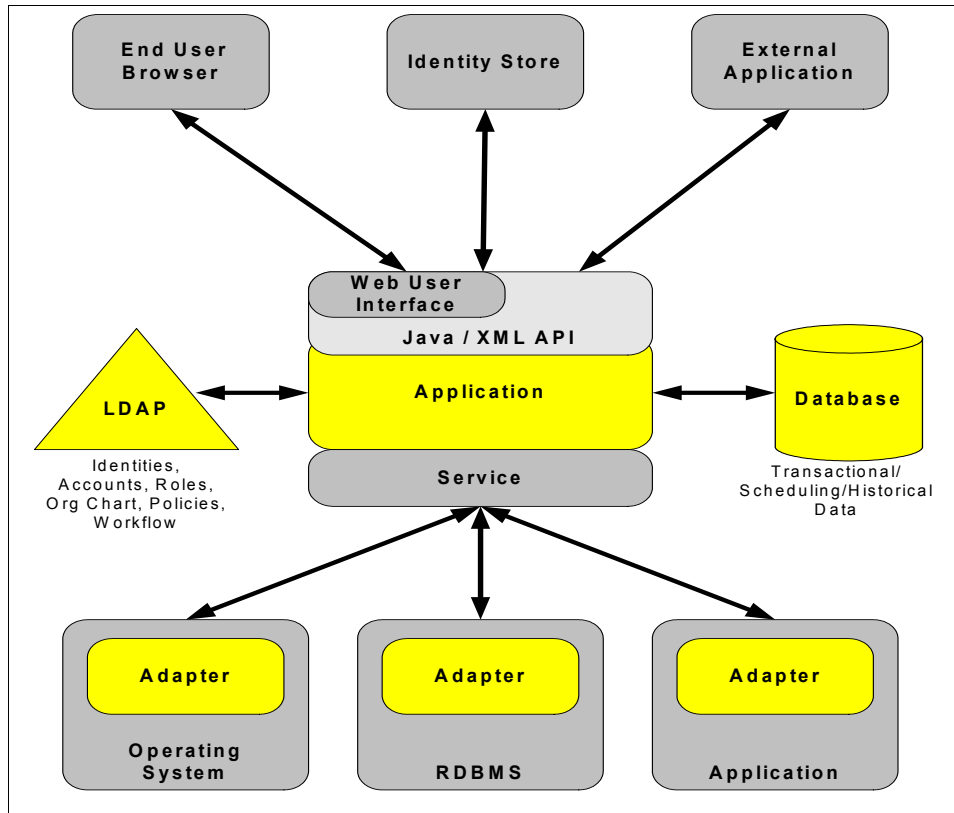


Figure 1-1 IBM Tivoli Identity Manager logical architecture

The following sections cover each individual layer in more detail.

1.1.2 Web User Interface Layer

The Web User Interface is a set of combined subprocesses that provide all the applications of the Applications layer content to a user's browser as well as the initiation of applets (both run on the client and the server), such as the Workflow Design and the Form Creation.

Figure 1-2 on page 6 is a graphical representation of the Web User Interface followed by a description of each of the modules shown.

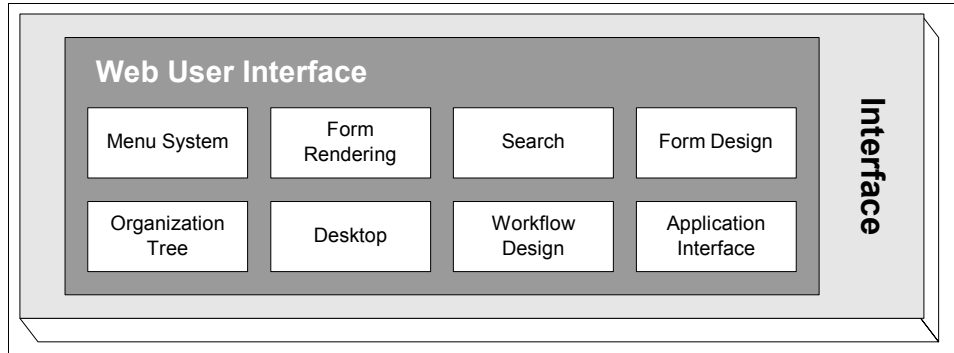


Figure 1-2 Web User Interface module subprocesses

Menu System

The Menu System module provides a consistent menu and shortcut mechanisms that are used for navigation throughout the user interface.

Form Rendering

The Form Rendering module provides the run-time interpreter to display the customized forms designed in the Form Design module.

Search

The Search module provides a framework for general and more specific search interfaces to be used throughout the user interface.

Form Design

The Form Design module provides a near WYSIWYG (What You See Is What You Get) user interface design environment for customizing the forms that display information about the entities managed through the user interface. This module makes use of applets for its more flexible demands.

Organization Tree

The Organization Tree module provides the graphical tree representation of the organizational structure in which entities managed through the user interface are logically stored.

Desktop

The Desktop module provides a framework for providing a consistent layout in the pages displayed in the user interface. It is within this framework that products of the other Web User Interface modules are displayed, such as the Menu System and Organization Tree.

Workflow Design

The Workflow Design module provides a graphical workflow design environment. A workflow process consists of a set of activities that are executed in an ordered fashion according to conditional transitions. The designer provides a graphical way of defining such a workflow process. This module makes use of applets for its more flexible demands.

Application Interface

The Application Interface module consists of all application-specific user interface components. For example, the interfaces required to create a provisioning policy or an account are organized into this module. This module makes use of other modules in the Web User Interface subsystem, such as the Form Rendering and Search modules.

1.1.3 Applications Layer

The applications layer is the remoteable external interface used to access all publicly available provisioning functions.

The Applications subsystem contains all modules that provide provisioning specific capabilities, such as identity management, account management, and policy management. Each application makes use of the core services in the Services layer to achieve its goals. It is the Applications module that provides the external interface to the provisioning platform. Figure 1-3 is a graphical representation of the Applications Interface followed by a description of each of the modules shown.

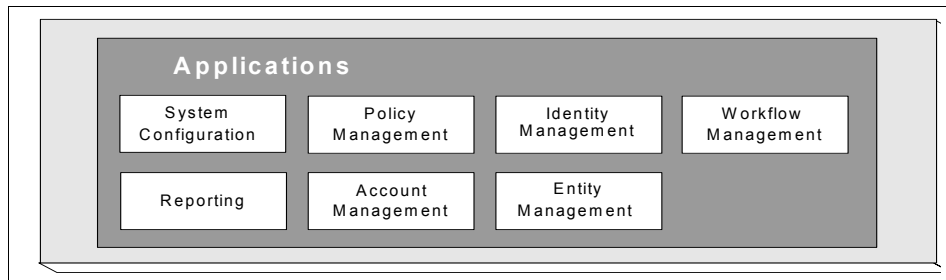


Figure 1-3 Applications module subprocesses

System Configuration

The System Configuration module provides the capabilities required to manage the IBM Tivoli Identity Manager system, such as defining behavioral properties.

Policy Management

The Policy Management module provides the capabilities to manage the policies in the system, including provisioning, password, service selection, and identity policies.

Identity Management

The Identity Management module provides the capabilities required to manage identities, such as their addition, removal, suspension, reinstatement, transferal, and modification, including the changing of roles. The definition of roles, including dynamic roles, is also included in this module.

Workflow Management

The Workflow Management module provides the capabilities required to manage workflow processes, such as their addition, modification, and removal. The ability to view the status and details of active and historical processes is also provided in this module.

Reporting

The Reporting module provides the canned report capabilities of the system. This module provides the query and formatting of the reports driven from the user interface.

Account Management

The Account Management module provides the capabilities required to manage accounts, such as their addition, removal, suspension, reinstatement, and modification.

Entity Management

The Entity Management module provides the capabilities required to manage the types of entities managed by the system, such as types of identities and accounts. This includes the ability to define the schema for the entity type, the operations the entity type can support, and the lifecycle of the entity type.

Note: Sitting between the Web user interface and the Applications layer in Figure 1-1 on page 5 is the public Java™ API. This API provides a set of Java classes that abstracts the more commonly used functions of the provisioning platform, such as identity management, password management, and account management. The classes that make up this API are the same classes the Identity Manager product uses for its out-of-the-box user interface.

For more information, refer to documentation provided with the Applications API in the <ITIM_HOME>/extensions/doc/applications directory.

1.1.4 Services Layer

The Core Services subsystem contains all modules that provide general services that can be used within the context of provisioning, such as authentication, authorization, workflow, and policy enforcement. These services often make use of other services to achieve their goals. Figure 1-4 is a graphical representation of the Services Interface followed by a description of each of the modules shown.

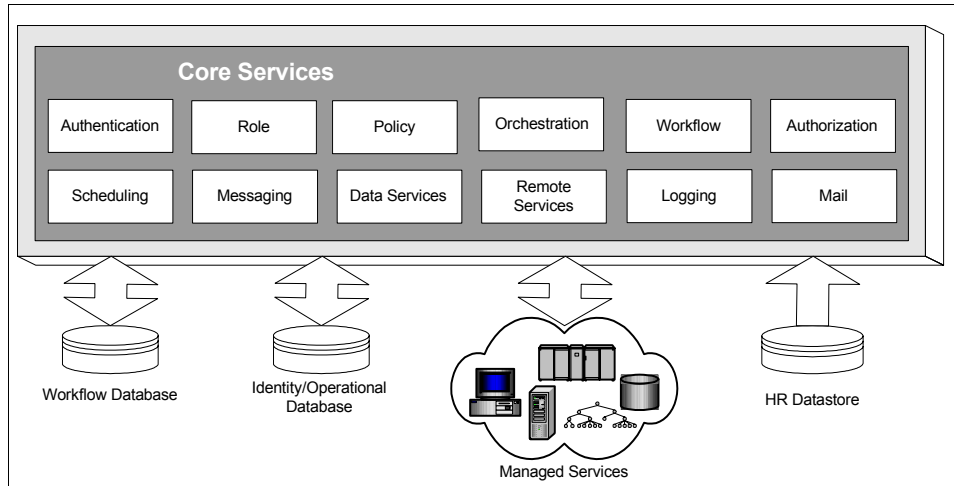


Figure 1-4 Core Services module subprocesses

Authentication

The Authentication module provides a set of authentication implementations that can be used by clients of the service. Examples of these implementations are simple password authentication and X.509 certificate authentication. The module is designed as a framework that can be extended by customers to provide their own implementations.

Role

The Role module evaluates dynamic memberships to roles. This module is called upon when an identity or dynamic role definition changes to identify which identities should be members of dynamic roles.

Policy

The Policy module enforces the policies that associate users with services. The module ensures that provisioning requests conform to the policies that are defined. The module resolves the appropriate policies that apply to a user and determines the services for which that user is authorized. The module validates

and generates passwords. The module generates identities for users and accounts.

Orchestration

The Orchestration module provides a coordination service for extensible operations that are performed on entities and manages the lifecycles of those entities. For instance, the orchestration module provides an abstraction layer to the Account Management application for executing the steps needed to provision an account of a given type. Regardless of the steps involved, which could be customized or changed, the Account Management module would always use the same interface to the Orchestration module.

Workflow

The Workflow module executes and tracks transactions within the system. This would include the provisioning and de-provisioning of a service, a user's status change, the custom process associated with a provisioning request in the system, or any other transaction that affects a user's, or group of users', access to services. Each of these transactions is persistent for fault-tolerant execution and historical auditing purposes. Clients can query the Workflow module for the status of the transactions being executed.

Authorization

The Authorization module provides an interface to enforce authorization rules as clients attempt operations in the system. These rules apply to accessing data within the system, as well as to operations that can be applied to the system data.

Scheduling

The Scheduling module provides a timer that notifies *clients* of timed events for which they are subscribed. The Scheduling module uses the Messaging module to notify those clients.

Note: The *clients* discussed in this section are internal to Identity Manager. For example, workflow is a client to scheduling; it uses scheduling to allow workflows to start at a later date instead of immediately.

Messaging

The Messaging module provides guaranteed asynchronous messaging to and between internal modules in the architecture. The module relies heavily on the Java Message Service (JMS) specification to provide support for multiple messaging middleware vendor implementations.

Data Services

The Data Services module provides a logical view of the data in persistent storage (LDAPv3 directory) in a manner that is independent of the type of data source that holds the data. The model abstracts the details of the stored data into more usable constructs, such as Users, Groups, and Services. The model also provides an extendable interface to allow for customized attributes that correspond to these constructs. Metadata information about the persistent data can also be retrieved using this module.

Remote Services

The Remote Services module provides the interaction with the external systems for provisioning and de-provisioning services. The synchronization of service information and user information is also performed within this module. The module is designed as a framework that can be extended by customers to provide their own implementations of provisioning and de-provisioning of services. This allows the platform to easily support different protocols and APIs that may be supported by the resources to be provisioned.

Logging

The Logging module provides a common logging interface to all other modules.

Mail

The Mail module provides an interface for notifying users via a messaging system, such as e-mail. The module is configurable to accommodate different messaging systems.

1.1.5 LDAP Directory

The IBM Tivoli Identity Manager system uses an LDAPv3 directory server as its primary repository for storing the current state of the enterprise it is managing. This state information includes the identities, accounts, roles, organization chart, policies, and workflow designs.

More details about the LDAP Directory and its schema are available in the *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6*, SC32-1769.

1.1.6 Database

A relational database is used to store all transactional, reporting, and schedule information. Typically, this information is temporary for the currently executing transactions, but there is also historical information that is stored indefinitely to provide an audit trail of all transactions that the system has executed.

More details about the database and its schema are available in *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6, SC32-1769*.

1.1.7 Resource connectivity

The back-end resources that are being provisioned by IBM Tivoli Identity Manager are generally very diverse in their capabilities and interfaces. The IBM Tivoli Identity Manager system itself provides an extensible framework for adapting to these differences in order to communicate directly with the resource. For a more distributed computing alternative, a built-in capability to communicate with a remote adapter is provided. The adapters typically use an XML-based protocol, either Directory Access Markup Language (DAML) or Directory Service Markup Language (DSML Version 2), as a communications mechanism.

Directory Access Markup Language connectivity

DAML is a proprietary XML message format used when communicating with one of IBM Tivoli Identity Manager's standalone adapters. These adapters are programs installed on either the managed resource or on a host that can manage the resource through a remote administration API.

DAML is a simple XML schema definition that enables the encoding of identity information in the form of an XML document so that it can be easily shared via IP protocols such as HTTPS, as shown in Figure 1-5.

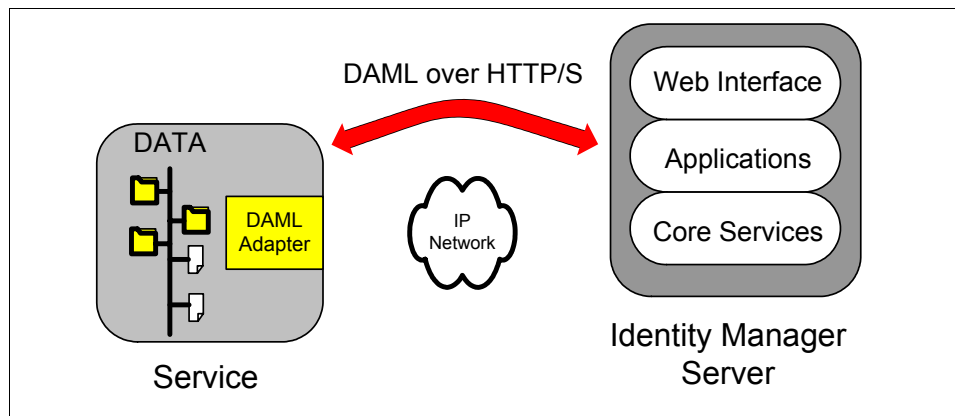


Figure 1-5 DAML connectivity to a service

Transactions from the IBM Tivoli Identity Manager Server are sent securely via HTTPS to the service adapter and then processed by the adapter.

For example, if a service has just been connected to the IBM Tivoli Identity Manager Server, the accounts that already exist on the server can be reconciled

or pulled back in order to import the users' details into the IBM Tivoli Identity Manager LDAP directory. If a password change or a provisioning of a new user occurs, the information is transferred to and then processed by the adapter. The adapter deposits the new information within the application or operating system that is managed.

Directory Service Markup Language connectivity

DSMLv2 is an industry standard XML message format for the representation of directory data and operations. There are no standard IBM Tivoli Identity Manager adapters that use DSMLv2. Instead, DSMLv2 is used in conjunction with IBM Tivoli Directory Integrator to create custom adapters.

Directory Integrator provides an easy and flexible way to link IBM Tivoli Identity Manager to a wide variety of managed resources. Directory Integrator offers connectors that can be used to manage data in files, directories, databases, message queues, and many other data sources. It allows you to define, using simple scripts, how DSMLv2 operations issued by IBM Tivoli Identity Manager should be translated into operations on the managed resource, as shown in Figure 1-6.

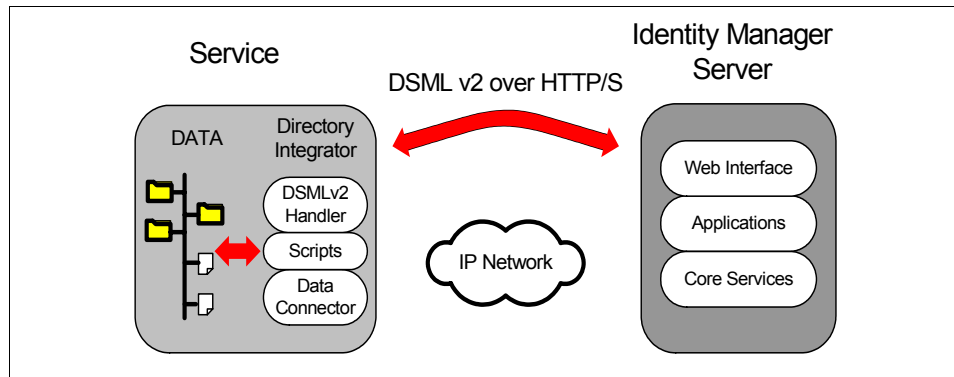


Figure 1-6 DSMLv2 service communication using IBM Tivoli Directory Integrator

1.2 Application Programming Interface (API)

The API provided for the system is organized into two categories based on their programming language, Java and JavaScript™. The *JavaScript API* is used directly by users of the graphical user interface when defining rules in the system, such as with identity policies, provisioning policies, and workflow designs.

The *Java API* can be further subdivided into two more categories, an API for interfacing with the system and an API for extending the system's behavior. In other words, the *interface API* allows external programs to direct the provisioning system to perform some actions or to simply query the provisioning system for some information. The *extension API* allows custom Java code to be called from the provisioning system during its regular flow within the same context (and JVM™).

Since the interface API can be called by any program that is out of the control of the provisioning system, a strict security layer is built into it. A JAAS implementation is provided and required to be used by clients to authenticate against the Identity Manager user store. Once authenticated, the JAAS principal must be authorized to execute each API against the requested objects in the system's data model. This is identical to the security provided by the graphical user interface of the system. The *interface API* consists of the following packages already described:

- ▶ Applications
- ▶ JAAS
- ▶ Identity
- ▶ Provisioning
- ▶ Workflow
- ▶ Search

The extension API can only be called by the provisioning system, or at least within the same application server. The API is not remoteable. This API is also used within the provisioning system itself. The functions provided by this API must be extremely fast and have very little overhead. Due to its restricted use and overhead requirements, this API does not have a strict security model. The *extension API* consists of the following packages already described:

- ▶ Authentication (all)
- ▶ Data Services
- ▶ Model
- ▶ Domain
- ▶ Logging
- ▶ Mail
- ▶ Workflow
- ▶ Password Rules (all)
- ▶ Remote Services
- ▶ Provider
- ▶ Workflow
- ▶ Applications
- ▶ Model
- ▶ Provisioning
- ▶ Query

We discuss each one of these API categories in more detail in the following section.

1.2.1 Application API

The *Application package* represents the Application subsystem of the System Architecture. This package contains classes that aggregate the capabilities provided in the Services subsystem into more abstract provisioning specific applications, such as those required to manage passwords or accounts. This package provides the external interface to the provisioning platform for presentation-oriented clients to use. The applications provided within this package enforce security for the platform by authenticating all clients and authorizing all operations.

There are two major areas of distinction in the Application package. One area contains packages that are not only used by the Identity Manager user interface tier, but also provided as a public interface, Application Programming Interface (API). The second area contains packages, which are used solely by the Identity Manager user interface, and the second area is not provided as a public interface.

For organization purposes, the public API portion of the Application package has been broken down into five sub-packages, *Identity*, *Provisioning*, *Search*, *Workflow*, and *JAAS*. The *Identity package* represents the Identity Management module in the Applications subsystem of the System Architecture. It contains the classes that provide the capabilities to manage identities, or people, including their placement in an organizational chart and their categorization through roles.

The *Provisioning package* represents the Account Management module in the Applications subsystem of the System Architecture. At the base level, the Provisioning package contains the classes that provide the capability for provisioning the identities to external resources via accounts. This also includes the management of passwords on those resources. Both packages have a dependency on the Identity package.

The *Workflow package* represents the Workflow Management module in the Applications subsystem of the System Architecture. It contains the classes that provide the auditing and management capabilities of workflow processes from the user's perspective. This package provides an authorization layer over the capabilities of the Workflow package in the Services subsystem. This package also has a dependency on the Identity package to provide an identity's workflow assignments.

The *Search package* contains the classes that provide general search capabilities over any of the objects managed in the system (identities, accounts,

and so on). This package does not, however, have a dependency on the other packages listed because it interacts directly with the Data Services architectural module for these managed objects.

The *JAAS package* contains JAAS extensions to provide an authentication mechanism for users of the Application package within the JAAS framework. All other Application packages have a dependency on this package for identifying the authenticated user. The JAAS package utilizes the Authentication service to implement the user authentication. Below in Figure 1-7 is a graphical representation of these packages and how they interact.

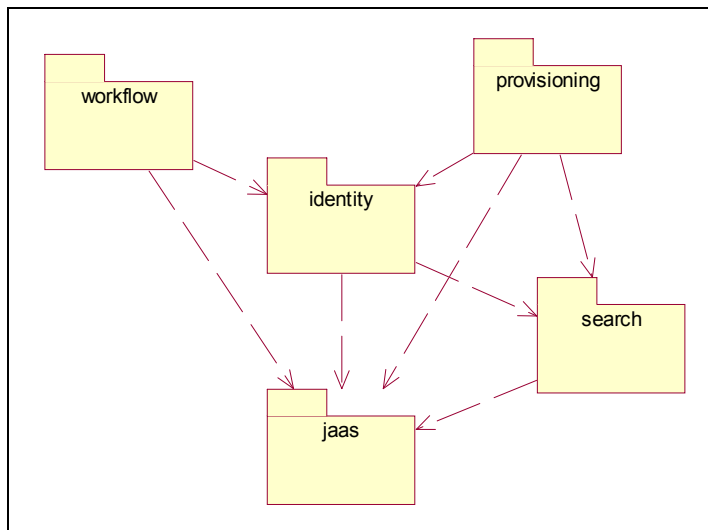


Figure 1-7 Application Public Interface Package Diagram

The root Application package provides general use classes that can be used by all sub-packages when implementing specific provisioning features. Below in Figure 1-8 on page 17 is a diagram of the Application class.

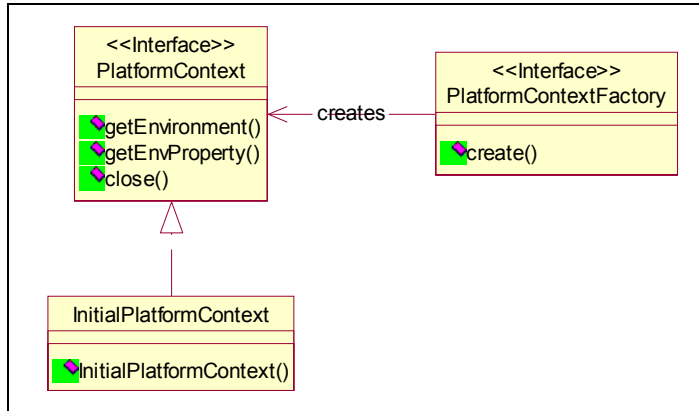


Figure 1-8 Application class diagram

The *PlatformContext* represents a connection to the provisioning platform. This *PlatformContext* is an interface that can be implemented differently depending on the deployment of the provisioning platform. The different implementations are constructed using a corresponding implementation of the *PlatformContextFactory*.

To help keep the client simple, the *InitialPlatformContext* class provides an implementation of the *PlatformContext* interface that constructs the proper *PlatformContext* implementation to communicate with the provisioning platform using the correct *PlatformContextFactory*. The client merely has to reference the correct factory class name during the *InitialPlatformContext's* construction.

1.2.2 Authentication API

The *Authentication package* represents the Authentication module within the Services module of the System Architecture. This package contains the classes needed to authenticate users of the system using an extensible framework to allow custom authentication mechanisms to be put in place. A default implementation for password credentials checked against the platform's data store is provided.

The Authentication package holds the classes that provide authentication services for clients. The design of this package is based on a framework to allow custom authentication implementations to be registered with the system. Below in Figure 1-9 on page 18 is a graphical representation of these classes and how they interact.

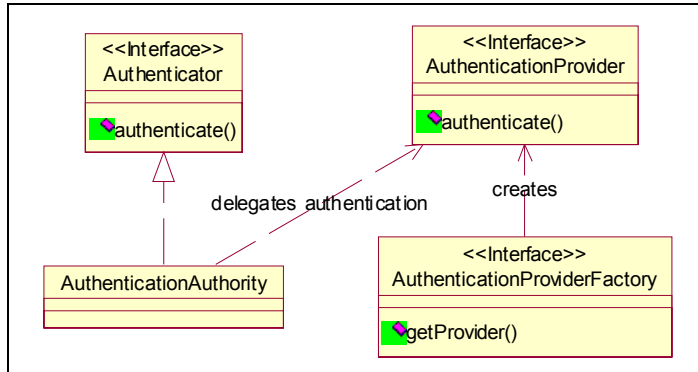


Figure 1-9 Authentication class diagram

The *Authenticator interface* provides the primary method signature for clients to request authentication of an identity using a set of credentials. The *AuthenticationAuthority* class implements that interface with a flexible implementation of delegating the implementation of the authentication to an *AuthenticationProvider* implementation. This bridge pattern allows for the *Authenticator* client interface to be specialized independently of the implementation mechanisms. An *AuthenticationProviderFactory* is used by the *AuthenticationAuthority* to construct the provider.

System

The *System package* is a sub-package of Authentication that provides the Identity Manager specific implementation of the *Authenticator interface* that provides the ultimate flexibility in both selecting implementations of authentication mechanisms as well as the number of authentication types that are supported (for example, user certificates, user passwords, distributed agent authentication, and so on). See Figure 1-10.

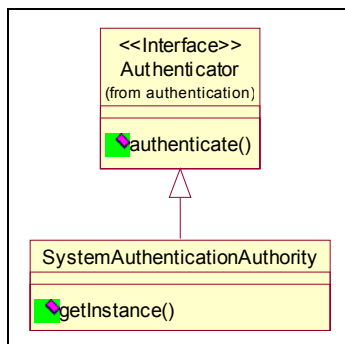


Figure 1-10 System authentication class diagram

The *SystemAuthenticationAuthority* is a singleton class that implements the *Authenticator interface* to provide the global authentication service to the provisioning platform. It is flexible in that it reads from a configuration file the types of authentication needs required by its clients and the implementations of those authentication needs. This allows for additional applications to be developed that can have new authentication needs met very simply without affecting the implementation of this module, as well as for deployment choices on how to implement the authentication (for example, using a Windows® NT domain controller as the authentication store).

1.2.3 Data Services

The *Data Services package* holds the object-oriented logical representation of the system's subject data held within persistent storage. The goal of the package is to provide an intuitive, flexible, and efficient interface to the data repository without revealing the details of the storage mechanism or format to the package's client. This package is further divided into the two packages, Model and Schema.

The *Model package* itself is quite large. For purposes of organization, the Model package has been divided into several sub-packages, Domain, Form, Policy, Workflow, and System. The *Policy package* holds the data objects relevant to policies. The *Workflow package* holds the data objects relevant to workflow designs. Since policies reference services, roles, and workflow designs, the Policy package has a dependency on the Domain package. The *Domain package* holds the data objects relevant to an organization, such as locations, people, services, and roles. The *System package* holds the data objects relevant to Identity Manager users specifically, such as Identity Manager accounts and groups for authorization. Since an Identity Manager account must be traced back to its owner, the System package has a dependency on the Domain also.

The Model and Schema packages represent the modules defined in the System Architecture for the Data Services subsystem. The Model package provides a thin abstraction layer above the data store for all persistent objects managed by the system. The Model package has a dependency on the Schema package for obtaining schema information needed to efficiently access system data from persistent storage. See Figure 1-11 on page 20.

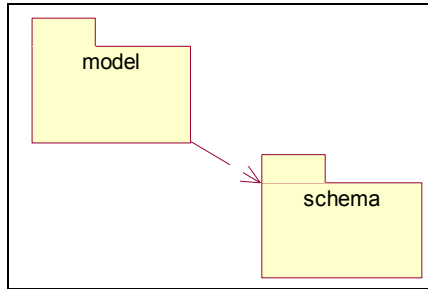


Figure 1-11 Data Services Package Diagram

Schema

The *Schema package* contains a set of objects that represent the schema of the data store. This interface provides clients the ability to query different aspects of classes and attributes in the data store, which is particularly useful in this system, which allows customers to use their own class and attribute definitions within the data model.

The *SchemaAttribute* class represents attribute information. This class can be retrieved independently. In addition to the standard LDAP schema information about an attribute, additional information, such as whether the attribute is enumerated and what those enumerations are, is provided by the *AttributeConstraint* class, which can be obtained from the *SchemaAttribute*.

Model

The *Model package* holds the system's subject data specific to the managed domain. This includes domain information such as the organization, people, services, accounts, and policies. Since there are a large number of objects in the Domain Model package, the Model package has been further subdivided into several other packages. At the base Model package level, some objects have been defined that can be used across all objects within the Model sub-packages.

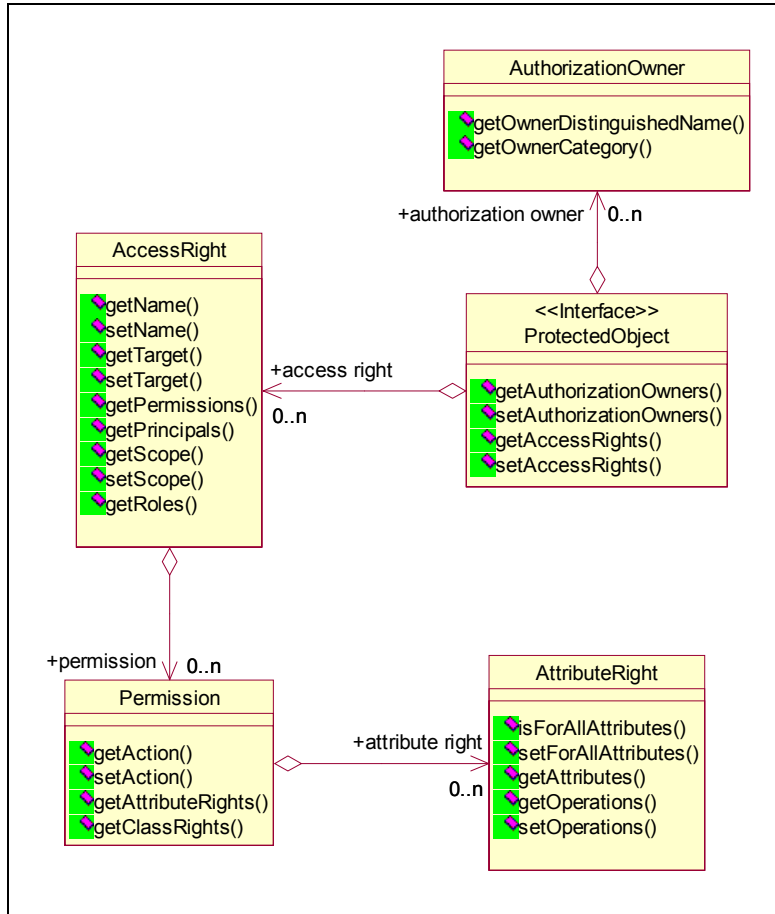


Figure 1-12 ProtectedObject class diagram

The *ProtectedObject*, depicted in Figure 1-12, is the base interface for all data objects that are protected through the use of access control information (ACI). This access control information is stored using the *AccessRight* class and its aggregates. The *AccessRight* objects are used by the Authorization package to determine whether users of the system are authorized to read or modify the associated data objects. The *AuthorizationOwner* class identifies the users of the system who are allowed to make changes to the *AccessRight* objects, therefore, changing the authorization model for the associated data objects.

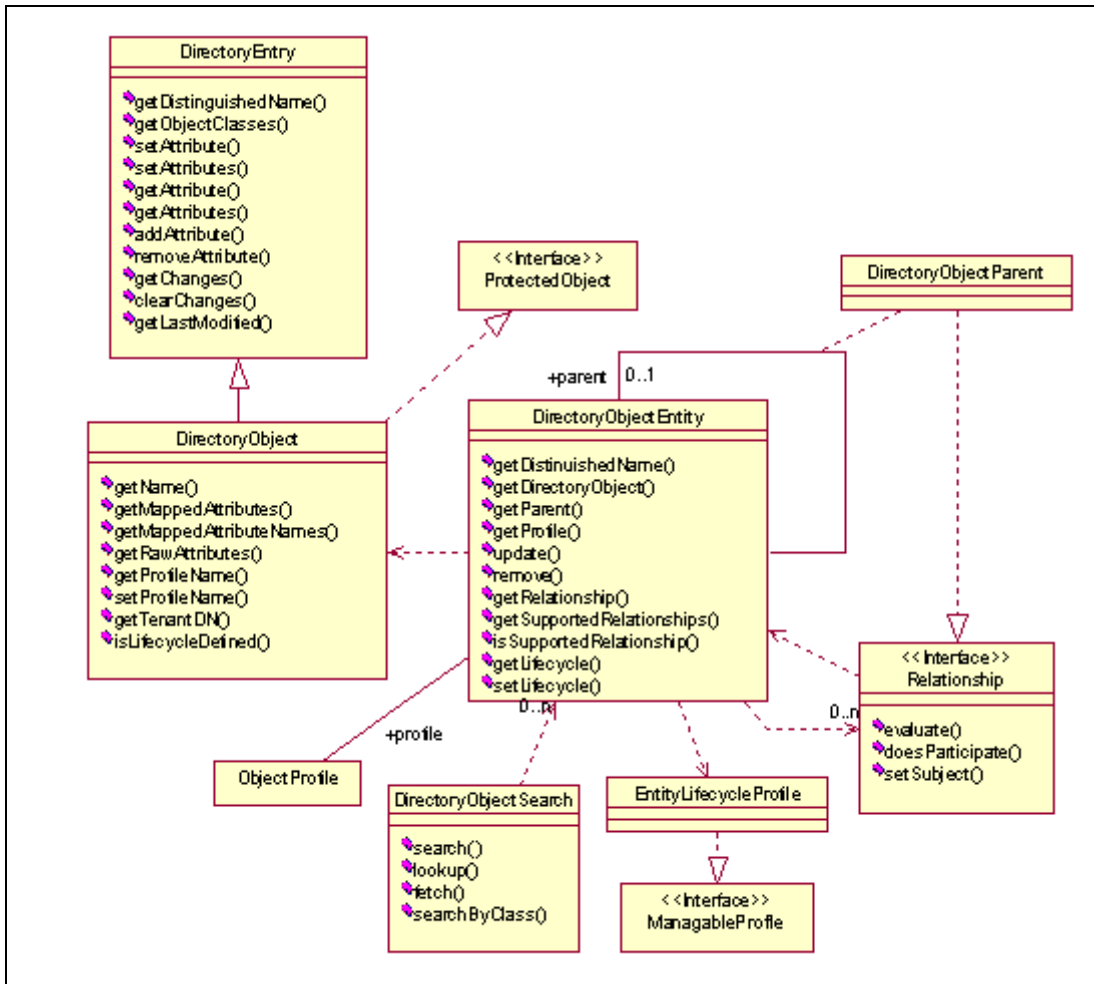


Figure 1-13 *DirectoryObject* class diagram

The *DirectoryObjectEntity*, shown in Figure 1-13, is the base class for all data objects that can be represented with a flexible attribute name/value pair retrieval and update approach. The interface provided easily allows the data storage representation of these objects to be extended without affecting the software interfaces. This class can be instantiated on its own if clients wish to retrieve data objects in a general fashion, or more commonly, this class is inherited from more specialized semantically meaningful classes found in the Model sub-packages.

The entity can have an optional *EntityLifecycleProfile*. The *EntityLifecycleProfile* holds all lifecycle characteristics that are defined at the entity level versus the *ObjectProfile* level. This class is used specifically for

workflow-based policy enforcement. Only *ServiceEntities* (see below) have *EntityLifecycleProfiles* in Identity Manager 4.6. The *DirectoryObject* is the base interface for all Value Objects that are provided by entities that implement the *DirectoryObjectEntity* interface. The *DirectoryObject* is a subclass of *DirectoryEntry*, which holds the interface for a raw directory entry. The *DirectoryObject* extends *DirectoryEntry* for the purposes of representing a semantic directory entry in the platform's data model. This requires the ability to map semantic attributes of the platform to customer-supplied raw directory attributes. The *isLifecycleDefined()* method is new in Identity Manager 4.6, so the client can quickly determine if the entity has any lifecycle characteristics. Each *DirectoryObjectEntity* has an *ObjectProfile* associated with it which defines how the data object fits within the system's semantics needed for provisioning business logic.

To provide support for a flexible data model where relationships can be added and queried at run time by clients, the concept of *relationships* is introduced. The *Relationship* interface defines the base requirements for all discoverable relationships in the data model. For example, simply querying a *DirectoryObjectEntity* for a relationship called "parent" can discover the relationship implementation *ContainedEntityParent*. An instance of the *DirectoryObjectEntity* class returns a *ContainedEntityParent* instance, but the client can perform an evaluation of this relationship using the interface provided by the *Relationship* object. Specializations of *DirectoryObjectEntity* might return a different implementation for the "parent" relationship, but to clients, the implementation difference is transparent.

There is one supporting search class, *DirectoryObjectSearch*, to the *DirectoryObjectEntity*. The *DirectoryObjectSearch* class provides the interface for retrieving *DirectoryObjectEntities* by either distinguished name or a filter in the RFC 2256¹ format.

¹ For details about RFC 2256, check <http://rfc.net/rfc2256.html>

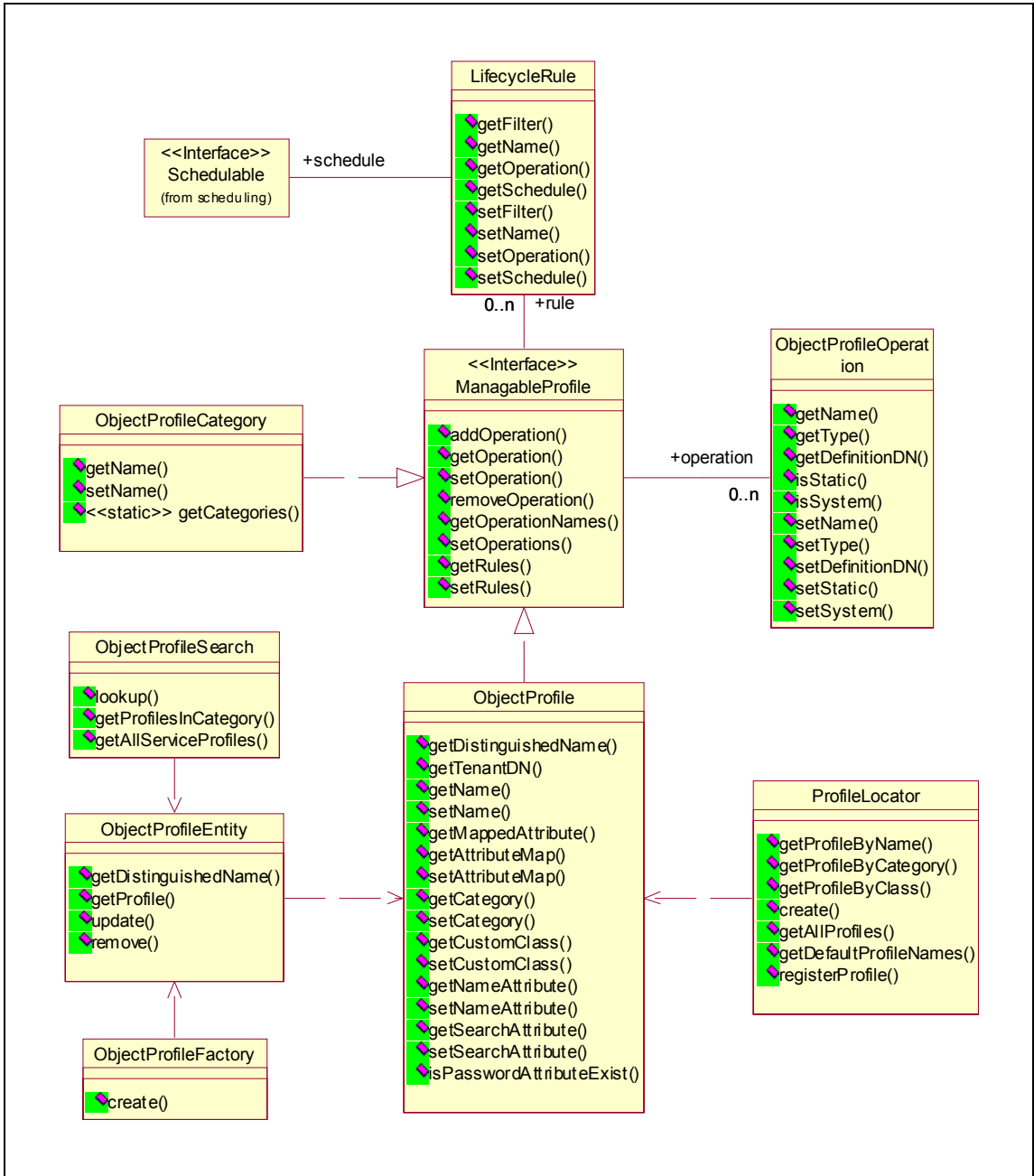


Figure 1-14 ObjectProfile class diagram

The *ObjectProfileEntity* class, depicted in Figure 1-14 on page 24, provides metadata information about how data store object classes represent semantic data model entities within the provisioning context of the system. For example, the system allows a customer to use their own LDAP class, Employee, to represent people within the system. A profile is set up that associates that class with the Person entity and maps the appropriate attributes of the Employee class to the semantic personal attributes, such as name, mail, and shared secret. The *ObjectProfile* class represents the Value Object for this *ObjectProfileEntity*. The *ObjectProfile* implements the *ManageableProfile* interface, which defines the common properties of profiles that have operations. The *ObjectProfileOperation* represents a lifecycle operation, which is a named business task implemented using a workflow process design. The *LifecycleRule* class represents a lifecycle rule, which is made up of a filter and an operation. The *ObjectProfile* and the *ObjectProfileCategory* both implement this interface because they both can have operations defined for them. An *ObjectProfile* is also defined by belonging to an *ObjectProfileCategory*.

The two supporting classes for *ObjectProfileEntities* are the *ObjectProfileSearch* and *ObjectProfileFactory* classes. The *ObjectProfileSearch* provides an interface to retrieve an *ObjectProfileEntity* by distinguished name or by category, since that name is unique within the scope of a single tenant within the system. The *ObjectProfileFactory* provides the interface to create a new *ObjectProfileEntity* in the data store.

Due to the frequency at which clients need to query *ObjectProfile* information, a caching strategy is used. The *ProfileLocator* provides fast access to *ObjectProfiles* based on predefined queries. The cache of *ObjectProfiles* that the *ProfileLocator* uses is refreshed on a customer-defined time interval. Changes made to the profiles are not loaded into the cache until that time interval.

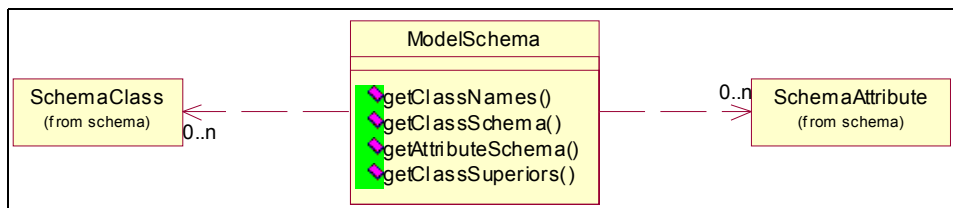


Figure 1-15 Schema class diagram

Although the Schema package, as described earlier, provides an interface for obtaining schema information about the data store, clients of the data model often require this schema information, but with the semantics of the system's data model intact. For example, customers can map existing classes in the data store to an entity in the system's data model. The object profile holds the

mapping information of the customer's class schema to the semantic schema information the system requires for its business logic. The *ModelSchema* class, shown in Figure 1-15 on page 25, is introduced here to execute these mapping rules before providing the schema information to the client so that the information is consistent with the system's business logic. The *ModelSchema* class also filters out attributes from classes that are used only internally by the system.

Domain

The *Domain package* is a sub-package of the Model package that holds the system's subject data specific to the customer environment, such as people, roles, services, and the organizational structure this data is placed in.

The organizational structure of the items within the data model is represented by a handful of classes as shown in Figure 1-16.

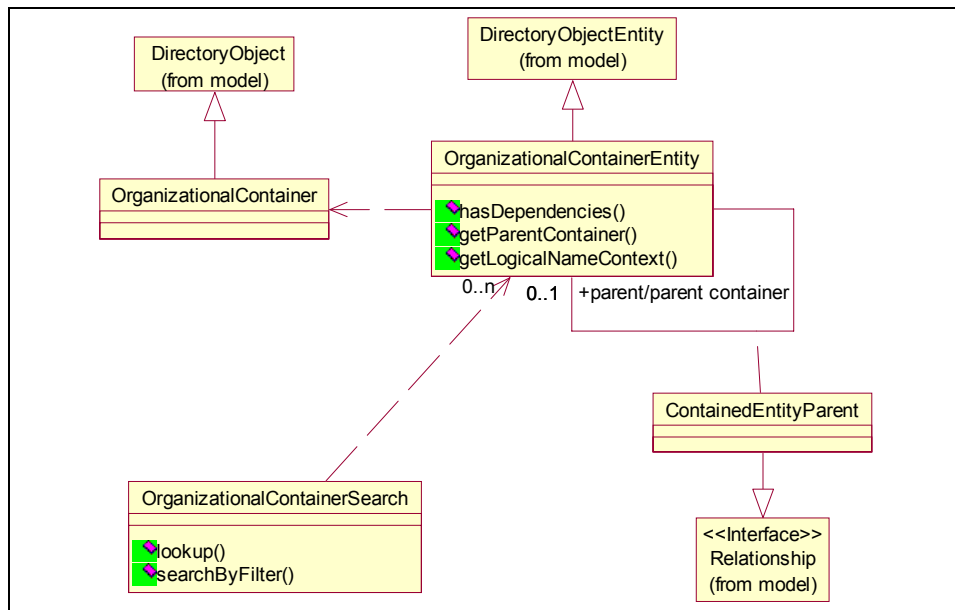


Figure 1-16 *OrganizationalContainer* class diagram

First, the *OrganizationalContainerEntity* class provides a base interface for all containers in the organizational structure. This class provides the interface for traversing the tree through parent relationships, as well as an interface for querying for dependent entities. The specializations of this class are the *AdminDomainEntity*, *BusinessUnitEntity* (represents locations and organizational units), *BusinessPartnerOrgEntity*, and *OrganizationEntity*.

The *DirectorySystemEntity* class, shown in Figure 1-17, represents the root of all domain entities in the system or the root tenant of all domain entities if the system is deployed as multi-tenant.

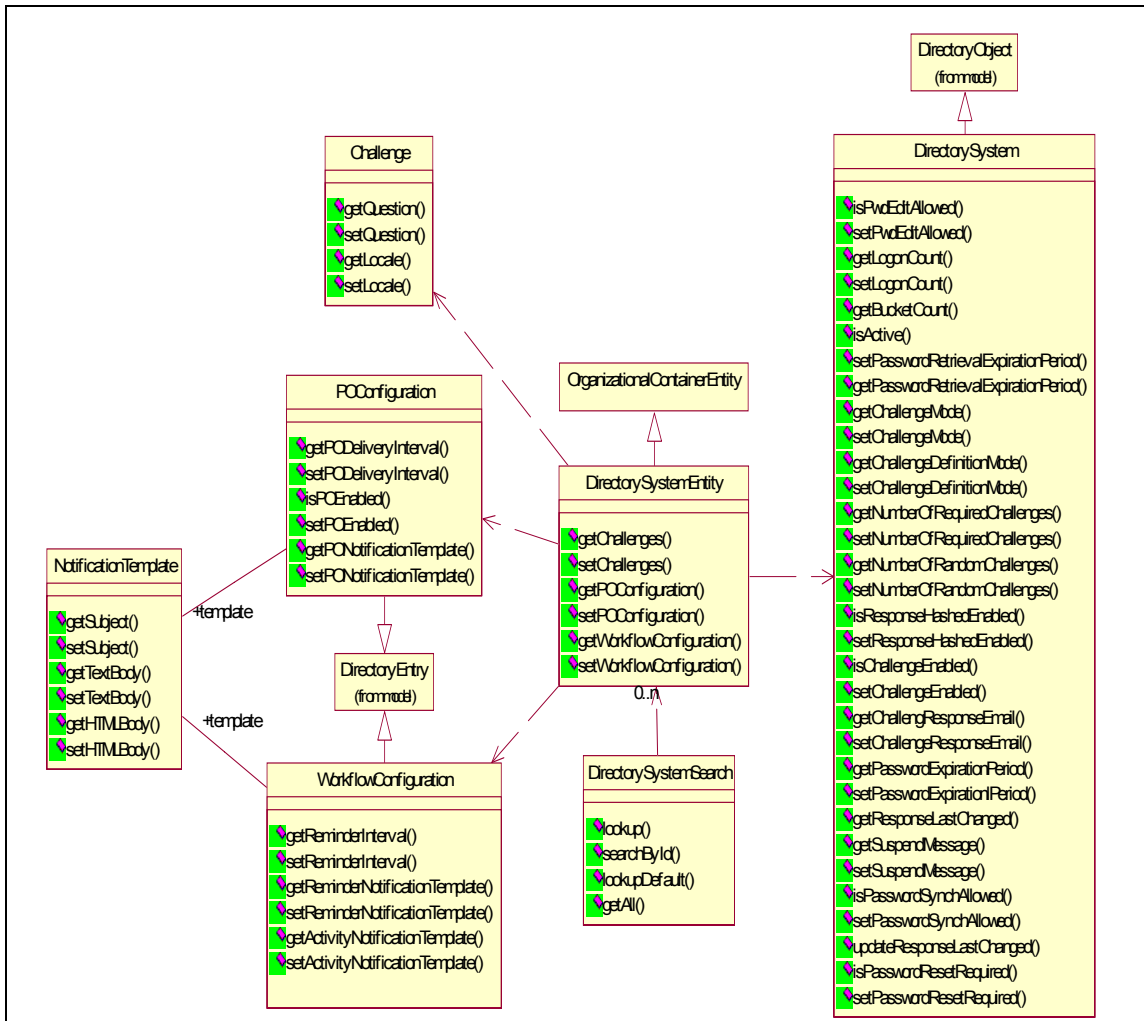


Figure 1-17 *DirectorySystem* class diagram

This entity holds the system-wide, or tenant-wide, properties. The *DirectorySystem* class is the value object class for *DirectorySystemEntity*. The *isPasswordResetRequired()* and *setPasswordResetRequired()* methods replace the *isLostPwByMail()* and *setLostPwByMail()* methods in Identity Manager 4.6 for the challenge/response login flow. The *Challenge* class is new in Identity Manager 4.6 so that a locale can be associated with the system-defined

challenges. The *NotificationTemplate* class is new in Identity Manager 4.6 to describe the template used for messages delivered to users for the Notification Post Office, To do Item Reminders, and Manual Activity Notification customization. The *POConfiguration* class is a value object holding post office configuration information. The *WorkflowConfiguration* class is a value object holding global workflow configuration information. The *DirectorySystemSearch* class provides a search capability. The only two search options provided are by distinguished name or by a unique relative name, the id.

The *BusinessUnitEntity*, depicted in Figure 1-18, represents both a *Location* and an *OrganizationalUnit* in the organization hierarchy. The entity abstracts organizational containers with no other distinction other than they all support being assigned a supervisor.

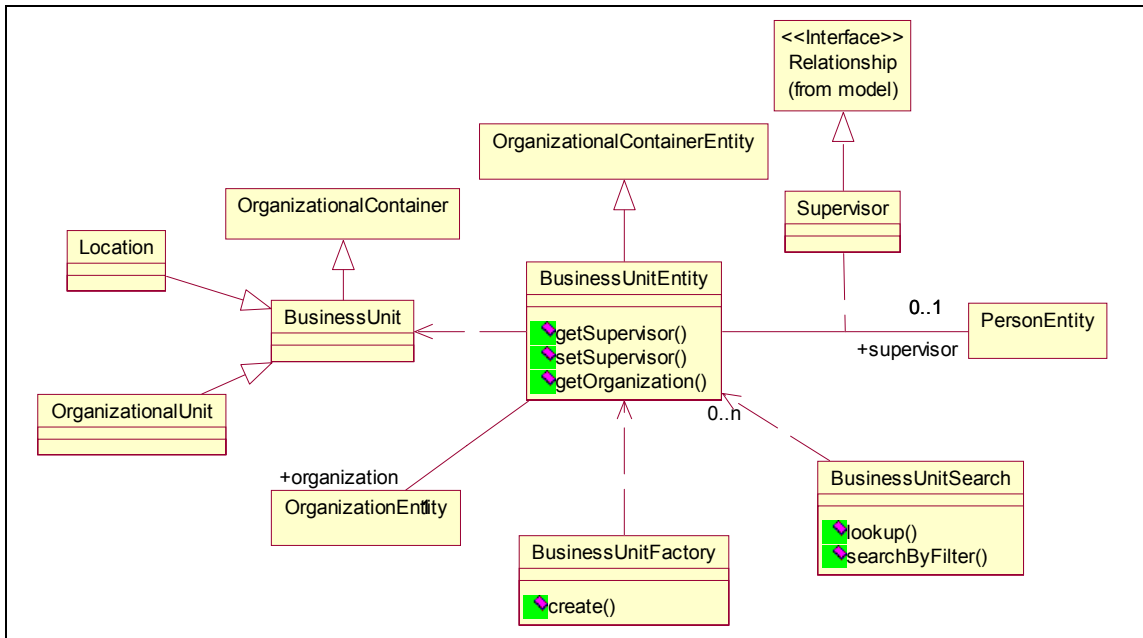


Figure 1-18 *BusinessUnit* class diagram

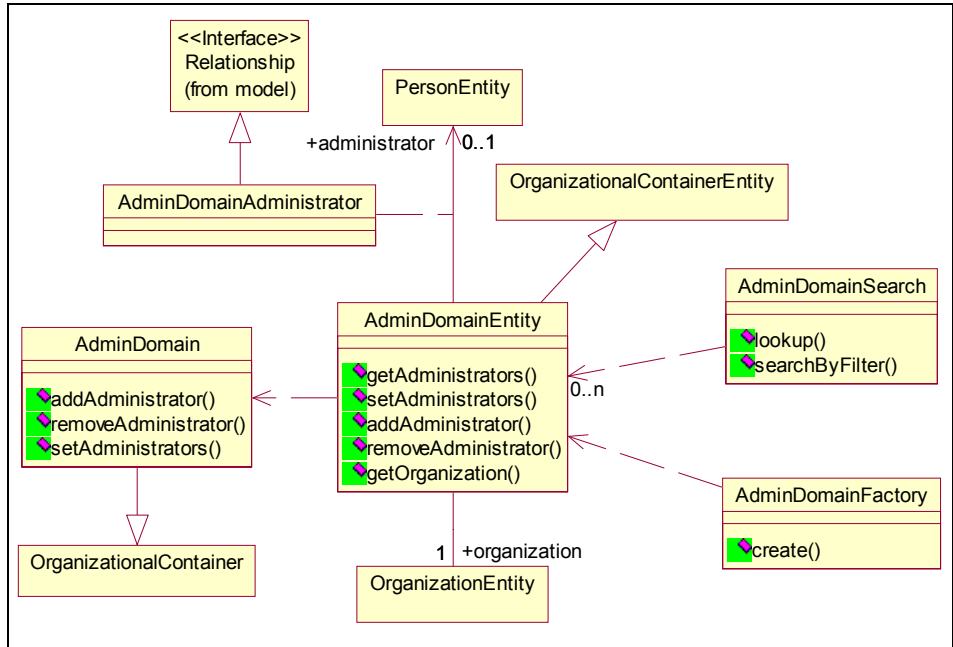


Figure 1-19 AdminDomain class diagram

The *AdminDomainEntity*, shown in Figure 1-19, represents an administrative domain container. It distinguishes itself by its ability to associate a set of administrators with the organizational container.

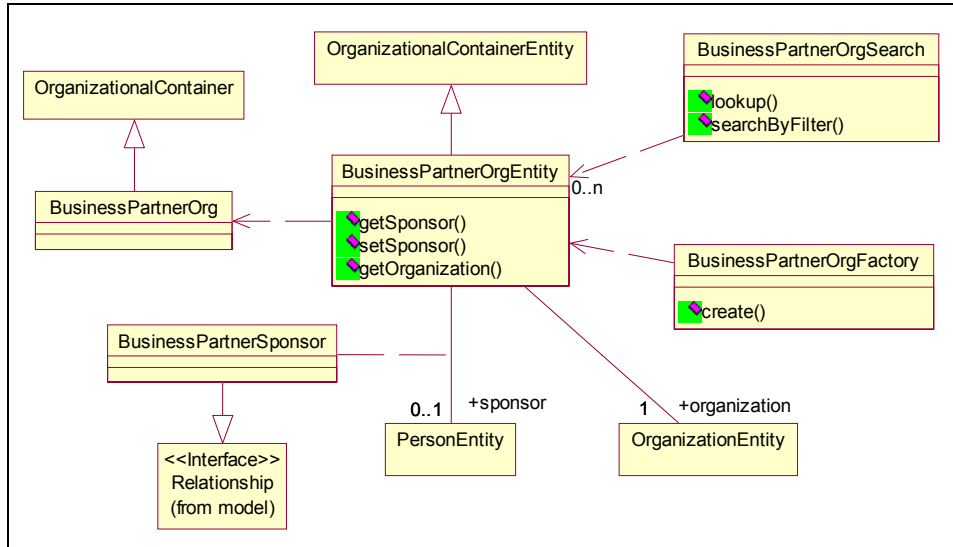


Figure 1-20 PartnerOrganization class diagram

The *BusinessPartnerOrgEntity* in Figure 1-20 represents a business partner organization that was designed for holding, but not limited to, *BusinessPartnerEntities*. The only real distinction between this container and other container types is the ability to associate a sponsor with the organizational container.

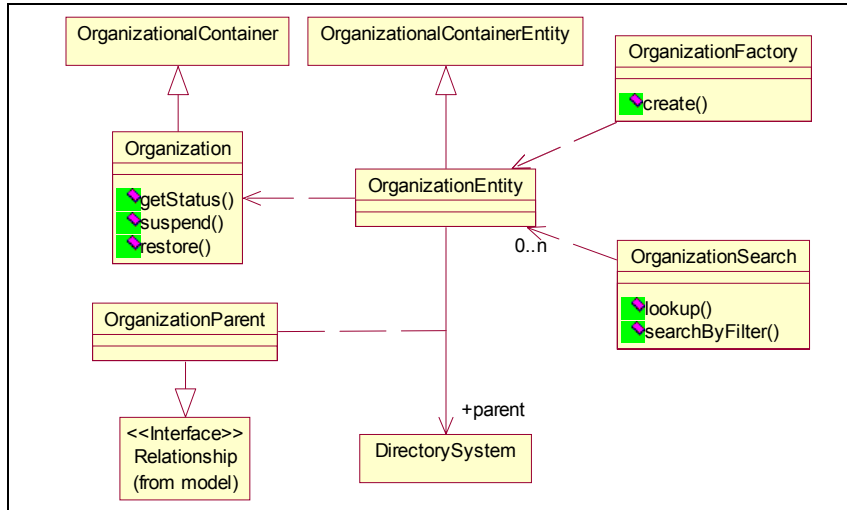


Figure 1-21 Organization class diagram

The *OrganizationEntity*, as shown in Figure 1-21, is a bit different in design to its organizational container predecessors, because it cannot be nested or have any parents itself. Other than that, it is quite similar to other organizational containers.

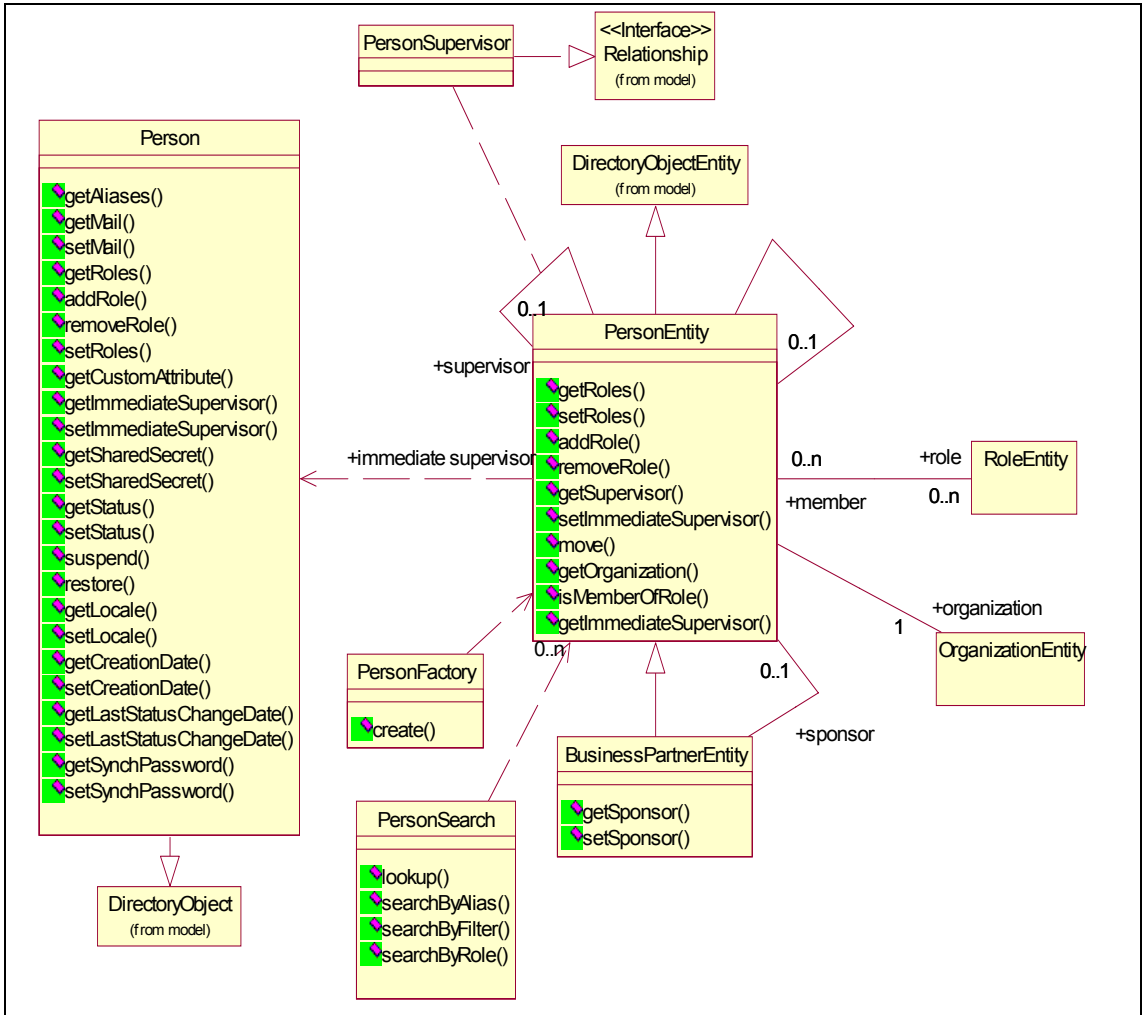


Figure 1-22 Person class diagram

The *PersonEntity* in Figure 1-22 represents any identity managed by the system in the data model. In general, these identities are truly human in nature, but can sometimes represent more, such as a system or job function (administrator). The *PersonEntity* provides an interface for all identity relationships needed within the system. The most noteworthy are an identity's relationship to its accounts and its affiliation with roles. The *Person* class is the value object holding attribute information for a person. The *getCreationDate()*, *setCreationDate()*, *getLastStatusChangeDate()*, and *setLastStatusChangeDate()* methods are used to support Lifecycle Event Information in the schema. The *getSynchPassword()* and *setSynchPassword()* methods are used to synchronize passwords. Two

supporting classes are provided, *PersonFactory* for creation of people in the data store, and *PersonSearch* for different searching capabilities.

The *BusinessPartnerEntity* class is a specialization of the *PersonEntity* to represent business partners in the data model. The only real difference for a business partner is the need to identify a sponsor for the individual.

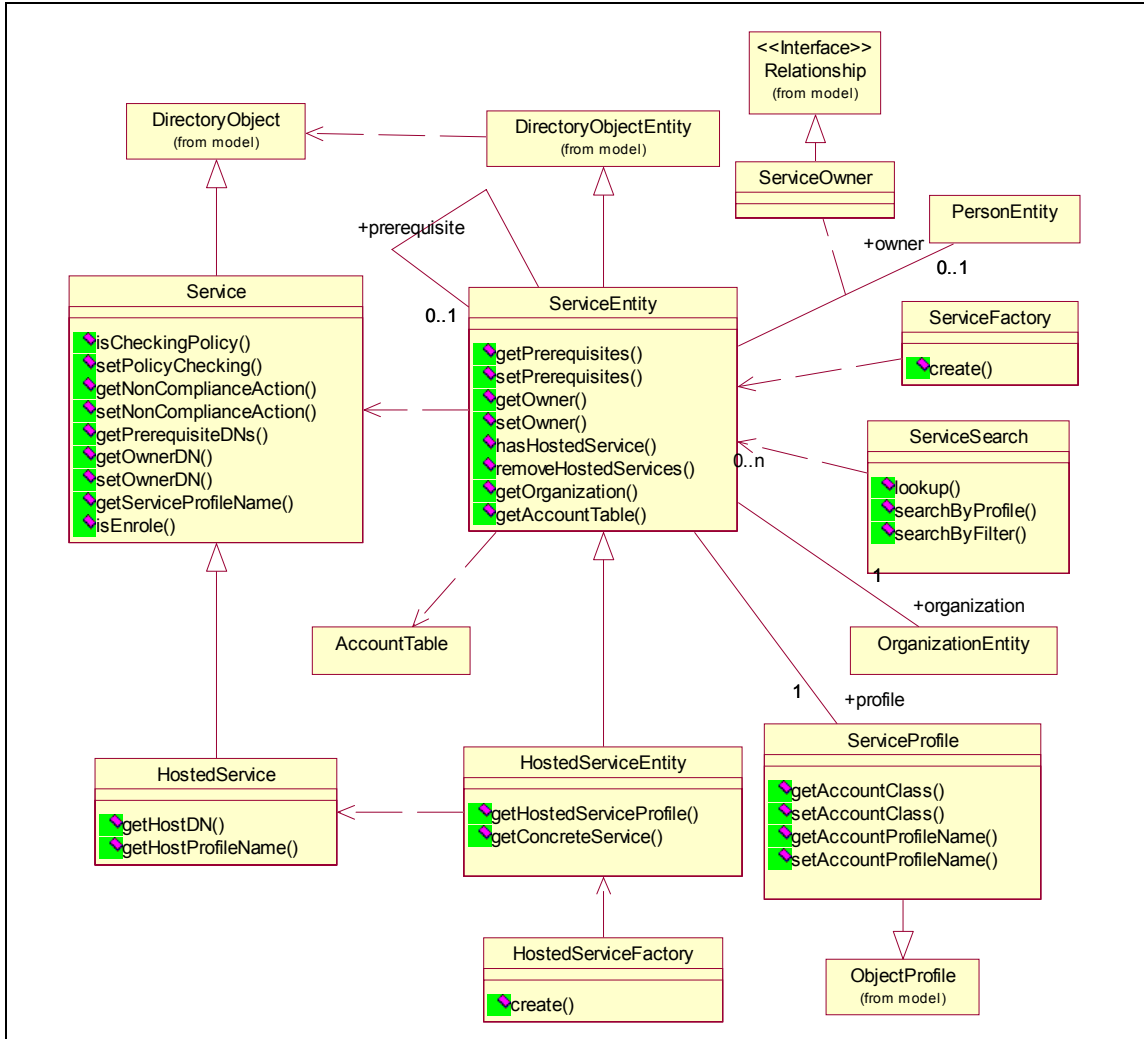


Figure 1-23 Service class diagram

The *HostedServiceEntity* class, shown in Figure 1-23, is a specialization of the *ServiceEntity* to represent a service hosted by another organization. This hosted service lies in the customer organization and is a virtual copy, or proxy, to the

concrete service within the owner organization. For the most part, this entity looks the same as concrete services but provides an interface to obtain profile information about itself and the proxied service. The profile information for all *HostedServiceEntities* is always the same, a hosted profile. The *HostedService* class represents the value object and *HostedServiceFactory* is the factory.

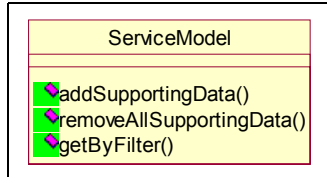


Figure 1-24 *ServiceModel* class diagram

The *ServiceModel* class in Figure 1-24, which can also be obtained from the *ServiceEntity*, provides the interface for the supporting provisioning model of a service, such as groups and other contextual information. This interface allows the client to add and remove objects from the directory that are linked to the specified service.

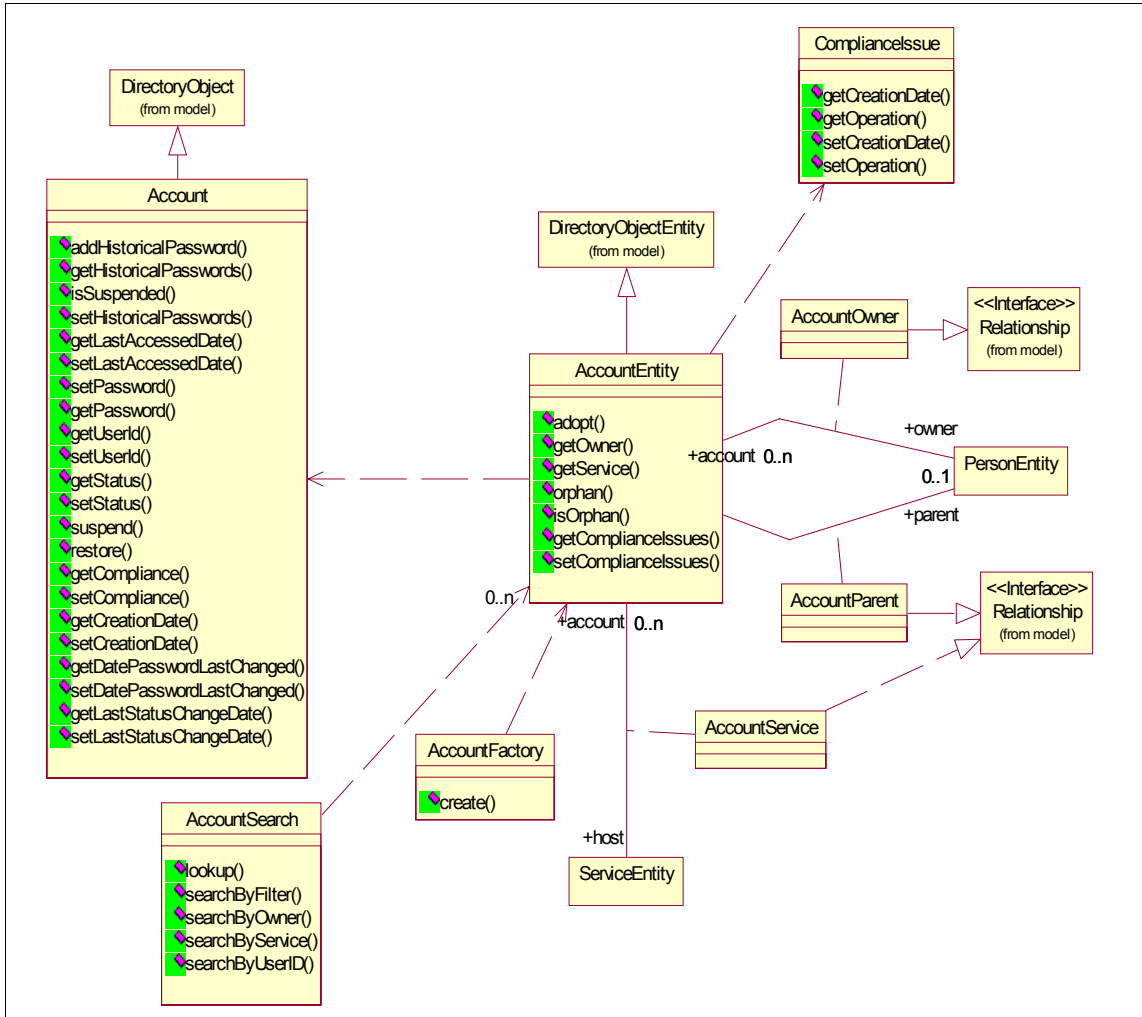


Figure 1-25 Account class diagram

The *AccountEntity* class, depicted in Figure 1-25, represents an account, or identity, that is provisioned on a service. The *AccountEntity* provides the interface for account relationships, such as ownership and business logic-like suspensions. An account compliance issues list has been added that consists of many possible *ComplianceIssue* objects to support workflow-based policy enforcement.

The *Account* class is the value object holding attribute information for an account. The *getCreationDate()*, *setCreationDate()*, *getLastStatusChangeDate()*, and *setLastStatusChangeDate()* methods support

lifecycle event information in the schema. Two supporting classes are provided, *AccountFactory* for creation of accounts in the data store, and *AccountSearch* for different searching capabilities.

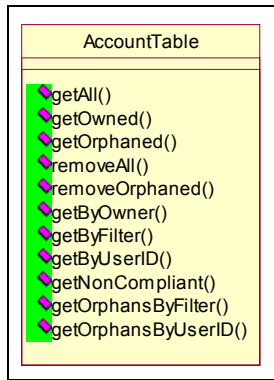


Figure 1-26 *AccountTable* class diagram

For traversing the various aspects of the service-to-account relationship, the *AccountTable* class, shown in Figure 1-26, is provided. The *AccountTable* class provides a variety of queries to obtain accounts for a service.

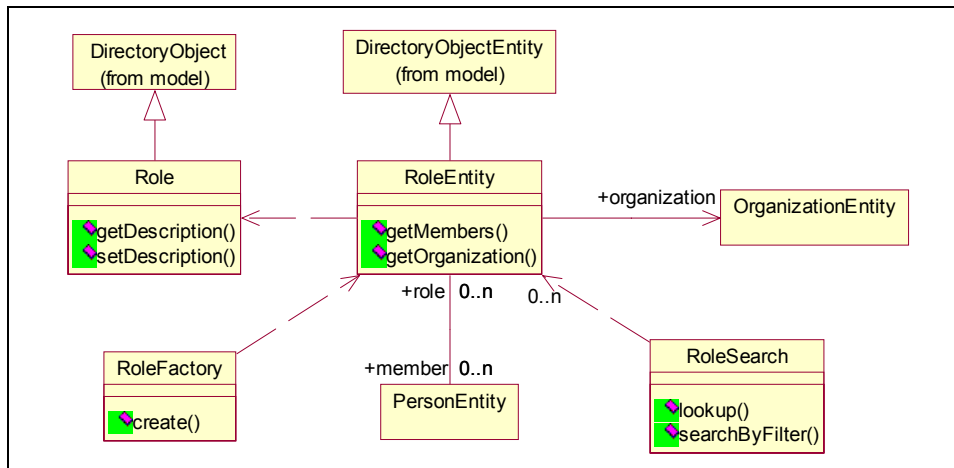


Figure 1-27 *Role* class diagram

The *RoleEntity* class in Figure 1-27 represents a role that categorizes identities for the purposes of provisioning. The *RoleEntity* provides the interface for identifying all role members. The *Role* class is the value object holding attribute information for a role. Two supporting classes are provided, *RoleFactory* for

creation of roles in the data store and *RoleSearch* for different searching capabilities.

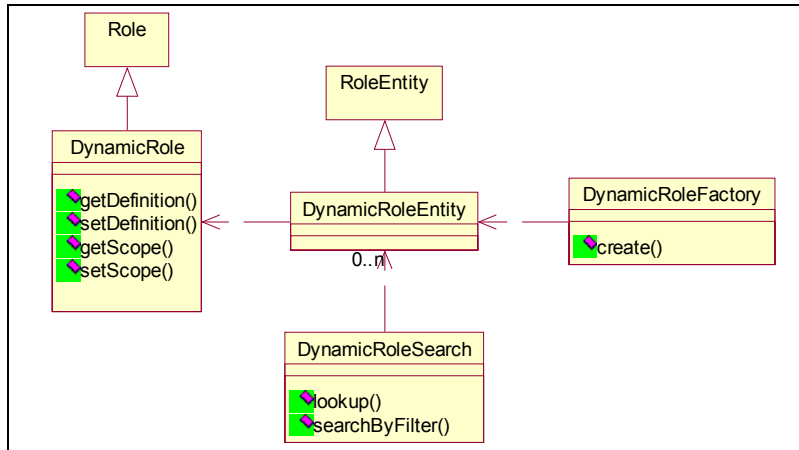


Figure 1-28 *DynamicRole* class diagram

The *DynamicRoleEntity* class, shown in Figure 1-28, is a specialization of the *RoleEntity* that represents a dynamic role in the system. The definition of a dynamic role is a rule that, when evaluated, identifies a group of people, or identities, that should be treated as members. The *DynamicRole* class is the value object holding attribute information for a dynamic role. Two supporting classes are provided, *DynamicRoleFactory* for creation of dynamic roles in the data store and *DynamicRoleSearch* for different searching capabilities.

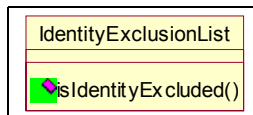


Figure 1-29 *IdentityExclusionList* class diagram

The *IdentityExclusionList* class in Figure 1-29 represents a list of identities that should be excluded when attempting to automatically adopt service accounts. The interface is very simple. The client can ask whether a given identity is excluded or not.

System

The *System* package is a sub-package of the Model package that holds classes that represent the users and groups in the Identity Manager system.

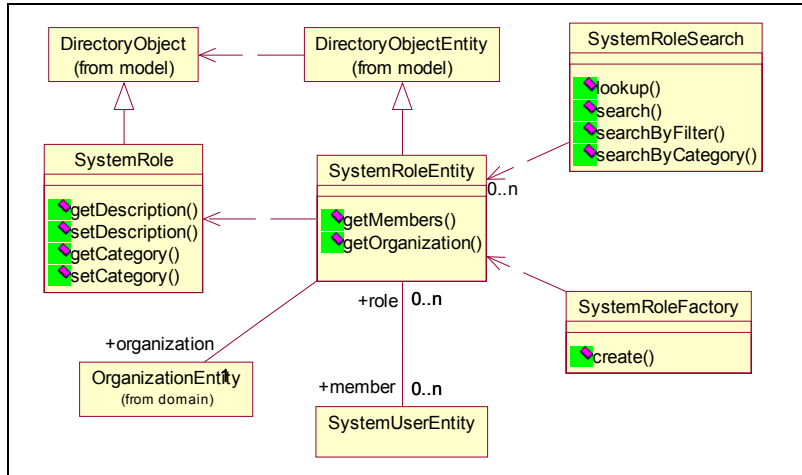


Figure 1-30 SystemRole class diagram

The *SystemRoleEntity* class in Figure 1-30 represents a named group of Identity Manager users that can be used within access control information to identify the governing users. The *SystemRoleEntity* provides the interface for identifying members. The *SystemRole* class is the value object holding attribute information for a group. Two supporting classes are provided, *SystemRoleFactory* for creation of groups in the data store and *SystemRoleSearch* for different searching capabilities.

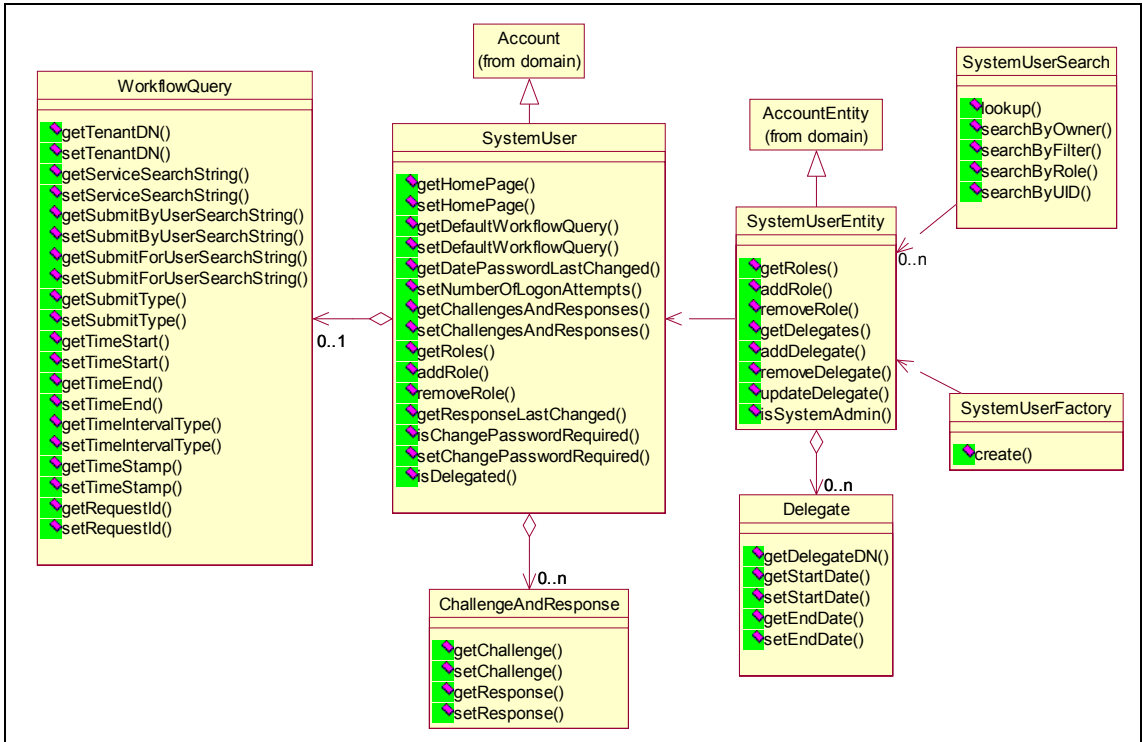


Figure 1-31 SystemUser class diagram

The *SystemUserEntity* class, shown in Figure 1-31, represents an Identity Manager user. The *SystemUserEntity* provides the interface for group and delegation management. The *Delegate* class represents a delegate for the user with time duration. The ability to define multiple challenges and responses for the user is also available. The *ChallengeAndResponse* class holds this information. The *SystemUser* class is the value object holding attribute information for an Identity Manager user. Two supporting classes are provided, *SystemUserFactory* for creation of users in the data store and *SystemUserSearch* for different searching capabilities.

1.2.4 Logging

The Logging package represents the Logging module defined within the Services subsystem of the System Architecture. This package contains classes that provide clients with a logging service. This service provides a consistent interface for logging information throughout the platform while also providing extensibility through a configurable implementation of filtering and formatting of the information.

The Logging package holds the classes that provide clients with a Logging service.

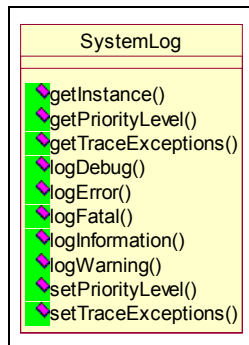


Figure 1-32 Logging class diagram

1.2.5 Mail

The *Mail package* represents the Notification module defined within the Services subsystem of the System Architecture. This package contains classes that provide clients with a service for mailing messages as a form of notification. The package provides value-added services, such as a message factory framework on top of its underlying delivery technology provided by the standard Java Mail API. In Identity Manager 4.6, a post office capability is provided to consolidate messages sent to individuals using a store and forward approach.

The Mail package holds the classes that provide clients with a mail delivery, or notification, service. The implementation of this package relies heavily on the Java Mail API for its implementation, but provides additional value through the use of more platform specific objects.

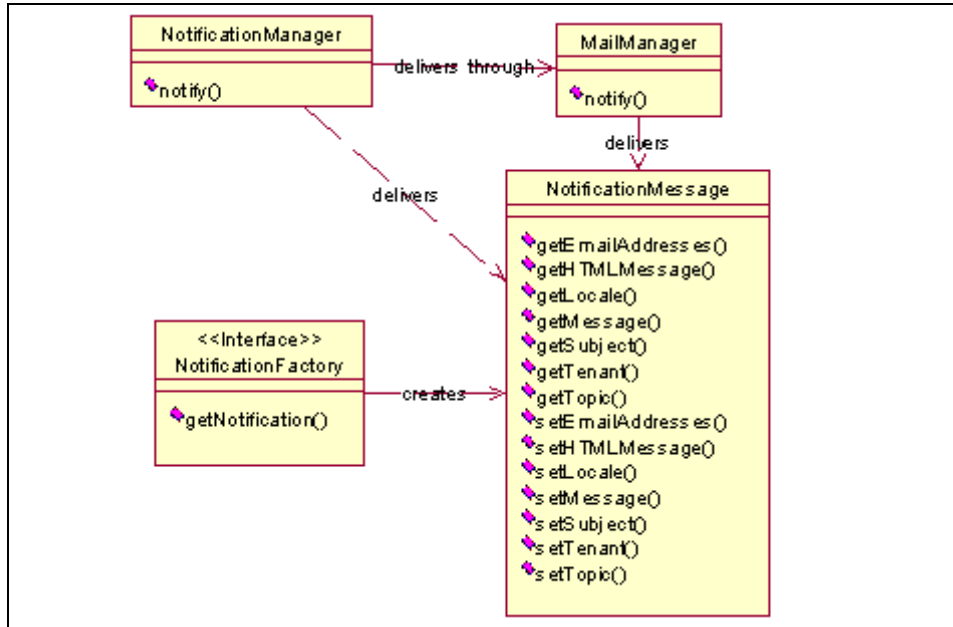


Figure 1-33 Mail class diagram

The *NotificationMessage* in Figure 1-33 represents the message itself that needs to be delivered. The message is defined by a set of recipients, a subject, a text body, an optional HTML body, and an optional topic name. The *getTenant()*, *setTenant()*, *getTopic()*, *setTopic()*, *getLocale()*, and *setLocale()* methods support the Notification Post Office. The *NotificationFactory* is an interface that can be implemented to create the appropriate content of a *NotificationMessage* based on the situation, or context, causing the need for a notification.

The *MailManager* is a simple class that merely delivers the *NotificationMessage* to its recipients using the delivery mechanism configured through the Java Mail API.

The *NotificationManager* provides a more comprehensive facade requiring only a category of notification and the current context. This object employs the correct *NotificationFactory* and *MailManager* to generate and deliver the appropriate *NotificationMessage* respectively. The choice of which *NotificationFactory* implementation to use is configured through a property file.

1.2.6 Policy

The *Policy package* has one sub-package named *Analysis* that provides an external API to the *Policy package* for analyzing policies. The classes in this

package leverage the policy engine classes in the base Policy package to perform their analysis.

Analysis

The *Analysis package* is a sub-package of the Policy package that holds classes that provide a public interface for analyzing the current policies in the system. This analysis enables clients to determine the rights, or entitlements, that individuals have been assigned. Clients also are able to identify which policies apply to given roles.

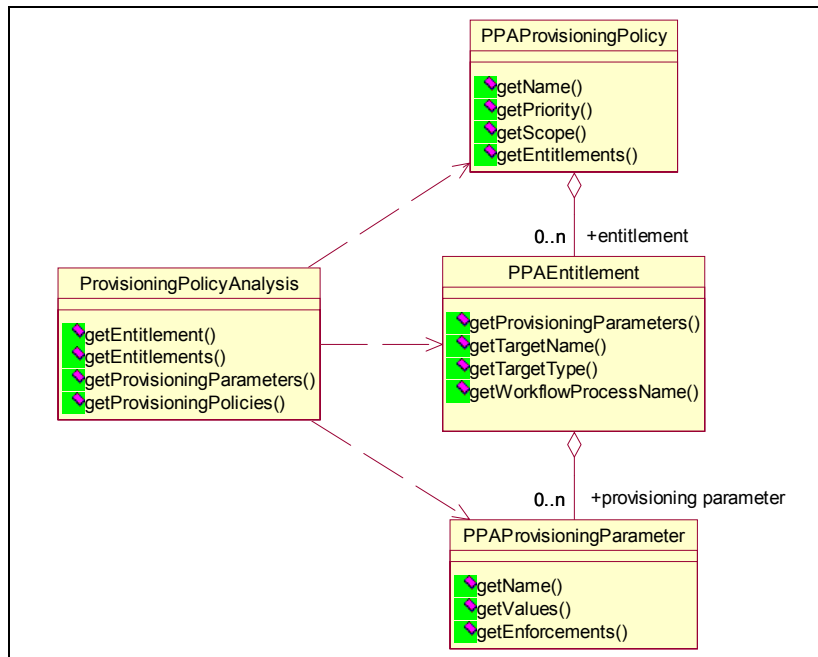


Figure 1-34 PolicyAnalysis class diagram

The *ProvisioningPolicyAnalysis* class, depicted in Figure 1-34, is the primary interface for analyzing provisioning policies in the system. From this class, the client can obtain the entitlements for a given individual, which are represented by the *PPAEntitlement* class. Clients can also obtain just the provisioning parameters for an individual on a given service, which are represented by the *PPAProvisioningParameter*. Clients can also obtain the list of provisioning policies that apply to a given role and that are represented by the *PPAProvisioningPolicy* class. However, due to the context driven approach to joining policies and generating dynamic parameters through scripts, these *PPAProvisioningPolicies* and their associated *PPAEntitlements* and

PPAProvisioningParameters are not fully evaluated by the policy engine. Instead, they are returned in definition form.

1.2.7 Password rules

The *Password rules package* provides an external API for customers to extend the capability of the password rules engine used by Identity Manager.

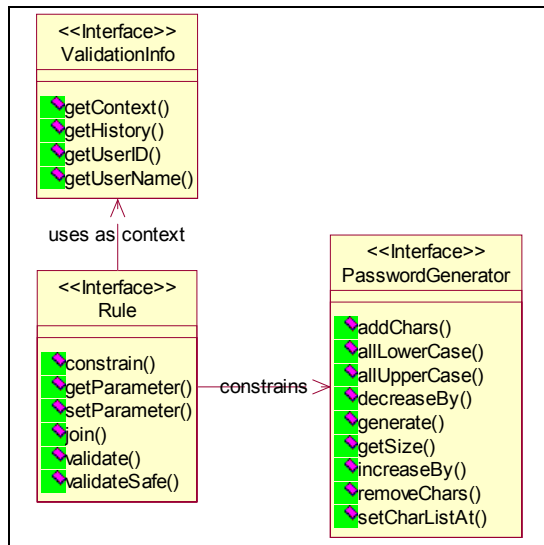


Figure 1-35 PasswordRules class diagram

The *Rule* interface in Figure 1-35 specifies the required signature that must be fulfilled by any password rule that can be interpreted by the rule engine. The *ValidationInfo* interface provides the *Rule* implementation access to contextual information about the validation process the current *Rule* is involved in. The *PasswordGenerator* interface specifies the required signature that must be implemented in order to generate passwords.

Provisioning

The *Provisioning* sub-package holds provisioning specific extensions to the Password Rules framework. Currently, there is only one interface, *ProvisioningValidationInfo*, which extends *ValidationInfo* with provisioning specific properties that are available to Rules. It is displayed in Figure 1-36.

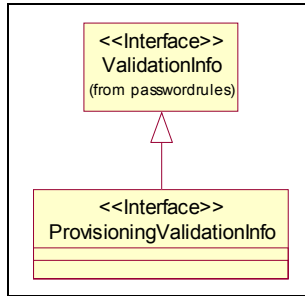


Figure 1-36 Provisioning Password Rules class diagram

Standard

The *Standard* sub-package, shown in Figure 1-37 on page 45, holds a set of standard rule implementations and a standard password generator implementation.

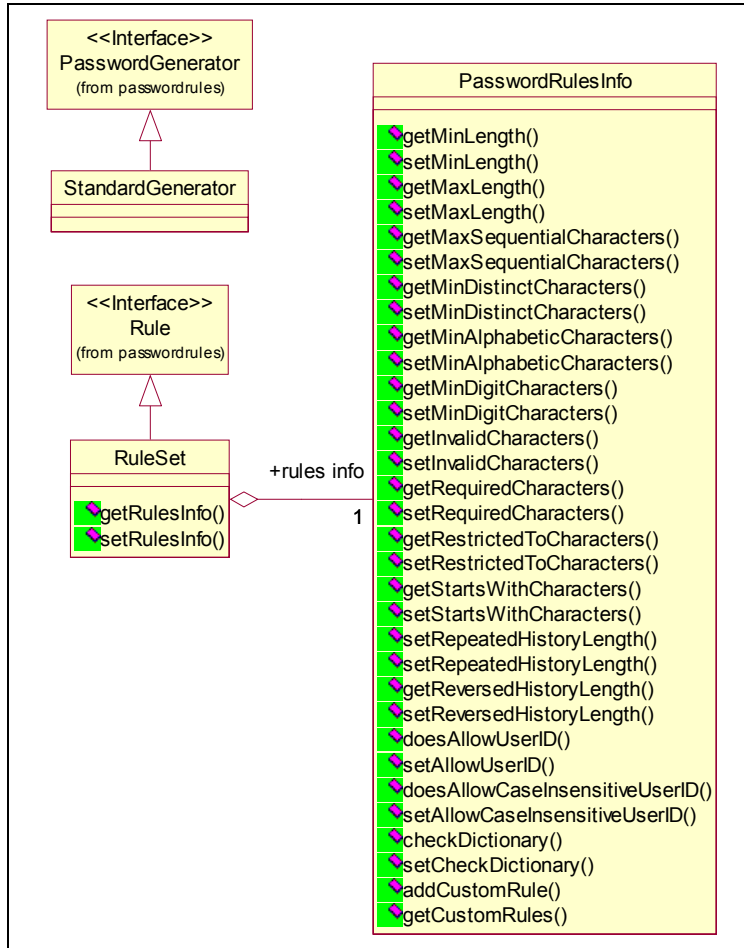


Figure 1-37 Standard PasswordRules class diagram

The *RuleSet* is a class that implements the *Rule* interface and represents a collection of standard *Rule* objects as defined through the *PasswordRulesInfo* class. The *PasswordRulesInfo* class holds the out-of-the-box password rule information as well as an interface to register custom rule classes. Customers can provide one or more of their own implementations of the *Rule* interface and register it with Identity Manager through a properties file setting.

The *StandardGenerator* is the *PasswordGenerator* interface that is packaged with Identity Manager. Customers can provide their own implementation of the *PasswordGenerator* and register it with Identity Manager through a properties file setting.

1.2.8 Remote Services

The *Remote Services package* represents the Remote Services module defined within the Services subsystem of the System Architecture. This package contains the classes that provide an extensible framework for abstracting the communications between the platform and the resources it manages. This abstraction supports the complete set of provisioning actions required of resources, such as the addition, removal, modification, and retrieval of provisioning specific objects (for example, users and groups) on the resource. The Provider package, a sub-package of remote services, provides an extensible framework for implementing all interactions with the managed services using different protocols and APIs.

Provider

The *Provider package* contains the classes that provide the extensible framework for implementing all interactions with the managed services possibly using different protocols and APIs, shown in Figure 1-38.

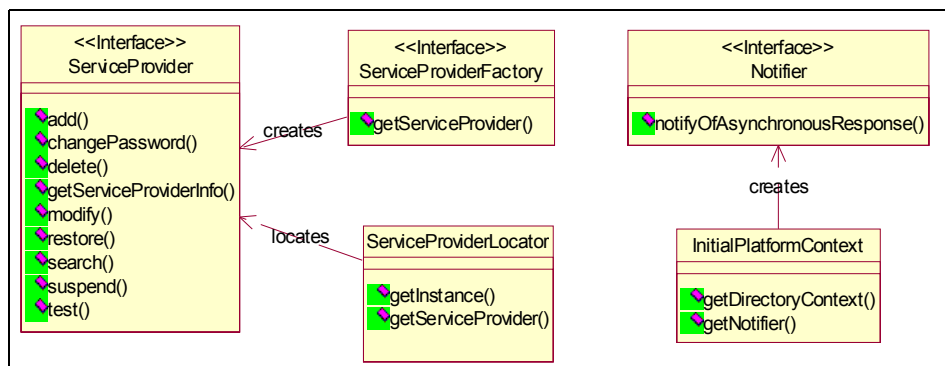


Figure 1-38 *ServicesProvider* class diagram

The *ServiceProvider* interface is the key interface for implementing the communications between the platform and a managed service. To support a new protocol or API to provision a managed service, the client implements this interface. The *ServiceProviderFactory* is another interface that the client implements to initialize their implementation of the corresponding *ServiceProvider*.

Each type of service managed by the platform has a *ServiceProviderFactory* implementation registered for it. This registration enables the platform to call the *ServiceProviderLocator* to obtain the correct implementation when business logic requires communication with a managed service.

The *InitialPlatformContext* is provided to implementers of a *ServiceProvider* for interacting with the provisioning platform in a safe controlled API that allows the Provider implementation to be safe from dependencies that might cause incompatibilities during platform upgrades. This object provides a key interface to obtaining a *Notifier* from the platform. The *Notifier* provides an interface for returning the results of an asynchronous request made to the managed service.

1.2.9 Workflow

The *Workflow package* represents the Workflow module defined within the Services subsystem of the System Architecture. This package, depicted in Figure 1-39, contains several other sub-packages that provide both the ability to define and execute workflow processes within the platform.

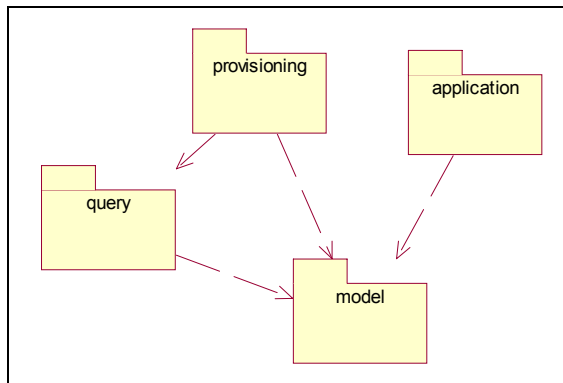


Figure 1-39 Workflow package diagram

The *Model* sub-package contains the classes needed to manage workflow processes within the platform. The workflow engine itself is designed to be able to call out to any block of code from an activity and to string those activities together in a variety of different sequences defined as processes. Although the engine might call out to several different packages in the platform through this mechanism, the workflow engine package itself really only has inherent dependencies on the Data Services and Mail packages.

The *Query* sub-package contains the classes needed for a more flexible query mechanism to the workflow engine. This package has a dependency on the Model package.

There are several more provisioning specific classes that are defined to more easily integrate provisioning concepts into the more general design of the workflow engine in the Model and Query packages provided in a sub-package

named *Provisioning*. This package has dependencies on the Model and Query packages.

The *Application* sub-package contains the interfaces for defining custom extensions that can be displayed in the workflow designer environment and called by the workflow engine.

Model

The *Model package* holds the classes that provide workflow management capabilities. These capabilities include the creation of workflow processes, managing the state of those processes, and auditing current and historical processes. The participation in a workflow process by an external resource is also provided in this package. The complete WorkflowModel class diagram is depicted in Figure 1-40 on page 49.

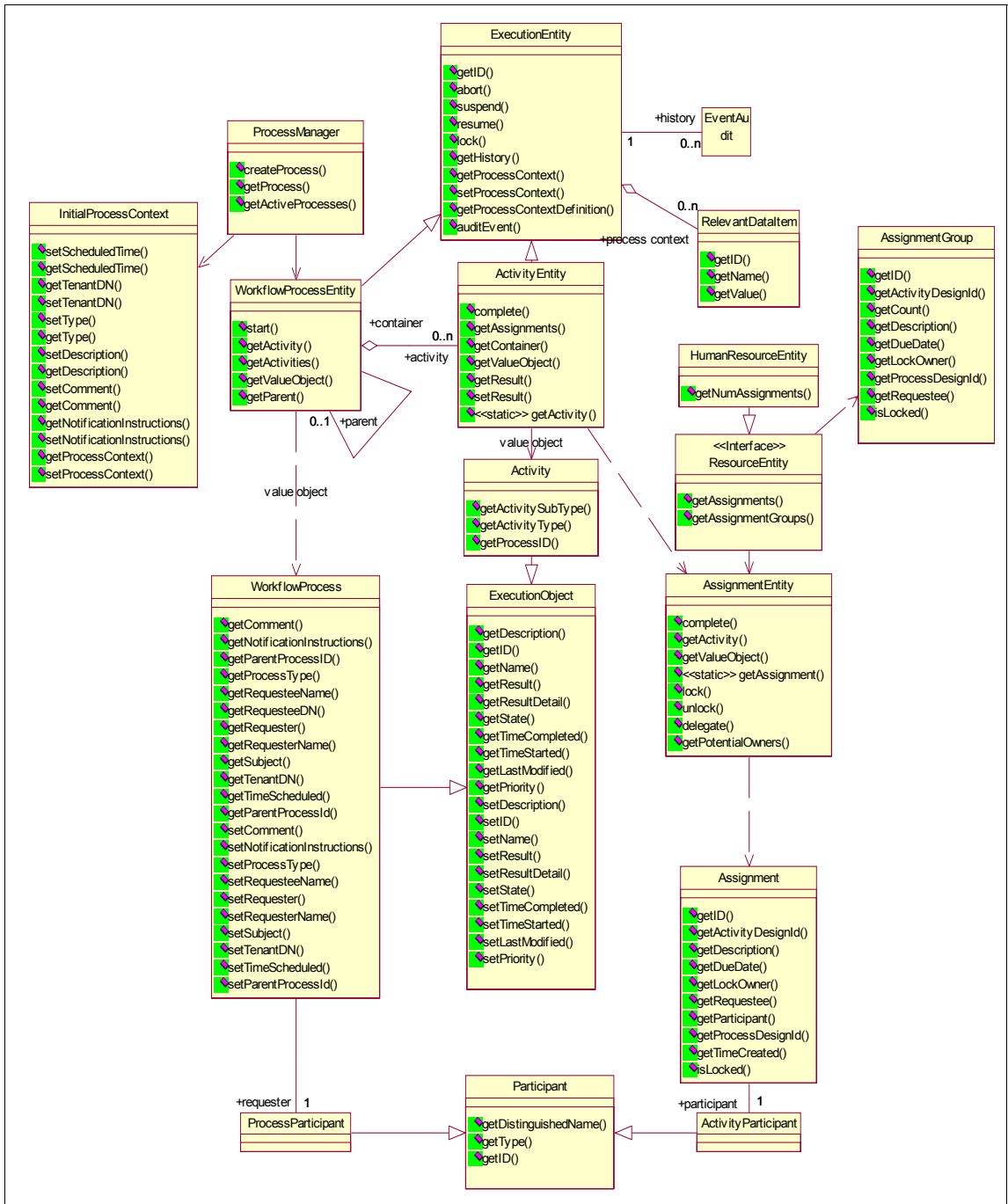


Figure 1-40 WorkflowModel class diagram

The *ProcessManager* class provides the central point of entry into the workflow system. This class provides interfaces for the creation of workflow processes and the querying of current and historical workflow processes. The creation of a workflow process requires configuration information for that process to be provided by the client. This is done through the use of the *InitialProcessContext* class. The *WorkflowProcessEntity* represents an individual workflow process. For active processes, this class provides interfaces for controlling state. It also provides interfaces for querying process characteristics (provided by the *WorkflowProcess* value object class) and querying process relationships, such as parent processes (if any) and child activities. Through the interface of the *ExecutionEntity* base class, the client can also obtain the history of the process implemented with event records.

The *ActivityEntity*, also a subclass of *ExecutionEntity*, represents an activity within a workflow process. This class provides information about the activity (through the *Activity* value object class), as well as the ability to complete the activity with a specified status.

An external actor (for example, a system or a human being) that might participate in a workflow activity is called a *workflow resource*. The *ResourceEntity* interface represents such a resource. The *HumanResourceEntity* class, however, can only represent a human resource. When the platform resolves the defined participant of a workflow activity to a resource, an assignment of that activity to the resource is made. The *AssignmentEntity* class represents this assignment. There can be multiple assignments for one activity. The *lock()* and *unlock()* methods are used in locking To do Items while the *delegate()* and *getPotentialOwners()* methods are used in the delegation of To do Items. The *ActivityEntity* has an interface for retrieving all of its associated *Assignments*. The list of assignments that a resource has makes up its “To do list”. This list can be obtained from the *ResourceEntity* and is typically made up of *Assignment* objects which are the value objects of *AssignmentEntities*. The *isLocked()* and *getLockOwner()* methods are used in the locking of To do Items. The *getActivityDesignId()*, *getProcessDesignId()*, *getDescription()*, and *getDueDate()* methods are used to complete multiple To Do Items. The *AssignmentGroup* class is a value object that is returned when obtaining assignments for a resource in a grouped fashion, for example, by activity design id. It provides most characteristics of the *Assignment* value object but with an additional count parameter of how many assignments are in this grouping. When the resource wishes to complete its assignment, it must obtain the *ActivityEntity* from the assignment it is associated with and use the interface provided by that class.

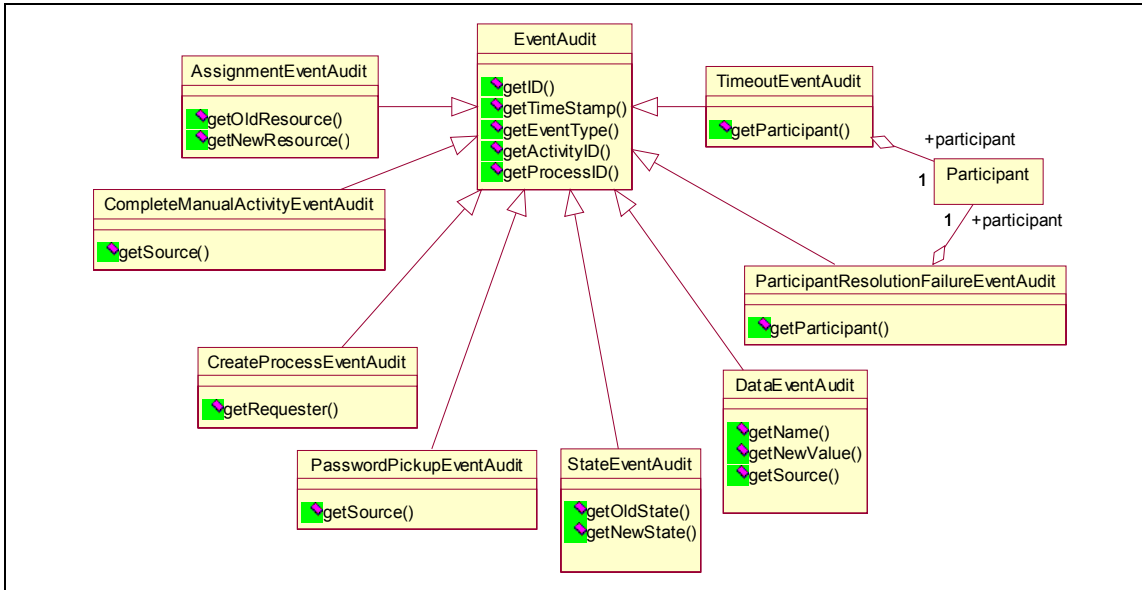


Figure 1-41 Workflow EventAudit class diagram

The *EventAudit* class, shown in Figure 1-41, represents an auditable event recorded by the workflow engine. The history provided about a process or activity is represented as a list of *EventAudit* objects. This history can be obtained through the *ExecutionEntity* interface. The *EventAudit* class has some general auditing information, but more specific information about a particular type of event can be defined by a subclass representing that type of event. Figure 1-41 illustrates the specific events modeled within this package.

Query

Although the *ProcessManager* provides interfaces for querying current and historical processes in the workflow engine, the interfaces that it provides are limited. The *Query* package provides classes, shown in Figure 1-42 on page 52, that support an extensible design for defining a more flexible query interface.

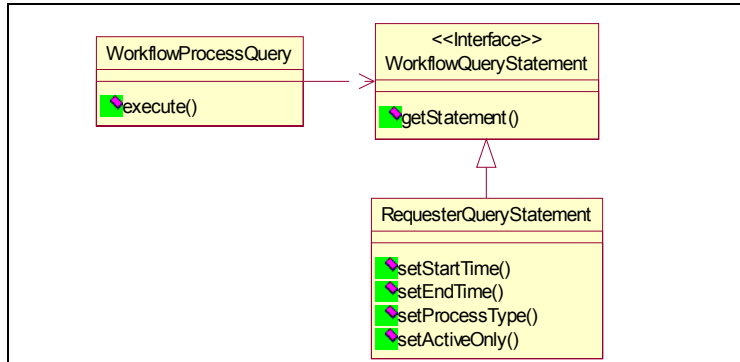


Figure 1-42 Workflow query class diagram

The *WorkflowQueryStatement* interface can be implemented to provide a specific query into the workflow engine that can provide more power to the client than what is currently provided in *ProcessManager*. The *WorkflowQueryStatement* just provides the definition of the query though. The *WorkflowProcessQuery* class performs the execution of the query. The *RequesterQueryStatement* is a default implementation provided in this package that provides several options on querying workflow processes created by a given requester.

Provisioning

The *Provisioning package* holds the classes that provide more provisioning specific interfaces to the capabilities provided by the *Model* and *Query* packages.

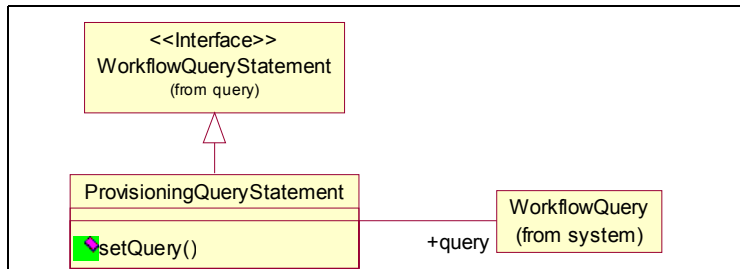


Figure 1-43 Provisioning WorkflowQuery class diagram

The *ProvisioningQueryStatement* class in Figure 1-43 provides a provisioning specific implementation to the *WorkflowProcessQuery* interface from the *Model* package. The query parameters are defined using the *WorkflowQuery* class defined in the *System* sub-package of *Data Services*.

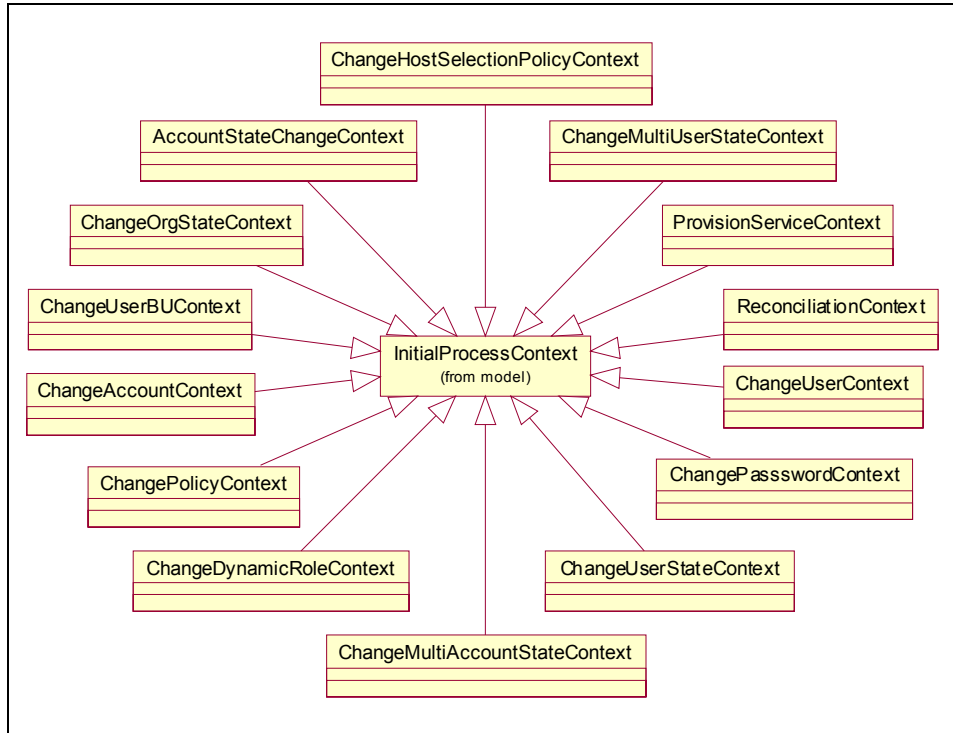


Figure 1-44 *InitialProcessContext* class diagram

There are several subclasses of the *InitialProcessContext* class, depicted in Figure 1-44, in the Model package, that have been provided to provide a more intuitive, operation specific mechanism for creating such an initial context. For example, the *ChangePasswordContext* class provides a constructor that takes in the password as a compile-time checked parameter, instead of using the run-time checking provided through the *setProcessContext()* method of the *InitialProcessContext* class.

Application

The *Application* package holds the classes that provide workflow integration and extension capabilities with Java applications, shown in Figure 1-45 on page 54. A Java application is one or more Java-based classes that make up a task that can be called from a workflow activity to extend the capabilities of the current workflow processes defined in the platform.

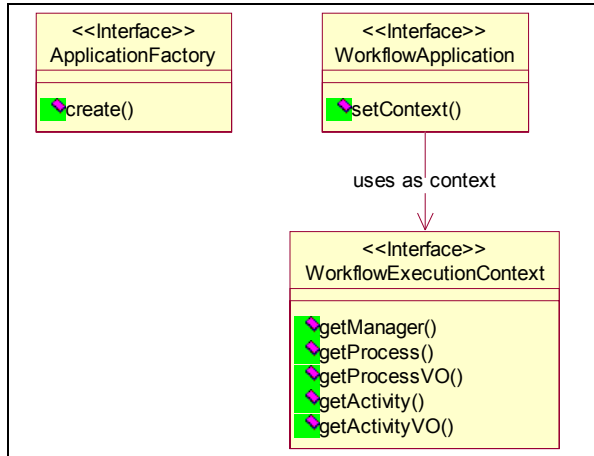


Figure 1-45 Workflow ApplicationIntegration class diagram

There are two basic scenarios for extending the capabilities of a workflow process by calling a Java application. The first scenario is one where an existing Java API (Java class) is available that needs to be integrated to perform a task. The second is one where custom code is written to perform a task where a tighter coupling is required with the workflow process, requiring workflow contextual information to be available to the application.

The purpose of the *ApplicationFactory* class is to help the first scenario by providing an interface to the integrator that the workflow engine can make use of to instantiate an instance of the predefined Java class that needs to be integrated. The engine already supports a flexible mapping scheme (through the process definition) for adapting to the signature of the Java class to call to execute the task, but the instantiation and configuration of the class is left to this mechanism. To keep things simple, however, the engine supports default constructors and static method calls without the use of a factory.

The *WorkflowApplication* is an interface that a custom workflow application developer can make use of when writing a Java class that needs to be called by the workflow engine. This biggest benefit to implementing this class is the availability of the context of the current activity and process calling the class made available in the *WorkflowExecutionContext* interface. This interface provides the custom application access to the current activity, process, and process manager.

1.2.10 FESI extensions

Free ECMA Script Interpreter (FESI) is a full implementation of the first version European Computer Manufacturers Association (ECMA) script language defined in ECMA standard 262 called EcmaScript. EcmaScript is roughly equivalent to the JavaScript Version 1.1 or to the core part of JScript®, but without the navigator.

The JavaScript extensible framework and API in IBM Tivoli Identity Manager is based on the use of the FESI JavaScript interpreter. The purpose of extending the framework is not to alter the capabilities of the interpreter, but to add additional objects and functions to the interpreter's glossary so more capabilities are available within the scripts interpreted by the FESI interpreter.

The API provided by the FESI JavaScript interpreter allows additional objects and functions to be created and registered with the interpreter so they can be executed at run time. The API used to extend the interpreter is based on a Netscape standard which makes it portable to other vendors' implementations that support it if necessary.

The JavaScript Extensions API consists of a set of classes that provide integration points with the JavaScript interpreter for creating and retrieving information available in the run-time context of the interpreter, as well as for registering new types of objects and functions the interpreter can interpret in users' scripts. In addition to the general mechanisms available for this purpose provided by the FESI interpreter, a more provisioning platform specific set of classes are provided for better integration with the extensions that are deployed with the platform already. The general FESI interfaces and classes used to extend the FESI interpreter can be found in the FESI.jslib package. The platform specific classes can be found in the com.ibm.itim.script and com.ibm.itim.fesiextensions packages.

1.3 Workflow

Two basic forms of workflow or business processes exist within Identity Manager. Before describing the different types of processes in more detail however, it is best to understand the concept of lifecycle management found within Identity Manager. Lifecycle management is the concept of managing the operations performed upon an identity object for the span of its entire life, starting with the creation of the identity object and ending with its deletion, should that occur, inclusive of all operations in between. Therefore, while there are two different process or workflow types, all operations that are performed on the given identity object are performed by the lifecycle management process.

The two types of workflow within Identity Manager are defined as entitlement and operation workflows or business processes.

An *entitlement workflow* implements the business processes that create the identity objects associated with an entitlement in a provisioning policy. An entitlement workflow gathers all of the information required to create the identity objects. The information that is included in identity objects depends on the definitions of the provisioning policy, which reflect the audit and compliance requirements of the organization.

The *operational workflow* is the business process that defines all the operations that are performed to create, modify, delete, suspend, and transfer identity objects.

To put this into better perspective, the entitlement workflow associated to an entitlement within a provisioning policy gathers and documents all the required information to create an identity object. For example, if the object meets the requirements defined in the given policy, then the gathered information is given to the operational workflow process to create the object. For more information about standard workflow processes and capabilities, refer to Chapter 4, “Detailed component design” on page 89 of the IBM Redbook *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996.

Both types of processes utilize the same extension capabilities of script nodes and extensions, which we discuss next.

1.3.1 Script nodes

Script nodes provide the capability of adding needed extensibility into a workflow process simply by dropping a node into the process and writing the needed JavaScript into the node. FESI extensions can also be used here for further extensibility. This type of extensibility is best when changes need to be made to the extension from the workflow designer. For more information about script nodes, refer to Chapter 4, “Detailed component design” on page 89 of the IBM Redbook *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996.

This is a more flexible solution than that of the workflow extension that follows.

1.3.2 Workflow extensions

Workflow extensions are similar to a script node in that they are a node which can be dropped into workflow to extend capability, however, it is an immutable node from the workflow designer once the workflow extension has been created originally. Therefore, this type of extensibility is perfect for situations that require

an extension to a process but should not be changed from within the workflow designer. For more information about workflow extension nodes, refer to Chapter 4, “Detailed component design” on page 89 of the IBM Redbook *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996.

1.4 Custom service provider

Custom service providers are remote services used to communicate with a *managed service* directly or to a *custom adapter*. An example of a managed service, which can be communicated with directly, is one that has a remoteable interface, such as a directory or a database.

The remote services provider interfaces and classes have been provided to enable the development of custom connectors that can be used from the Identity Manager provisioning platform, or any other Java-based provisioning platform that supports the same interface. The provisioning platform by itself is expected to perform all of the operations needed to determine the operations and their parameters that are to be executed against resources. The connector is responsible for executing those operations on the resource in whatever resource specific manner is required. The interface between the platform and the connector used to implement this procedure is defined with this API.

There are eight operations that make up the interface between the provisioning platform and the connector: add, modify, delete, suspend, restore, change password, search, and test. Each operation, with the exception of suspend, restore, and change password, is defined in a manner to support the provisioning of any object supported on the resource, but typically, provisioning systems only focus on provisioning users. Because of the initial focus on user provisioning, the suspend, restore, and change password operations are key user-focused operations that provide convenience.

When performing these eight provisioning operations, some resources require either asynchronous or synchronous communications. The remote services API supports both modes of operation, but the custom connector implementation is responsible for performing the communication with the resource itself. See Figure 1-46 on page 58.

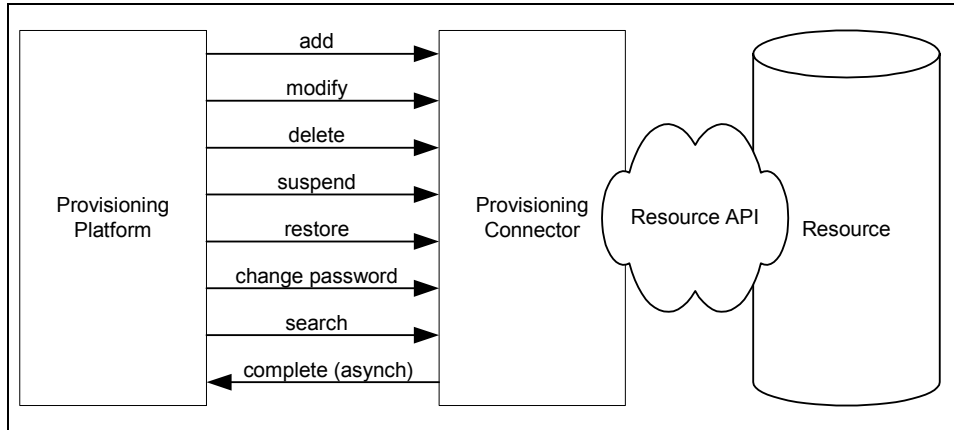


Figure 1-46 Provisioning platform connector

You can find more information about custom service providers in the data/extensions directory of the Identity Manager installation directory, specifically <ITIM HOME>/extensions/doc/serviceprovider/serviceprovider.html.

1.5 Custom reporting

Reporting capabilities can be extended in several fashions with Identity Manager. The most common fashion is to create custom reports with ad hoc reporting provided by Identity Manager custom reports. However, if complex multiple joins, beyond what is offered by Identity Manager, are required, then Crystal Reports can provide another solution. Crystal Reports can be plugged into Identity Manager and can be used to generate reports with multiple complex joins. Further, Identity Manager can be configured to use an incremental data synchronizer on a remote machine with a different operating system, for example. However, Identity Manager still supports the current or existing report extensions previously available by providing the necessary reporting information within the report.xml file in the <ITIM HOME>/data/report.xml file. Lastly, reports can be cloned within the database, if there is a requirement for this. For more information about extending the reporting capabilities within Identity Manager, refer to the Identity Manager documentation in the installation directory <ITIM HOME>/extensions/examples/reports.

1.6 Conclusion

This concludes the comprehensive discussion of the IBM Tivoli Identity Manager system architecture, the application programming interface, workflow, custom

service provider, and reporting. In the next chapter, we take a closer look at a very important infrastructural topic: high availability.



Architect a high availability solution

In this chapter, we discuss the aspects of a solution design for Identity Manager high availability (HA). We elaborate on the considerations to take into account when designing the operational aspects of implementing and maintaining an HA Identity Manager deployment.

High availability is a general term and can mean different things to different people. We define high availability as follows: *High availability combines software with industry-standard hardware to minimize outages by quickly restoring essential services when a system, component, or application fails.*

In the context of this discussion, we address the high availability aspects of the Identity Manager software components. The concepts of a fault-tolerant hardware configuration and high-availability operating system-based infrastructure should be considered and evaluated for each deployment, and the costs weighed against the risks. These aspects, however, are more generic systems design concepts common to most projects and specific to an organization's operational environment. They are not necessarily specific to Identity Manager, and, hence, are not within the scope of this discussion. In most cases, a combination of the various approaches is used. No two environments typically use a standard uniform approach due to the unique constraints, dependencies, and priorities of each approach. The environment should be

evaluated and planned for by the project team in consultation with stakeholders and system owners.

The software components relevant to an Identity Manager deployment when considering a high-availability solution are as follows:

- ▶ Application server
- ▶ Relational database
- ▶ Directory server
- ▶ Identity Manager adapters

The primary business goal of the design approach adopted in the current implementation is to reduce down time to a minimum, using software components available with Tivoli Identify Manager, and to manage the failover mechanism, avoiding operating system or external high availability components when it is possible.

2.1 Application server

The application server runs the Identity Manager application that performs all the business-related operations and provides the Web interface to users. There is only one scenario to consider when designing the application server component for high availability: run the application server in its native clustered mode of operation.

The application server used by Identity Manager is IBM WebSphere® Application Server, which provides the ability to run as a WebSphere Application Server cluster.

Note: If you are planning to implement a high availability solution for Identity Manager at some point in time, even when your first implementation phase will only be based on a single Identity Manager Server, we suggest you consider the configuration of a WebSphere Application Server cluster solution with the deployment manager component and only one cluster member. These two components can be installed on the same machine. When you need to implement the high availability solution, it is a simple task to add another WebSphere Application Server cluster member on a new machine and install the Identity Manager Server component.

The WebSphere Application Server is the primary component of the WebSphere environment. The WebSphere Application Server runs a Java virtual machine, providing the run-time environment for the enterprise application code. The application server provides containers that specialize in enabling the execution of specific Java application components.

A cluster configuration contains WebSphere Application Server nodes, which are logical groups of one or more application servers on a computer. Nodes reside within an administrative domain called a *cell*, which the *deployment manager* manages. A node agent manages all managed processes on the node by communicating with the deployment manager to coordinate and synchronize the configuration. The deployment manager is the administrative process that provides a centralized management view and control for all elements in the cell, including the management of clusters.

Note: Tivoli Identity Manager does not support:

- ▶ Vertical cluster configuration that has more than one cluster member within a WebSphere Application Server node
- ▶ Functional cluster configuration that separates workflow processing and user interface processing on separate machines

Multiple machine environments extend basic single machine WebSphere Application Server configurations by distributing the Application Server over multiple machines, increasing the overall processing power from one machine to contributions from all machines in the configuration.

The flow of data in a WebSphere Application Server environment starts with a Web server receiving requests and routing them to the Application Server for processing. A WebSphere Application Server node stores administrative configuration data in XML files. A database can hold application data for applications that require a place to store data, such as user session information. There are also one or more administrative clients, such as the administrative console, for manipulating configuration data.

In a configuration such as the one depicted in Figure 2-1, each computer shape represents one WebSphere Application Server node in one cell. WebSphere Application Server also permits you to install both the WebSphere Application Server base product and the deployment manager on the same computer.

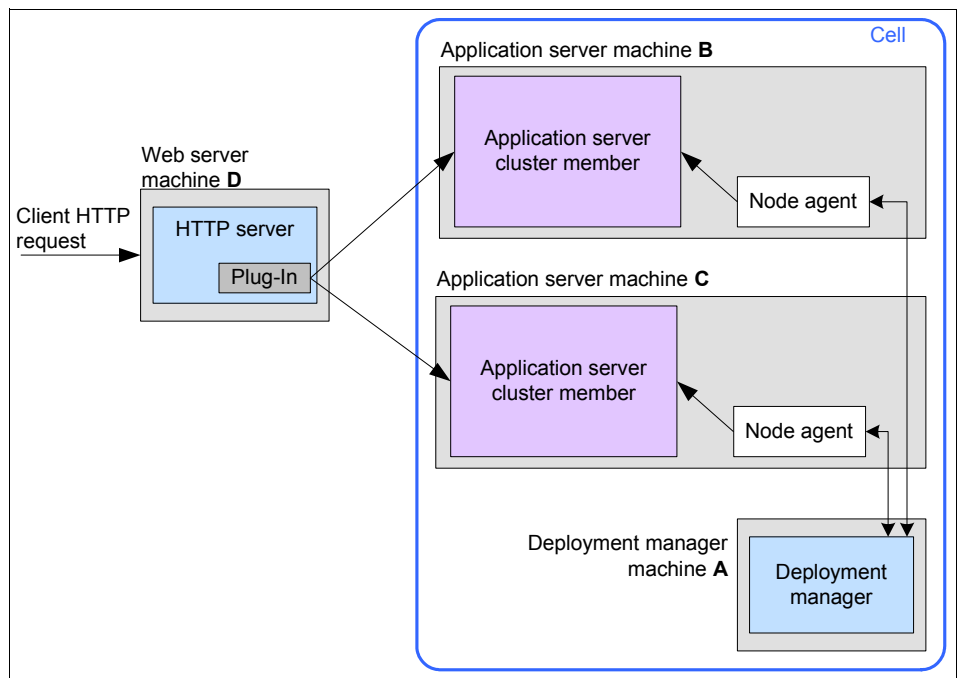


Figure 2-1 Horizontal scaling for WebSphere Application Server environment

In this example, the Web server on machine D distributes requests to clustered Application Servers on machines B and C. Cluster members on machines B and C are created in the same cluster.

Note: The TAA design approach leverages the functionality available in the security reverse proxy component, Tivoli Access Manager WebSEAL, to perform the authentication and authorization for users into Identity Manager. WebSEAL also automatically performs the load balancing and failover aspects in the event of an application server instance failure. This is especially useful if you are running separate instances of the Identity Manager application on separate application server instances.

We are not going to discuss the WebSphere Application Server general high availability methodologies in this redbook. There are many guides and Web sites that contain this information. The following redbooks might be of interest:

imitations about high availability of the Identity Manager messaging service solution.

The Java Message Service (JMS) server enables the Tivoli Identity Manager application to exchange information with other applications by sending and receiving data as messages. The messaging module provides assured asynchronous messaging to and between internal modules in the Identity Manager architecture.

An example of the JMS queue topology used on a WebSphere Application Server cluster environment is shown in Figure 2-2.

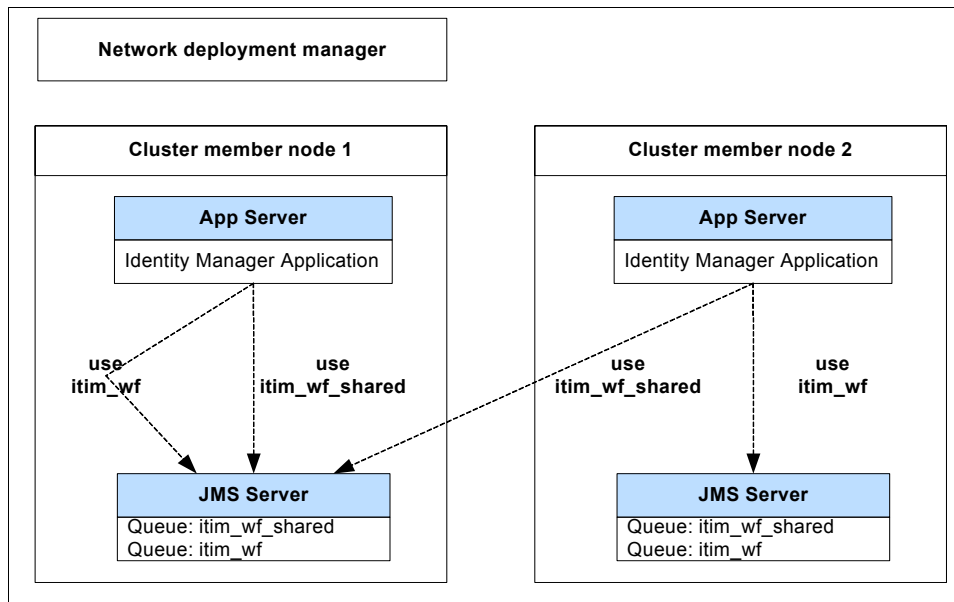


Figure 2-2 Workflow JMS queue topology (cluster)

In the example shown in Figure 2-2 on page 65, the workflow JMS queues are defined on all cluster members. The shared queue is only used on one cluster member. All application servers will send and receive messages from the shared queue.

- ▶ The shared workflow queue (itim_wf_shared) is only active on one JMS server. All cluster members send and receive messages to and from the same shared queue.
- ▶ The local workflow queue (itim_wf) is active on all cluster members. Only the Application server running on the same node will communicate with the local workflow queue.
- ▶ If the shared queue is not available, the message is sent to the local queue instead. This limits the impact on the overall workflow environment of the shared workflow queue JMS Server, if it is down.

WebSphere Application Server, Version 5, includes an implementation of the JMS 1.0.2 application program interface (API) as part of its support for J2EE™ 1.3.

The Embedded WebSphere JMS service is accessible only from WebSphere Application Server containers, and it is not interoperable with WebSphere MQ. The concept behind the Embedded WebSphere JMS server is to provide easy access to JMS for J2EE programmers. Although the underlying technology is provided by WebSphere MQ, the Embedded WebSphere JMS server is not meant to replace any external WebSphere MQ environment.

An approach to handle the WebSphere Embedded JMS server single point of failure (SPOF) is to use hardware-software clustering software (such as HACMP™). The network and hardware system can be used in conjunction with WebSphere Embedded JMS server in WebSphere Application Server Network Deployment V5.1 to build a highly available message-oriented middleware (MOM) system within a WebSphere V5 domain (cell).

Figure 2-3 on page 67 shows that such a system can tolerate any failure from the JMS server process, disk, operating system, host, and network. Therefore, this system negates the WebSphere Embedded JMS server single point of failure. The system provides a transparent and single image view to application clients that need to put or retrieve messages.

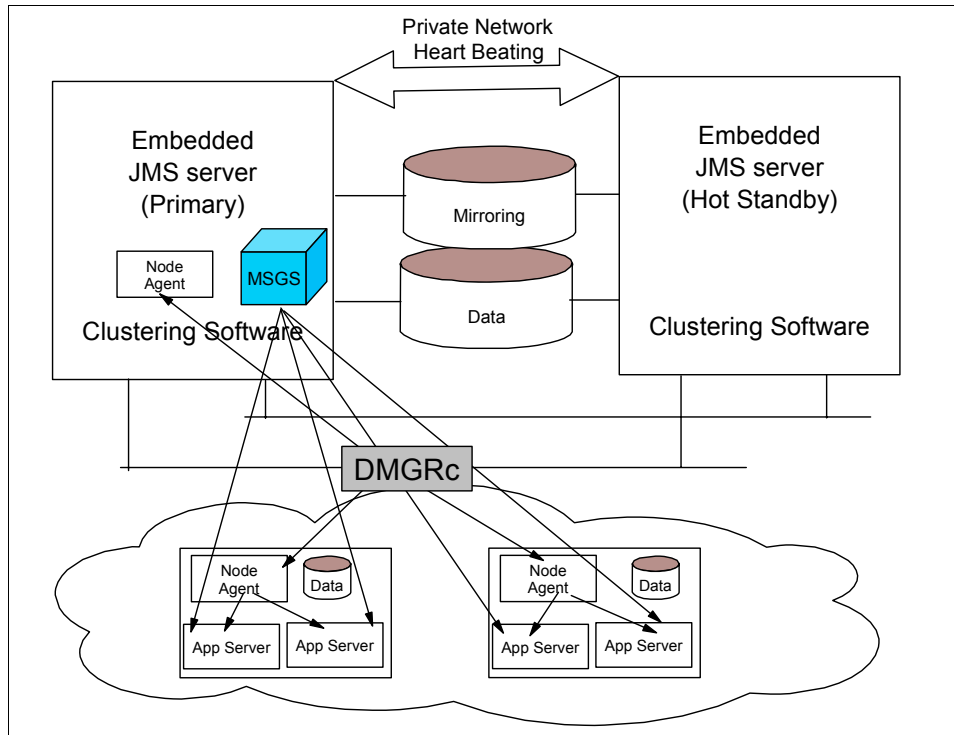


Figure 2-3 JMS server with clustering software

For details about the JMS WebSphere embedded messaging providers, refer to Chapter 11 of the IBM Redbook *IBM WebSphere V5.1 Performance, Scalability, and High Availability: WebSphere Handbook Series, SG24-6198-01*.

2.2 Directory server

Identity Manager requires an LDAP directory server to store essential data such as users, accounts, policies, and so on. As a result, it is an extremely critical component. Most LDAP servers have some level of functionality to allow for a high availability deployment through the use of data replication, as shown in Figure 2-4 on page 68.

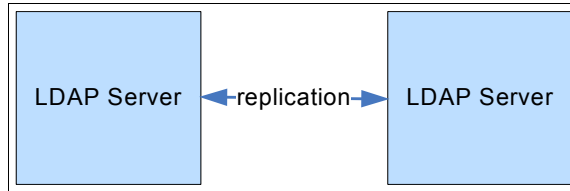


Figure 2-4 LDAP server replication

IBM Tivoli Directory Server allows for multiple LDAP servers to be configured with replication between them to ensure data integrity is maintained. Each Tivoli Directory Server server can be configured as a read/write-enabled server or as a read-only replica. The replication process keeps the data in multiple directory servers synchronized.

Replication provides three important benefits:

- ▶ Redundancy of information: Replicas back up the content of their supplier servers.
- ▶ Faster searches: Search requests can be spread among several servers, instead of a single server. This improves the response time for the request completion.
- ▶ Security and content filtering: Replicas can contain subsets of the data in a supplier server.

Through replication, a change made to one directory is propagated to one or more additional directories. In effect, a change to one directory shows up on multiple directories. The IBM Tivoli Directory Server supports an expanded master-replica replication model. Replication topologies are expanded to include:

- ▶ Replication of subtrees of the Directory Information Tree to specific servers
- ▶ A multi-tier topology, referred to as *cascading replication*
- ▶ Assignment of server role (supplier or consumer) by subtree
- ▶ Multiple master servers, referred to as *peer-to-peer replication*
- ▶ Gateway servers that replicate across networks

The term *master* is used for a server that accepts client updates for a replicated subtree. The term *replica* is used for a server that only accepts updates from other servers designated as a supplier for the replicated subtree.

The master/peer server contains the master directory information from where updates are propagated to the replicas. All changes are made and occur on the master server, and the master is responsible for propagating these changes to the replicas.

There can be several servers acting as masters for directory information, with each master responsible for updating other master servers and replica servers. This is referred to as *peer replication*. Peer replication can improve performance and reliability.

Inside the scope of our design approach, the peer replication topology is used in order to improve reliability by providing a backup master server ready to take over immediately if the primary master fails.

Identity Manager allows for configuration against a *single logical LDAP*. Note the use of the word *logical*. This means that it needs to refer to a Uniform Resource Identifier (URI) that allows access to an LDAP. Given this, consider the following scenarios where we describe two replication topologies usable for a high availability Tivoli Identity Manager solution, Manual failover to secondary LDAP and Automated failover to secondary LDAP.

Note: Each of the scenarios presented below details two LDAP servers for illustrative purposes. The scenarios can easily be extrapolated to a topology that uses multiple (more than two) LDAP servers.

2.2.1 Manual failover to secondary LDAP

Identity Manager is usually configured to reference the physical location of the master LDAP server. The issue is that if the primary LDAP server becomes unavailable, there needs to be manual intervention to configure Identity Manager to use another LDAP server. Such a configuration looks similar to Figure 2-5.

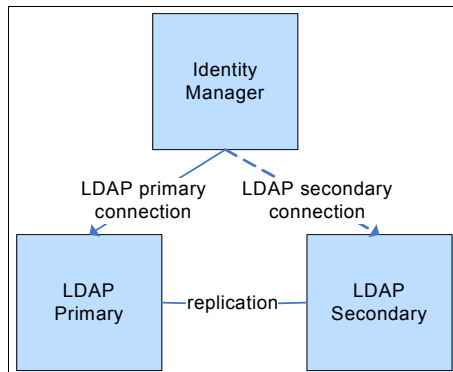


Figure 2-5 LDAP manual failover to replica

In the event of the primary LDAP becoming unavailable, Identity Manager needs to be configured manually to reference the secondary LDAP. This involves

modifying the relevant Identity Manager configuration file and restarting the Identity Manager application.

There are options to consider here in terms of the type of secondary LDAP to use. That is, should the secondary LDAP be read-only or read/write? Having a read-only secondary causes Identity Manager not to be able to perform operations where data needs to be written to the LDAP. This includes (but is not limited to) provisioning-related operations on users and accounts, and modifications to policies. Having a read/write secondary LDAP provides the benefits of having a fully functional Identity Manager application in the event that the primary LDAP fails. The only issue is that having a read/write secondary can potentially pose a data integrity and security risk if it is not properly secured. The same can be said of the primary LDAP. This is a completely acceptable and potentially more desirable option (from a functional perspective), provided the correct security measures are taken, and this is in fact a more likely deployment approach. One thing to note is that if the approach selected is to have a read/write secondary, both the primary and secondary LDAPs need to have replication to each other enabled. This is required to assist with recovery from a failure.

The main disadvantage of this approach as a whole is the manual intervention involved. The system is unavailable for a period of time due to the manual tasks involved and the requirement to restart the application.

Recovery

The secondary LDAP server is configured as read/write. The first step is to bring the failed primary LDAP back to a state where it is usable. Recall that in this type of configuration, both LDAP servers should be configured to accept and publish replication updates to each other. Thus, restarting the failed primary LDAP allows all the updates made to the secondary LDAP to be updated on the now functional primary. There is now the option to modify the Identity Manager configuration to once again reference the primary LDAP and restart the application or to simply leave Identity Manager referencing the secondary LDAP. From a logical perspective, in the case where the Identity Manager LDAP configuration is not modified, the secondary LDAP has now been promoted to primary and the primary demoted to secondary. That is, they have swapped roles. This can be the approach taken to reduce the Identity Manager application down time.

For our customer scenario in Part 2 this topology configuration is not acceptable for Tivoli Austin Airlines in order to achieve the primary business goal of continuous operation on this high availability environment.

2.2.2 Automated failover to secondary LDAP

The use of a *smart* IP load balancer negates the need for Identity Manager to be reconfigured in the event of an LDAP server failure. There are many load balancers on the market and the selection of a specific type is not discussed. The high-level requirements for a load balancer to use within the discussed environment are:

- ▶ Must be able to route application network traffic seamlessly and not modify data being routed
- ▶ Must be able to detect failure of a process accepting requests from the network
- ▶ Must be able to handle priority of potential destinations and route requests accordingly

In this case, Identity Manager is configured to use the URI of the load balancer for its LDAP requests. The load balancer then forwards the request to the primary LDAP server. In the event that the primary LDAP becomes unavailable, the request is forwarded to the secondary LDAP, which needs to be *promoted* to be the new primary LDAP. The logic is handled at the load balancer (some load balancer products might require manual configuration) and Identity Manager does not need to be reconfigured in the event of an LDAP server being unavailable. An illustration of this can be seen in Figure 2-6.

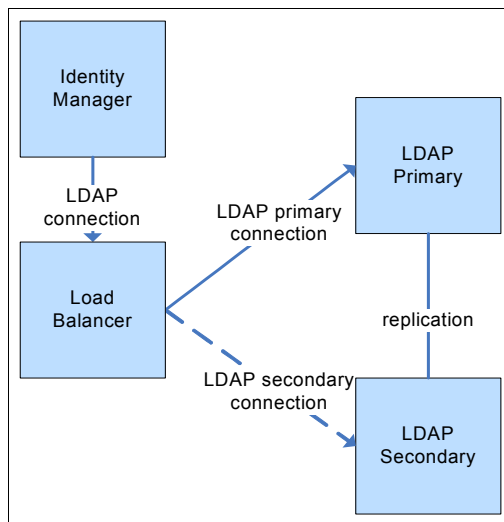


Figure 2-6 LDAP failover through a load balancer

The obvious disadvantage with this approach is the fact that the secondary LDAP is not being utilized until the primary LDAP fails. It is essentially functioning as a *hot standby*.

Instead of using a separate load balancer product, we can leverage an IBM Tivoli Directory Server Version 6.1 component called the *Tivoli Directory Server proxy server*. It acts as an intermediary for a client wanting to connect to an LDAP server. In other words, it proxies requests to a Tivoli Directory Server in the environment depending on the configuration specified and the request it receives from the client. This proxy server component can take the place of the load balancer as shown in Figure 2-7.

In case of a failover situation, the proxy server automatically promotes the secondary LDAP server to be the new primary LDAP server.

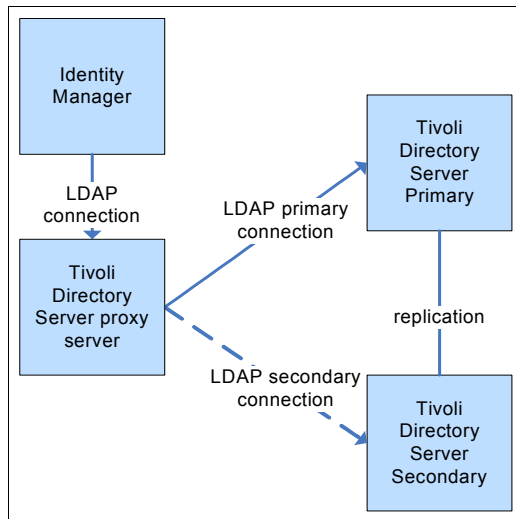


Figure 2-7 Automatic failover using Directory Server proxy server

The proxy server can actually help you achieve many other things that are not of any consequence for our high availability scenario. For a more general, non Identity Manager related introduction to the proxy server, consult Appendix C, “Tivoli Directory Server proxy server” on page 385 and the *IBM Tivoli Directory Server Administration Guide Version 6.1*, SC32-1564.

Dealing with a recovery situation after the primary LDAP server has failed can be approached the same way for either the load balancer or the Directory Server proxy server configuration.

Recovery

In both cases, load balancer or Directory Server proxy server, the secondary LDAP server has taken over the role of the primary LDAP server when a failover situation occurred. When we bring the old server (or a replacement) back online, it *automatically* becomes the secondary LDAP server until another failover situation occurs if we use the Directory Server proxy server. Some load balancer products might require manual configuration to implement this.

As soon as the old server is available again, the LDAP replication process starts to synchronize it with the primary LDAP server.

Caveats to the automated failover scenarios

As already mentioned above an obvious disadvantage is the fact that the secondary LDAP is not being utilized until the primary LDAP fails. It is essentially functioning as a *hot standby*.

The restriction in using the Tivoli Directory Server proxy server 6.1 is that it currently does not support LDAP paging, which is necessary for very large scale directories. So this approach should not be used under these circumstances and you need to fall back to the generic load balancer solution.

If you want to use the Tivoli Directory Server proxy server you may want to limit the maximum recommended size for your LDAP directory to 75,000 persons, each with an account on a single service.

However, it is Tivoli's stated intention to provide paging functionality in a future version of the Tivoli Directory Server proxy server.

2.3 Relational database

Some potential relational database high availability scenarios for Identity Manager are similar to the scenarios detailed in 2.2, "Directory server" on page 67. The concepts are similar and can be extrapolated from the LDAP discussion. That is, the functionality required to achieve the end result is usually available in most high-end relational databases. For a complete list of relational databases supported by Identity Manager, consult the latest release notes document.

In addition to this, there are also more sophisticated high availability solutions available for common relational database products (such as IBM DB2®) than for LDAP products. Relational database high availability strategies can generally be more tightly coupled with high availability operating system configuration options, systems management (automation, virtualization), and storage solutions (for example, Storage Area Networks) to offset some of the issues mentioned. That

is, there are prescribed methods to deploy a high availability relational database within specific product documentation.

Taking IBM DB2 UDB Version 8.2 as an example and using a combined approach leveraging operating system high availability features, the solution design team might choose to deploy DB2 in the following way.

2.3.1 Operating system cluster with DB2 active/standby

In the event of the active DB2 instance failing, the operating system clustering software starts the same instance on another node in the operating system cluster. This requires that all nodes in the operating system cluster have access to the same shared disk. While relatively simple in terms of DB2 clustering, this introduces delays during failover while the new processes are started and any in-flight transactions are rolled back. The database is accessed through the cluster address, so that no change in the Identity Manager database configuration is required during failover. See Figure 2-8 on page 74.

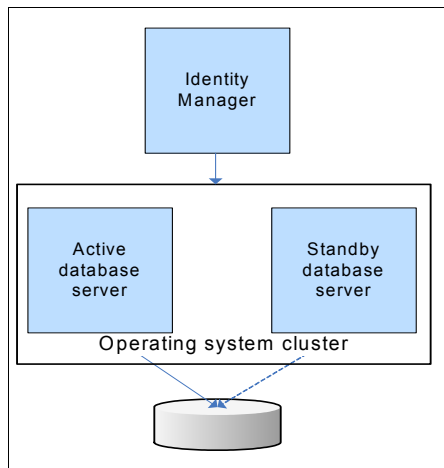


Figure 2-8 Operating system cluster with DB2 active/standby

2.3.2 DB2 mutual takeover multiple partition

All nodes in the database cluster operate in parallel. The database is partitioned so that if any server in the cluster fails, its partitions are failed over to the remaining nodes in the cluster. As with other strategies, there are various considerations in using this approach. The configuration still requires time for the *failed-over* partitions to be recovered, although as each partition has less than the whole volume of data, it is generally faster than an active/standby configuration. Database analysis needs to be performed to determine an

appropriate database schema, which is required for constructing a partitioned version of the Identity Manager database. This approach also requires that all servers have access to the file systems containing the database and transaction logs. The database is accessed through the cluster address, so that no change in the Identity Manager database configuration is required during failover. See Figure 2-9 on page 75.

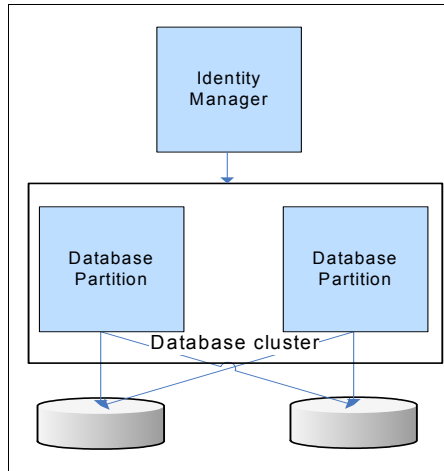


Figure 2-9 DB2 mutual takeover multiple partition

2.3.3 DB2 High Availability Disaster Recovery (HADR)

This involves using the DB2 automatic log shipping functionality to a secondary standby server, which applies the logs as it receives them. If the primary active server fails, then the DB2 client is automatically rerouted to the secondary failover server. Because no crash recovery is required, the failover to the secondary can be achieved in a minimal amount of time. Note that only one server can be active, read, and written to by a client. The secondary standby servers cannot participate in reads. There also needs to be careful planning if multiple failover standby databases are to receive updates. The DB2 client takes care of the failover; hence, there is no need for manual intervention in the Identity Manager database configuration during failover. See Figure 2-10 on page 76.

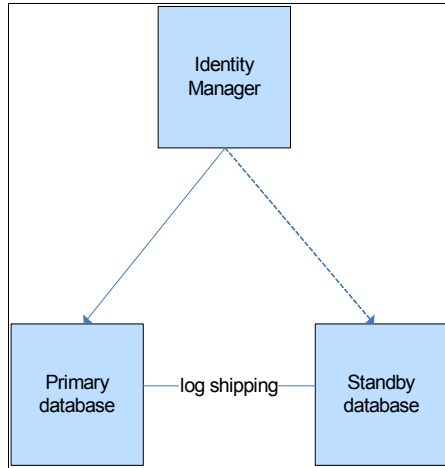


Figure 2-10 DB2 High Availability Disaster Recovery

Restriction: Before the DB2 client is rerouted to the standby server, the role of the standby server needs to be promoted to primary server. The current DB2 HADR implementation does not provide automatic promotion of the role; hence, you might need additional cluster manager software to be responsible for fault detection and for driving the failover activities. Although no licenses of cluster manager software are bundled with Identity Manager, you can find white papers describing how to automate the failover of HADR with various cluster manager software on the DB2 product domain.

Note: We intend the outlined DB2 options as examples. These are not the exhaustive high availability strategies available with DB2. For specific details and other options, refer to the DB2 8.2 product documentation.

2.4 Identity Manager adapters

Identity Manager manages accounts on managed resources through the use of adapters. This section speaks about adapters in general. Note that there are different adapters for each distinct type of managed resource, but you can generally apply the concepts discussed across the different types of adapters. For example, the concepts apply to the Windows Active Directory® adapter as well as the Tivoli Access Manager adapter, and so on.

Account operations issued by Identity Manager are executed by the relevant adapter for the type of managed resource the accounts reside on. This includes

provisioning, password management, and reconciliation operations. You might say that account operations are not mission critical and hence do not need to have high availability requirements factored in for a solution design. In many cases, this might be true. As with many things however, there are exceptions to the rule. Each deployment has specific requirements and it might be decided that certain operations must be highly available. For example, there can be cases where password resets, account suspensions, and account de-provisioning are deemed critical operations and must be highly available.

Note: The design consideration shown in the following section can be applied to all Tivoli Identity Manager adapters that can be implemented using an agent-less approach. This also includes the Tivoli Identity Manager adapters based on Tivoli Directory Integrator technology.

There are two aspects to consider when dealing with high availability for Identity Manager interactions with its adapters and subsequently the adapter interactions with the managed resource hosting the accounts being managed as illustrated in Figure 2-11. The first is the adapter interactions with the managed resource, for example, the Identity Manager Windows Active Directory adapter and its interactions with Windows Active Directory. The high availability aspects between these two components are not within the scope of this discussion because each managed resource has different approaches to high availability and they can vary in completely different ways. For example, a Windows Active Directory environment has a different high availability design and implementation approach compared to a relational database environment, both of which are different from a Tivoli Access Manager environment, and so on. The managed resource is viewed as a logical entity in the context of this discussion and assumed to have been designed for high availability by the solution design team responsible for the managed resource in question. The focus of this discussion is on the Identity Manager specific components required for ensuring account operations are highly available.

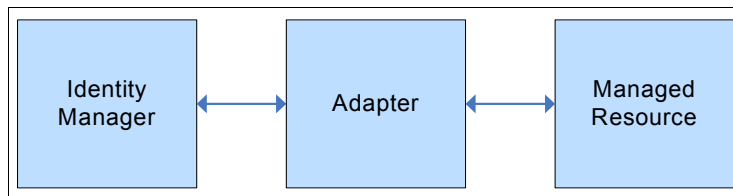


Figure 2-11 Identity Manager interactions with managed resources

Identity Manager, as per the approach taken with the LDAP and database, references its adapters via a URI (Uniform Resource Identifier). As previously mentioned, this is a *logical* location.

Given this, consider the following scenarios:

- ▶ Manual failover to secondary adapter
- ▶ Automated failover to secondary adapter
- ▶ Event notification on an HA adapter configuration

Note: The scenarios we present next are failover scenarios. We do not advise that you consider using a load balancing strategy for the adapters. This can cause issues with operations such as reconciliations, which rely on using a dedicated adapter instance.

The following scenarios consider the use of two adapters in each case. This is a solution that can handle the single point of failure on the Identity Manager adapter layer. In a high availability solution design, adding a third adapter in failover mode does not carry real benefits to the solution.

2.4.1 Manual failover to secondary adapter

This scenario involves having a secondary adapter available for use in the event that the primary adapter is unavailable as shown in Figure 2-12 on page 79. The URI reference to an adapter is not stored within a configuration file in Identity Manager, it is stored as an attribute of the service definition. Because of this, a change in this attribute does not require a restart of the Identity Manager application. All that needs to be done is for the value to be modified and saved within the service definition for it to take effect. The assumption we make here is that the secondary adapter is configured exactly the same way that the primary adapter is configured. If not, additional attributes need to be modified within the service definition. This depends on the difference in configuration settings between the two adapter instances. Refer to 2.4.3, “Event notification on an HA adapter configuration” on page 81 for adapter configuration when the event notification feature on the adapter is enabled.

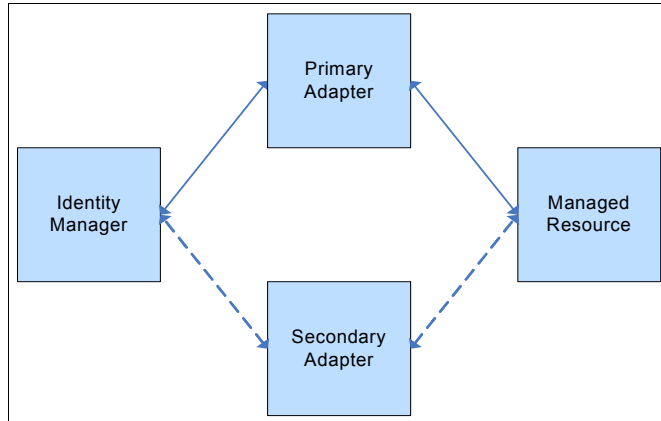


Figure 2-12 Manual failover to secondary adapter

If the primary adapter becomes unavailable, manual user intervention is required to make the change in the relevant service definition to reference the secondary adapter. This secondary adapter can be an active standby or it can be brought up to an active running state when it is required. An active standby secondary adapter will result in a shorter amount of down time but will require the system resources to be available for use by the secondary adapter. If the secondary adapter is brought to an active running state, it might take a little longer to achieve availability, but it does not consume system resources (other than disk space) while it is not required. The resources consumed while in an active state but not being utilized by the Identity Manager Server are relatively insignificant, however. The approach taken depends on operational requirements of the organization, for example, the amount of system resources available or service level agreements that must be met.

Recovery

It might be acceptable to use the secondary adapter as the active adapter until such a time where it is unavailable and perform the steps mentioned to have Identity Manager use the primary adapter again. If it is essential that Identity Manager always use the primary adapter if possible, then a suitable approach is to wait for the next available change window to do so (following the change control procedures of the environment in question), preferably when the Identity Manager Server is not performing any tasks related to the adapter in question. For example, we do not advise that you do this while a reconciliation is running.

2.4.2 Automated failover to secondary adapter

This scenario relies on the secondary adapter being available for use at all times. There is also a requirement to leverage the use of a suitable IP load balancer as

detailed in 2.2.2, “Automated failover to secondary LDAP” on page 71. The Identity Manager Server is configured to reference the URI of the load balancer which then routes requests to the relevant available adapter as shown in Figure 2-13.

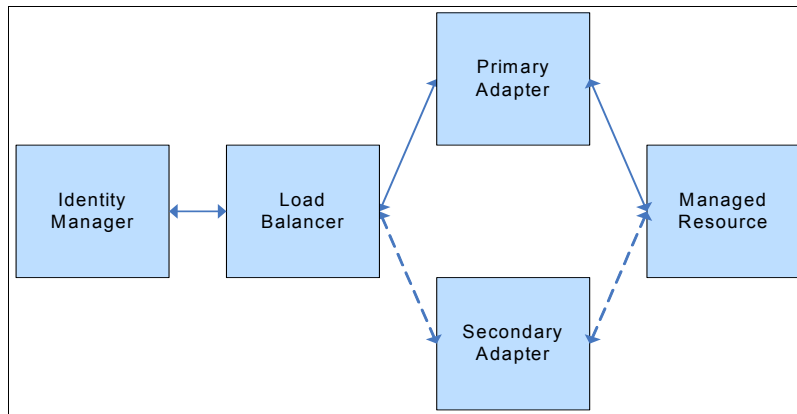


Figure 2-13 Automated failover to secondary adapter

An important point to note with this approach is the use of the word *failover*. The load balancer is configured to prioritize requests to the primary adapter. That is, it attempts to use the primary adapter at all times until such a time when it is unavailable. In the event of unavailability, the load balancer uses the secondary adapter. Note that the secondary adapter must be configured exactly the same way the primary adapter is configured, for example, the primary and the secondary adapter have to use the same keys and certificates, the same authorized user ID and password, and the same ports. If the event notification is enabled on the adapter, see 2.4.3, “Event notification on an HA adapter configuration” on page 81 for design and implementation considerations.

Recovery

There are two distinct recovery scenarios that we need to examine.

- ▶ If the load balancer allows modification of priorities while it is running it should be modified to give priority to the secondary adapter while the primary adapter is being brought back to a functional state. If the desire is to use the primary adapter where possible, the load balancer needs to be reconfigured to prioritize requests to be routed to the primary adapter. This should be done within the change control procedures of the environment and during a suitable change window, preferably when the Identity Manager Server is not performing tasks related to the adapter. For example, we do not advise that you do this while a reconciliation is running.

- ▶ If the load balancer does not allow modification of priorities while it is running, the network connection between the load balancer and the primary adapter should be disabled (either directly at the network configuration level or via a device such as a firewall) while the primary adapter is brought back to an active functional state. The network connectivity can be restored within the change control procedures of the environment and during a suitable change window, preferably when the Identity Manager Server is not performing tasks related to the adapter.

2.4.3 Event notification on an HA adapter configuration

Event notification is a feature of Identity Manager adapters that updates the Tivoli Identity Manager Server at set intervals. Event notification detects changes that are made on the managed resource and updates the Tivoli Identity Manager Server with the changes. You can enable event notification if you want to have updated information from the managed resource sent back to the Tivoli Identity Manager Server between full reconciliations.

For the high availability adapter design, you have to consider that you can only enable the event notification feature on the primary adapter.

Event notification on the primary adapter

When event notification is enabled, a database of the reconciliation data is kept on the machine where the adapter is installed. The database is updated with the changes that are requested by the Tivoli Identity Manager Server and will remain synchronized with the server. You can specify an interval for the event notification resource. When the interval has elapsed, any differences between the managed resource and the database are forwarded to the Tivoli Identity Manager Server and updated in the local snapshot database.

Figure 2-14 on page 82 describes the components involved in the synchronization process of the changes that occurred on the managed resources:

1. The local database on the adapter machine server is updated with the changes that are requested by Tivoli Identity Manager Server. The local snapshot will be synchronized with the server (step 1).
2. The adapter, at each event notification interval, compares the local information with the managed resource (steps 2 and 3).
3. The differences between the managed resource and the local database are forwarded to the Identity Manager Server (step 4).
4. The differences are updated on the local database (step 5).

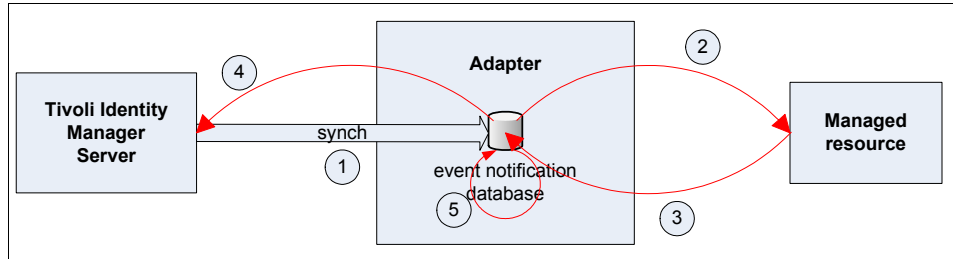


Figure 2-14 Adapter event configuration process

A suitable approach is to not enable the event notification feature on the secondary adapter when an automatic failover mechanism through a load balancer has been implemented for the adapter's high availability. Since in this configuration both the primary and secondary adapters are active, both will update the Identity Manager Server with unpredictable effects on the Identity Manager Server side. Figure 2-15 shows this design approach when event notification is enabled. For the recovery, see the procedures described in 2.4.2, "Automated failover to secondary adapter" on page 79.

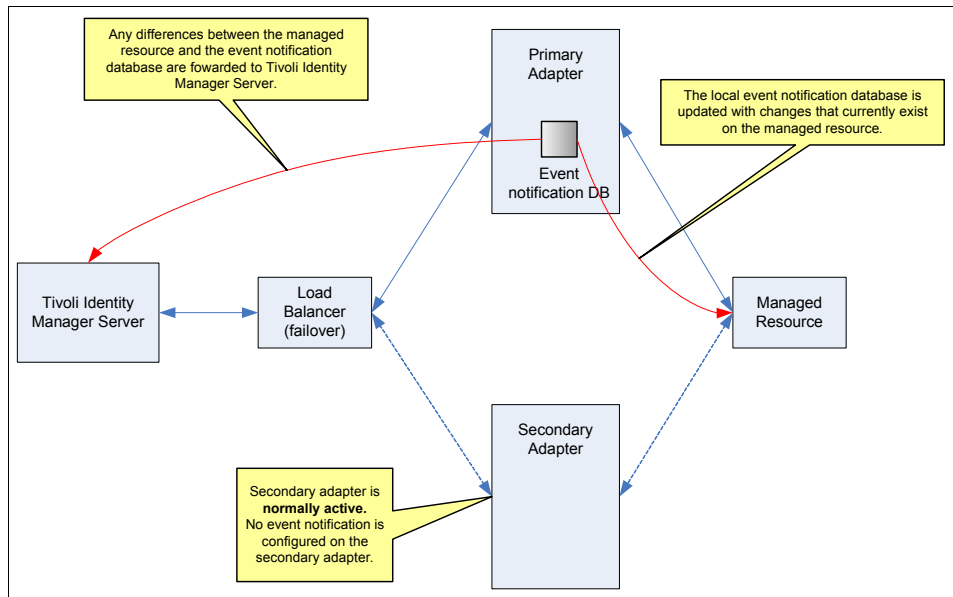


Figure 2-15 Event notification configured only on the primary adapter

Benefits

By implementing this solution (event notification not enabled on the secondary adapter), no extra cost (operating system cluster or automatic scripting to

change adapter configuration during the failover) is required, and there is no synchronization problem between the Identity Manager Server and the managed resource.

Limitations

When the primary adapter is unavailable, the secondary adapter does not update the Identity Manager Server based on the event notification feature. A suitable approach is to wait for the next available change window to re-prioritize the load balancer request to the primary adapter as soon as it is available again. This should be done within the change control procedures of the environment and during a suitable change window.

2.5 Physical HA component architecture

The Identity Manager system is always deployed as part of an enterprise environment. A goal of the physical component architecture is to be flexible enough to support different configuration options. This section discusses the different network zones that you find within an enterprise environment and the placement options for the design of different Identity Manager components to support continuous operations.

For general considerations about the physical component architecture of an Identity Manager solution, refer to Chapter 3. “Identity Manager components structure” in the *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996-01.

2.5.1 Component configuration and placement

In the design of the Identity Manager architecture for a production setting, we do not recommend that you deploy all the Identity Manager components within a single network.

In the next sections, we discuss how various Tivoli Identity Manager components, designed to accomplish high availability requirements, relate to the network. We provide recommendations for how they should be distributed in a typical architecture.

2.5.2 Network zones

We have to consider four types of network zones in our discussion of Tivoli Identity Manager component placement:

- ▶ Private Network (the extranet)
- ▶ Controlled (an extranet-facing DMZ and the intranet)
- ▶ Restricted (a production network)
- ▶ Secure (a management network)

An *extranet* is a private network that uses the Internet protocol and the public telecommunication system to securely share part of a business's information or operations with partners, customers, or other businesses. An extranet can be viewed as part of a company's intranet that is extended to users outside the company. Think of an extranet as a private portion of the Internet.

In this section, we branch the extranet from the intranet to emphasize that some Identity Manager services, such as self-care management and delegated administration, can be shared with business partners. This is also useful to describe the security and hardening considerations in 2.6, “Security and integrity for high availability” on page 87.

Since we do not place any components in an Extranet zone except a user's Web browser, we take a closer look at the remaining zones.

For a general overview of network zones, refer to Chapter 2, “Common security architecture and network models” in the IBM Redbook *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014-02.

Extranet DMZ (controlled zone)

The extranet DMZ is generally a controlled zone that contains components with which clients can directly communicate. It provides a “buffer” between the private network zone (extranet) and internal networks. Because this DMZ is typically bounded by two firewalls, there is an opportunity to control traffic at multiple levels:

- ▶ Incoming traffic from the extranet to hosts in the DMZ
- ▶ Outgoing traffic from hosts in the DMZ to the extranet
- ▶ Incoming traffic from internal networks to hosts in the DMZ
- ▶ Outgoing traffic from hosts in the DMZ to internal networks

Because a typical Tivoli Identity Manager deployment integrates with a Tivoli Access Manager environment, we need to consider the placement of WebSEAL in the DMZ. WebSEAL can be used to protect access to the HTTP server used by the Tivoli Identity Manager GUI server and to a customized self-care service used by business partners and other external users. The DMZ is an appropriate location for the WebSEAL component of Tivoli Access Manager, and in

conjunction with the available network traffic controls provided by the bounding firewalls, it provides the ability to deploy a highly secure Web presence without directly exposing components that may be subject to attack by network clients.

Note: You can obtain more information about IBM Tivoli Access Manager design architecture consideration by reading the IBM Redbook *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014-02 and *Enterprise Business Portals II with IBM Tivoli Access Manager*, SG24-6885.

Production or management networks (restricted/secure zones)

One or more network zones can be designated as *restricted* or *secure*, that is, they support functions to which access must be strictly controlled, and of course, direct access from an uncontrolled network and also an extranet network should not be permitted. As with an Internet DMZ or extranet DMZ, a restricted network is typically bounded by one or more firewalls, and incoming and outgoing traffic can be filtered appropriately. Access to a secure zone is only available to a small group of authorized staff. Access into one area does not necessarily give you access to another secured area.

These zones typically contain Tivoli Identity Manager and infrastructure server components.

Intranet (controlled zone)

Typically, a controlled zone, such as a corporate intranet behind one or more firewalls, is not heavily restricted in use, but an appropriate span of control exists to assure that network traffic does not compromise the operation of critical business functions.

Other networks

Keep in mind that the network examples we use do not necessarily include all possible situations. There are organizations that extensively segment functions into various networks. However, in general, the principles discussed here can easily translate into appropriate architectures for such environments.

Placement of various Tivoli Identity Manager components within network zones is a reflection of the security requirements in play, and alternatively, a choice based upon an existing and planned network infrastructure and levels of trust among the computing components within the organization. While requirement issues might often be complex, especially with regard to the specific behavior of certain applications, determination of a Tivoli Identity Manager architecture that appropriately places key components is generally not difficult. With a bit of knowledge about the organization's network environment and its security policies, reasonable component placements are usually easily identifiable.

Figure 2-16 summarizes the general Identity Manager component type relationships to the network zones we have discussed.

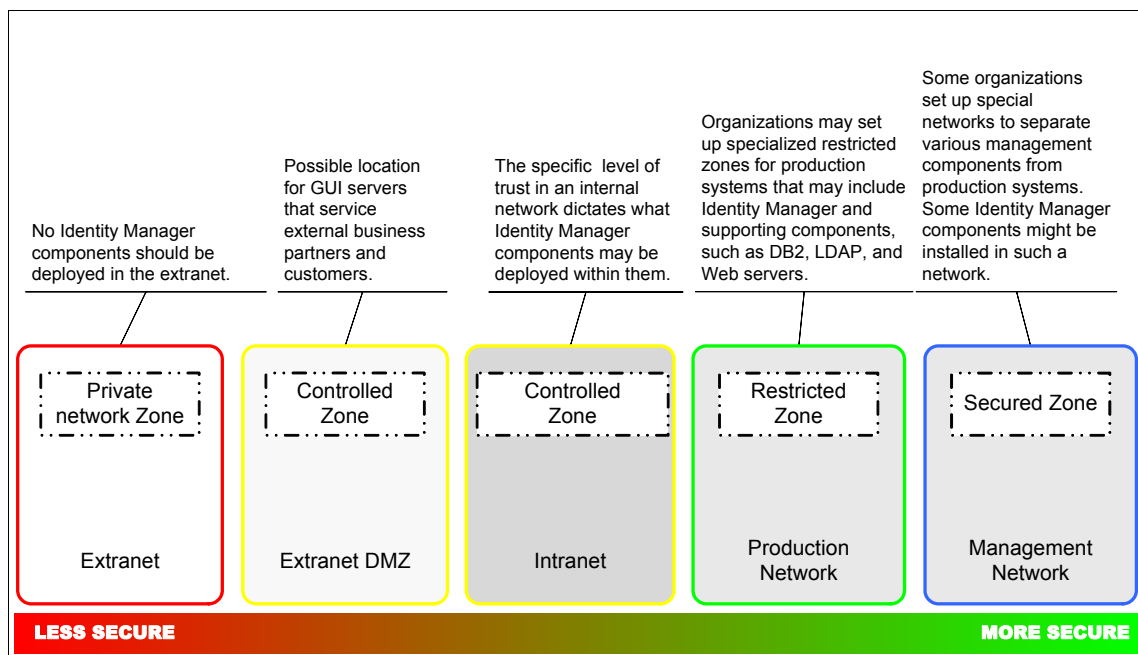


Figure 2-16 Network zones for Identity Manager placement

Because all the components of Tivoli Identity Manager have either information that access should be restricted to, or support such resources, we recommend that they all are placed in a restricted zone. An exception to this might be to place a Web server in the extranet DMZ to manage external requests from business partners if no general access control solution, such as Access Manager WebSEAL, is in place.

Figure 2-17 on page 87 shows an example architecture for integrating Identity Manager high availability design components. Note that firewalls are introduced to separate the networks and permit access only through specified ports. In this example, access from the extranet is only allowed on the listening ports to the WebSEAL server in the DMZ, and the WebSEAL server is configured to access the back-end Web servers through alternative ports, hence forcing all users requesting access to the back-end Web servers to be authenticated by WebSEAL.

We discuss the security hardening requirements for the integrated Identity Manager high availability design architecture in 2.6, "Security and integrity for high availability" on page 87.

In Figure 2-17, the self-care application represents the logical components for all Identity Manager services published towards the extranet. The WebSphere Application Server cluster that hosts the self-care services has been separated from the Identity Manager Server cluster essentially for performance reasons.

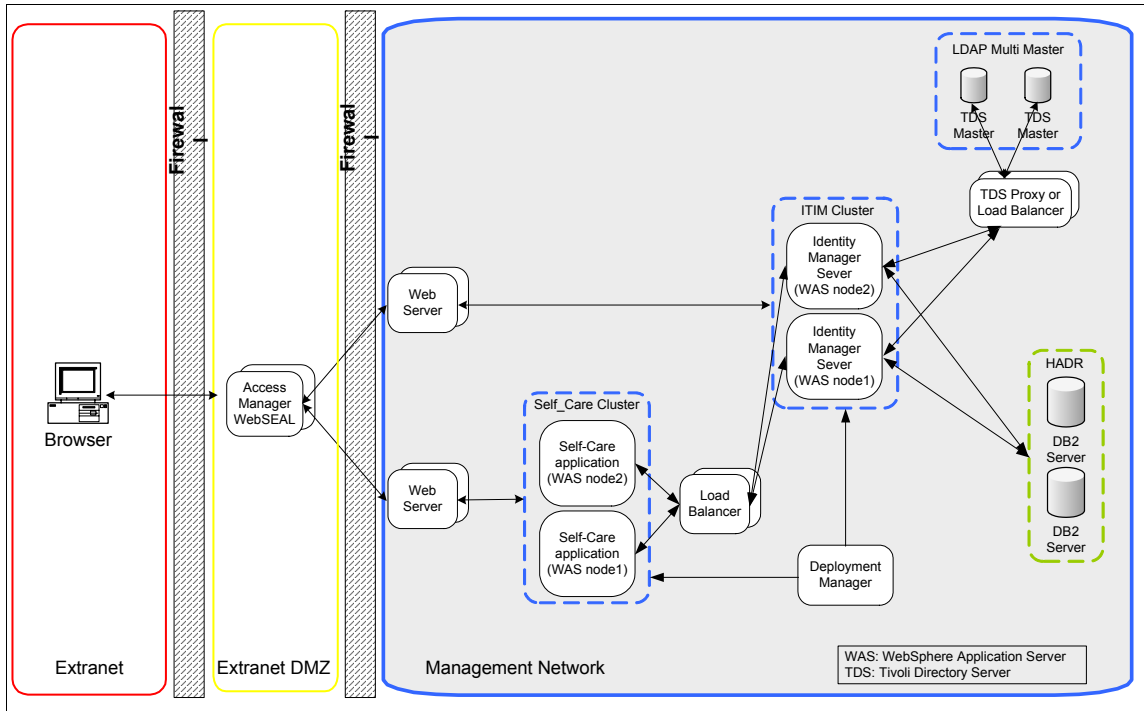


Figure 2-17 Integrated architecture for Identity Manager high availability design components

2.6 Security and integrity for high availability

The Identity Manager software components allow flexibility to enable or disable various security features within each component and between the components. Many of these are generally disabled by default to allow for ease of development and initial deployment. As part of the design and planning, these configuration differences should be documented and decisions made with regards to which features need to be enabled and in which environments.

This section discusses several design considerations to note within a high availability architecture for Identity Manager as shown in Figure 2-17.

For more general security considerations, refer to Chapter 5, “Operational Solution design”, in *Identity Manager Design Guide with IBM Tivoli Identity Manager*, SG24-6996-01.

The main security considerations to note within an Identity Manager high availability environment are:

- ▶ Identity Manager and LDAP communication via SSL.

Typically configured for SSL in production. Setting the SSL option on the Tivoli Directory Server proxy server (`ibm-slapdSecurity` attribute on Tivoli Directory Server) enables the proxy server (SSL server) to receive either secure (default port 636) or insecure (default port 389) communications from Identity Manager (SSL client). Setting the `SSL only` option on the Tivoli Directory Server proxy server enables the proxy server (SSL server) to receive only secure (default port 636) communications from Identity Manager (SSL client).

If you only need *encryption* on the LDAP communication, it is enough to select *Server authentication* for the Tivoli Directory Server proxy server (`ibm-slapdSslAuth` attribute on Tivoli Directory Server). For server authentication, the Directory Server proxy server supplies Identity Manager with the proxy server’s X.509 certificate during the initial SSL handshake. If Identity Manager validates the proxy server’s certificate, then a secure, encrypted communication channel is established between the proxy server and the Identity Manager Server. For server authentication to work, the Directory Server proxy server must have a private key and associated server certificate in the server’s key database file.

If you need *encryption* on the LDAP communication and *two-way authentication* between the LDAP client (Identity Manager) and the LDAP server (Directory Server proxy server), you have to select the *Server and client authentication* type for the Directory Server proxy server. With client authentication, the LDAP client (Identity Manager) must have a digital certificate available (based on the X.509 standard). This digital certificate is used to authenticate the LDAP client to the LDAP server (Directory Server proxy server).

- ▶ Tivoli Directory Server proxy server and Tivoli Directory Server back-end communication via SSL.

In this case, you want to use the same considerations described in the previous “Identity Manager and LDAP communication via SSL” bullet. In this case, however, the LDAP server is represented by the back-end Tivoli Directory Server and the LDAP client is the proxy server.

- ▶ Tivoli Directory Server peer-to-peer topology communication via SSL.

In this case, you want to use the same considerations described in the “Identity Manager and LDAP communication via SSL” bullet. Here, each back-end LDAP peer serves both LDAP server and LDAP client role.

- ▶ Pick a random password encryption key.

A default value for an encryption key is given during installation. To ease your change management tasks the same password encryption key should be used on the deployment manager and on each node where an Identity Manager instance is installed.

- ▶ Identity Manager and adapter communication via SSL.

This option is typically enabled in production. You have the option to configure the SSL communication to be one-way or mutually authenticated. Certain environments only require one-way SSL while others with strict security policies might mandate that the SSL communications are mutually authenticated. The same SSL configuration (same SSL option and same certificate) has to be used on the adapters in a high availability configuration.

- ▶ Identity Manager Web application access only via HTTPS.

This option enforces that users can access the Identity Manager application only over a secure SSL channel. This requirement varies between deployments depending on the security policies of the organization.

- ▶ Application Server security enabled.

Enable WebSphere global security on the Identity Manager Server WebSphere Cluster environment. WebSphere global security ensures that authenticated users have the necessary permissions to access Identity Manager EJB™ components. Configuring this security component involves configuring an authentication mechanism, a user registry, and optionally, Java 2 security.

Use the same security consideration to enable security on the application server cluster where the self-care Identity Manager services are installed. An SSL connection has to be enabled between the WebSphere cluster where the self-care application is installed and the Identity Manager WebSphere Application Server cluster to encrypt the communication between the Identity Manager Server cluster and the self-care server cluster.

Refer to the online *IBM Tivoli Identity Manager Information Center Version 4.6, SC23-5267*, and the *IBM Tivoli Identity Manager Server Installation and Configuration Guide for WebSphere Environments Version 4.6, SC32-1750*, for detailed specifics about performing these actions.

- ▶ DB2-encrypted communications.

Enforce data encryption between the DB2 server and the DB2 client. Configure the DB2 database server to use DATA_ENCRYPT as the authentication type. A value of DATA_ENCRYPT means the server accepts encrypted SERVER authentication schemes and the encryption of user data.

Refer to the online IBM DB2 UDB Version 8.2 Information Center for detailed specifics about performing these actions.

- ▶ DB2 HADR environment secure communications.

Enforce data encryption communication using a private network for the TCP/IP communication between the primary and standby databases. DB2 HADR uses TCP/IP for communication between the primary and standby databases.

As shown in the list above, many of the connections between Identity Manager components can be secured using SSL. Figure 2-18 shows these connections.

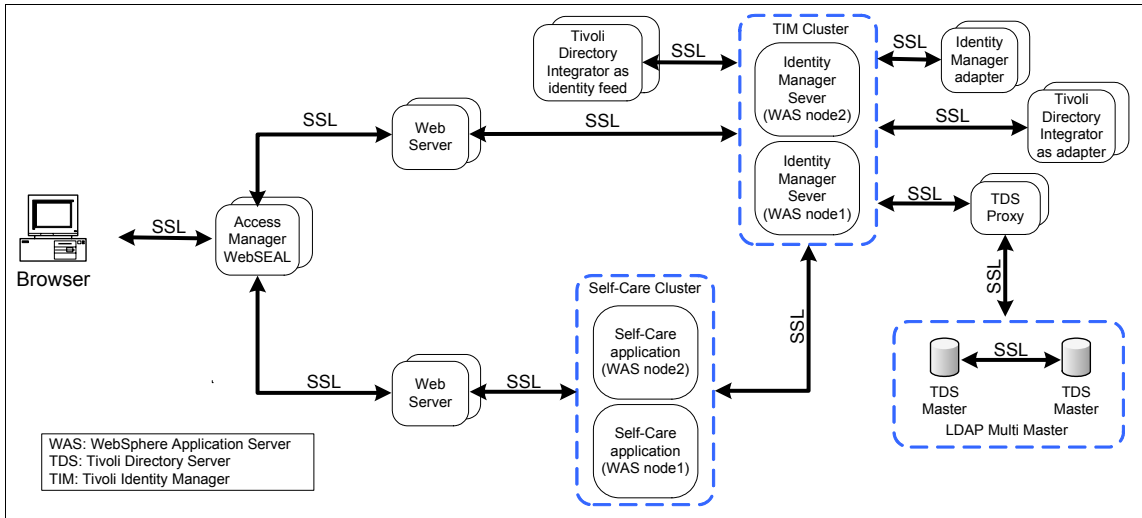


Figure 2-18 Identity Manager component connections with SSL support

The most critical of these connections from a security perspective are the connections from the Web browser to the Web server and on to the Identity Manager server, and the connections from the Identity Manager Server to its adapters. These connections transmit account passwords, so the privacy of the data on these connections is very important. The remaining connections do not transmit clear text account passwords, but they do transmit credentials that are used for authenticating the Identity Manager components and might transmit sensitive user data.

2.7 Conclusion

This concludes our comprehensive discussion about high availability and failover situations for Tivoli Identity Manager. We have addressed the application server clustering techniques along with availability issues for the LDAP directory and the relational database server infrastructures. In addition to these infrastructure

components, we have also discussed the Identity Manager adapters. Finally, we covered best practices considerations about high availability on physical component design, security and integrity, and component communications.

The Tivoli Austin Airlines customer scenario will implement a high availability solution based on their individual business requirements.



Part 2

Customer Scenario

In this part, we describe a customer scenario based on the fictive Tivoli Austin Airlines corporation that has been introduced in the first Identity Manager-related IBM Redbook *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996.

This time, the company is challenged to address advanced requirements, including high availability for a 24x7 implementation and deployment of a self-care application to its partners and customers as well as its employees.



Tivoli Austin Airlines, Inc.

This chapter provides an introduction to the overall structure of Tivoli Austin Airlines (TAA) corporation, including its business profile, current IT architecture, and infrastructure, as well as their business vision and objectives.

Note: All names and references for company and other business institutions used in this chapter are fictional. Any match with a real company or institution is coincidental.

3.1 Company profile

Tivoli Austin Airlines (TAA) is one of the major airlines within the continental United States. It has been in business for 14 years now and has increased its flight operations to over 750 daily flights nationwide, with the motto “To fly anything, anywhere.” This is an increase of some 150 flights a day over approximately the past two years and sales continue to increase.

The following sections describe:

- ▶ The geographic distribution of TAA
- ▶ The company organization
- ▶ HR and personnel procedures

Note: The following sections describe the company information relevant to an Identity Manager implementation and are not intended to be a complete description of the company.

3.1.1 Geographic distribution of TAA

TAA is based in Austin, Texas, with the corporate head office and central IT data center located near the Austin International Airport. TAA further operates the following three regional centers:

RW	Regional center West (San Francisco)
RA	Regional center Austin (Austin, within the central IT data center)
RE	Regional center East (New York)

These regional data centers service the IT needs of the region, such as LAN support, help desk support, and user administration. The corporate IT staff, such as systems programmers and developers, are located at the central IT data center.

TAA operates full aircraft service and maintenance centers in the Austin, New York, and San Francisco airports with satellite maintenance operation centers located in the rest of the major airports. These service and operation centers are responsible for the repair and servicing of the TAA fleet. All major repairs and servicing are done at the full service centers in Austin, New York, and San Francisco, while all the other sites are responsible for emergency and light maintenance. In the case where an aircraft must undergo extensive emergency mechanical repairs, the necessary repairs are made where the aircraft is in order to get the aircraft to one of the regional centers for the remainder of the repairs and flight certification.

TAA also runs multiple Customer Service Centers (CSCs) in the major airports, servicing front-office functions, such as ticketing, member lounges, baggage management, and staff HR systems. The CSCs have no local staff, so they contact the regional centers for technical support.

The TAA sites are:

- | | |
|--------------------------|--|
| Austin, TX | This is the IT center housing the core IT infrastructure and staff. It is also the Regional Center for the Austin region and contains the technical support staff for the Central region. It is the home of the largest aircraft service and maintenance center TAA operates with most of its parts vendors co-located to this site. It is the distribution hub for parts heading to its two other full service regional sites. Furthermore, TAA also operates a Customer Service Center at this location. |
| San Francisco, CA | This site is the Regional Center for the West region. It contains the technical support staff for the West region. This site is for the West region aircraft service and maintenance center, and there is also a Customer Service Center. |
| New York, NY | This site is the Regional Center for the East region. It contains the technical support staff for the East region. This is the site for the East region aircraft service and maintenance center, and there is also a Customer Service Center. |
| Seattle, WA | This site contains a Customer Service Center, is part of the West region, and is supported by the regional center in San Francisco. |
| Los Angeles, CA | This site contains a Customer Service Center, is part of the West region, and is supported by the regional center in San Francisco. |
| Denver, CO | This site contains a Customer Service Center, is part of the Austin region, and is supported by the regional center in Austin. |
| St. Louis, MO | This site contains a Customer Service Center, is part of the Austin region, and is supported by the regional center in Austin. |
| Detroit, MI | This site contains a Customer Service Center, is part of the East region, and is supported by the regional center in New York. |
| Raleigh, NC | This site contains a Customer Service Center, is part of the East region, and is supported by the regional center in New York. |

The geographic distribution of TAA is shown in Figure 3-1 on page 98. The figure shows the three regions, West, Central, and East.

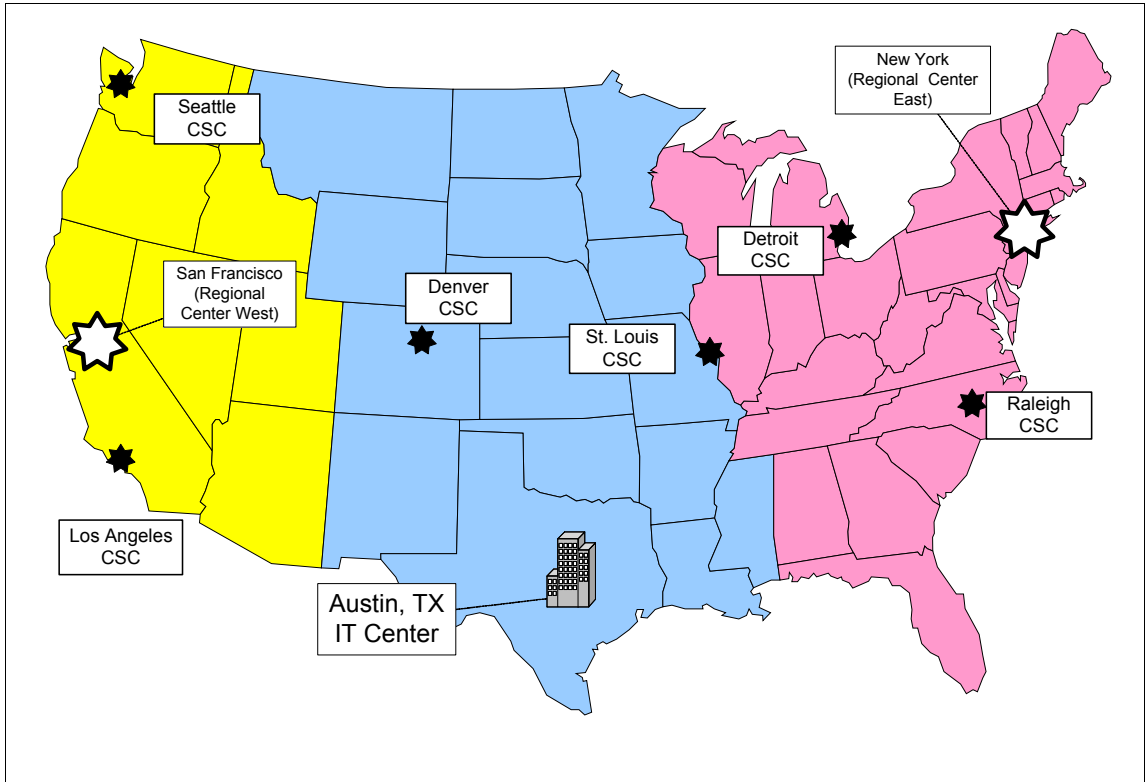


Figure 3-1 TAA geographic distribution

TAA's extension

“To fly anything, anywhere” is the motto for TAA. Since this represents the business purpose of the company, TAA has just opened a new customer service center outside of the United States of America. This CSC is located in Mexico City inside the Mexico City International Airport to provide all kinds of services to customers from Mexico including online ticket sales and account management for the “TAA frequent passenger benefits.” This effort complies with the business objective of TAA becoming one of the premiere airlines in the world. Figure 3-2 on page 99 depicts the location of the new CSC.



Figure 3-2 TAA expansion into Mexico City

The CSC in Mexico City provides the same services as the US locations, such as servicing front-office functions, ticketing, member lounges, baggage management, aircraft maintenance and repair, and staff HR systems. This CSC has no local support staff; they contact the Regional Center, Austin, for technical support and parts distribution.

Since the official language in Mexico is Spanish, TAA has implemented language support on most IT systems to provide Spanish local support to messaging services, including e-mail, alerts, and self-service Web pages.

3.1.2 Organization of TAA

The company is split into four key areas, the three regions and a core services division. This is shown in Figure 3-3 on page 100.

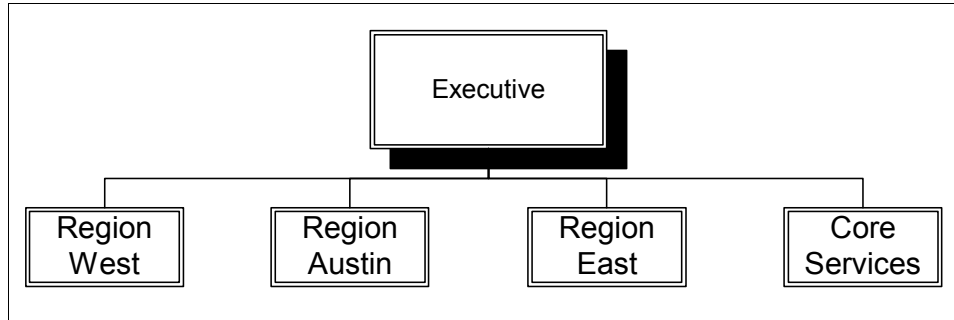


Figure 3-3 High-level organization chart

Each of the regions is responsible for the operation of the local services in that region, including customer service, baggage handling, ground services, aircraft maintenance, airport liaison, and staffing. The three regions have the same structure. The organization chart for the central region is shown in Figure 3-4.

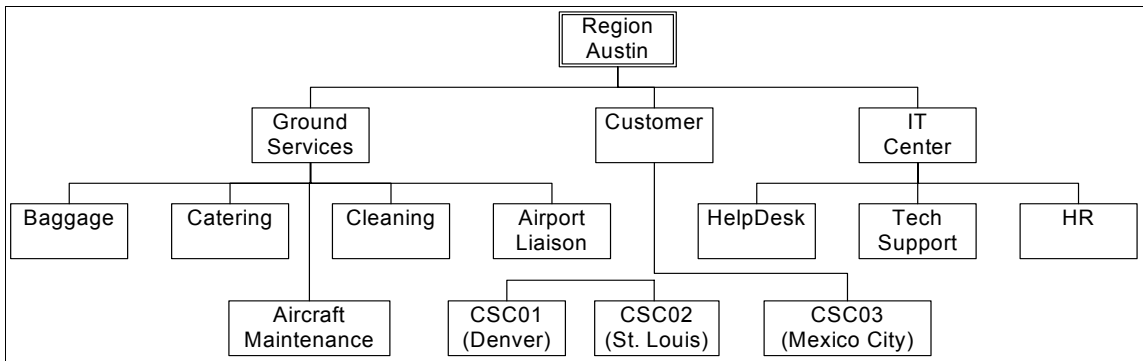


Figure 3-4 Central region organization chart

The core services division acts on a company wide scale. It is split into three departments, Sales, Support, and Flights. Each of these departments has a number of teams, as shown in Figure 3-5 on page 101.

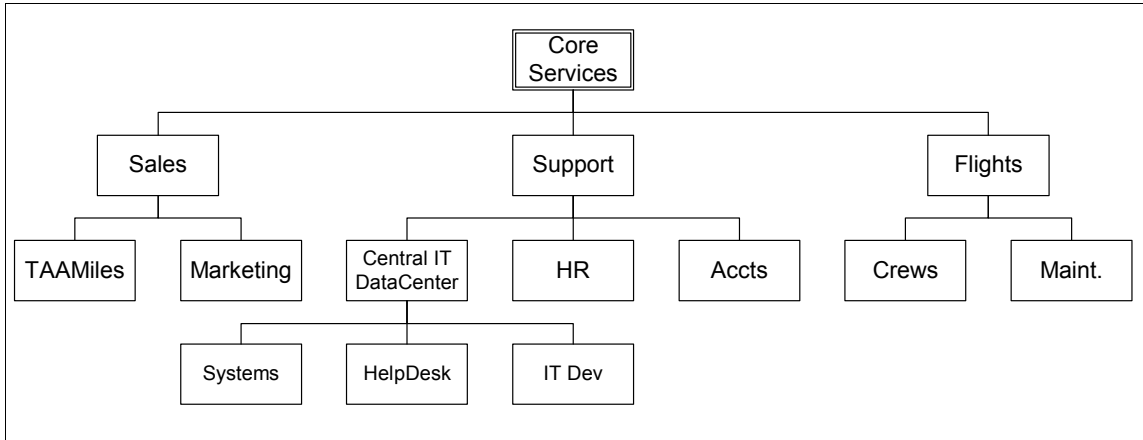


Figure 3-5 Core services organization chart

Each team within a department has a unique business code identifying the team and its location.

3.1.3 HR and personnel procedures

Personnel are managed by HR utilizing the centralized HRMS in the Austin HR site and IBM Tivoli Identity Manager. The following procedures apply to personnel management:

- ▶ When a new employee joins the company, the new employee is added to the HR system with the corresponding location and employee codes matching the new employee's job. An HR feed then updates Identity Manager with the new user. The new user is added to the appropriate role in Identity Manager according to the new employee's location and employee codes. This role is associated with those applications which that user requires to perform that user's duties at TAA. Identity Manager then automatically requests the required accesses to those applications, network, and the operating system which that role has been associated with. E-mails are sent (using Lotus® Notes®) to the required approvers and all other accounts are created immediately. Upon approval, those accounts which require approvals are also created. After all required accounts are created, an e-mail is sent to the new employee's manager indicating when the person is starting work along with the new employee's e-mail account information in order for that user to log in and collect the new employee's account information. When an employee needs additional access to resources, the employee asks the employee's manager to ask the appropriate support team, via e-mail, for the access. As with new accounts, the support teams grant the additional access. This currently takes up to two days to complete due to an administrative backlog.

- ▶ When an employee forgets the assigned password or has an account locked due to invalid passwords, the employee logs into Identity Manager via the forgotten password functionality and resets the password. This resets all of the employee's passwords to the new password. Further, when required to change a password, the employee uses the standard password change functionality from the Windows desktop environment. This password change is caught by Identity Manager and is then synchronized across all of the appropriate applications.
- ▶ When an employee leave the company, the employee is marked with the appropriate inactivity code within the HR system and the next HR feed to Identity Manager modifies the employee status within Identity Manager. This modification of the person status causes the accounts to be suspended, however, the employee is not deleted from Identity Manager according to security procedures since the employee identity is never to be used again in the future and the identity should still be able to be referenced. Furthermore, any information that the user may have worked with is required to be available for at least three months, at which time the user's accounts are deprovisioned and any access to the user's work must be restored from backup.

3.2 Current IT architecture

In this section, we describe the current IT environment at TAA. We cover:

- ▶ An overview of the TAA network
- ▶ The recently implemented e-business application
- ▶ The security infrastructure deployed for the e-business application
- ▶ The secured e-business initiative architecture
- ▶ User administration issues

3.2.1 Overview of the TAA network

TAA's central IT data center has implemented a back-end datastore, which is based on DB2 running on z/OS®. They use an MQ Series infrastructure for asynchronous transactions between the central IT data center, the CSCs, and the regional data centers.

The high-level network diagrams of TAA's network are shown in Figure 3-6 on page 103.

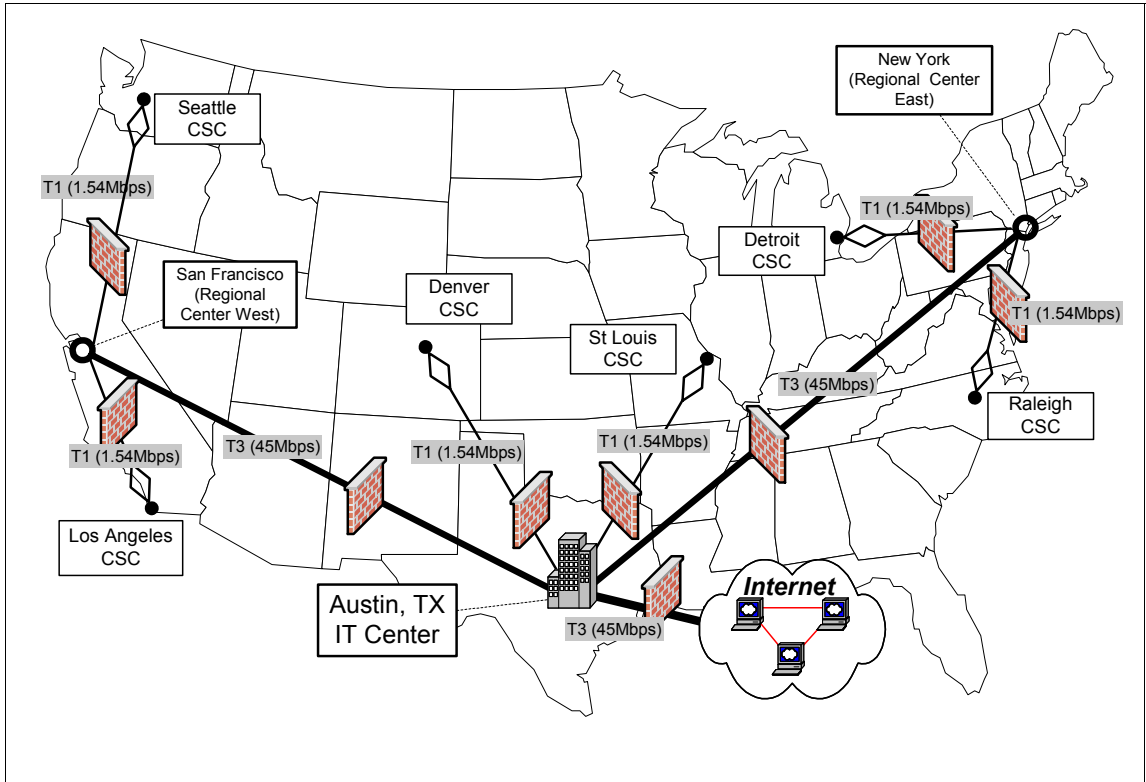


Figure 3-6 The TAA network

For the CSC in Mexico City, a high-level network diagram is shown in Figure 3-7 on page 104.

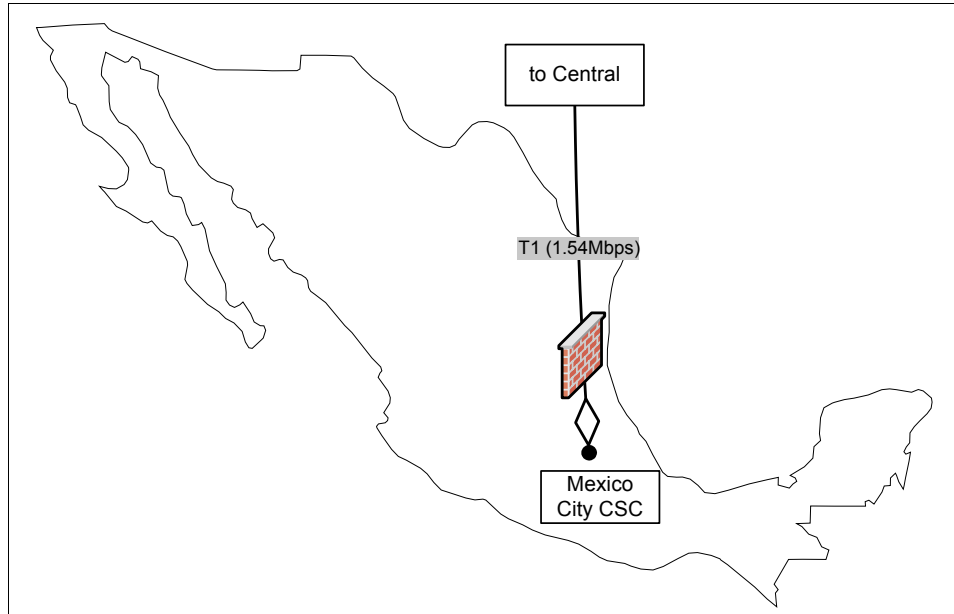


Figure 3-7 Mexico City CSC connectivity diagram

The location of firewalls is shown in Figure 3-6 on page 103. All external access to the TAA network is channeled through the firewalls and routers in Austin. There are also firewalls between the regional centers and the IT data center, as well as between the regional centers and the CSCs.

All T1 and T3 links are leased services. TAA relies on the service provider to ensure the necessary uptime, as agreed in the service level agreement. The diagrams in Figure 3-8 on page 107 and Figure 3-9 on page 108 are therefore logical and do not show redundant and standby links or triangulation of the network. TAA relies on the service provider for this.

TAA uses Lotus Notes for their e-mail system. This application is not available in the CSC and at the Gate terminals.

3.2.2 TAA's e-business initiative

Most of the business applications have been migrated to a distributed WebSphere Application Server implementation based on Linux systems, which are located in every regional center. All these systems communicate with the back-end database through the high-speed network.

The only application that has not been implemented using the WebSphere model is the *Gate Terminal Application*. This application runs on a Windows Active

Directory-based network on Windows terminals in each CSC (that is, the CSC employees cannot use a browser to access this application). The Gate Terminal Application uses MQ Series calls to check the appropriate passenger data.

3.2.3 Security infrastructure for the e-business initiative

In the past, TAA's application system experienced many unauthorized access attempts to critical business data. Recently, TAA has deployed a security solution that implements a centralized access control mechanism enforcing authentication and authorization of users before they actually access the applications and critical data via their Web browser. This solution is implemented based on IBM Tivoli Access Manager for e-business with WebSEAL for the access control component.

Note: In this redbook, we omit any detailed description about the IBM Tivoli Access Manager and WebSEAL solution, because our focus is on the identity management system. For further details, you might want to consult the following IBM Redbooks:

- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *Enterprise Business Portals with IBM Tivoli Access Manager*, SG24-6556
- ▶ *Enterprise Business Portals II with IBM Tivoli Access Manager*, SG24-6885

A typical user access with WebSEAL controls looks like this:

1. A user in a CSC logs on to the Windows domain specifying a Windows user ID and password.
2. The user starts a Web browser and accesses a login page for a specific application. The user logs in with the application user ID and password. A credential is used for access control by WebSEAL in the regional center the CSC belongs to.
3. WebSEAL accepts or denies the login. WebSEAL works as a reverse proxy between the user's Web browser and the application hosting Web server, controlling whether a user can access the requested resource or not.
4. WebSEAL's access control decisions are based on the information held within the Access Manager Policy Server and the relevant LDAP repository. The Policy Server stores the access control information used by WebSEAL and distributes access control information database replicas to all defined WebSEAL servers while the LDAP server has the user credential information created and used by Access Manager. The Policy Server is located in the Austin site, but WebSEAL and LDAP replica servers are made available in

each regional center. The LDAP server in Austin is the master server, which can be modified; the LDAP replica servers are read-only.

Only the Web applications can be secured by WebSEAL using Web user accounts, but there are other types of accounts necessary to run standard operations, such as Windows, Linux, and z/OS. These accounts can only rely on the native operating system security. That is why TAA puts the employees under an obligation to follow additional security policies to strengthen the levels of security, such as a periodic password change and other password policies for all types of accounts.

3.2.4 Secured e-business initiative architecture

Figure 3-8 on page 107 contains only the Austin site, which consists of the central IT data center, Regional Center Austin, and CSC, in order to make it simple. While there are strong grounds for altering the network topology and firewall configuration so that the Austin Regional Site is separate from the Austin Corporate site, the risk assessment carried out once again showed that there were higher priorities (those addressed by Access Control and Identity Management) than this internal network topology change. The full existing TAA topology is shown in Figure 3-9 on page 108.

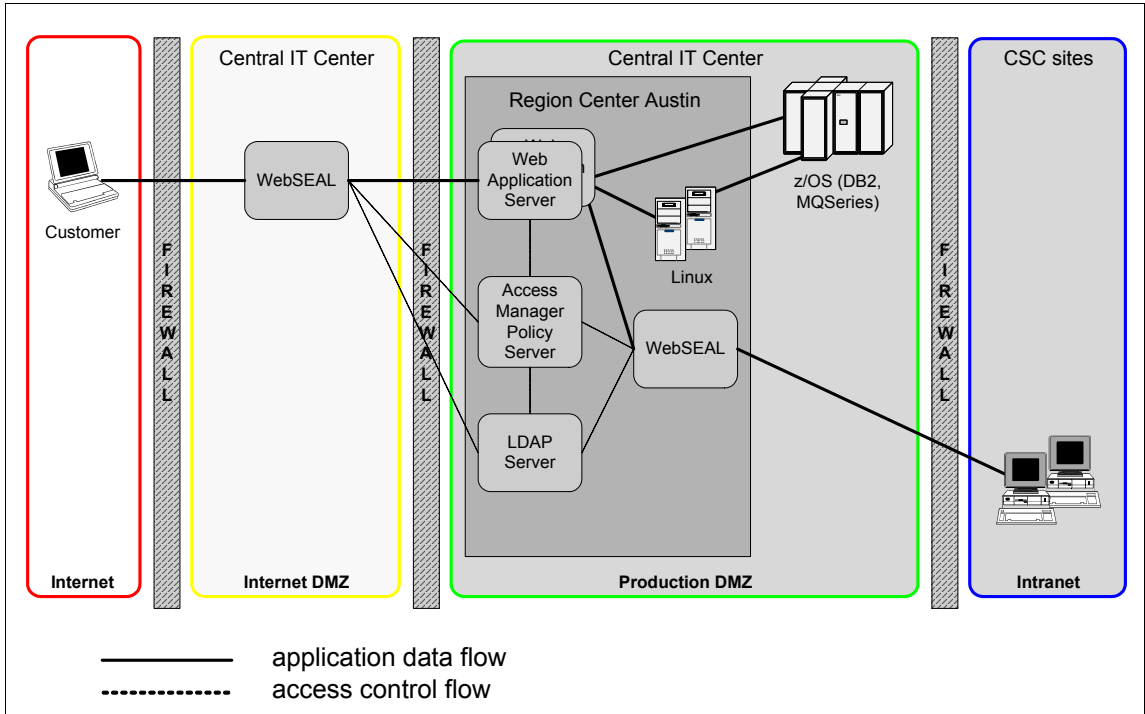


Figure 3-8 Current IT architecture in Austin

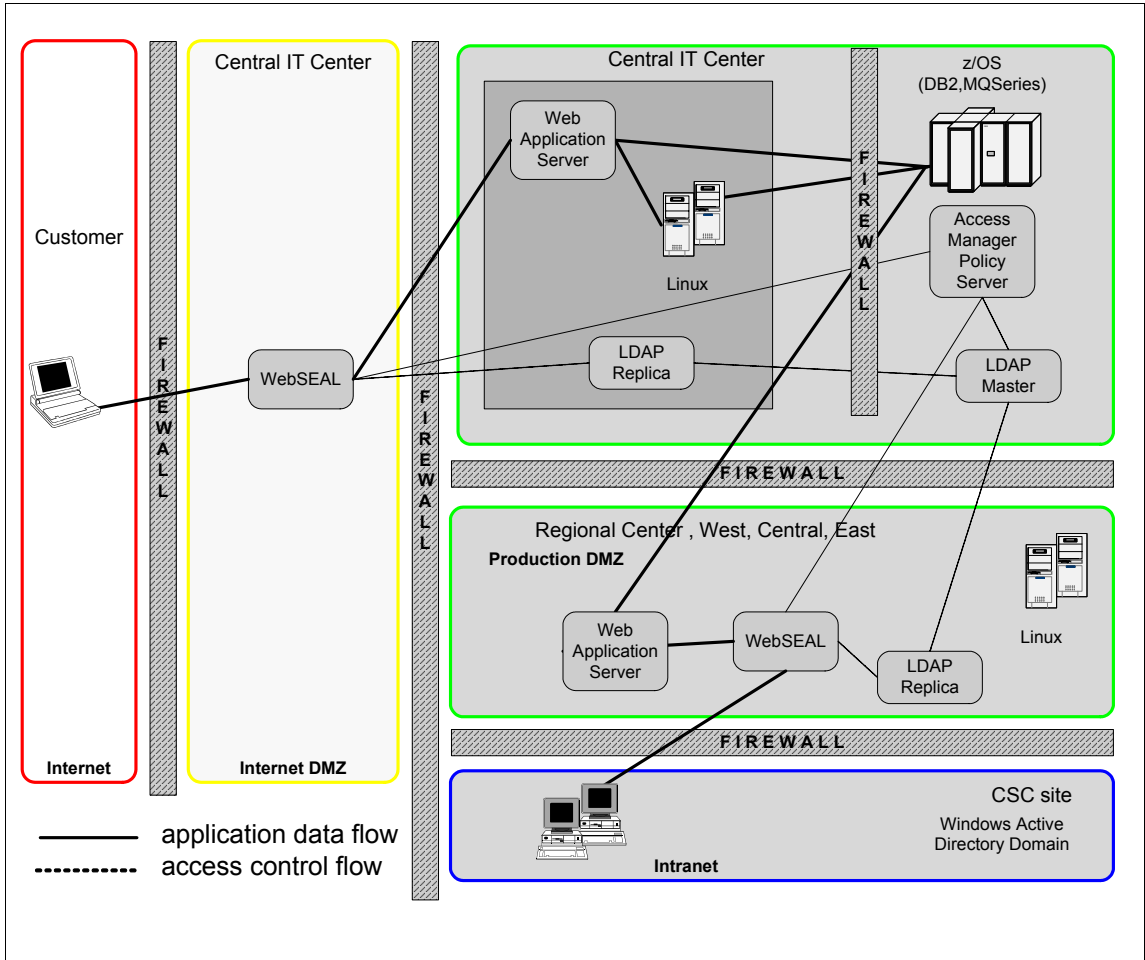


Figure 3-9 Current entire TAA architecture

Ultimately, TAA will aim at segregating the data (DB2 on zOS) and systems management zones (LDAP, Security management, and so on) from the Austin Corporate Network. The Austin Regional Network could then further be separated by a firewall and be treated exactly as though it were a remote regional center, even though it is not remote from Austin. Evolving toward this security best practice also creates further operating gains on the system management side of the organization and, therefore, in the user experience.

3.2.5 Identity management and emerging issues

TAA is trying to closer integrate business partners and customers into their current infrastructure in order to increase efficiency; however, this has raised infrastructure, identity, and access management issues. Before we discuss these issues in detail, we give you an overview of the current user management.

Current user management

TAA has been using IBM Tivoli Identity Manager in production for a little over a year now with its internal employees.

Note: You can read everything about the first Identity Manager installment at TAA in the IBM Redbook *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996.

Users request new all accounts through either their manager or second-line manager, who then requests the necessary access from the Identity Manager administrators. The CSC staff have their accounts in a Windows domain and the Tivoli Access Manager LDAP directory (for WebSEAL-controlled Web application access). The regional center administrators have their accounts on Linux systems, adding to Windows domains in their region and the Tivoli Access Manager LDAP directory. Currently, the provisioning time for the Windows accounts can take up to a day due to the low bandwidth between certain sites and the time for Windows Active Directory (AD) to replicate provisioning changes.

Programmers and developers in the central IT data center have their accounts on all platforms in the enterprise, including z/OS. Accounts on z/OS are created using the RACF® interface shipped with z/OS.

Customers' accounts are created in the Tivoli Access Manager LDAP registry located in Austin. They are managed by administrators in the regional center Austin.

Business partners do not have accounts and, therefore, must go through their management counterparts in TAA to access certain information that is critical for them to supply the necessary parts and services required of them. This builds extra time and cost into the process that would otherwise not be there.

Emerging issues

As we have mentioned, all employee accounts are created and maintained with Identity Manager. All customer accounts are created and maintained by administrators, and business partners do not have accounts; however,

customers and business partners require certain resources that are in the TAA IT infrastructure.

Too many requests cause regional administrators and managers not to reply immediately to customer and business partner requests, therefore, lengthening services and part requests as well as reducing customer satisfaction.

Currently, each new access request for resources not managed by Identity Manager has to be initiated by a manager. And even if the resource is being handled by Identity Manager, the approval has to be processed outside of Identity Manager whatsoever; it can take days to complete causing non-productive time for the employee.

A similar situation prevails in the administration of customer and business partner information: requests for creating their accounts, password reset requests, and so on. Some customers and business partners have already contacted IT management, because it takes too much time and effort to manage their account information.

While the introduction of Identity Manager has greatly reduced this time and backlog, this is an area that still needs streamlining especially considering TAA is looking to expand its IT offerings.

TAA has now decided to implement an additional security management solution focusing on user and identity management. The main objective in this case study scenario is to use the Tivoli Identity Manager self-care solution with high availability.

While TAA has implemented procedures and processes to mitigate the needs of the Sarbanes-Oxley Act of 2002, TAA has seen record increases in profit margin in the last year and is under more intense scrutiny from its shareholders as well as governmental agencies. Because of this, TAA is now looking to form a standing audit and compliance team that can field all requests from shareholders, board members, and auditors. With this action, TAA realizes that it must change how it currently reports and tracks information in order to make the team's mission successful.

3.3 Corporate business vision and objectives

TAA has implemented their e-business application system to employees and expanded their services on the Web to their customers. This system relies on a Web-based application infrastructure provided by the IBM WebSphere Application Server and a centralized access control solution using IBM Tivoli Access Manager for e-business.

In order to increase TAA's productivity and decrease costs, the user management processes for all the involved platforms have to be streamlined, and access to these processes have to be made more widely available as well as self-controlled.

The TAA vision is as follows:

- ▶ TAA has deployed a corporate wide user and identity management system, which has successfully operated according to its original requirements since going live in production. However, TAA is looking to leverage new IT offerings to its employees, customers, and business partners in order to increase efficiency of its operations but at the same time decrease overall expenditures.
- ▶ Given TAA's vision of expanding its IT offering to its customers, business partners, and employees, TAA sees that it must make its IT infrastructure as robust as possible so as to make these capabilities attractive and accessible for all to use in order for it to be successful.
- ▶ TAA has recently come under more scrutiny from its board members and financial watchdog agencies as it has become more profitable. To satiate these new inquiries and audits, TAA is looking to form an internal audit and compliance team that can prepare the necessary documents.

3.4 Project layout and implementation phases

Based on the corporate business vision, TAA has decided to implement the new solution in two phases:

- ▶ The first phase, Phase I, focuses on the tasks necessary to make the IT infrastructure robust enough to support the planned extra provisioning, access, and service usage that TAA is looking to offer. Specifically, these tasks include making the current Tivoli Identity Manager system highly available via clustering and redundancy. The different subsystems of Tivoli Identity Manager will be made highly available (HA) by use of proven best practices and vendor recommended approaches. Refer to Chapter 5, "Technical implementation phase I" on page 121 for details.
- ▶ Phase II focuses on the extensions to the current identity management system with regards to the new offerings which TAA is looking to make available to its employees and business partners. Specifically, this phase encompasses the creation and installation of a new self-care Web application and the creation of an extranet that makes this application available to its business partners. Furthermore, this phase encompasses the configuration of a delegated administration capability necessary for TAA's business partners

to administer their own users. Refer to Chapter 6, “Technical implementation phase II” on page 209 for details.



Project design

In this chapter, we describe the business requirements, functional requirements, security design objectives, and design aspects for an identity management self-care application based on Tivoli Identity Manager.

Most implementations are done in multiple phases to mitigate risk but also to gain return on investment as rapidly as possible. TAA has decided to use a multi-phased approach as well, first to create a needed foundation, and then to build on this foundation in the second phase. The content of each phase is decided by analyzing the priorities of the business requirements and mapping these through their functional requirements to Identity Manager capabilities. The earlier phases are dedicated to satisfying those requirements associated with high-priority business requirements and required capabilities for the second phase.

Implementation details for each of the phases are in 3.4, “Project layout and implementation phases” on page 111.

4.1 Business requirements

Based on the business visions introduced in 3.3, “Corporate business vision and objectives” on page 110, we need to take a closer look at the individual business requirements.

These can be lined out as follows:

- ▶ Introduce an extranet to business partners and customers

Tivoli Austin Airlines (TAA) would like to implement a solution suggested by a recently completed study, which found that TAA could expedite services to its airplanes and reduce costs if it created an extranet where both TAA and its business partners could share information.

- ▶ Reduce help desk overhead and costs

TAA is keen to gain cost savings by reducing the IT help desk costs which have been increasing over time; however, TAA is also extremely aware of the need for this function, especially when you consider TAA's plans to implement an extranet. Therefore, the reduction in cost cannot result in the reduction of capability.

- ▶ Reduce license and maintenance fees for deployed applications

An analysis commissioned by TAA has shown evidence of a high total cost of ownership for software. The analysis noted that if TAA provided metrics for the usage of its software licenses, TAA could potentially reduce its overall license cost by reducing the number of seats it was currently paying for.

- ▶ Improve audit and compliance reporting

TAA recently failed certain areas of a security audit that showed exposures in internal security compliance rules. A number of areas were seen to be lacking:

- There is no periodic certification of users' access rights.
- The reporting available is insufficient to verify security compliance.

TAA is extremely motivated to rectify these issues.

- ▶ Reduce provisioning time

TAA has been made aware from its IT organization that there are certain situations that require its managers to be able to delegate certain responsibilities to their subordinates in an expeditious manner in order for the subordinates to conduct business on the managers' behalf. TAA would like to make this happen since it was shown that having this type of capability gave their management more flexibility and, therefore, was a good return on investment.

4.2 Functional requirements

We extract functional requirements by mapping business requirements to their underlying reasons. We expand the reasons in increasing detail until we find problems that can be solved using capabilities of Identity Manager. Our

functional requirements will tie these low-level reasons for a business requirement to the Identity Manager capability that fulfills that business requirement.

Let us examine the business requirements and find the functional requirements for each.

- ▶ Business requirement 1: Introduce an extranet to TAA business partners and customers.

Tivoli Austin Airlines is aware that in order to increase efficiency and productivity, it must integrate its internal processes with its business partners more closely. Furthermore, TAA has come to realize that while customer satisfaction increased since the installation and rollout of the current identity management solution, there are still issues that need to be addressed. This can be written as the functional requirements described in Table 4-1.

Table 4-1 Functional requirements for extranet

Requirement	Description
A	Identity Manager installation to be highly available.
B	Provide self-care application to empower employees, customers, and business partners to administer their own account needs.
C	Provide self-registration capabilities for customers and business partners.

- ▶ Business requirement 2: Introduce a recertification process to gain metrics about license usage.

TAA conducted an internal audit that found that its software licensing and maintenance had increased substantially over the last several fiscal years. This fact is an opportunity that could potentially provide savings for the company. However, in order to capitalize on this potential savings, the software license usage needs to be measured to identify whether any licenses could be trimmed, and therefore, savings realized. To do this, TAA wants to implement a recertification process for all software currently in use to identify those licenses that could be cut along with the associated maintenance.

See Table 4-2 on page 116.

Table 4-2 Functional requirements for software license and maintenance cost reduction

Requirement	Description
D	All TAA employees and business partners that utilize TAA applications must have an e-mail account or access to the self-care application.
E	Create an Identity Manager recertification process for all TAA applications.
F	Make recertification process available company-wide and extranet-wide.

- Business requirement 3: Enhance audit and reporting capabilities.

TAA wants to enhance its audit and compliance solution with the formation of a specialized team to aggregate audit and compliance information in such a way that efficiently provides more detailed and focused reporting. To facilitate this, TAA wants to implement an audit and compliance reporting process that provides a streamlined solution.

Table 4-3 Functional requirements for enhanced audit and reporting capabilities

Requirement	Description
G	Automate audit and compliance report generation.
H	Automate audit and compliance report delivery.
I	Automatically update audit and compliance team with recertification information.

- Business requirement 4: Reduce application administration cost.

TAA has been made aware from the internal audit that was conducted that a cost savings could be realized from its application administration. TAA would like to implement a solution to make its application administration more efficient and productive and, therefore, reduce its overhead.

Table 4-4 Functional requirements for application administration cost reduction

Requirement	Description
J	Decrease the time required to debug application issues.
K	Mitigate error-prone administrative functions.
L	Reduce repetitive administrative duties.
M	Delegate administration of business partner users to business partner administrators.

4.3 Design approach

In the design approach section, we consider how identity management design objectives can be realized using Identity Manager. Our goal is to produce a plan containing a phased set of implementation steps where the end result satisfies the functional requirements and, therefore, also satisfies the original business requirements.

While business and functional requirements are the main parts of the identity management objectives, we also have to consider other non-functional requirements and constraints. These might include objectives that are necessary to meet general business requirements or practical constraints on constructing security subsystems. Identity Manager implementations often involve non-functional requirements relating to:

- ▶ High availability
- ▶ Backup and recovery
- ▶ Performance and capacity
- ▶ Change management
- ▶ Training
- ▶ Existing infrastructure
- ▶ Budget and staffing

Because we are focused on advanced designs of identity management with Identity Manager software in this book, we do not look at all of these non-functional requirements in detail.

The steps involved in producing an implementation plan are:

1. Prioritize the requirements.
2. Map the requirements to Identity Manager features.
3. Define the tasks involved in using those features to satisfy the requirements, and estimate the effort required for each task.
4. Divide the tasks into phases.

Prioritizing the requirements is important because the priorities are one of the primary factors used to decide which implementation tasks will be done in which phase of the project. It is rare that an identity management solution can be created as a single deliverable satisfying every requirement. It is far more likely that it will be delivered in phases, and the highest priority requirements should be addressed in the earliest phases.

Assigning priorities to the requirements is often difficult because “They are all important.” You can more easily compare the priorities of requirements by asking questions that gauge the positive and negative impacts of the requirements:

- ▶ How much money will be saved when the requirement is met?
- ▶ Are there penalties if the requirement is not met?
- ▶ Is there a date by which the requirement must be met?
- ▶ Are there other requirements with dependencies on this one?
- ▶ If this requirement is not met, is the company any worse off than they are now?

After mapping the requirements to Identity Manager features and creating a list of implementation tasks, the requirement priorities and the effort of each task can be used to decide how to break up the project into phases. The goal of breaking the project into phases is to quickly deliver solutions to some high-priority requirements. This allows the company to begin seeing a return on their investment, while lower priority and more difficult tasks are still being executed.

4.4 Implementation approach

This section applies the design approach described in 4.3, “Design approach” on page 117 to TAA’s specific requirements.

4.4.1 Non-functional requirements

The non-functional design objectives are those that do not relate specifically to the functional requirements but are items that should be addressed in the design. For TAA’s project, these include:

- ▶ Reuse of the existing identity management infrastructure
- ▶ Standards to be used
- ▶ Maintainability and configuration management
- ▶ High availability and disaster recovery

Reuse of the existing infrastructure

The design must allow for the reuse of the existing identity management design, except where it conflicts with the new requirements. The identity management solution must therefore be deployed into the existing architecture in a way that allows the accommodation of network and application changes with the least possible interruption to current identity manager services.

Standards to be used

Where possible, the design must comply with standards in order to make subsequent implementation easier, more secure, and audit compliant. You can find further details about policies and standards in Appendix A, “Corporate policy and standards” on page 313.

Maintainability and configuration management

The design must allow for the maintainability of the system. This may involve deployment of some form of configuration management methodology or system management tool set.

High availability and disaster recovery

The design requires high availability for the delivery of the project; therefore, the design needs focus on this as a priority.

4.4.2 Requirement priorities

TAA has analyzed their business requirements and has made cost savings and productivity their highest priorities; however, in order to realize these priorities, their dependencies must be implemented first.

4.4.3 Implementation tasks and efforts

The details of the implementation tasks are not described here. They are described in detail in the technical implementation chapters of this book.

4.4.4 Project phases

Based on the priorities of their business requirements and the levels of effort of the different implementation tasks, TAA has decided to split the project into two phases, as follows.

Phase 1: High availability (HA)

The goal of this phase is to complete all of the work necessary to create a highly available Identity Manager installation. At the completion of this phase, all of the Identity Manager components will be operational, and Identity Manager will have an availability of 24x7x364, with 24 hours of service time broken up throughout a given year. This phase is a prerequisite to the work done in the following phases. Tasks in this phase include:

- ▶ Installation of required middleware components for redundancy
- ▶ Configuration of components for HA functionality
- ▶ Security hardening of the components

Phase 2: Extranet

The goal of this phase is to implement an extranet to deploy a self-care application to both TAA's business partners and customers as well as its employees. Phase 2 involves implementation tasks that address high-priority requirements. However, one of the factors governing how quickly a feature can be implemented and placed into production use is the number of people who must be trained to use the new feature.

The features included in this phase are:

- ▶ Enhanced audit and reporting capabilities
- ▶ Enhanced troubleshooting capabilities
- ▶ Recertification capabilities
- ▶ Self-care application for employees, business partners, and customers

The self-care application will enable users, internal employees as well as business partners, and certain customers to access and manipulate specific information concerning their accounts.

- Users will be able to request the creation, modification, and deletion of accounts that they require to do their job.
- Account creation and modification may require approval by a member of an application administration team for the account's service.
- ▶ Delegated administration for business partners and customers

The goal of this task is to delegate Identity Manager management activities where possible to system administrators designated by business partners and customers for their employees, but also to the employees themselves. This will require the preparation of documentation and training for the delegated administrators. The implementation of these features will also require more extensive requirement gathering than the previous phases.

- Identity Manager maintains centralized control and an audit trail regardless of whether the account management is done by TAA system administrators or by delegated administrators.
- User management includes members of business partner and customer user administration teams as well as those of the TAA administration teams in order to administer users of each of the customer and business partner organizations.



Technical implementation phase I

This chapter describes the tasks, considerations, and implementation details of the first phase of the IBM Tivoli Identity Manager deployment at Tivoli Austin Airlines (TAA).

5.1 TAA's high availability scenario

This section discusses the requirements, the design considerations, and the implementation of the Identity Manager high availability scenario implemented by TAA.

5.1.1 Requirements

TAA requires a 24x7x365 uptime for its identity management services accessed by internal employees, business partners, and external users.

The continuous availability of the services has become a main business requirement that the company must consider on the design and technical implementation of the Tivoli Identity Manager solution.

Continuous availability can be separated into two categories:

- ▶ High availability

This means that a specific resource is available for use at all times, implying that if it becomes unavailable on a particular machine, another one starts to provide access to this resource. The business impact of these outages specifically refers to the desired improvements in terms of high availability.

- ▶ Continuous operation

This signifies the ability to avoid planned and unplanned outages. This includes the ability to upgrade and maintain processors, operating systems, business software, and the applications, where the business does not stop, even in a failure event. This is accomplished by adding duplicate resources to the environment.

Analyzing the business need to extend service hours produces a specification for the necessary degree of continuous operation. This is defined in the service level agreements (SLAs).

An unplanned outage of computer systems results in several business processes coming to a standstill. From a business perspective, the financial impact of a unplanned outage of the identity management system can occur in several ways:

- ▶ Lost productivity time of the users

Users who need the systems for their work might be idle for as long as the systems are down. For instance, user accounts will not be created in a timely manner for a newly hired employee.

- ▶ Loss of transactions

If the outage occurs during a password synchronization process, the unique password for all the target systems managed by Tivoli Identity Manager could be out-of-sync.

- ▶ Loss of compliance with security policies

If the outage occurs when an employee is changing a business role or is leaving the company, accounts could be still available on the target systems that should have been deleted or they could provide wrong levels of access.

- ▶ Extra cost and time for support service recovery

The work that was interrupted through an outage needs to be repeated at a later time. Repetition of workload requires extra system resources and extra time and cost for the support services.

In large computer networks, the sum of these cost items can amount to a substantial figure. After a business impact analysis, TAA, Inc. decided to

implement a high availability software solution for their identity management environment.

5.1.2 TAA's high availability planning

This section describes the high availability topology for TAA's identity management system. In addition, this section provides information regarding the software we used to implement the identity management environment.

TAA's Identity Manager high availability environment consists of the following five nodes (see Figure 5-1):

- ▶ Two Identity Manager nodes
- ▶ Directory Server Proxy Server and Deployment Manager node
- ▶ Two Identity Manager LDAP Directory Server and Identity Manager datastore nodes

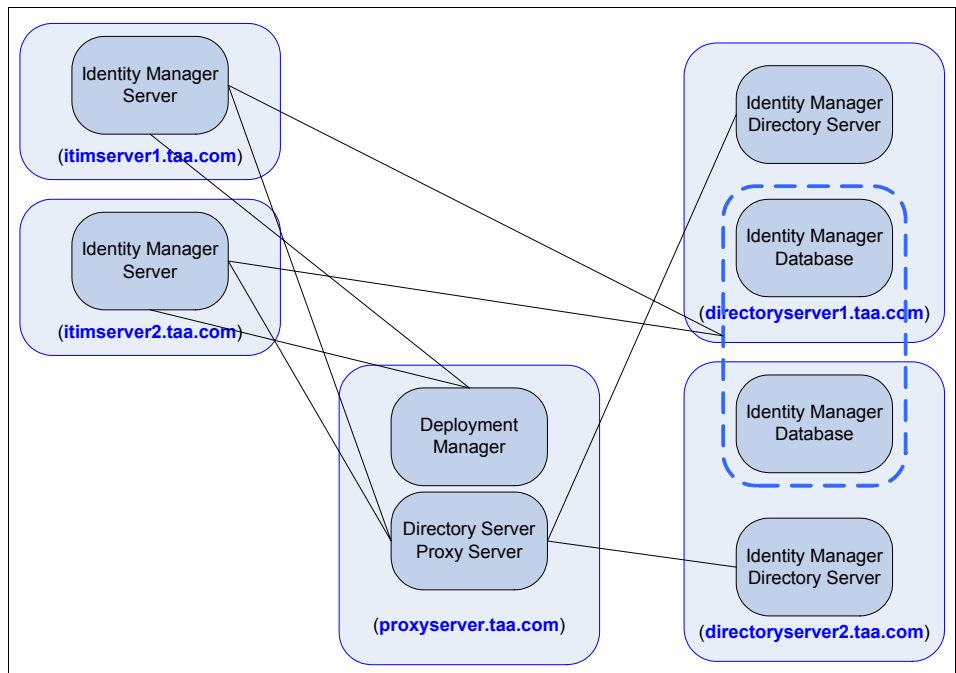


Figure 5-1 TAA high availability identity management environment

Note: For detailed and official product information about the software requirements, we recommend that you refer to the product planning and installation guides.

Software used within the TAA high availability environment

The TAA identity management system was implemented using the following software listed by node.

Tivoli Identity Manager Server nodes

TAA implemented the following:

- ▶ IBM WebSphere Application Server, Version 5.1.1.4
- ▶ IBM Tivoli Identity Manager, Version 4.6 + FP08
- ▶ IBM DB2 UDB client, Version 8.1.9

These software components were installed on the machines: itimserver1.taa.com and itimserver2.taa.com.

Proxy Server and Deployment Manager node

TAA implemented the following:

- ▶ IBM WebSphere Application Server Network Deployment, Version 5.1.1.4
- ▶ IBM Tivoli Directory Server, Version 6.1, Proxy Server component only
- ▶ IBM DB2 UDB client, Version 8.1.9

These software components were installed on the machine proxyserver.taa.com.

Since TAA's implementation does not encompass a very large scale LDAP population there are no restrictions in using the proxy server.

Note: Even though you install the proxy server as a package, which is part of the IBM Directory Server 6.1 media, it requires an independent license and maintenance entitlement. Check with your IBM Sales Representative for more details about how best to leverage the proxy server component.

Directory Server and database nodes

TAA implemented the following:

- ▶ IBM DB2 UDB, Enterprise Server Edition, Version 8.1.9
- ▶ IBM Tivoli Directory Server, Version 6.1

These software components were installed on the machines: directoryserver1.taa.com and directoryserver2.taa.com.

5.2 Application server high availability

This section discusses the requirements, the design considerations, and the implementation of the Identity Manager application server high availability scenario that was implemented by TAA.

5.2.1 Requirements

The continuous availability of the Identity Manager services is the primary business requirement that TAA needed to respond to service levels agreed to with the business partners.

5.2.2 Design considerations

The unique issue to consider when designing a high availability solution for the application server is the WebSphere cluster topology supported by the Identity Manager application: the horizontal cluster configuration is the only one supported by Identity Manager and the only one that can provide continuous operation.

In a configuration such as shown in Figure 5-2, each computer shape represents one WebSphere Application Server node on one computer. The configuration specifies the deployment manager on one computer but it could be installed on the same machine as the first WebSphere Application Server node. The remaining application servers are configured on additional computers.

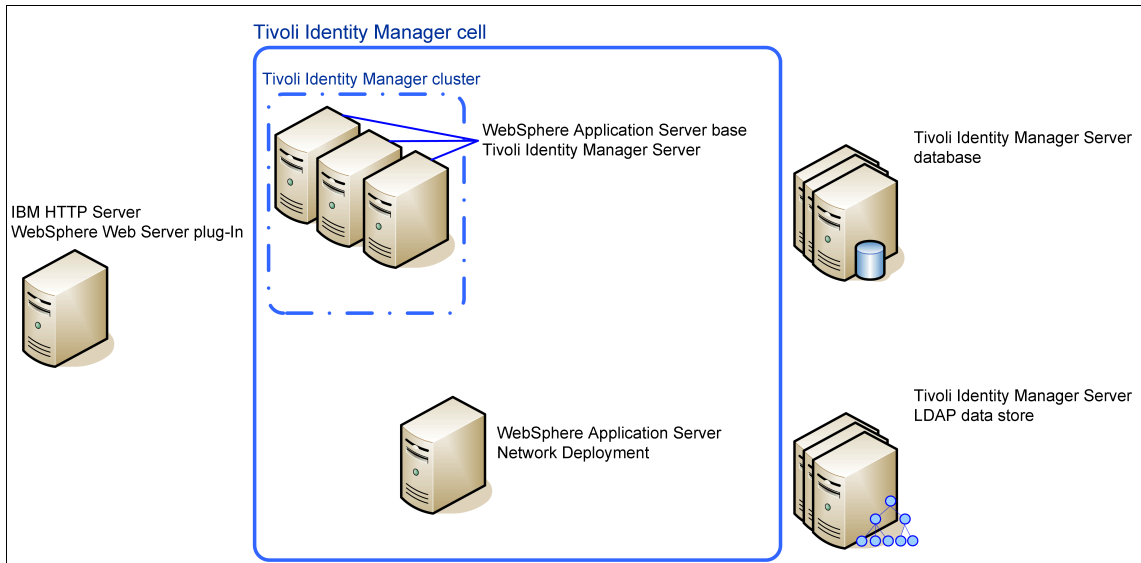


Figure 5-2 Cluster configuration on multiple computers

Note: Tivoli Identity Manager does not support:

- ▶ Vertical cluster configuration that has more than one cluster member within a WebSphere Application Server node.
- ▶ Functional cluster configuration that separates workflow processing and user interface processing on separate machines.

5.2.3 Application server high availability implementation

The following section shows all steps necessary to configure the Tivoli Identity Manager Server on a horizontal WebSphere Application Server cluster.

We are not going to describe the general steps required to implement a WebSphere cluster topology. There are many guides and Web sites that contain such information. The following Redbooks may be of interest:

WebSphere Application Server cluster solution with the deployment manager component and only one cluster member. These two components can be installed on the same machine. You only have to ensure that the computer has the required memory, speed, and available space to meet the additional load. Starting from this base configuration it will be an easy task to add a new cluster member to implement an Identity Manager cluster at application server level in the future if needed.

If you already have an installed Tivoli Identity Manager environment on a single server configuration, you have to reinstall the software in order to migrate to a Tivoli Identity Manager cluster configuration. There is no supported way to migrate a single application server install onto a cluster; but all the important information about the LDAP directory server and the Tivoli Identity Manager database will be maintained.

In the following description we consider a WebSphere cluster with two cluster members installed on two separate machines. This configuration is sufficient to avoid a single point of failure on the application server.

To configure an Identity Manager cluster server starting from an Identity Manager single server configuration, execute the following steps:

1. Back up the files in the ITIM_HOME/data directory. After the Identity Manager installation on the cluster server replace the custom properties files on the new Identity Manager installation with the one you have backed up.
2. Configure a WebSphere Application Server cluster with a unique cell and two cluster members. The WebSphere cluster topology used on the current implementation is shown on Figure 5-3 on page 128. For detailed information about WebSphere cluster installation and configuration, refer to Chapter 4, “Installing and configuring WebSphere Application Server” in the *IBM Tivoli Identity Manager: Server Installation and Configuration Guide for WebSphere Environments*, SC32-1750-00.

In the current implementation, the cluster topology is:

► Deployment manager

- WebSphere Application Server Network Deployment, Version 5.1.1.4

hostname	proxyserver.taa.com
Cluster name	ITIM_Cluster
Cluster nodes	itimserver, itimserver2

► Cluster members

- WebSphere Application Server, Version 5.1.1.4

- Cluster member A

hostname	itimserver1.taa.com
node	itimserver
cluster member name	serverp

- Cluster member B

hostname	itimserver2.taa.com
node	itimserver2
cluster member name	serverp

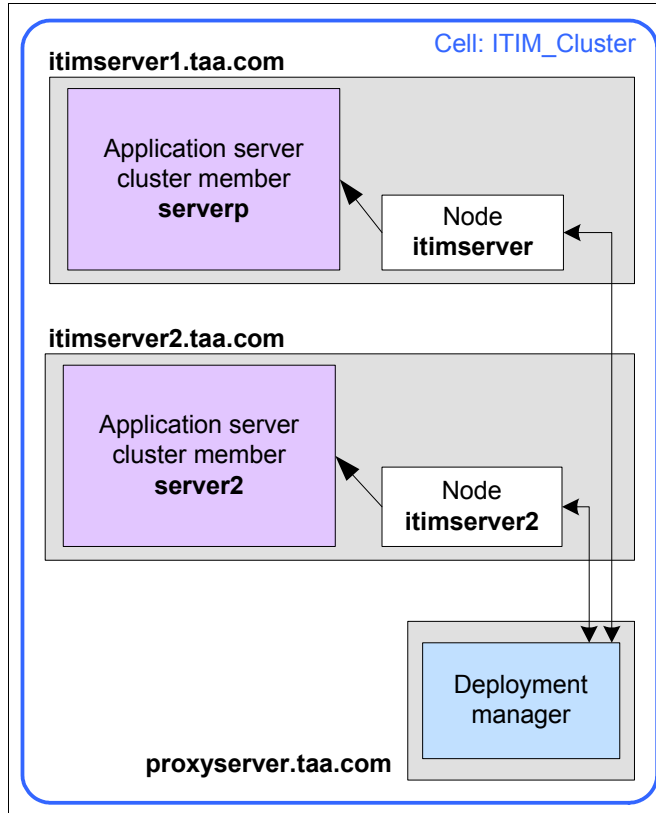


Figure 5-3 WebSphere Application Server cluster topology

The Figure 5-4 on page 129 shows a cluster view from the WebSphere Network Deployment administrative tool.

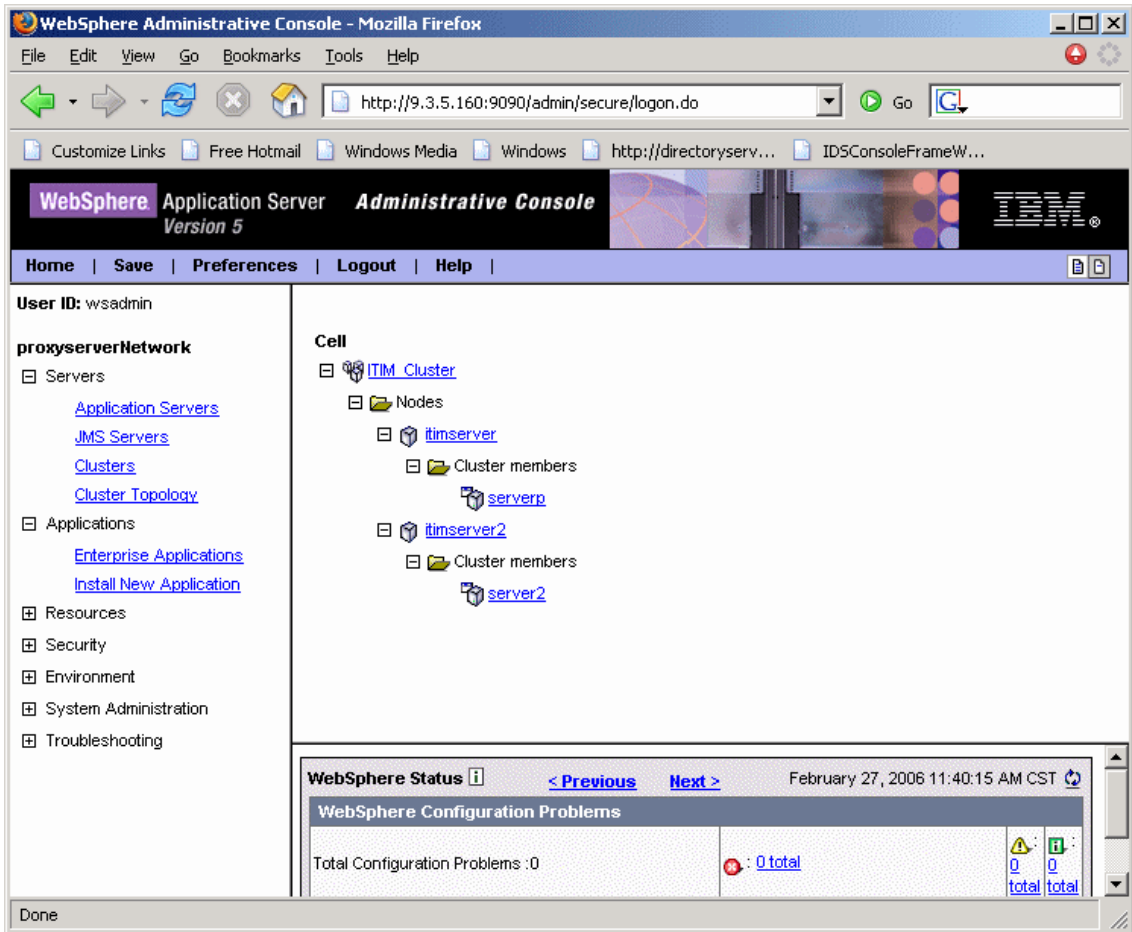


Figure 5-4 WebSphere Application Server Network deployment Administrative Console

3. Before we proceed with the next steps, we have to verify that the deployment manager and all the cluster members are started using the WebSphere Administrative Console. The cluster status displays as in Figure 5-4.

See Figure 5-5 on page 130.

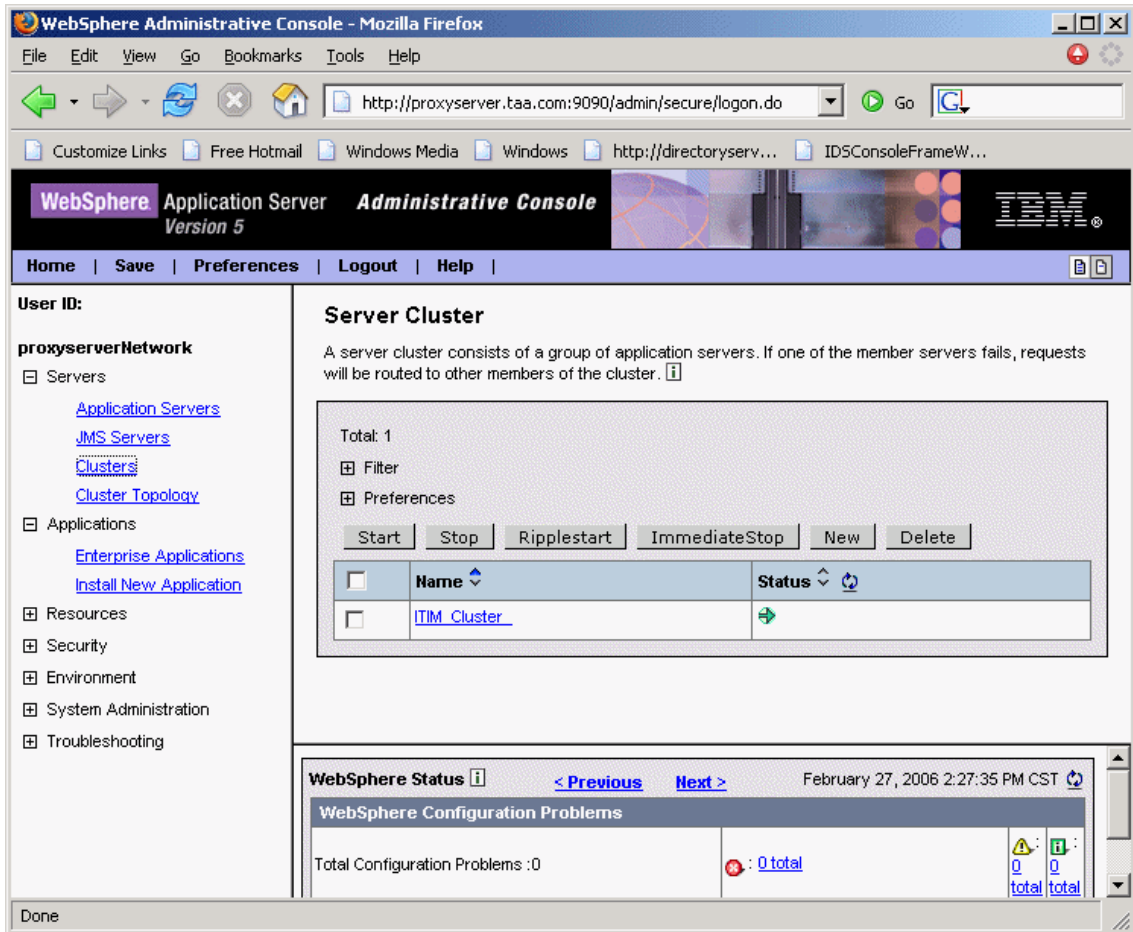


Figure 5-5 WebSphere Application Server cluster status

Install the Identity Manager in a cluster configuration. For detailed information about Tivoli Identity Manager cluster installation and configuration, refer to Chapter 6, “Installing Tivoli Identity Manager in a cluster configuration”, in the *IBM Tivoli Identity Manager: Server Installation and Configuration Guide for WebSphere Environments*, SC32-1750. To continue:

- a. Install Tivoli Identity Manager on the deployment manager using the installation wizard.
- b. Skip the steps to configure the Identity Manager directory server and database store.

Note: When the Identity Manager installation is complete on the deployment manager, use the **runConfig** command with the **install** option to configure Identity Manager to use the already configured LDAP directory and datastore.

- c. On the deployment manager machine, run the **runConfig** command with the **install** option to update identity Manager properties.

```
ITIM_HOME/bin/runConfig install
```

- d. On the system configuration tool, select the **Directory** tab and modify the following information for the directory server:

Principal DN and password

The principal DN and password that the Tivoli Identity Manager uses to log on to the directory server, for example, `cn=root`

Host name

Directory server host name of the already configured directory server. In our current implementation, `directoryserver1.taa.com`

Port

Port number for the directory server. In the current implementation, `389`.

- e. Click **Test** to verify the connection to the directory server.
- f. Select the **Database** tab and modify the following information for the Identity Manager datastore:

Database Type

In the current implementation, the database type is `DB2`.

Database Name or Alias

In this implementation, the database is already installed remotely and the value represents the local alias name of the remote database.

- g. Click **Test** to verify the connection to the database.
 - h. Click **OK**.
4. Restart the WebSphere deployment manager.

The Tivoli Identity Manager application is now installed on the WebSphere deployment manager and configured to use the previously installed directory server and database.

5. Ensure that the deployment manager and all WebSphere Application Server node agents are running. Complete the following steps using a command line interface:
 - a. To determine the status of the deployment manager, run the following command on the computer on which the deployment manager is installed:

```
WAS_HOME/bin/serverStatus.sh -all
```
 - b. To determine the status of the node agents and the JMS server, run the following command on the computer on which the WebSphere Application Server base product is installed:

```
WAS_HOME/bin/serverStatus.sh -all
```
6. Install Tivoli Identity Manager on each cluster member using the installation wizard.
7. Restart each WebSphere cluster member.

The Tivoli Identity Manager application is now installed on a cluster configuration as shown in Figure 5-3 on page 128.

5.3 Relational database high availability

This section discusses the requirements, design considerations, and implementation of the Identity Manager database high availability scenario implemented by TAA.

5.3.1 Requirements

The requirements for TAA's database server are basically similar to the common requirements described in 5.1.1, "Requirements" on page 121. In addition, TAA also wants to reduce costs for hardware and software licenses.

5.3.2 Design considerations

TAA is using DB2 as Identity Manager's relational database. The available high availability designs with DB2 are described in 2.3, "Relational database" on page 73. TAA has decided to use the HADR feature for their high availability scenario, because it allows the most rapid failover, software upgrades without interruption of service, and extensibility for cross-site replication with no additional software licenses and shared disks.

However, HADR does not automatically monitor the primary database server in case any outages might occur, so TAA has decided to use a custom shell script to issue appropriate takeover commands in the event of a primary database

server failure. There are several ways to monitor the status of the primary database:

- ▶ Monitor server machine failure.
- ▶ Monitor network failure.
- ▶ Monitor the *db2sysc* process on the primary server.
- ▶ Monitor the log shipping connection of HADR.

In TAA's case, we use a script that monitors the status of the log shipping connection and issues the takeover command at its disconnection.

5.3.3 Relational database high availability implementation

This section describes the necessary steps to set up the standby database for HADR with the existing primary database used by TAA's Identity Manager. In this section, we use the DB2 Control Center for the configuration of HADR.

Create standby database environment

First of all, we have to prepare the new DB2 environment. Install DB2 with the same fix level as the primary database on a separate server machine and create users, groups, and a new database instance in the same way as on the primary server.

Catalog the standby database instance

In order to set up the HADR configuration, the instance of the standby database must be cataloged on the primary database as a remote node. Start the DB2 Control Center on the primary database server as shown in Figure 5-6 on page 134.

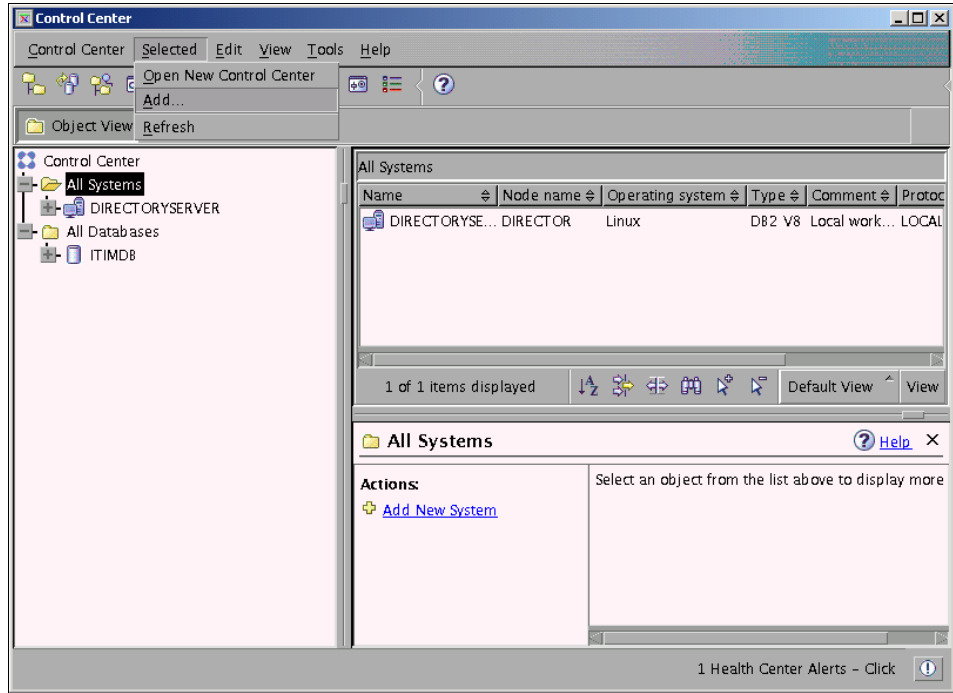


Figure 5-6 DB2 Control Center

To start the DB2 Control Center on the primary database server:

1. Click **All Systems**, and select **Selected** → **Add** from the menu bar.

See Figure 5-7 on page 135.

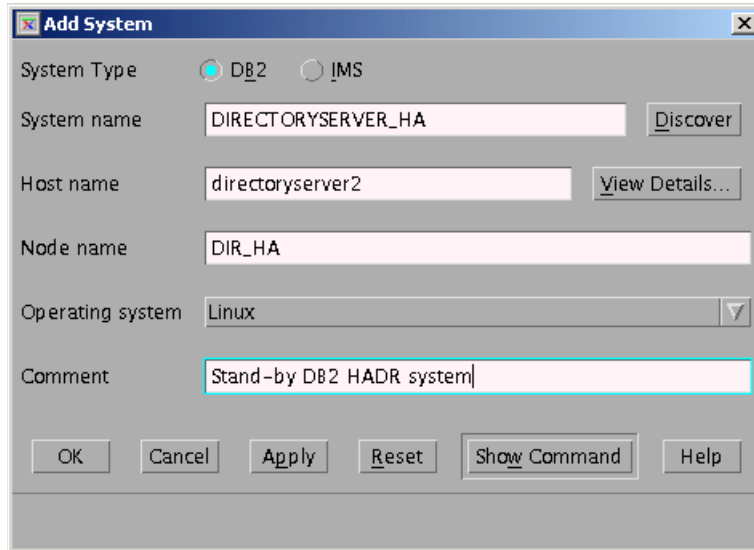


Figure 5-7 Add new system details

- As shown in Figure 5-7, enter the standby database server information. In TAA's environment, these are:

System Type	DB2
System name	DIRECTORYSERVER_HA
Host name	directoryserver2
Node name	DIR_HA
Operating system	Linux
Comment	Standby DB2 HADR system

- Click **OK**. The new system is added to the list of systems as depicted in Figure 5-8 on page 136.

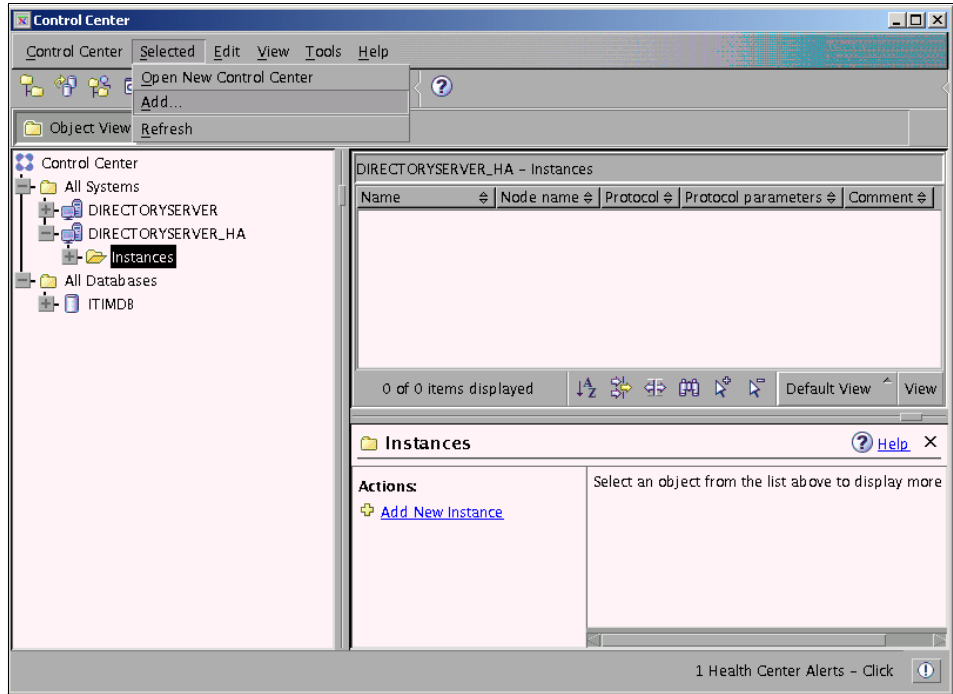


Figure 5-8 New system added in the DB2 Control Center

4. Expand the DIRECTORYSERVER_HA system and click **Instances**, select **Selected** → **Add** to catalog the standby database instance.

See Figure 5-9 on page 137.

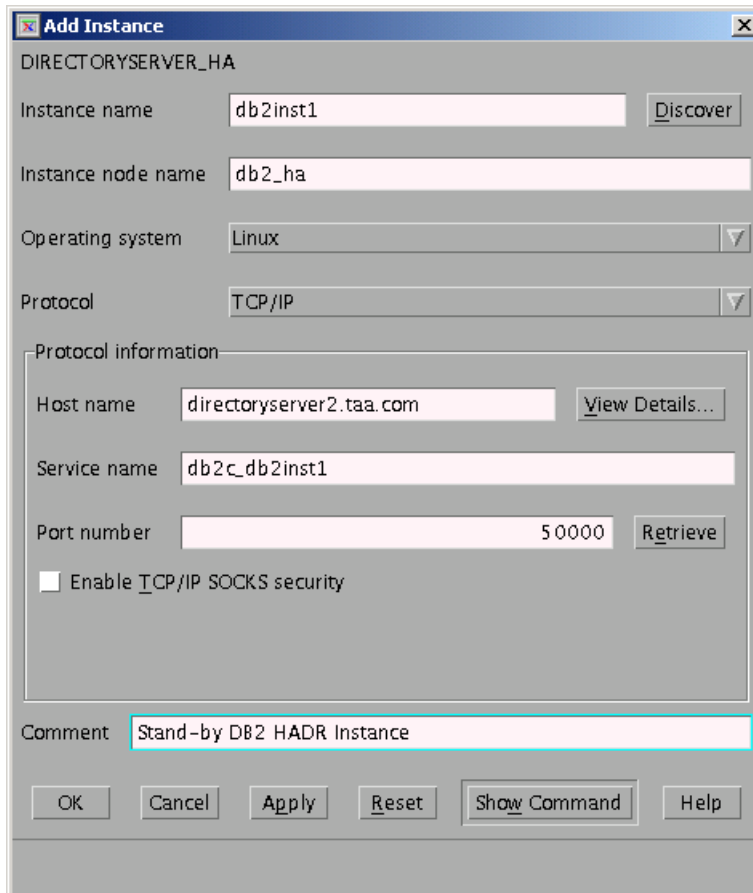


Figure 5-9 Add new database instance details

- As shown in Figure 5-9, enter the information about the standby database instance. In TAA's case, we entered:

Instance name db2_inst1
Instance node name db2_ha
Operating System Linux
Protocol TCP/IP

Enter Protocol information:

Host name directoryserver2.taa.com
Service name db2c_inst1
Port number 50000
Enable TCP/IP SOCKS security Off

Comment Standby DB2 HADR Instance

6. Click **OK**.

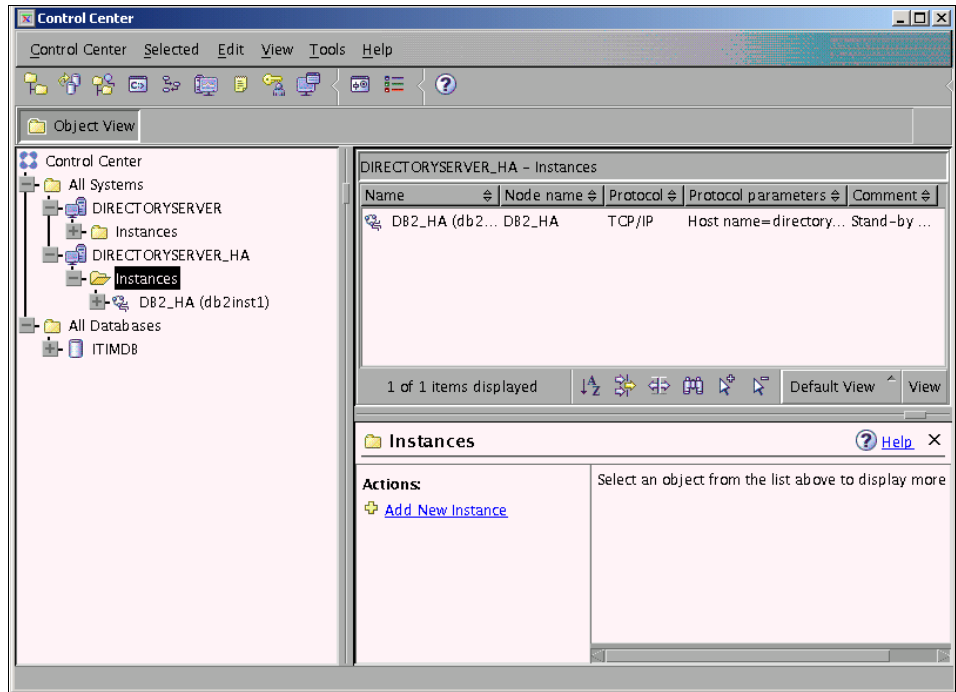


Figure 5-10 New database instance added to the Control Center

7. Ensure that the new instance is added as a remote node, shown in Figure 5-10.

Configure the HADR databases

To set up the HADR pair, we use the HADR wizard of Control Center.

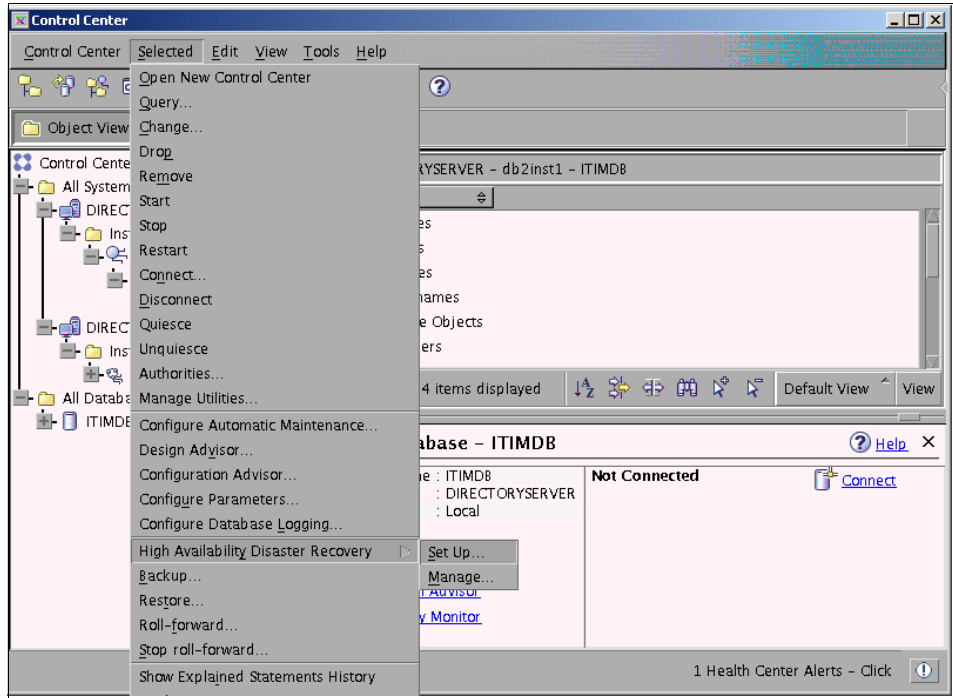


Figure 5-11 Using the HADR Control Center wizard

To set up the HADR pair:

1. Select **ITIMDB**, which is used as the primary database by Identity Manager, and select **Selected** → **High Availability Disaster Recovery** → **Set up** from the menu shown in Figure 5-11. After the Introduction panel of the HADR wizard appears, click **Next** to continue.

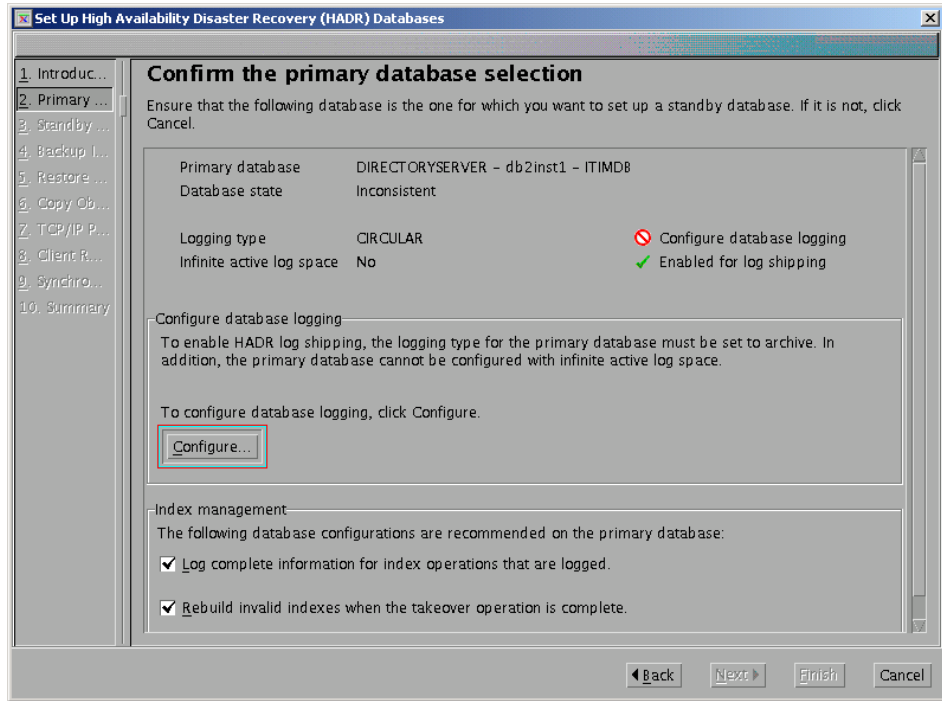


Figure 5-12 HADR configuration wizard

2. Because the HADR database must not use circular logging, we need to change the logging type for the ITIMDB from circular to archive. Click **Configure** to start the Configure Database Logging Wizard, as depicted in Figure 5-12, and follow its guides. In this configuration step, a full database backup is performed by the wizard and we use the backup image to initialize the standby database.

After the configuration of the logging type, continue with the HADR wizard in Figure 5-13 on page 141.

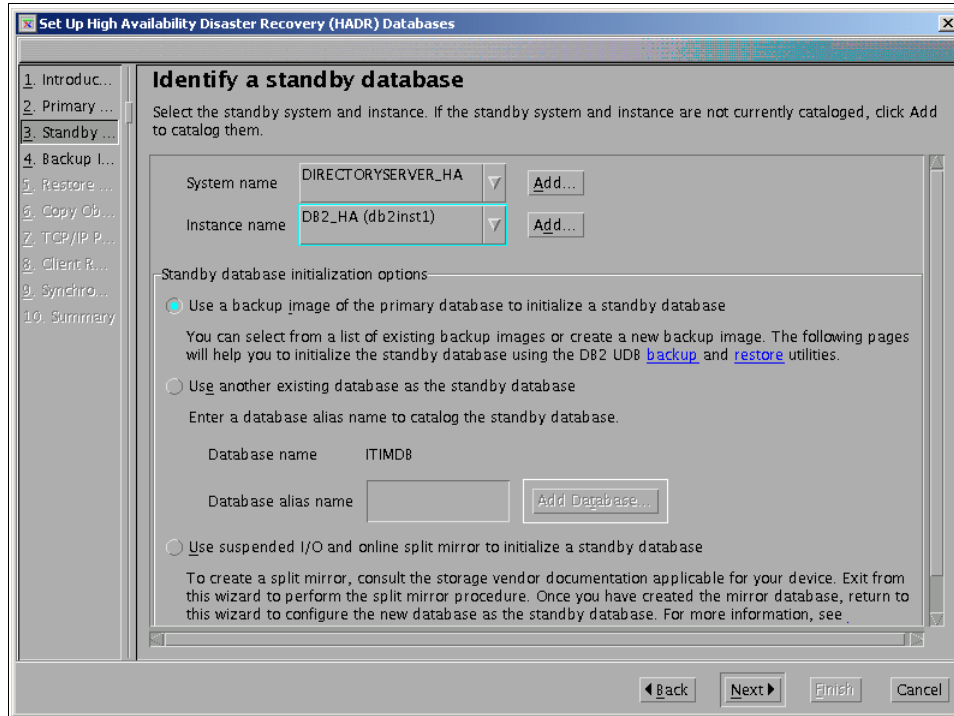


Figure 5-13 HADR configuration wizard: continued

3. Enter the standby database information, shown in Figure 5-13, as follows:

Specify the database instance to set up the standby database:

System name DIRECTORYSERVER_HA

Instance name DB2_HA

Select **Use a backup image of the primary database to initialize a standby database** as initialization options.

Click **Next**.

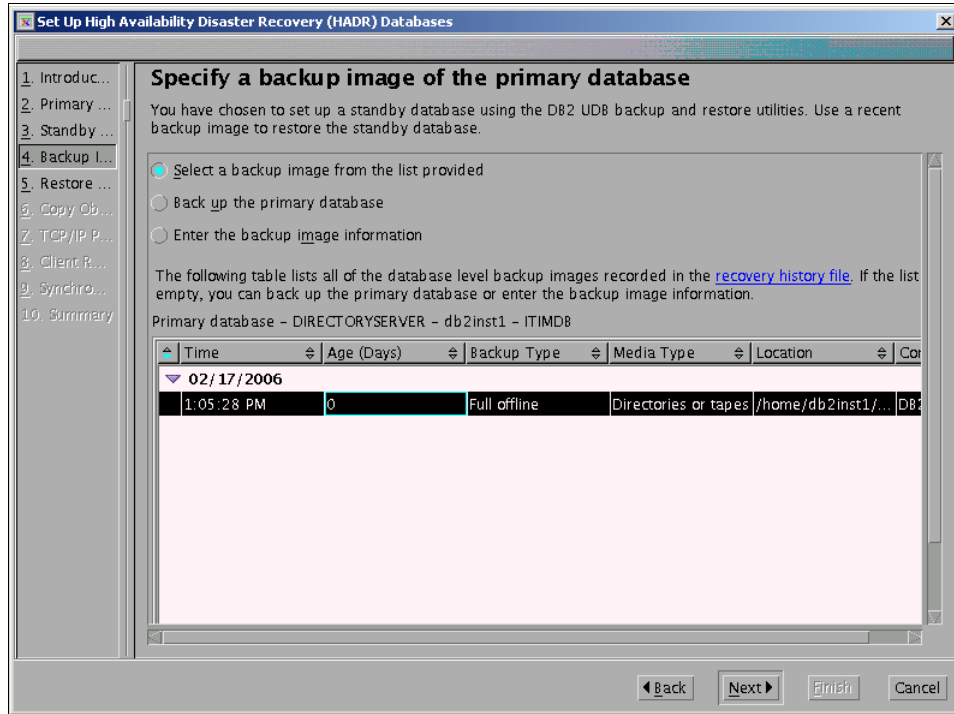


Figure 5-14 HADR configuration wizard: continued

4. Specify the backup image to initialize the standby database.

Choose **Select a backup image from the list provided** as shown in Figure 5-14.

Select the backup image of the primary database created at the change of the logging type.

Click **Next**.

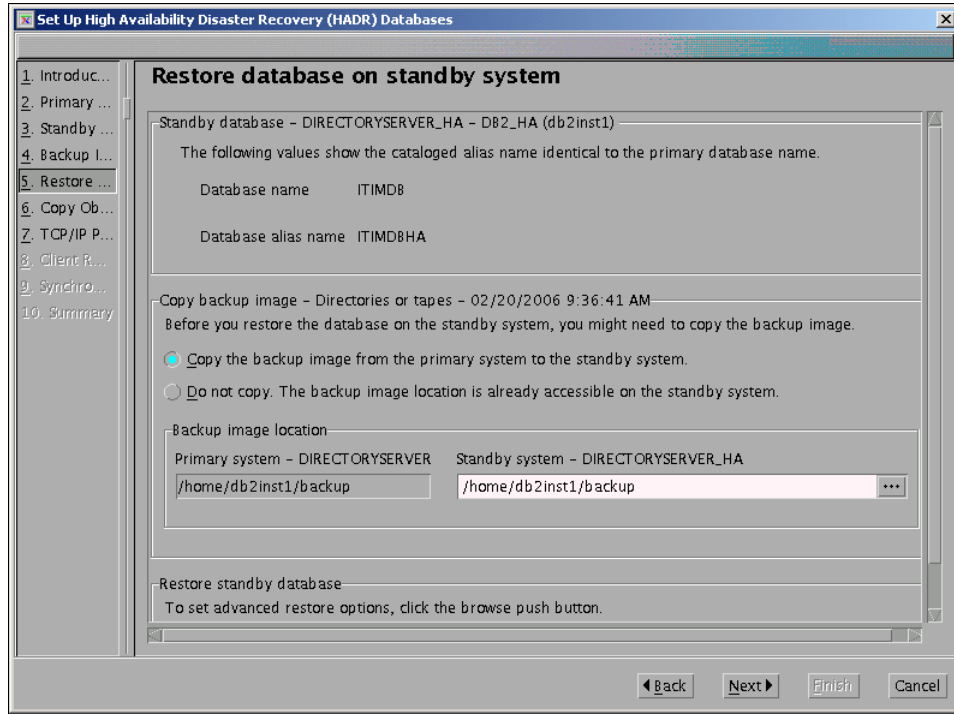


Figure 5-15 HADR configuration wizard: continued

5. Enter the standby database information.

Click **Add Databases** to specify the standby database alias name.

Alias ITIMDBHA

Comment Standby ITIMDB

6. As shown in Figure 5-15, ensure that **Copy the backup image from the primary system to the standby system** is selected and enter the directory path on the standby database server where the backup image is copied to. The directory must be created and given write permissions in advance.

Backup image location /home/db2inst1/backup

Click **Next**.

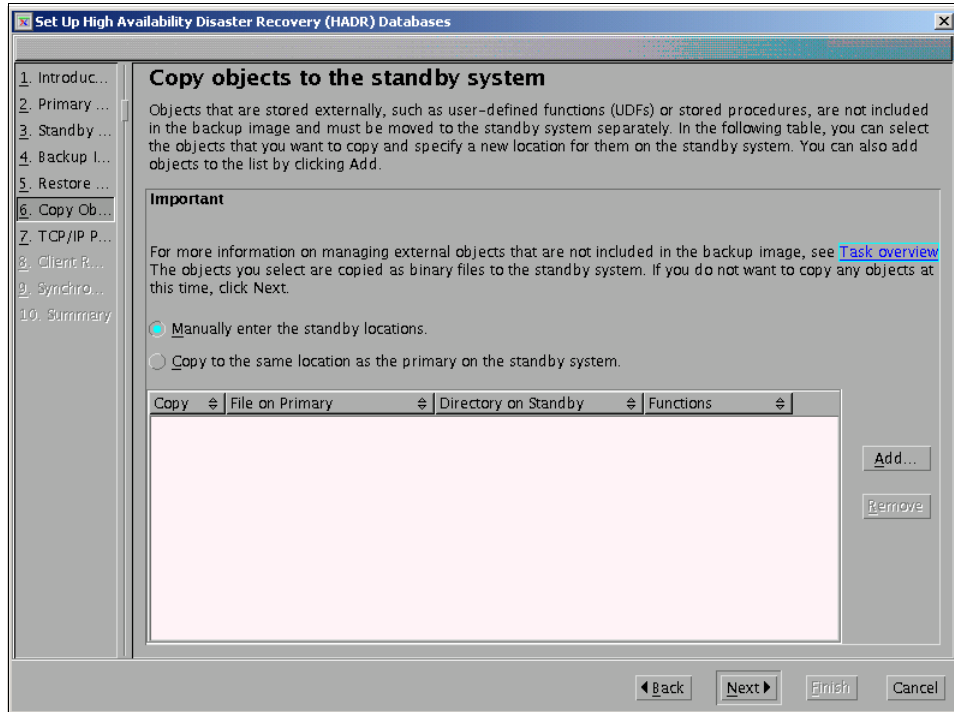


Figure 5-16 HADR configuration wizard: continued

7. Ensure **Manually enter the standby locations** is selected as shown in Figure 5-16 and click **Next**.

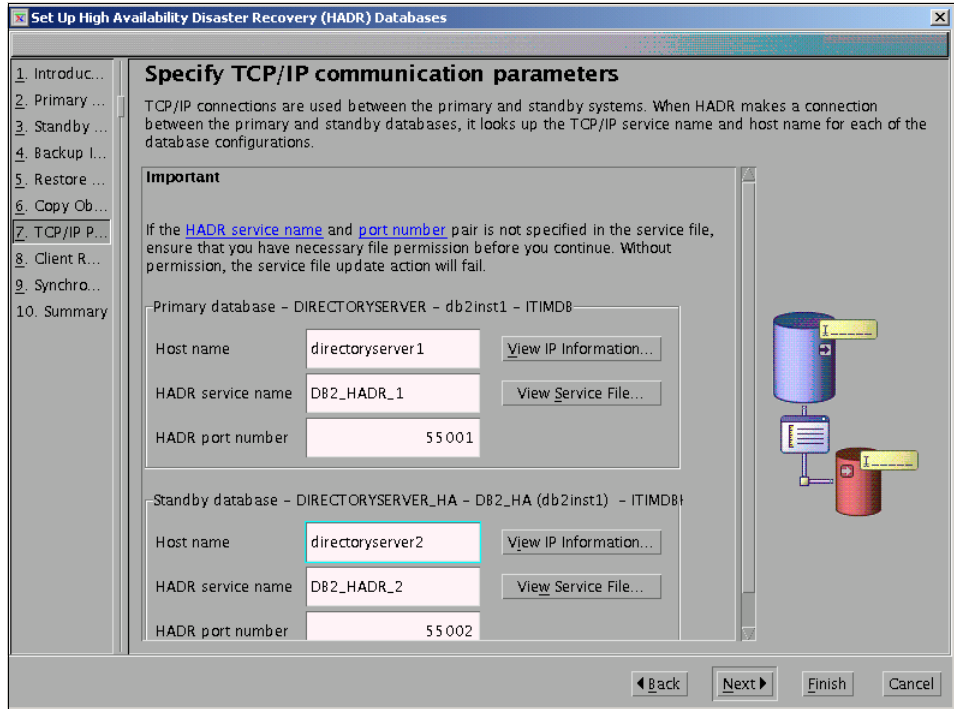


Figure 5-17 HADR configuration wizard: continued

8. Specify the information for the log shipping connection as depicted in Figure 5-17.

Enter the primary database TCP/IP configuration.

Host name directoryserver1
HADR servicename DB2_HADR_1
HADR port number 55001

Enter the standby database TCP/IP configuration.

Host name directoryserver2
HADR servicename DB2_HADR_2
HADR port number 55002

Click **Next**.

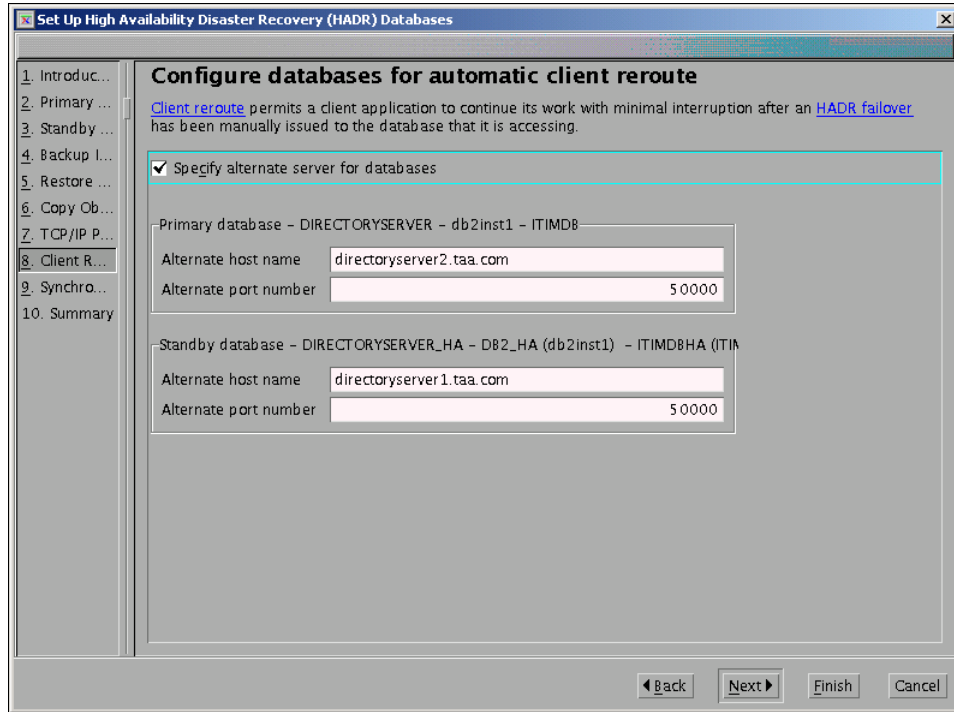


Figure 5-18 HADR configuration wizard: continued

9. To enable automatic rerouting of the connection from Identity Manager at database failure, configure the alternate host name and port number for each database.

As shown in Figure 5-18, ensure that **Specify alternate server for databases** is checked.

Enter the alternate server information of the primary database:

Alternate host name directoryserver2

Alternate port number 50000

Enter the alternate server information of the standby database:

Alternate host name directoryserver1

Alternate port number 50000

Click **Next**.

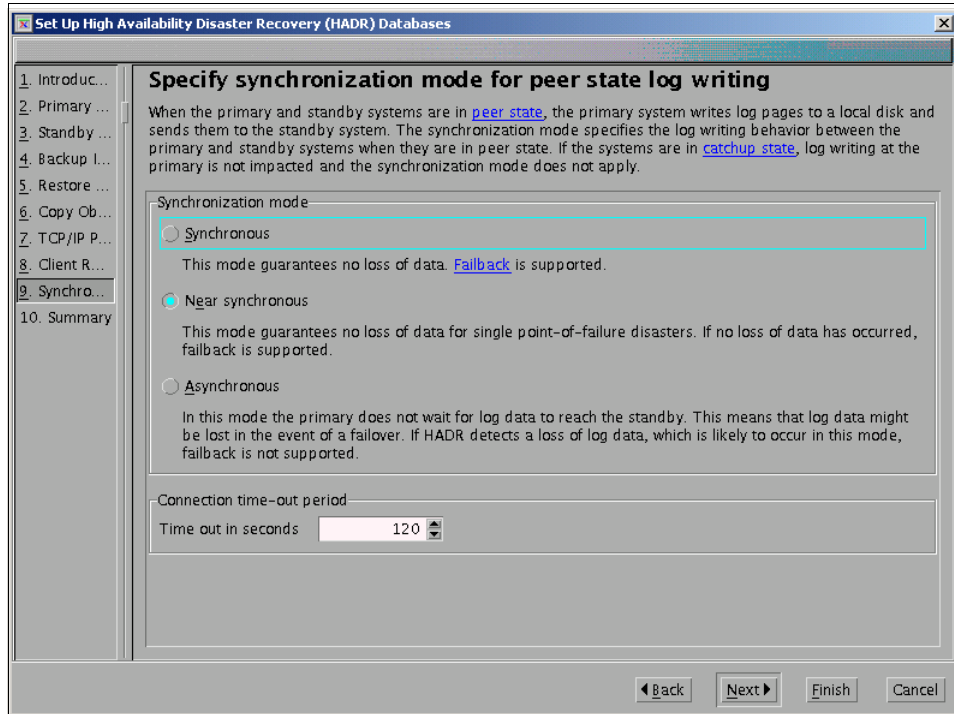


Figure 5-19 HADR configuration wizard: continued

10. In TAA's case, we use *Near synchronous* mode for peer state log writing.

Ensure that **Near Synchronous** is selected as shown in Figure 5-19 and click **Next**.

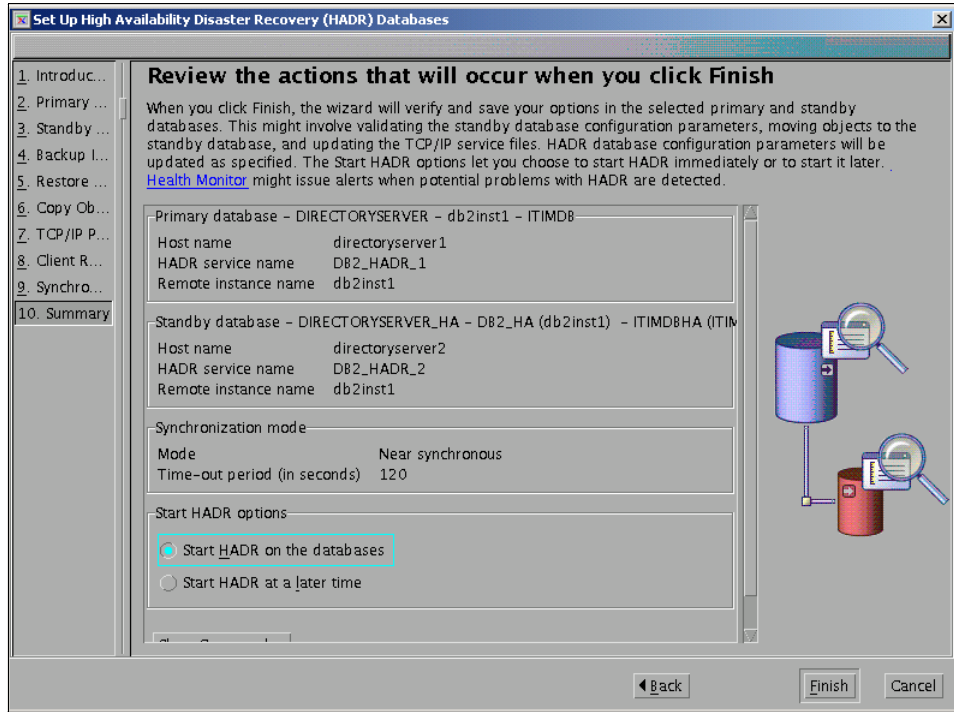


Figure 5-20 HADR configuration wizard: continued

11. Review the configuration of the HADR pair shown in Figure 5-20 and ensure that **Start HADR on the databases** is selected.

Click **Finish**.

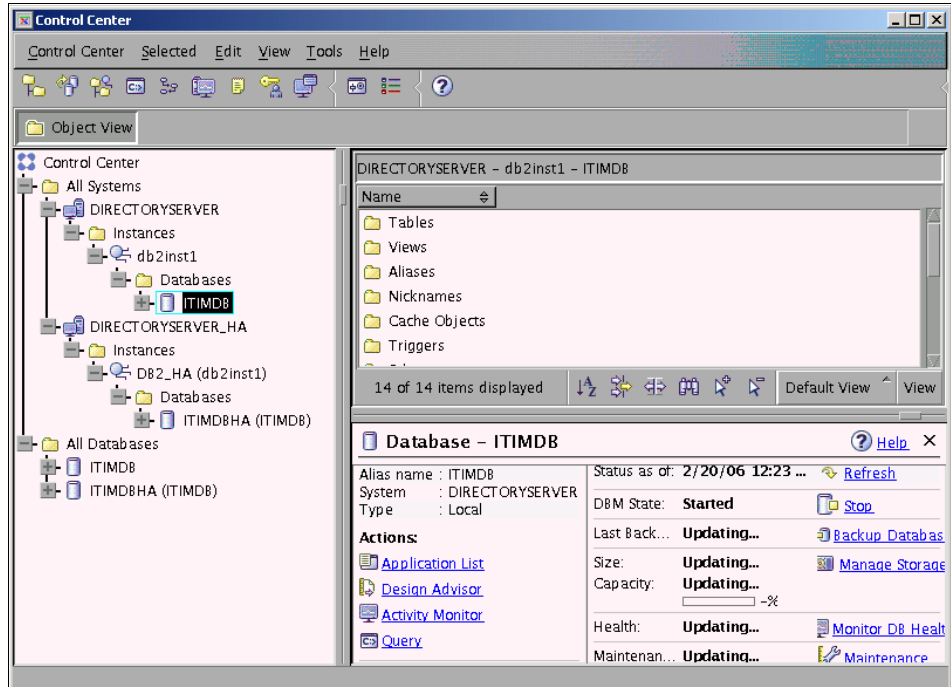


Figure 5-21 Finished HADR configuration in the Control Center

12. We now have completed the setup of the HADR pair. To verify the HADR configuration, open an HADR Action window.

Select **ITIMDB** and select **Selected** → **High Availability Disaster Recovery** → **Manage** from the menu bar.

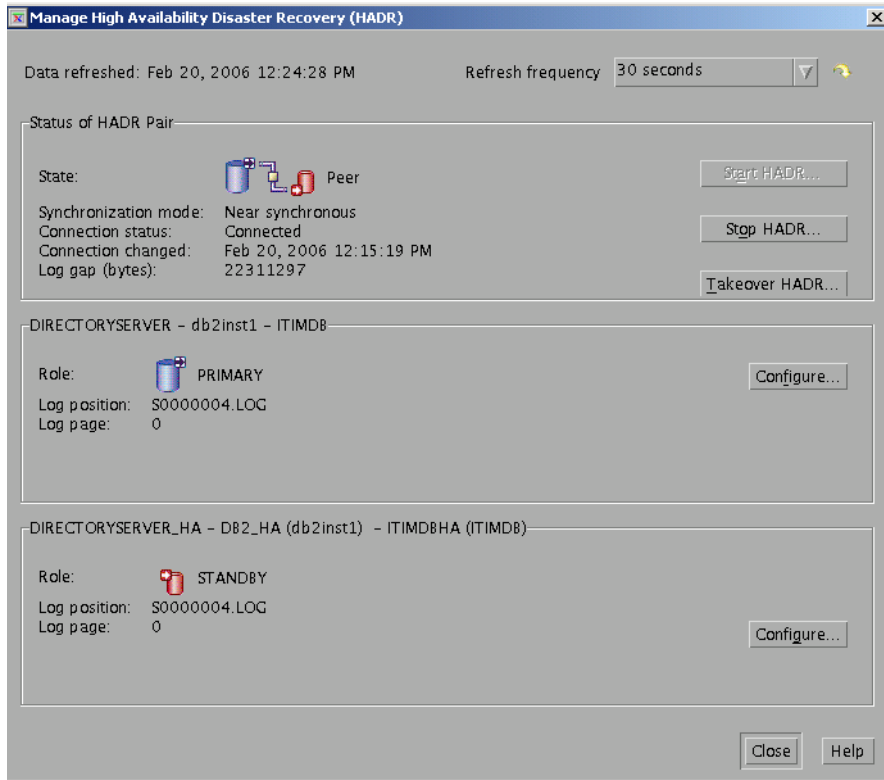


Figure 5-22 HADR action window

13. Ensure that the state of the HADR pair is *Peer* and the connection status is *Connected* as shown in Figure 5-22.
14. Click **Takeover HADR** and ensure that **Switch roles** is selected on the next panel, then click **OK**.
15. Ensure that the takeover completed successfully.

Implement automatic failover

To automate the failover of the database, TAA has decided to place a shell script on the database servers. The script monitors the connection status of the HADR pair on the standby server and issues a takeover command if the status turns into a disconnected status.

The script code used in TAA's environment is shown in Example 5-1 on page 151.

Example 5-1 Sample script to automate database failover

```
#!/usr/bin/ksh

#####
# auto_takeover.ksh
# This script must be called by database instance user.
# This script monitor the HADR status at 30-seconds interval
# and issue takeover if HADR pair is disconnected.
# Exit with:
# 0 - HADR takeover is completed successfully.
# 255 - The database is not a standby database.
# 254 - Failed to get the status of the HADR pair.
# 253 - Failed to takeover.
#####
#set -x

#####
# Set parameters and error codes
#####
DBNAME=ITIMDB
MONITOR_INTERVAL=30
EXIT_ROLE_ERROR=255
EXIT_STATUS_ERROR=254
EXIT_TAKEOVER_ERROR=253

#####
# hadr_monitor
# Monitor the HADR status
# Exit with:
# 0 - HADR pair is working normally.
# 1 - HADR pair is disconnected.
# EXIT_ROLE_ERROR - The database is not a standby database.
# EXIT_STATUS_ERROR - Failed to get the status of the HADR pair.
#####
function hadr_monitor
{
hadr_role=$( db2 get snapshot for all on ${DBNAME} | awk ' $1 == "Role" && $2
==
    "=" {print $3}' )
hadr_role_cfg=$( db2 get db cfg for ${DBNAME} | grep 'HADR database role' | awk
'{print $5}')
if [[ "${hadr_role_cfg}" != "STANDBY" || "${hadr_role}" != "Standby" ]];then
    return $EXIT_ROLE_ERROR
fi

hadr_connection_status=$( db2 get snapshot for all on ${DBNAME} |awk '$1 ==
"Con
nection" && $2 == "status" && $3 == "=" {print $4}')
```

```

if [[ "${hdr_connection_status}" = "" ]];then
    return $EXIT_STATUS_ERROR
elif [[ "${hdr_connection_status}" = "Disconnected," ]];then
    return 1
fi
return 0
}

#####
# main
# Call hdr_monitor every $MONITO_INTERVAL and issue takeover
# if HADR pair is disconnected.
#####
while true
do
    hdr_monitor
    HADR_STATUS=?

    if [[ $HADR_STATUS = $EXIT_ROLE_ERROR ]];then
        echo $DBNAME is not a standby database. Exiting.
        exit $EXIT_ROLE_ERROR
    elif [[ $HADR_STATUS = $EXIT_STATUS_ERROR ]];then
        echo Failed to get the connection status of HADR. Exiting.
        exit $HADR_STATUS=ERROR
    elif [[ $HADR_STATUS = 1 ]];then
        echo HADR connection status is "Disconnected". Calling takeover
command.
        db2 takeover hdr on db $DBNAME by force
        if [[ $? = 0 ]];then
            echo HADR takeover completed successfully. Exiting.
            exit 0
        else
            echo HADR takeover failed. Exiting.
            exit $EXIT_TAKEOVER_ERROR
        fi
    elif [[ $HADR_STATUS = 0 ]];then
        echo $DBNAME is working as a standby database normally.
    fi
    sleep $MONITOR_INTERVAL
done

```

5.4 Directory Server high availability

This section discusses TAA's requirements, design considerations, and implementation of the Tivoli Directory Server high availability environment.

5.4.1 Requirements

TAA wants to increase efficiency and productivity on its identity management services. In order to achieve this, TAA must closely integrate its internal processes with its business partners. The introduction of an extranet to its business partners and customers requires that the user registry used by Identity Manager must be highly available and implemented with automatic failover mechanisms.

5.4.2 Design considerations

The design and implementation of a distributed directory can be approached by selecting different types of directory topologies and different types of components to manage the failover and load balancing. The implementation described in 5.4.3, “TAA’s Directory Server high availability implementation” on page 153 is based on the following design considerations:

- ▶ Identity Manager mainly executes write mode requests on the directory server services.
- ▶ A manual failover within a directory server cluster produces an unacceptable down time for the service level defined by TAA. TAA requires an automatic failover between two back-end directory servers.
- ▶ In order to keep the costs related to high availability as low as possible, TAA wants to use the software components already available with Identity Manager software licenses.
- ▶ In a directory server failure situation, an administrator or user currently using the Identity Manager Web administrative console must reauthenticate.

Based on these design considerations, the high availability directory server environment will look like this:

- ▶ Peer-to-peer topology for back-end Tivoli Directory Server
Multi-master environment for all the servers available on the distributed directory environment
- ▶ Proxy server used to manage failover requests to the peer-to-peer topology
Proxy server component available with Tivoli Directory Server Version 6.1 package

5.4.3 TAA’s Directory Server high availability implementation

The following section shows all steps necessary to configure the Tivoli Directory Server in a peer-to-peer topology using the Tivoli Directory Server proxy server as a load balancer.

These steps explain the implementation of a Tivoli Directory Server peer-to-peer topology starting from an already implemented Tivoli Directory Server instance being used as the Tivoli Identity Manager user registry.

Here is a brief list of steps required to implement a Tivoli Directory Server high availability solution:

1. Create a second Tivoli Directory Server instance.
2. Synchronize the encryption key and the schema between the two Tivoli Directory Server instances.
3. Enable master-slave replication between the two Directory Server instances.
4. Promote the secondary Directory Server instance to master.
5. Create a third Directory Server instance for the Tivoli Directory Server proxy server.
6. Synchronize the encryption key and the schema between the Tivoli Directory Server proxy server and the back-end Directory Server.

Note: In this example, TAA first creates a master-slave replication and then promotes the slave to a master. This step is no longer needed now with the new IBM Directory Server 6.1, and the replica can start as master, which is different that it was in previous versions. See the *IBM Tivoli Directory Server Installation and Configuration Guide Version 6.1*, SC32-1560, for more details.

Create Tivoli Directory Server secondary instance

To configure the secondary instance, issue the following steps:

1. If the `idsxinst` application is not running execute the command `idsxinst &` from the command prompt. The ampersand (&) specifies that the process will run in the background. When returning to the command prompt, press the Enter key to get the command prompt back.

The window in Figure 5-23 on page 155 shows the result of the `idsxinst` command.

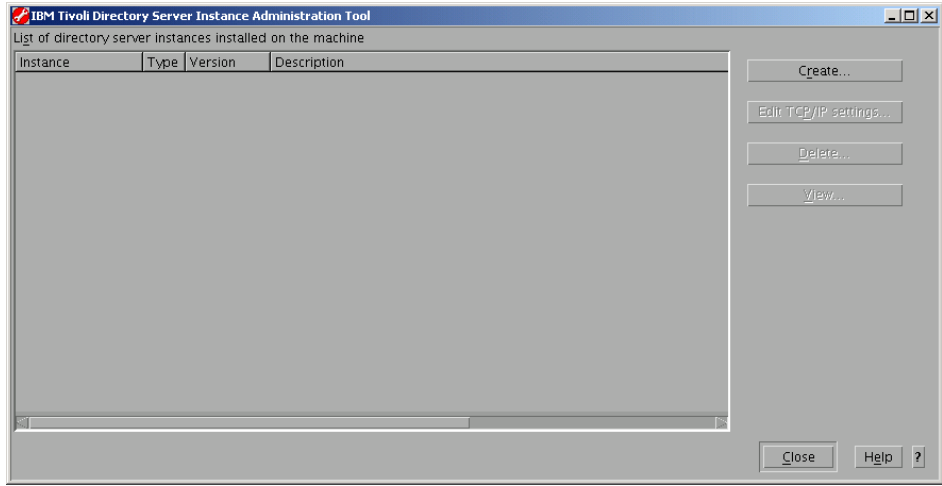


Figure 5-23 Tivoli Directory Server administration tool

2. Click **Create**.
3. Select **Create a new directory server instance**, and click **Next**.

Enter the following values in the instance details fields shown in Figure 5-24 on page 156:

User name	ids1dap1
Install location	Leave blank
Encryption seed string	sunshine (use the same seed string used for Directory Server primary configuration)
Instance description	For example, TDS Server #2

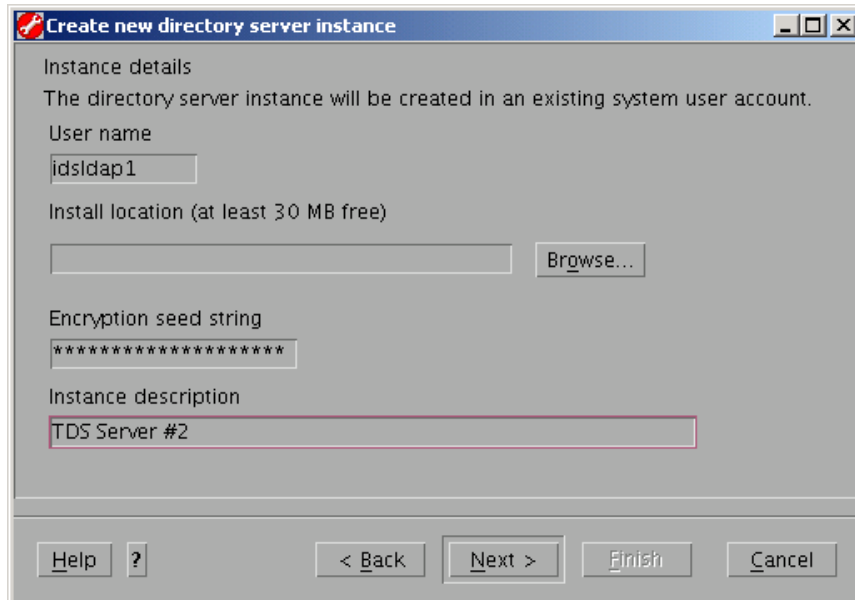


Figure 5-24 Tivoli Directory Server second instance creation

4. Click **Next**.
5. On the DB2 instance details window, ensure that **idsldap1** is selected as the DB2 instance name, and click **Next**.
6. On the TCP/IP settings for Multihomed¹ hosts window, ensure that the **Listen on all IP addresses** check box is selected, and click **Next**.
7. On the TCP/IP port settings, shown in Figure 5-25 on page 157, enter the following port details:

Server port number	389
Server secure port number	636
Admin daemon port number	3538
Admin daemon secure port number	3539

¹ Multihomed means that a server supports a configuration on multiple IP addresses.

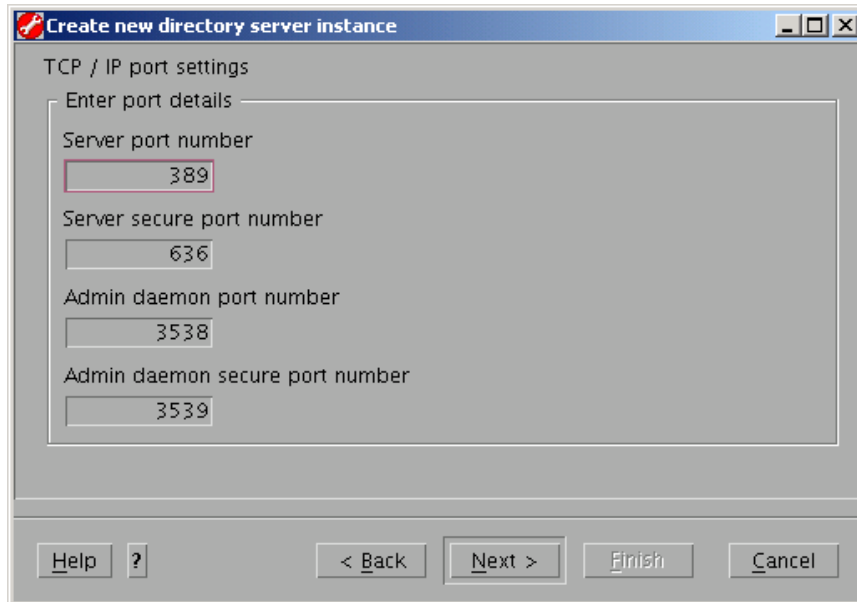


Figure 5-25 Tivoli Directory Server second instance listening port configuration

8. Click **Next**.
9. On the Optional Steps window, ensure that the **Configure Admin DN and Password** and **Configure database** check boxes are selected, and click **Next**.
10. On the Configure administrator DN and password window, enter the following values:

Administrator DN	cn=root
Administrator password	<admin_password>
Confirm password	<admin_password>
11. Click **Next**.
12. On the Configure database window, enter the following values:

Database user name	idsldap1
Password	<idsldap1 OS password>
Database name	idsldap1
13. Click **Next**.
14. Keep the defaults on the Database options window, and click **Next**.

15. Review the settings, and click **Finish** to create the directory and database instances.

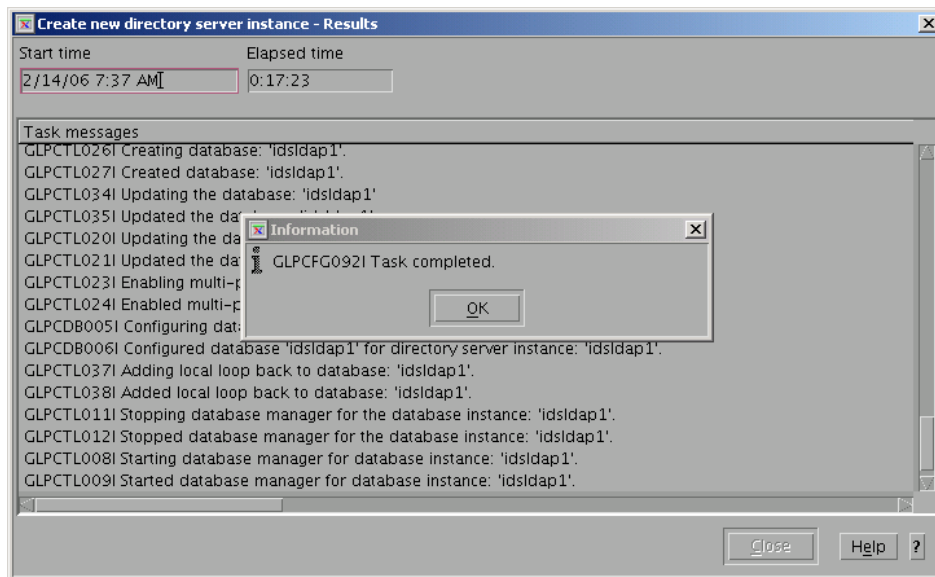


Figure 5-26 Tivoli Directory Server secondary instance creation summary

16. On the Results window shown in Figure 5-26, click **OK**, then click **Close**, and finally, click **Close** again to exit the idxsinst application.

Note: Do not start the Tivoli Directory Server instance you have created now. This instance will be started after the suffix configuration and the cryptography keyring file synchronization. Follow the sections “Create the suffix on Tivoli Directory Server” on page 158 and “Copy modified schema to the secondary Directory Server” on page 159 before starting the Tivoli Directory Server instance.

Create the suffix on Tivoli Directory Server

The Tivoli Directory Server process must be stopped to create the suffix. The Tivoli Directory Server instance must not be started until after the next section.

To add the Tivoli Directory Server suffix, we use the Tivoli Directory Server configuration console:

1. From a command line window, run the command **idsxcfg &**.
2. Select **Manage Suffix** on the left menu.
3. Add the suffix DN **dc=com** and press **Add** as depicted in Figure 5-27.

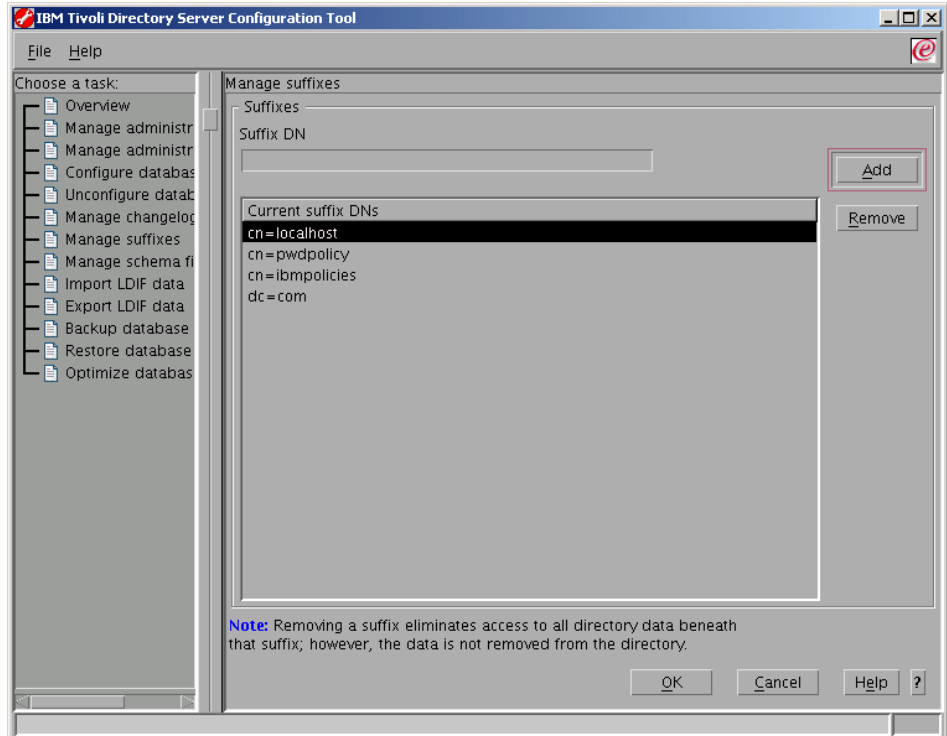


Figure 5-27 Tivoli Directory Server graphical configuration tool

4. Click **OK** to close the Tivoli Directory Server configuration console.

Copy modified schema to the secondary Directory Server

The Tivoli Identity Manager installation adds new objectclasses and attributes to the Directory Server user registry. In order to replicate the modified LDAP schema on the new Tivoli Directory Server instance, it is necessary to copy the modified schema on the new server instance.

Copy the V3.modifiedschema file from the primary Tivoli Directory Server `/<idsslapd_instance_primary>/etc/V3.modifiedschema` to the secondary Tivoli Directory Server `/<idsslapd_instance_secondary>/etc/V3.modifiedschema`.

Synchronize the cryptography keyring file

Note: Tivoli Directory Server V6.1 uses a default password encryption method - AES256. Because of this it is necessary for the Tivoli Directory Server primary and secondary instance to share an encryption keyring. The other option would be to go into the Tivoli Directory Server Web Administration Tool and change the security settings to another encryption algorithm, such as SHA or Crypt.

Follow these instructions to synchronize the cryptographic keyring file:

1. Run the bulkload command **idsbulkload** to create the initial cryptographic keyring file:

```
idsbulkload -I idsldap1
```

Disregard the `-i` message.

```
GLPBLK025E The -i option is required.
```

2. Copy the cryptographic keyring file:

```
/<idsslapd_instance_primary>/idsslapd-idsldap/etc/ibmslapddir.ksf
```

from the primary instance to the secondary instance.

```
/<idsslapd_instance_secondary>/idsslapd-idsldap/etc/ibmslapddir.ksf
```

3. Start the secondary instance, to load the new cryptographic keyring file.

```
idsslapd -I idsldap1
```

4. Run the bulkload command **idsbulkload** to finalize the synchronization of the cryptographic keyring file.

```
idsbulkload -I idsldap1
```

Load Identity Manager data on the secondary server instance

To import data created by Tivoli Identity Manager on the secondary Tivoli Directory Server instance, these have to be exported to a file in LDIF format and imported on the secondary Tivoli Directory Server instance.

Follow these instructions to export data from the primary Directory Server instance and to import onto the secondary Directory Server instance:

1. On the Tivoli Directory Server primary instance, export data to a file in LDIF data format:

```
idsdb2ldif -I <instance_name> -o <output_ldif_file>
```

2. Copy the `output_ldif_file` on the secondary Tivoli Directory Server instance.

3. Import LDIF exported data on the secondary Tivoli Directory Server instance.

```
idsldif2db -I <instance_name> -i <output_ldif_file>
```

There are several methods of loading LDIF formatted data into the directory. To import data from an LDIF file, you can use either the `idsldif2db` or the `idsbulkload` utility. The quickest way is to use the `idsbulkload` utility. The `idsldif2db` and `idsbulkload` utilities must be used while the directory server is offline. The `idsbulkload` utility bypasses the LDAP front end and loads the data directly into the back-end DB2 database. The `idsbulkload` utility usually is significantly faster than `idsldif2db` and `ldapadd` when loading approximately 100,000 to a million entries.

In our implementation, we used the `idsldif2db` utility to check data consistency during the import.

Enable Master-Slave replication

This phase of the configuration involves the task of setting up replication. There are two main types of replication:

- ▶ Master-Slave (also known as Supplier / Consumer)
- ▶ Multiple-Master (also known as peer-to-peer)

Master-Slave replication

Master-Slave replication involves a read-write-enabled master server sending updates to a read-only slave server. Many environments use this setup to help with application performance, because slave servers are not busy sending updates to other servers and can provide better performance to applications that require only search functionality.

Multiple-Master replication

Multiple-Master replication involves two or more read/write-enabled master servers in the environment. This is useful for an environment that requires 24x7x365 uptime. In this case, both servers can be "hot" in case there is a need for failover.

This section demonstrates how to set up both types of replication. Master-Slave replication is set up first, and then the slave replica is promoted to be a Master server. To do this, we use the primary and secondary Tivoli Directory Server instances.

The following processes must be started to complete this section:

1. The primary Tivoli Directory Server must be started.
2. The secondary Tivoli Directory Server must be started.
3. WebSphere Application Server must be started prior to accessing the Tivoli Directory Server Web Administration Tool.

Setup replication on the primary Tivoli Directory Server

To configure the replication on the primary Tivoli Directory Server instance, issue the following steps:

1. Access the Tivoli Directory Server Web Administration tool (Figure 5-28):

<http://directoryserver1.taa.com/IDSWebApp/IDSjsp/Login.jsp>

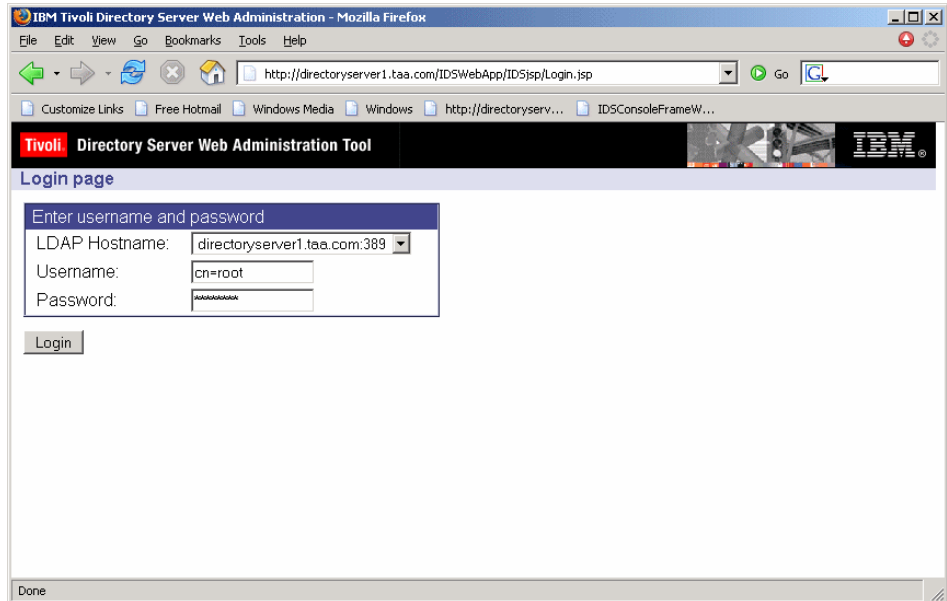


Figure 5-28 Tivoli Directory Server Web Administration Tool

2. After logging into the primary Tivoli Directory Server, select **Replication Management**.
3. Select **Manage topology**.
4. Click **Add**.
 - a. Enter `dc=com` in the Subtree DN box.
 - b. Check to ensure that `ldap://directoryserver1.taa.com:389` is in the master server referral URL field.

c. Click **OK** to save the changes.

Figure 5-29 shows the results of this configuration.

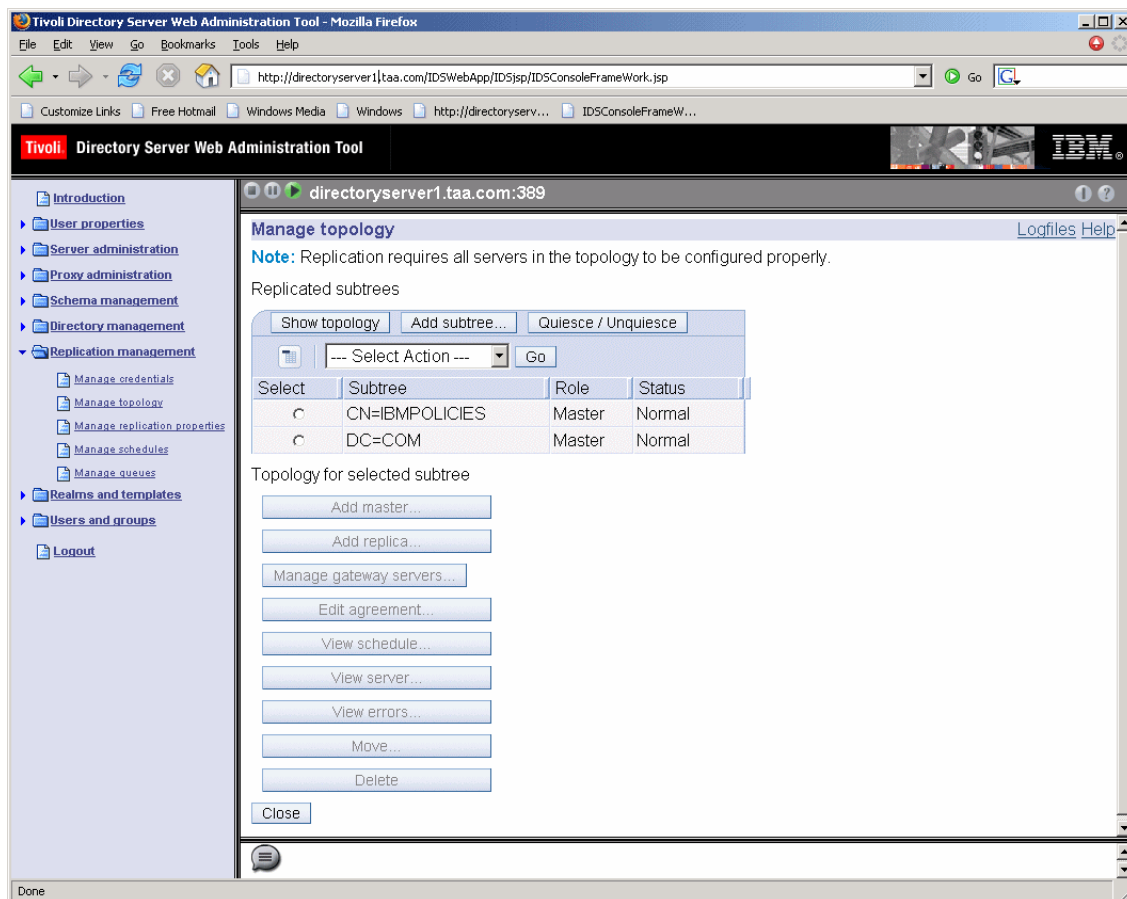


Figure 5-29 Manage suffix for Tivoli Directory Server replication

5. Select **Manage credentials**.
6. Select **DC=COM** from the subtree list.
7. Click **Show Credentials**. There should be no credentials listed for the DC=COM tree.

Figure 5-30 on page 164 shows what these steps should look like.

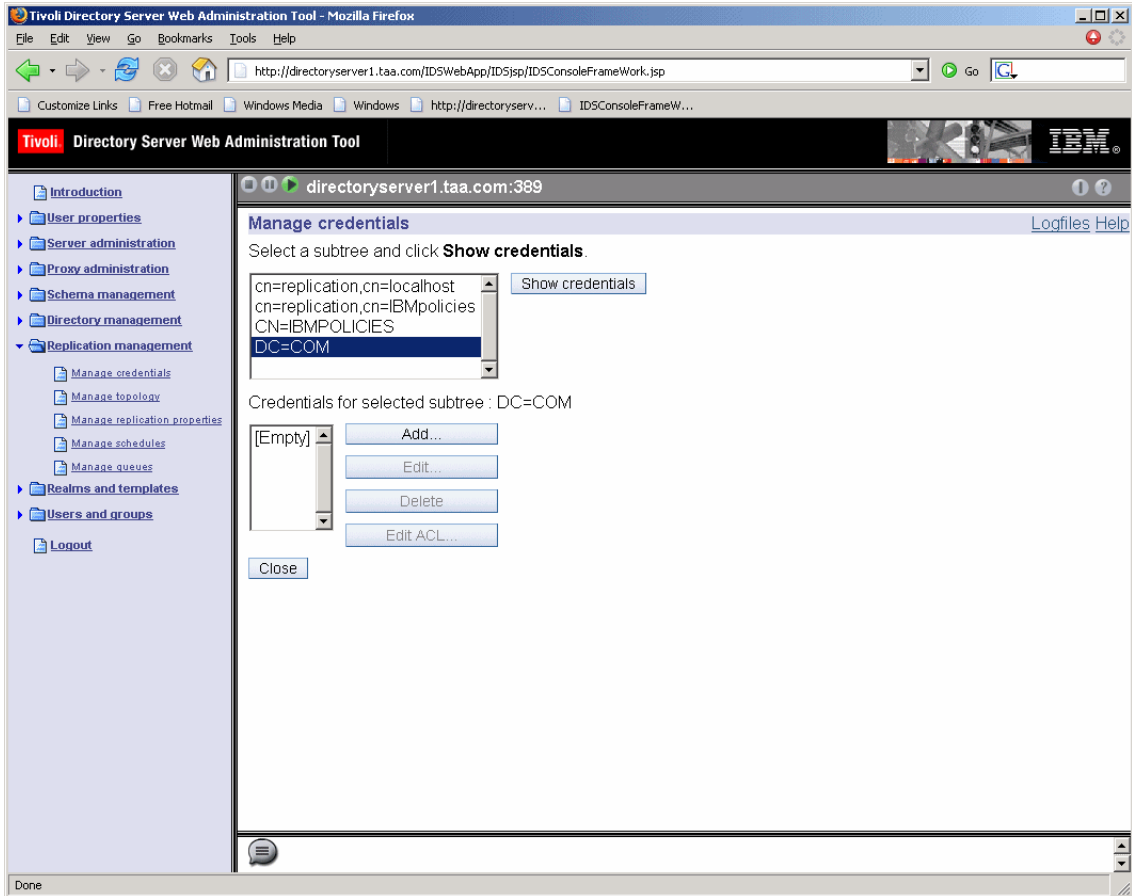


Figure 5-30 Manage credentials for Tivoli Directory Server replication

8. Click **Add** to add the credentials for the replicated subtree.
9. Add the following credential information:

Credential Name	cn=replicamanager
Authentication method	Simple bind
10. Click **Next**.
11. Enter the Simple bind information as shown in Figure 5-31 on page 165:

Bind DN	cn=replicamanager
Bind password	passw0rd
Confirm password	passw0rd
Description	leave blank

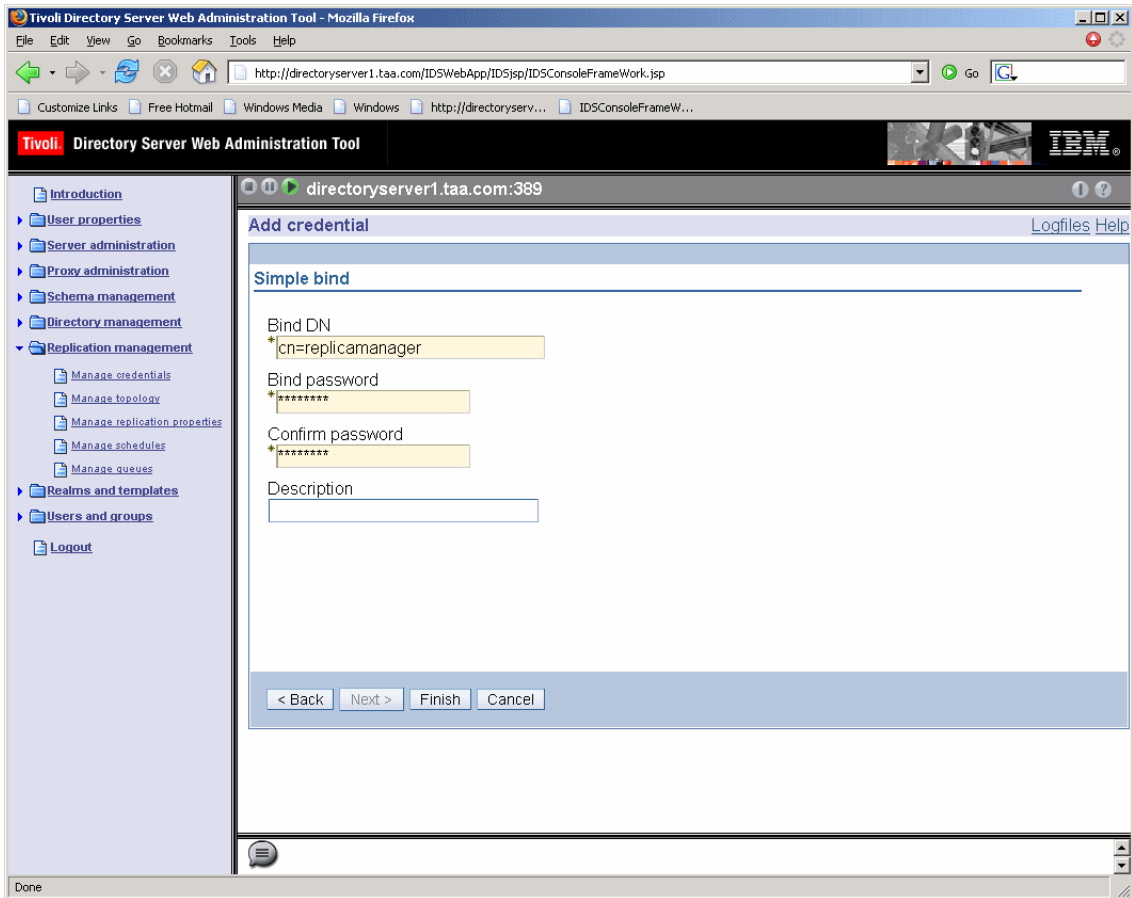


Figure 5-31 Create credential for replication

12. Click **Finish** to save the changes.
13. Click **Close** to complete this step.

Now that the credentials are configured for the DC=COM subtree, it is time to configure the replication topology. The following steps define the server that will be the replica of the master server, Tivoli Directory Server secondary instance:

1. Under Replication management, select **Manage topology**.
2. With the **DC=COM** subtree selected, click **Show topology**.
3. From the Topology for the selected subtree panel shown in Figure 5-32 on page 166, click **Add replica**.

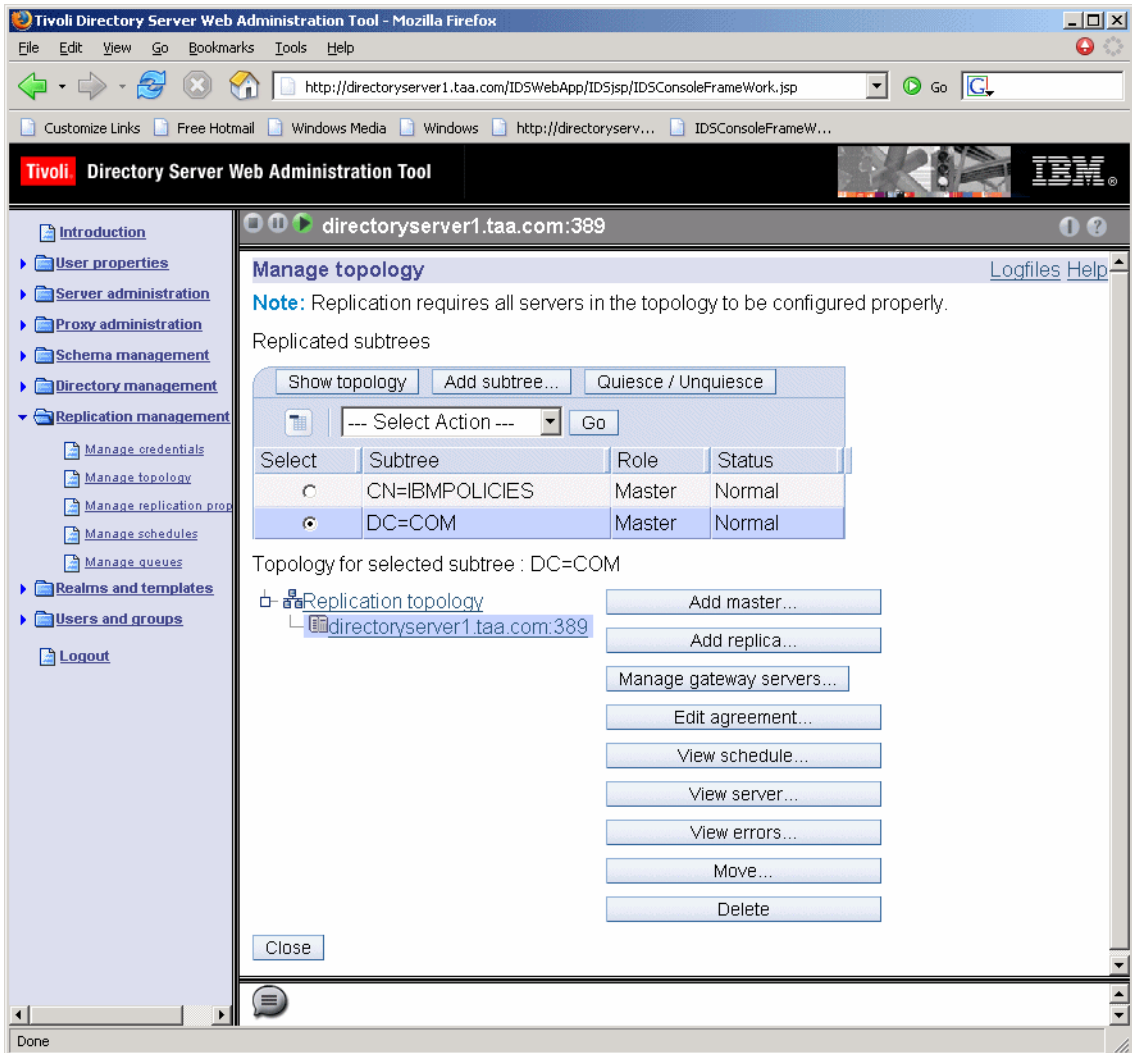


Figure 5-32 Manage topology view

- On the Add replica window, shown in Figure 5-33 on page 167, enter the following information:

Hostname	directoryserver2.taa.com
Port	389
Enable SSL	Leave unchecked.
Replica Name	Leave blank.

Replica ID	Click Get replica ID .
Description	Leave blank.
Credential Object	Click Select , which opens a new window. In the Select Credential window, select the radio button next to the DC=COM entry. Click Show Credentials to show the previously configured credential information. With the replicamanager credential displayed, click OK .

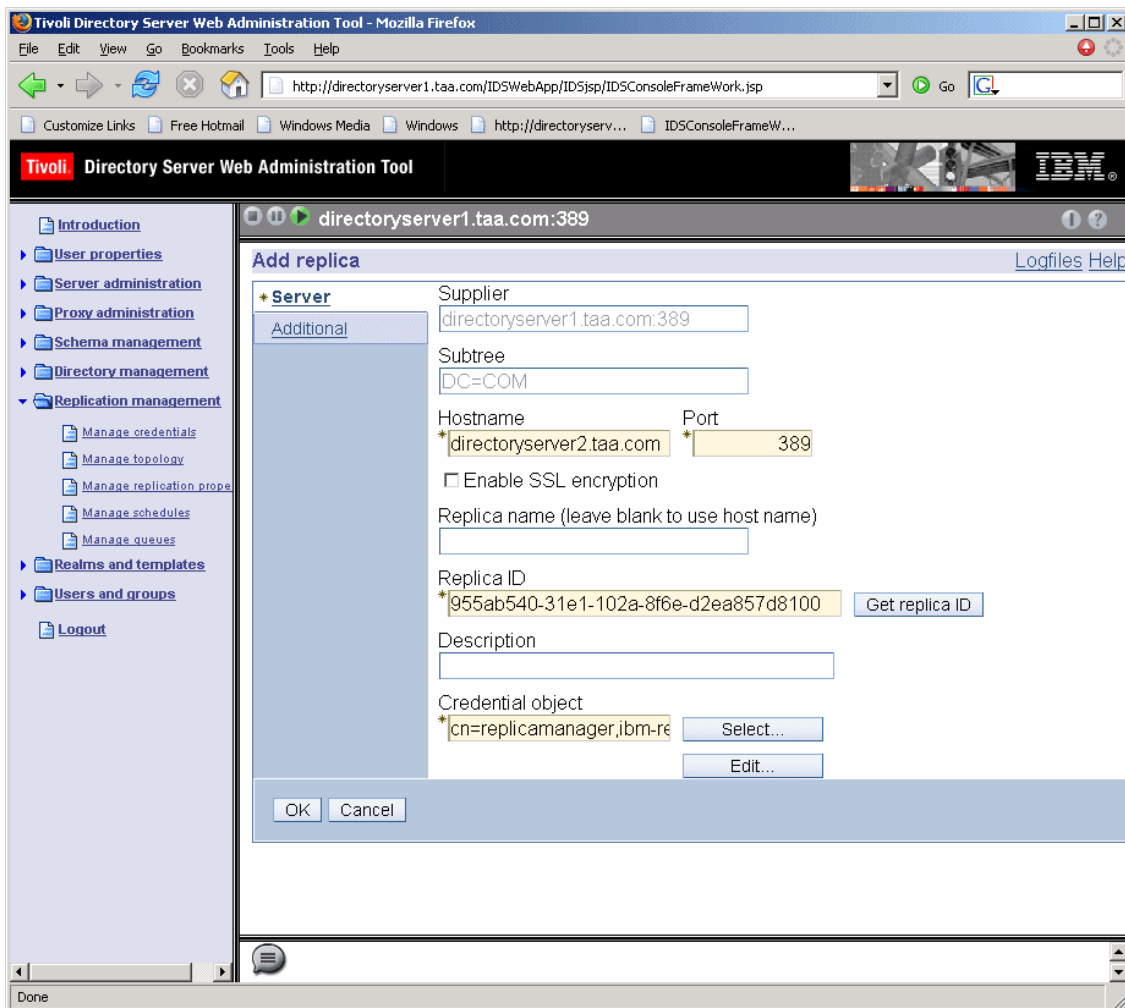


Figure 5-33 Create agreement for Directory Server replica

5. Click the **Additional** menu tab to continue to the next step.

The Add Replica - Additional window allows the administrator to add further details about the replica, including the new feature allowing multithreaded replication to help with replication performance.

On this window, the only change is to add the credentials to the consumer machine:

Note: During this process, the setup logs in to the replica server and makes changes to the configuration file. The system needs the admin DN and password of the replica server to complete this task.

- a. Select the check box next to **Add credential information on consumer**.
- b. Consumer admin DN: `cn=root`
- c. Consumer admin password: *<cn=root password on the secondary>*

Figure 5-34 on page 169 shows the filled in values.

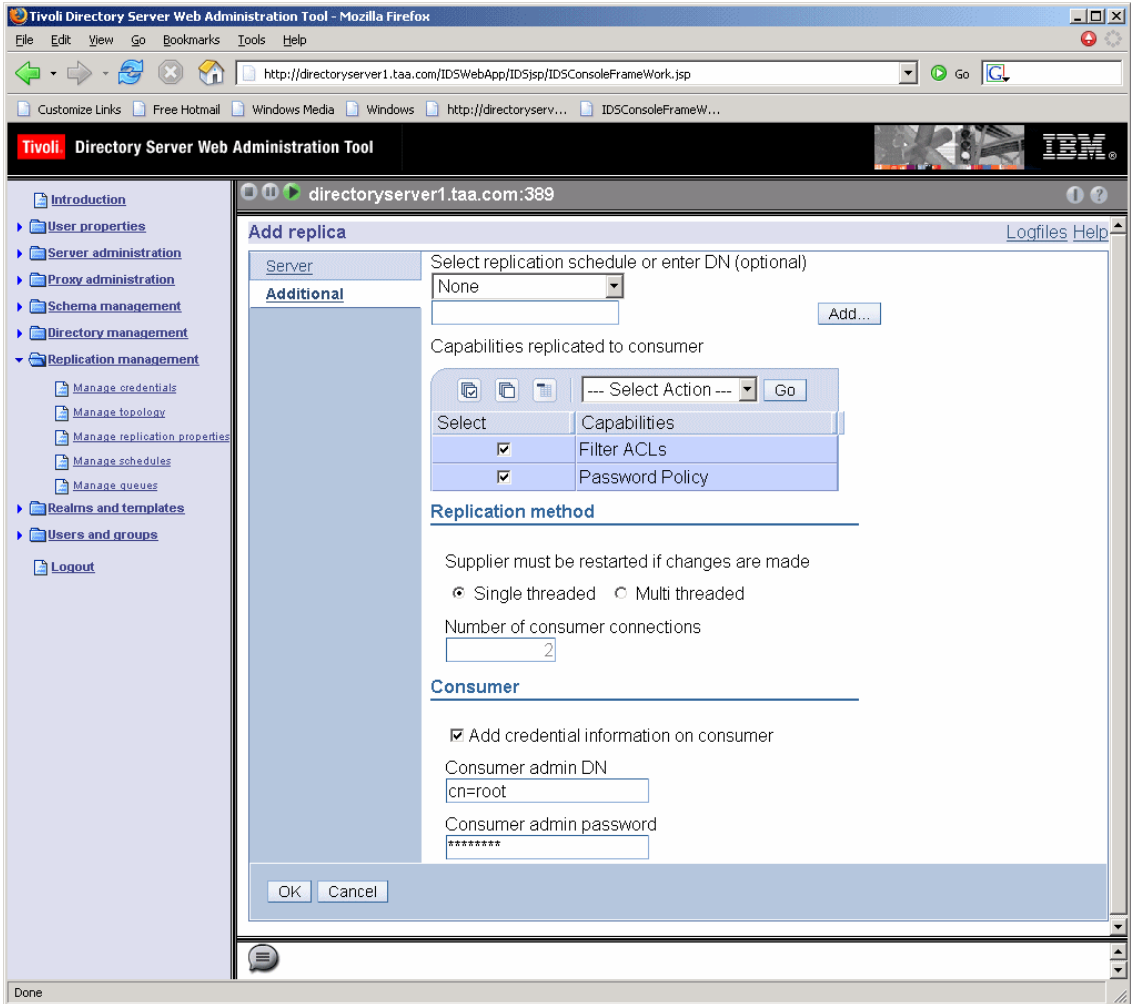


Figure 5-34 Configure credentials to add on the consumer machine

6. Click **OK** to continue.

The following output is depicted in Figure 5-35 on page 170.

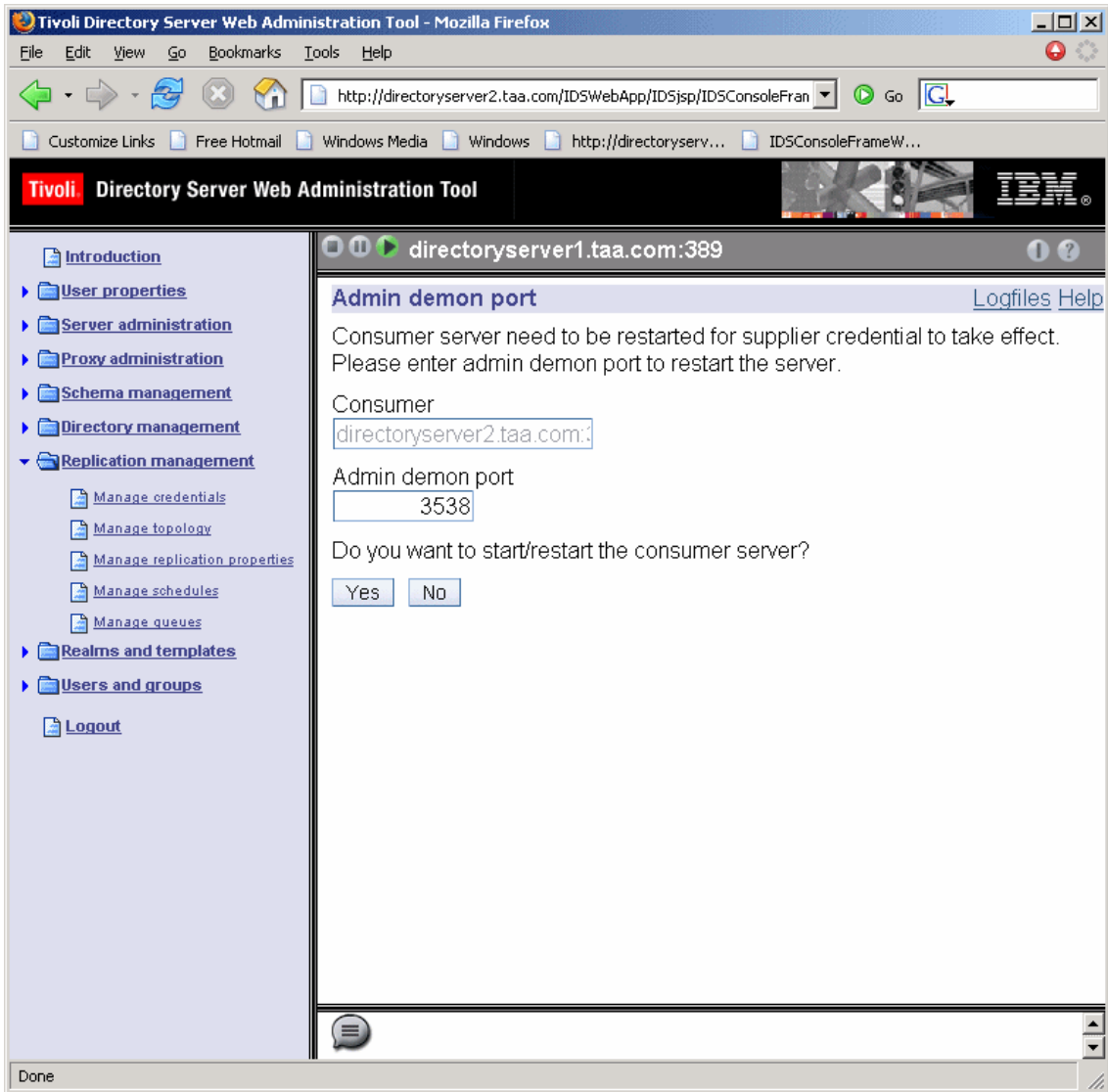


Figure 5-35 Add credentials on the consumer machine

7. Click **Yes** to restart the secondary server.

When the secondary server is restarted, the message in Figure 5-36 on page 171 displays.

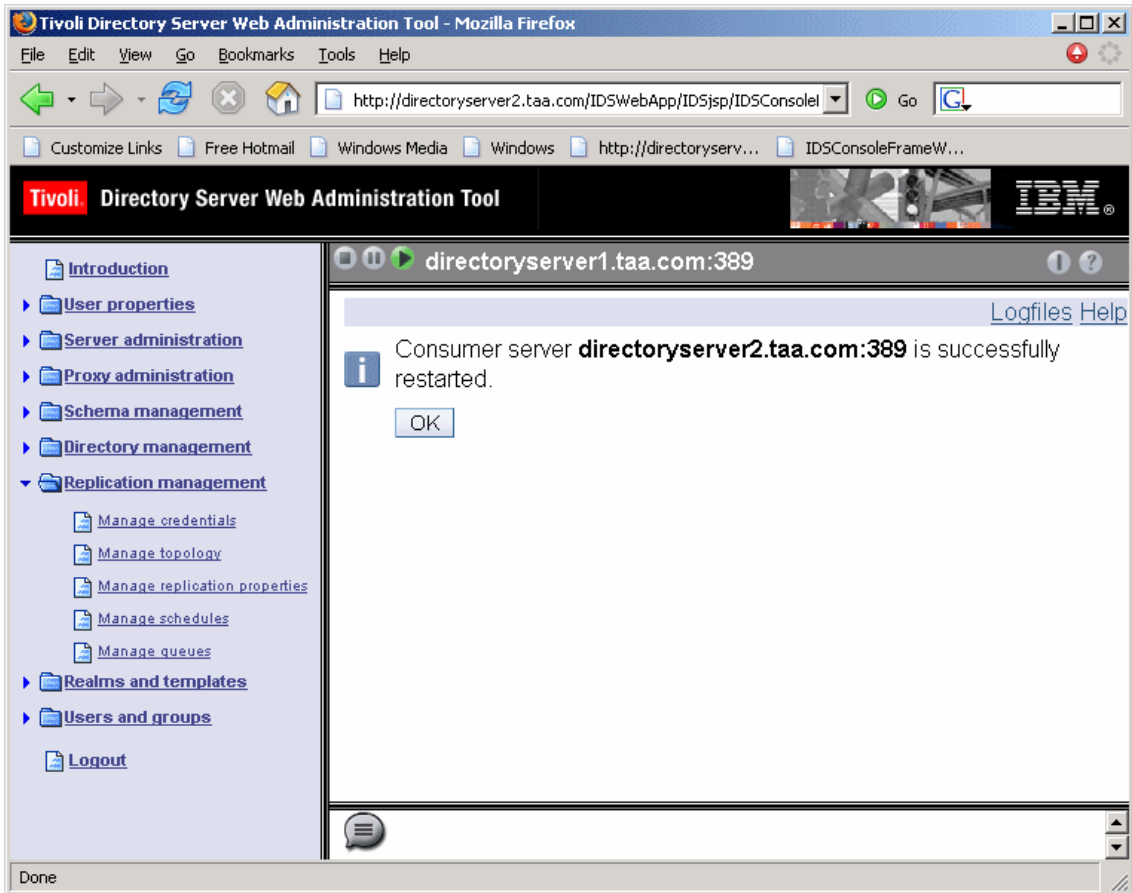


Figure 5-36 Credential successfully added on the consumer machine

8. Click **OK** to continue and to see the information window displayed in Figure 5-37 on page 172.

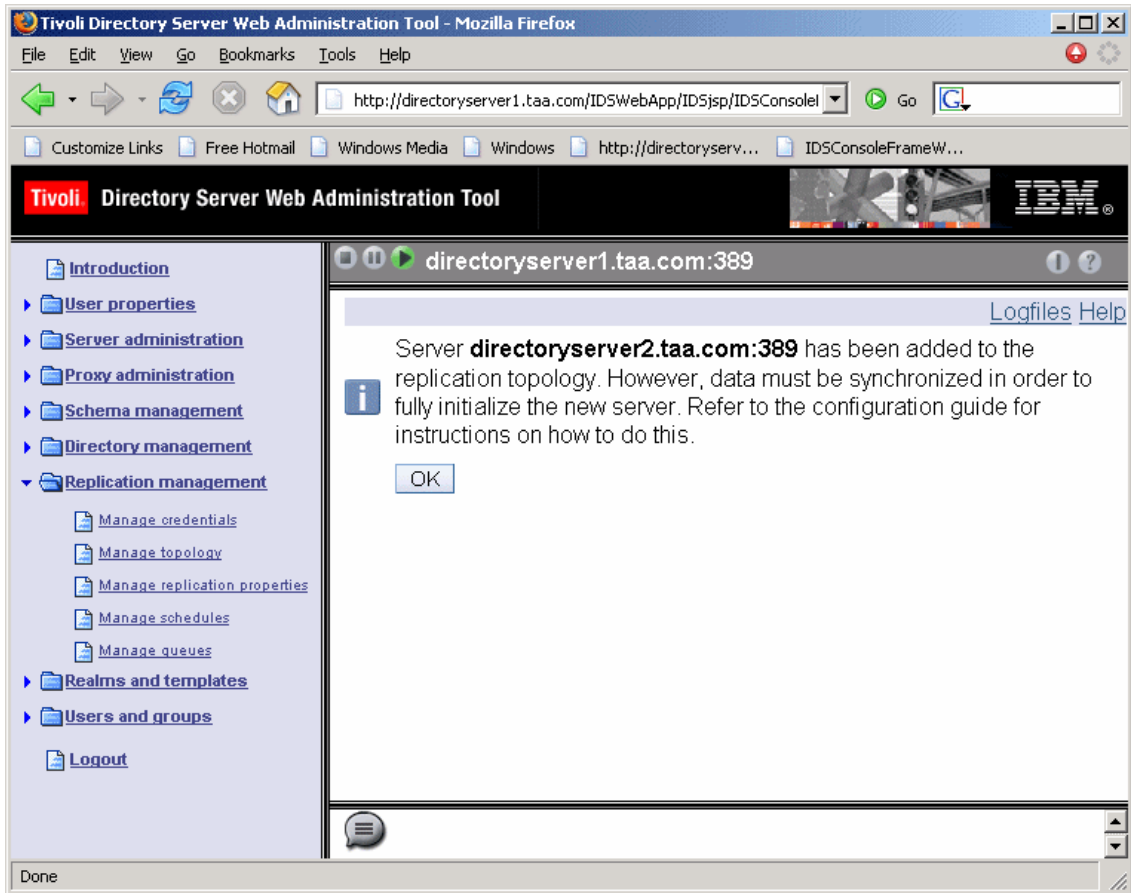


Figure 5-37 Directory Server replica added on replication topology configuration

9. On the Manage Topology window, depicted in Figure 5-38 on page 173, select the **DC=COM** subtree radio button, and click **Quiesce / Unquiesce**.
10. A window asking for confirmation to “quiesce the subtree DC=COM” is presented. Click **OK**.

The Manage Topology window shows that the DC=COM subtree is in a quiesced status.

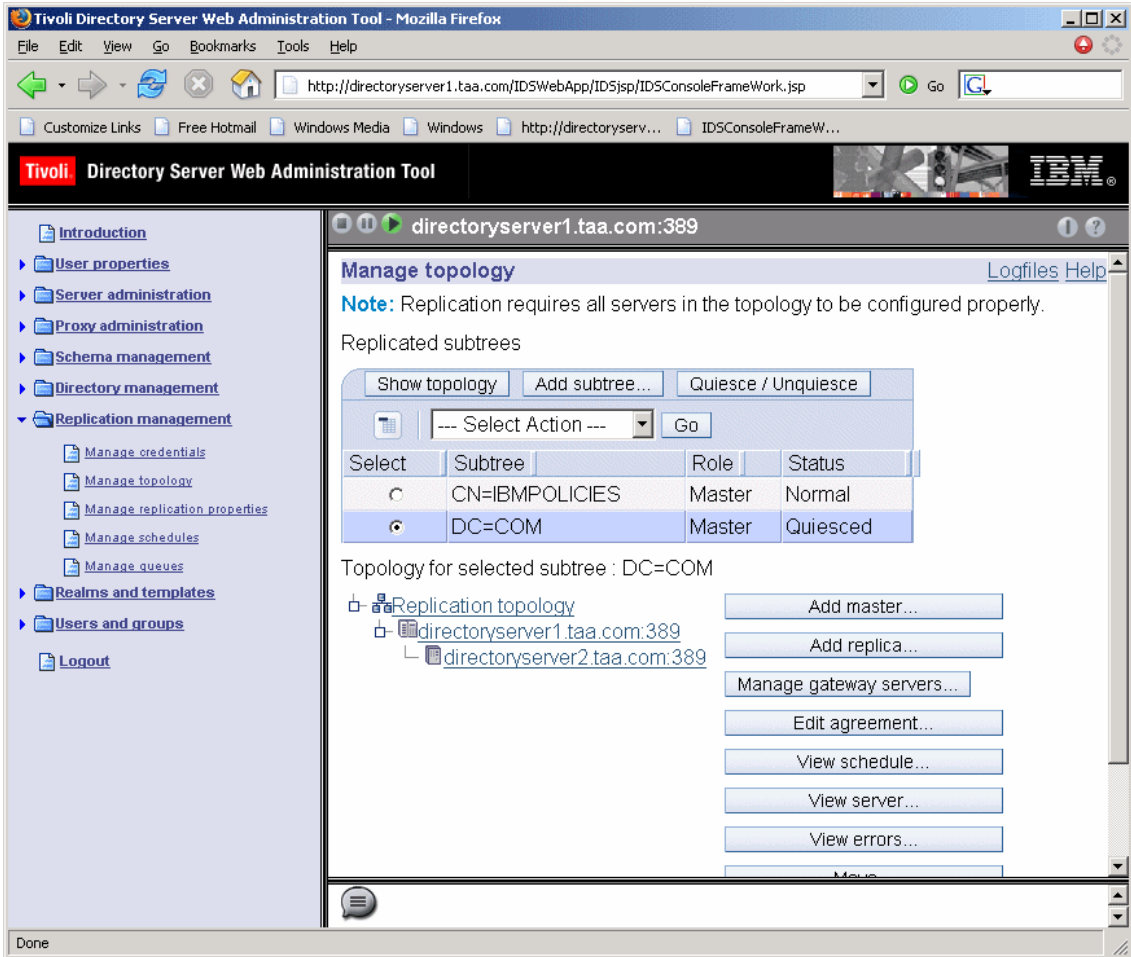


Figure 5-38 Manage topology view: master - replica topology

Note: The process of moving the data from the master server to the replica server is a manual process.

It is not an automated process because moving the data from the master server to the replica server using the normal replication process can be extremely time-consuming, since it is sending the data over the network and then processing the data.

It is a best practice to export the data from the master server, transfer it to the replica server (using ftp or another transfer protocol), and then load the data.

The `db2ldif` (`idsdb2ldif`) utility is used to export the data from the master server.

The `ldif2db` (`idsldif2db`) or `bulkload` utilities can be used to import the data into the replica server.

To continue building the Replica, the master server must be quiesced, so it does not accept any updates while the synchronization operation is in progress.

11. Synchronize the data from the master server to the Replica:

- a. To export the data from the primary Tivoli Directory Server, issue the following command at the command prompt:

```
idsdb2ldif -I idsldap1 -s dc=com -o /tmp/data.ldif
```

Where:

-o specifies the output file (in our sample, `/tmp/data.ldif`)

-I specifies the instance from which to export the data

-s specifies the suffix from which to export the data

Note: The secondary Tivoli Directory Server instance must be stopped for the following procedure.

- b. Load the data on the secondary Tivoli Directory Server by issuing the following command at the command prompt:

```
idsldif2db -i /tmp/data.ldif -I idsldap2 -r no
```

Where:

-i specifies the input file (in the sample, /tmp/data.ldif)

-I specifies the instance from which to import the data

-r specifies not to replicate the data as it loads

When the load is complete, the master server and the replica server will be in synch.

Note: The secondary Tivoli Directory Server instance must be restarted.

When the replica creation process completes, Tivoli Directory Server starts the replication queue in a suspended state. The queue can be restarted at this point so it can be tested.

12. To unsuspend the newly created replication queue, select **Manage queues**. The state should be suspended, as shown in Figure 5-39.

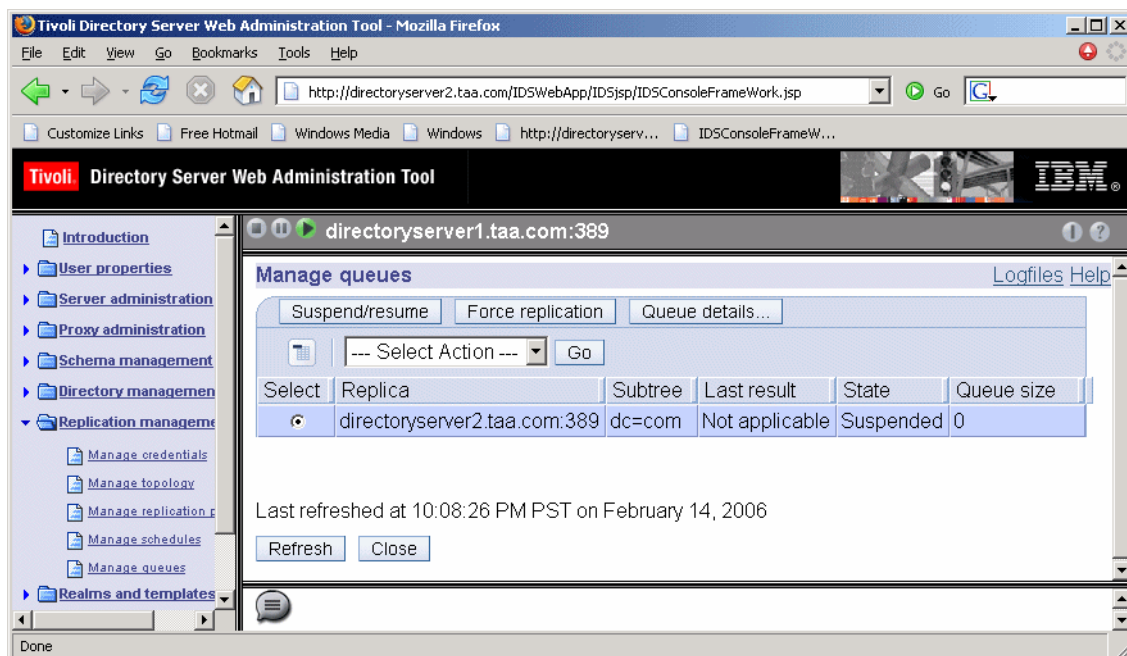


Figure 5-39 Manage queues panel after adding Directory Server replica

13. Click **Suspend/resume** to start the replication queue. The queue should display the Last result as OK and have a State showing Active if everything is correct. Click **Refresh** to make sure the queue shows zero, which is depicted in Figure 5-40.

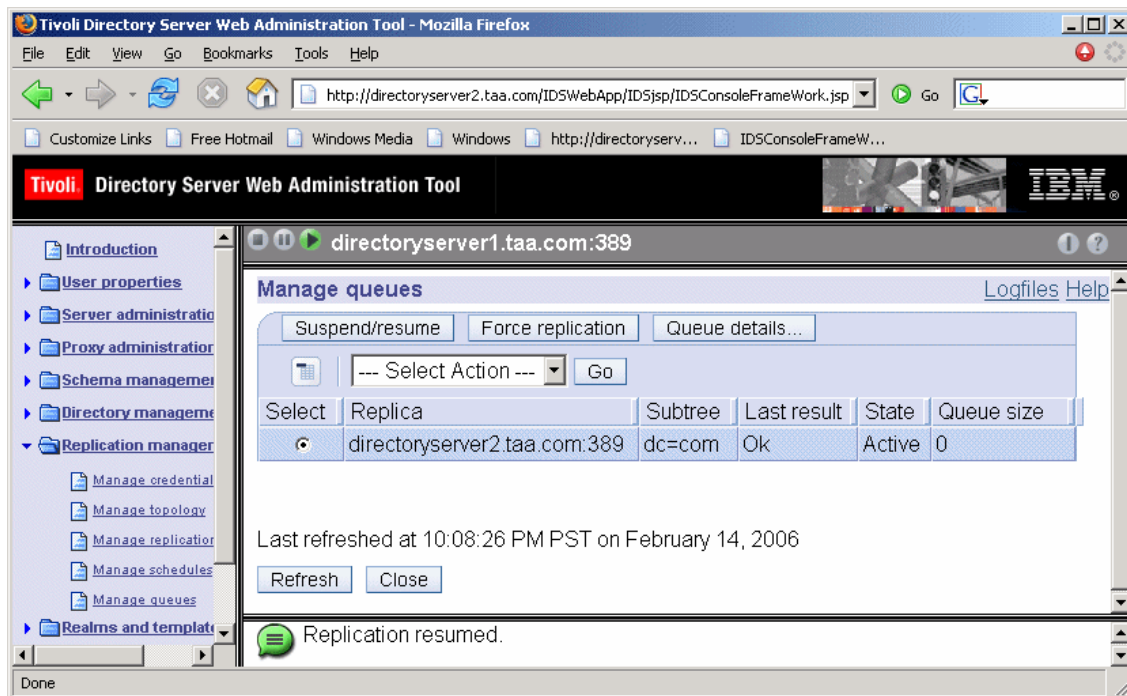


Figure 5-40 Manager queues view active state for queue on Directory Server replica

14. Click **Refresh** to ensure that the queue empties.
15. To further investigate, click **Queue details**. When on the Queue details window, select **Pending changes** to see if replication is flowing smoothly. If there are problems with the replication queue, it most likely has to do with problems in the username and password of the user credentials. Further information can be found in the `ibmslapd.log` file.

Test replication from Directory Server master to replica

Once all looks good with the replication queue, it is time to test. This test uses an object add, for instance an Organizational Unit add, from the Identity Manager console (Identity Manager is using the primary Directory Server instance). Then we return to the Manage Queue window in the Directory Server console to ensure the data was sent. If the data was sent, we examine the secondary

Directory Server instance to ensure the change was replicated to the directory server tree:

1. Log in to Tivoli Identity Manager as an administrator (for example, `itimanager`)

`http://itmsserver1.taa.com/enrole/logon`

2. Click **My Organization**.
3. Click **Manage Organizational Units**.
4. Click expand node **Tivoli Austin Airlines**.

You see the organizational unit branches already created under Tivoli Austin Airlines.

5. Click **Tivoli Austin Airlines branch** to select this organizational unit.
6. Click **Add**.
7. Insert the **Organizational Unit Name**, for example, `Test Organizational Unit`, as shown in Figure 5-41.

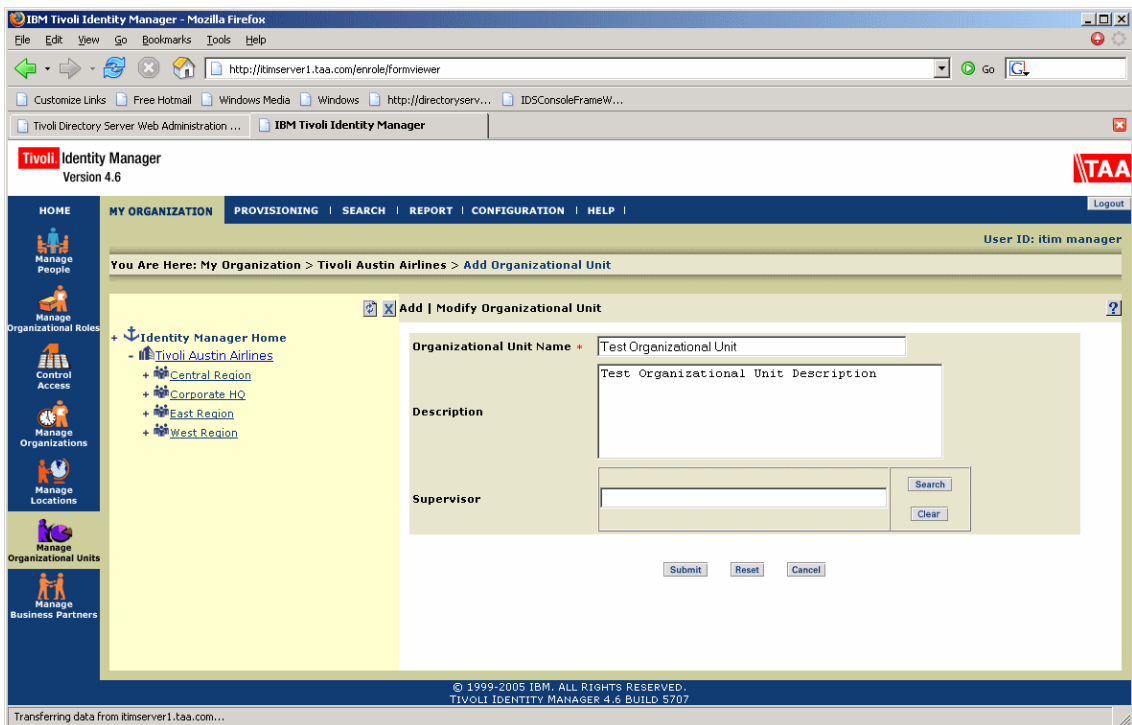


Figure 5-41 Create an Organizational Unit

8. Click **Submit**.

The new organizational unit appears on the organizational unit list.

Note: Tivoli Directory Server only replicates the attribute that has been changed rather than the entire directory entry. In this example, the new entry created on the primary directory server tree is replicated.

9. Log in to the Tivoli Directory Server Web Administration Tool on the primary Directory Server as an administrator (for example, cn=root).
`http://directoryserver1.taa.com/IDSWebApp/IDSjsp/Login.jsp`
10. Select **Replication management** and then select **Manage queues**.
11. Hopefully, the state of the queue is *Ready* with the last result *OK*. Click **Queue details**.
12. Select **Last attempted details**. The last attempted details show the DN information of the entry that was just added, depicted in Figure 5-42 on page 179.

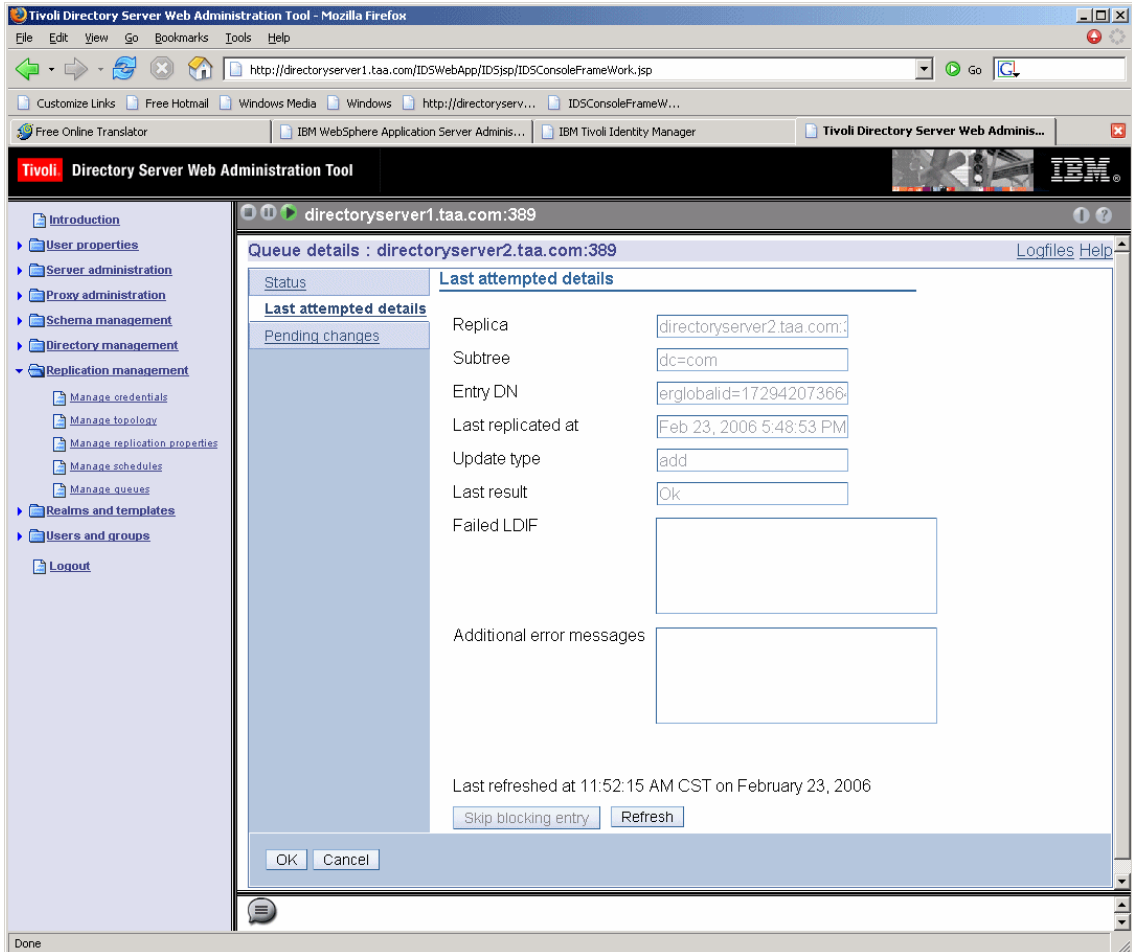


Figure 5-42 Replication queue status, last attempted details

13. Ensure that the change has been replicated to the replica by logging in to the Tivoli Directory Server Web Administration Tool on the replica Directory Server as an administrator (for example, cn=root).
14. Select **Directory management** and then select **Find entries**.
15. In the Search filter page, select **organizationalUnit** as objectclass, **ou** as Attribute.
16. Insert Test Organizational Unit in the search filter text box as shown in Figure 5-43 on page 180.

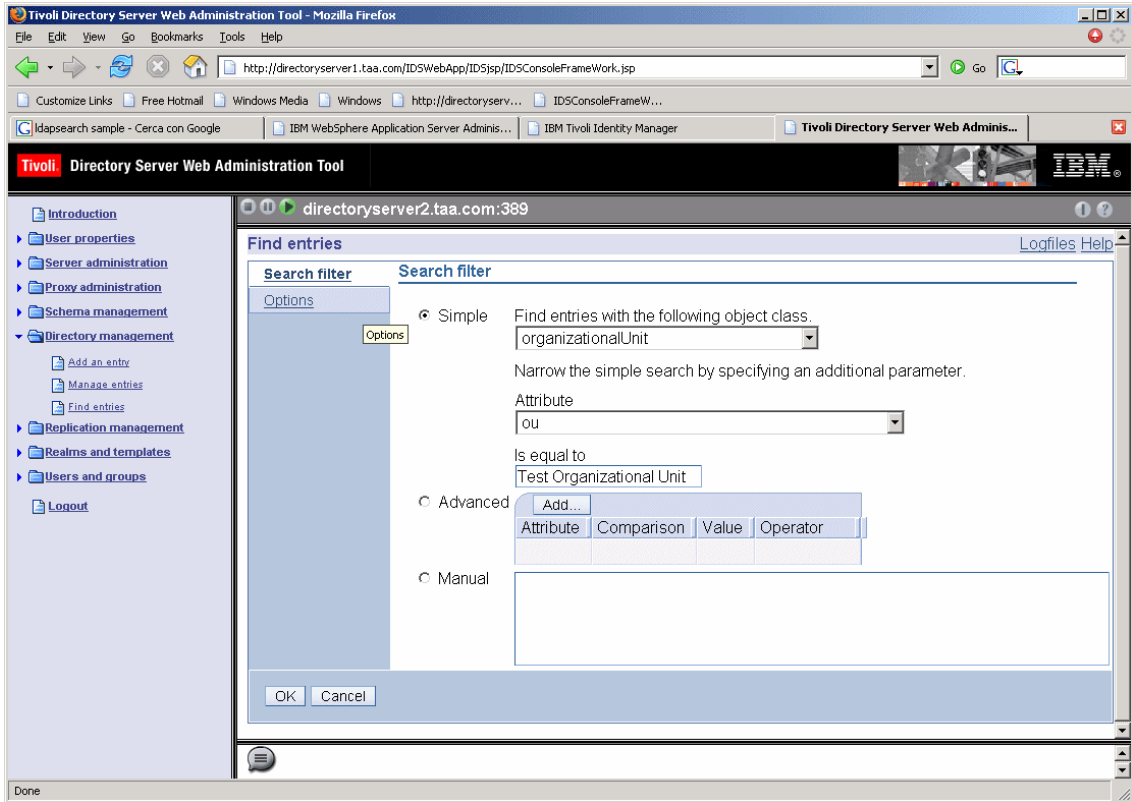


Figure 5-43 Search entry added on the Directory Server replica

17. Click **OK**.

18. The entry replicated on the secondary Directory Server is displayed in Figure 5-44 on page 181. Our replication process to the newly configured replica was successful.

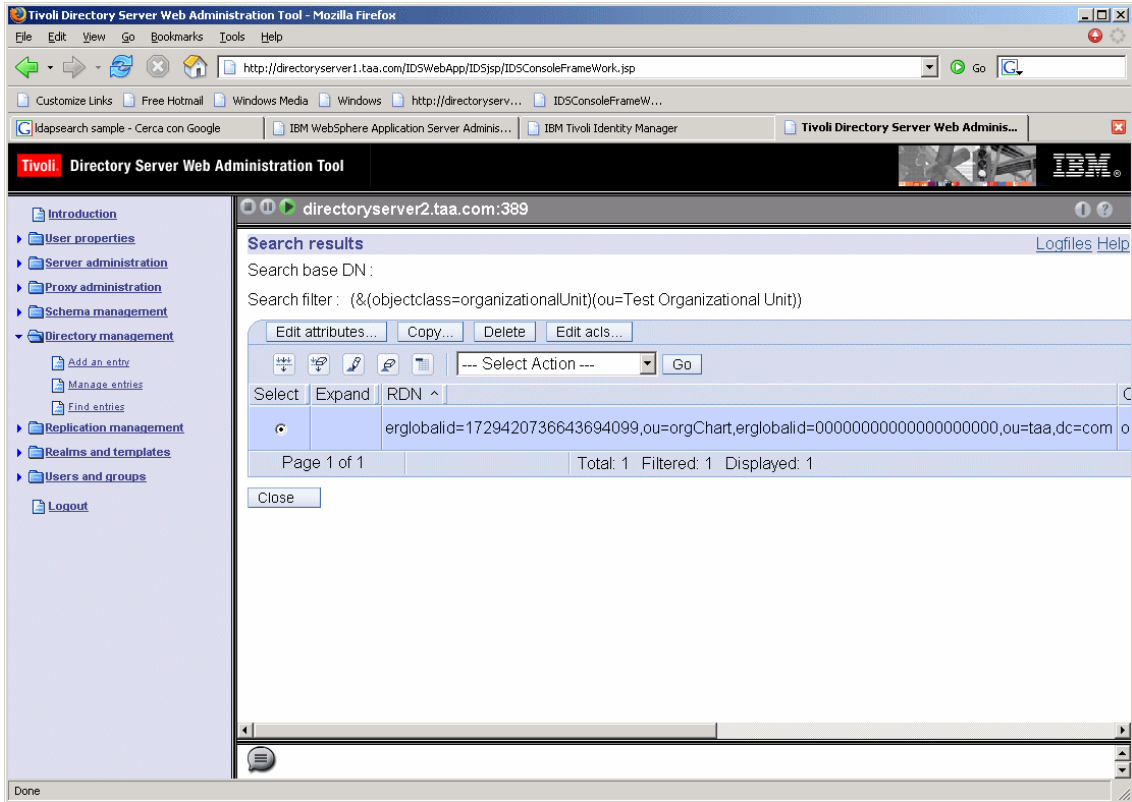


Figure 5-44 Find entry result

Promote replica server to a master server

This next phase involves promoting the replica on `directoryserver2.taa.com` to a master server. This allows both servers to be available for writes, providing a 24x7 availability.

The following steps are executed on the primary Tivoli Directory Server instance:

1. In the Tivoli Directory Server Web Administration tool, select **Replication management**, then select **Manage topology**.
2. With the **DC=COM** subtree radio button selected, click **Show topology**.
3. In the Topology for selected subtree section, expand the **directoryserver1.taa.com:389 topology** by clicking on the box to the left of the item.
4. Select **directoryserver2.taa.com:389**.
5. Click **Move**. The following window displays. See Figure 5-45 on page 182.

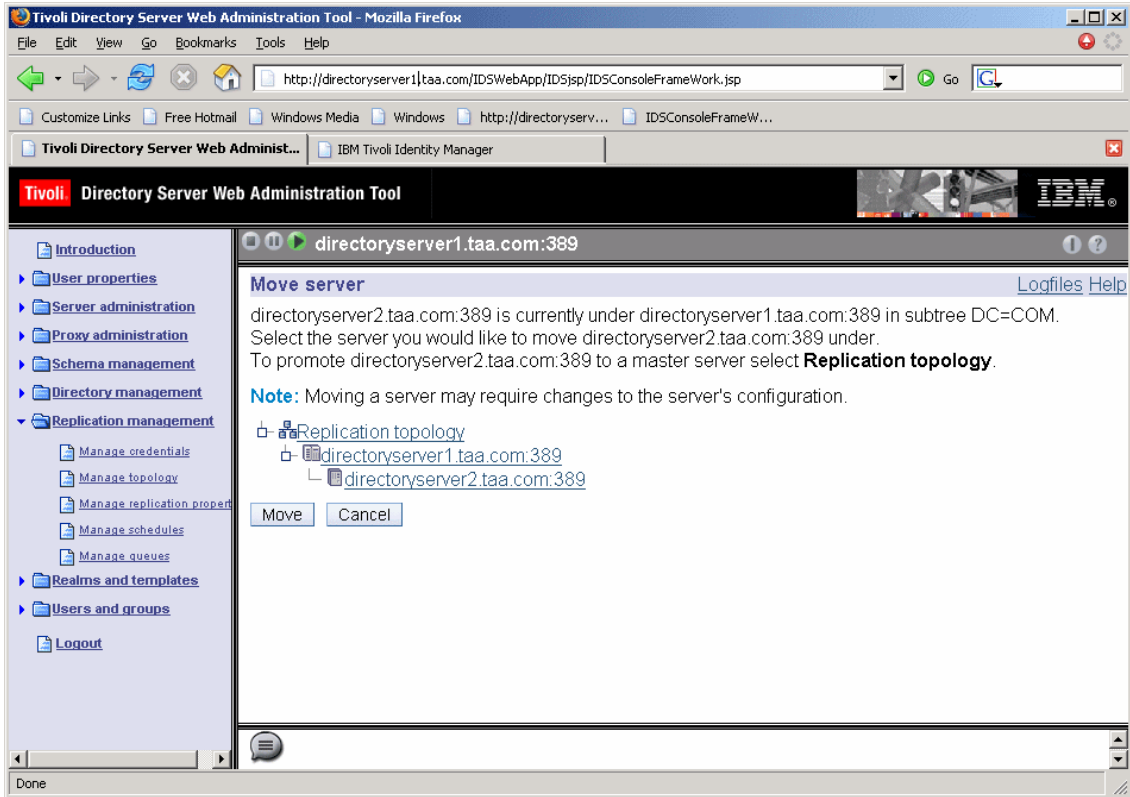


Figure 5-45 Move replica to master

6. Select **Replication topology** to move directoryserver2.taa.com to be a master server in the topology.
7. Click **Move**.
8. On the Create additional supplier agreements window, click **Continue**.
9. The message “The manage topology function will now collect information” displays. Click **OK**.

Configure credentials to be used for peer-to-peer replication

This step defines the credentials that are used during the replication process.

The following steps are executed on the primary Tivoli Directory Server instance:

1. With the **DC=COM** radio button selected, click **Show credentials**.
2. You can use the same credentials that were defined earlier. Using the box under the **Select credentials** section, select **replicamanager** from the list.

3. Add the login information for the consumer server, so the credentials can be added to the configuration file. Click **Add credential information on consumer** check box, and add the following credential information:

Consumer admin DN cn=root

Consumer admin password <consumer password on the secondary>

Figure 5-46 shows the details.

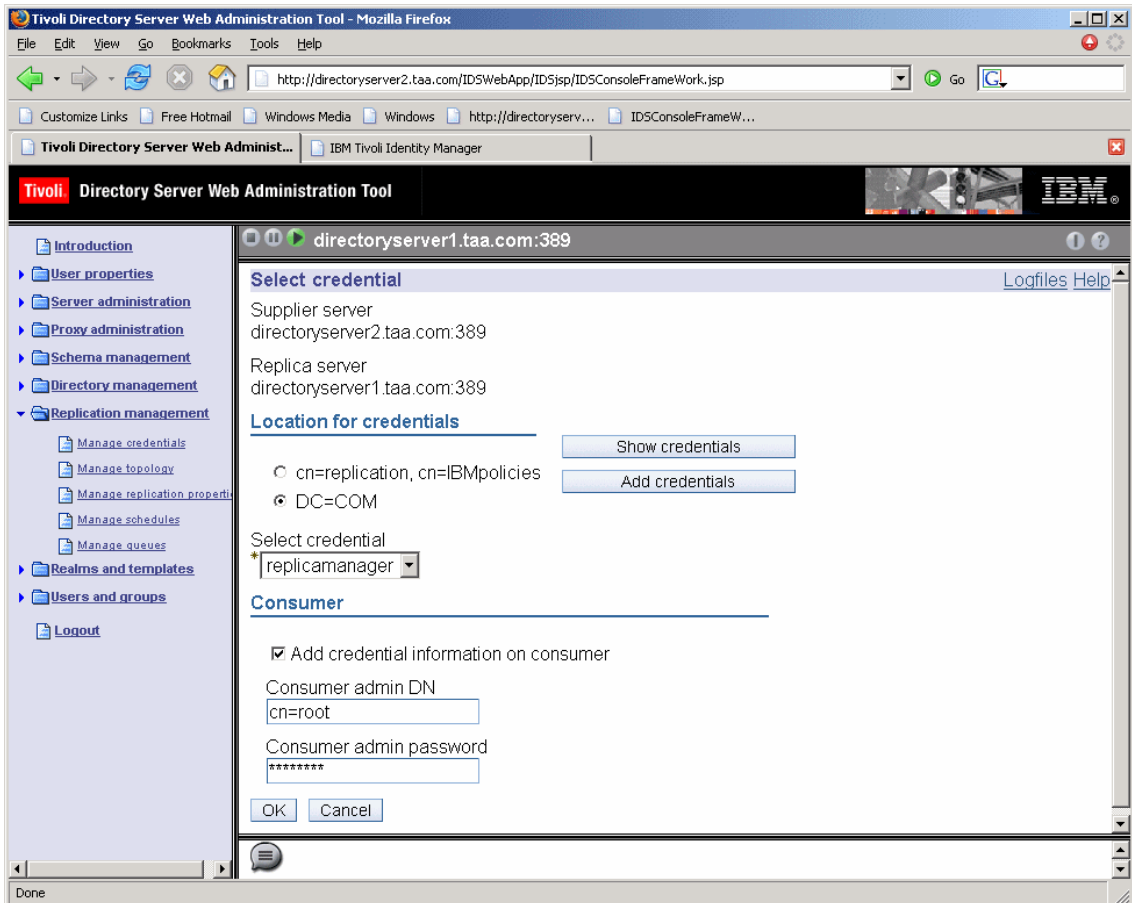


Figure 5-46 Configure credentials for peer-to-peer replication

4. Click **OK**.

A message is displayed saying that directoryserver1.taa.com must be restarted. This is done in the following step.

- The Replication topology window displays. Figure 5-47 shows the new replication topology, with directoryserver2.taa.com now at the same level in the topology as directoryserver2.taa.com.

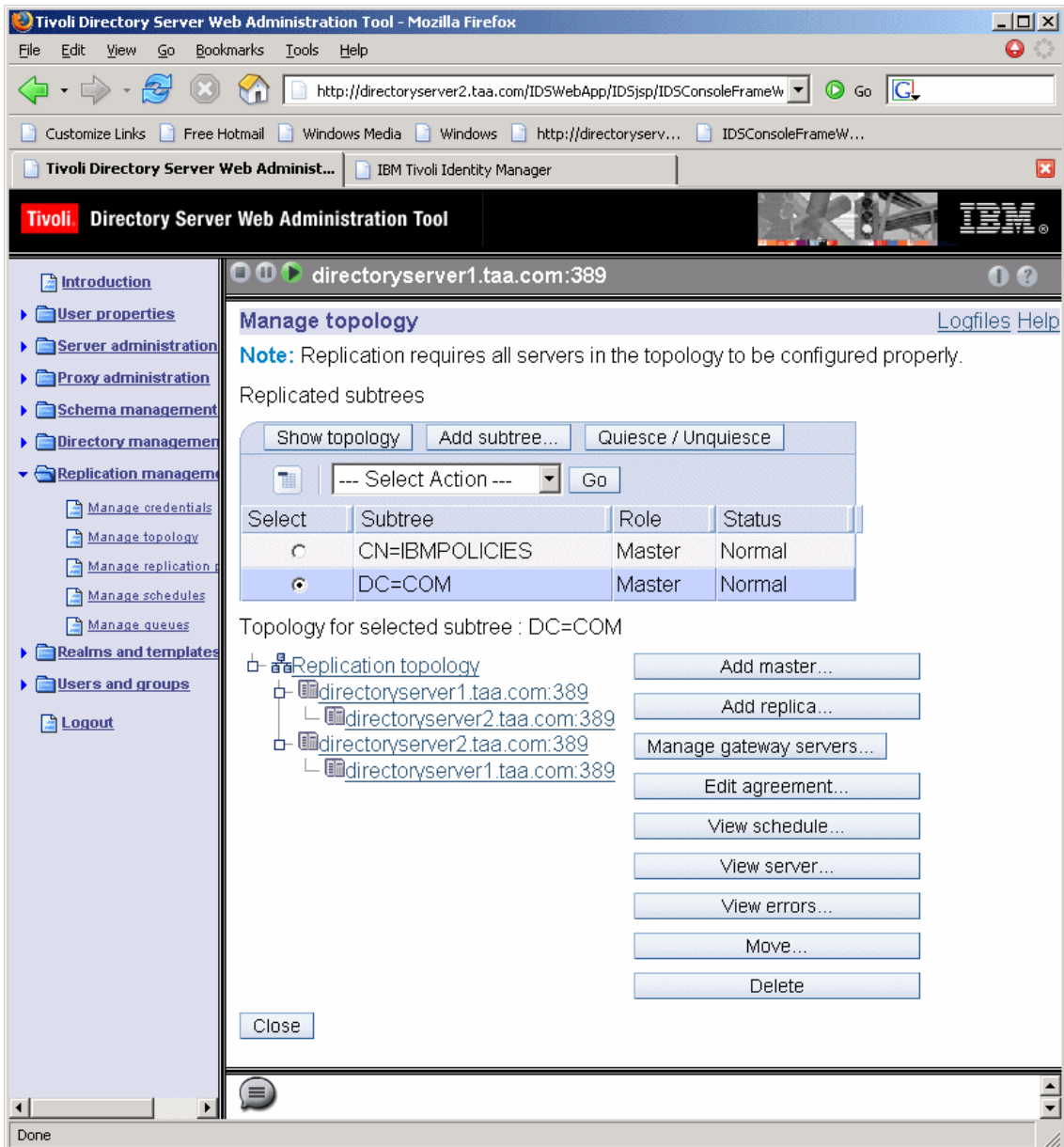


Figure 5-47 Replication peer-to-peer topology

6. Restart the primary Tivoli Directory Server instance.
When the replication is first started, Tivoli Directory Server starts the replication queue in a suspended state. The queue can be restarted at this point, so it can be tested.
7. Log out of the primary Tivoli Directory Server Web Administration Tool and log in to the secondary.
8. Select **Replication Management**.
9. Select **Manage queues**. The state should be *suspended*.
10. Select **Suspend/resume** to start the replication queue. The queue should show the Last result as *OK* or have an active state if everything is correct. Click **Refresh** to make sure the queue shows zero.

To further investigate, click **queue details** and **pending changes** to see if replication is flowing smoothly. If there are problems with the replication queue, it most likely has to do with problems in the username and password of the user credentials. Further information about the issue can be found in the `ibmslapd.log`.

Test peer-to-peer topology

Once all looks good with the replication queue, it is time to test the peer-to-peer replication sending an LDAP write operation on both the back-end directory servers. This test follows the same steps shown in “Test replication from Directory Server master to replica” on page 176 to add an object Organizational Unit on the `directoryserver1.taa.com`. This time the entry is deleted using the Web Administration console from `directoryserver2.taa.com`.

Configure Tivoli Directory Server proxy server component

Now we have two Directory Server instances configured on a peer-to-peer topology, as shown in Figure 5-48. Both the instances can be accessed in read/write mode from an LDAP client.

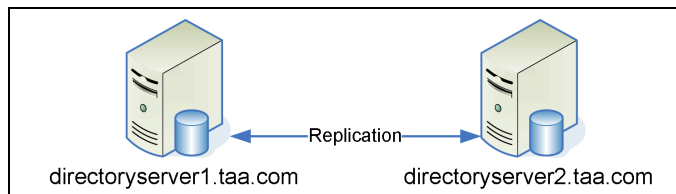


Figure 5-48 Peer-to-peer topology

To implement a Directory Server high availability environment, we need to add a component that can manage and balance the read/write requests to the peer-to-peer topology in failover mode. The Tivoli Directory Server proxy server

component will be used to manage the failover request on the Directory Server topology, as shown in Figure 5-49.

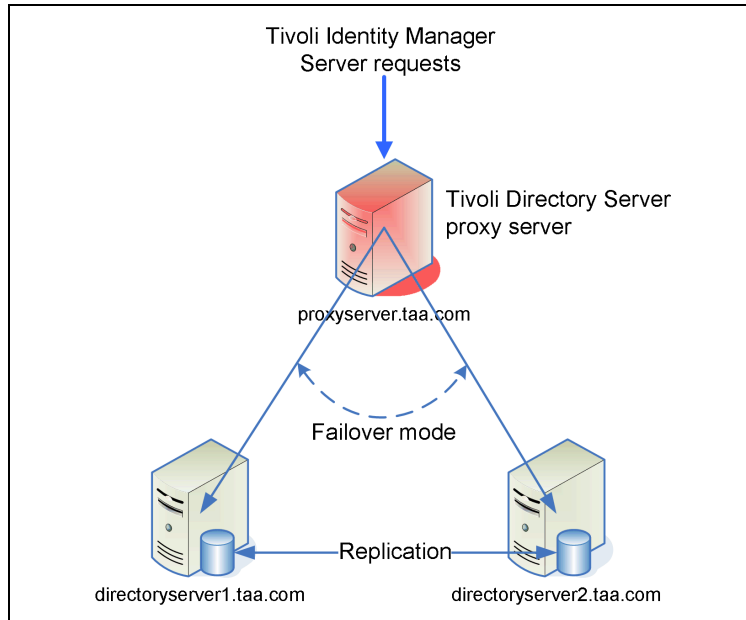


Figure 5-49 Proxy server as a load balancer configured in failover mode

The proxy server does not have an RDBM back-end and cannot take part in replication. The directory proxy server sits at the front end of a distributed directory and provides efficient routing of the Identity Manager Server requests. In order to obtain high availability, another server machine (in this scenario with hostname proxysvr.taa.com) has been added to the TAA IT environment.

Note: In a proxied directory, failover support between proxies is provided by creating an additional proxy server that is identical to the first proxy server. These are not the same as peer masters, the proxy servers have no knowledge of each other and must be managed through a third-party load balancer.

A load balancer, such as the IBM WebSphere Edge Server, is configured to send requests to only one proxy server. If that proxy server is down or unavailable because of a network or system failure, the load balancer sends the updates to the next available proxy server until the first server is back online and available.

Refer to your load balancer product documentation for information about how to install and configure the load balancing server.

This section demonstrates how to set up and configure a proxy server in a peer-to-peer topology. For general information about the Tivoli Directory Server proxy server component and different topology configurations, refer to the *IBM Tivoli Directory Server Administration Guide Version 6.1*, SC32-1564.

With IBM Tivoli Directory Server Version 6.1, you can install two types of servers: the full server and the proxy server. To install a proxy server, you do not need to have DB2 installed on the computer.

For the proxyserver.taa.com server machine, we use an xSeries® Linux and to install the Tivoli Directory Server proxy server, we use the Linux utilities.

Note: In our architectural and planning steps we are referring to IBM Tivoli Directory Server 6.1. However, at the time of writing this Redbooks publication we only had access to IBM Tivoli Directory Server 6.0. This is why the individual installation and configuration steps and screenshots may refer to v6.0. If you will be deploying v6.1 of the product please make sure you use the appropriate files and commands.

To install the proxy server, follow these steps:

1. Log in as root.
2. Install the 32-bit client by typing the following at a command prompt:

```
rpm -ihv idsldap-cltbase60-6.0.0-0.i386.rpm  
rpm -ihv idsldap-clt32bit60-6.0.0-0.i386.rpm
```

3. Install the proxy server component by typing the following at a command prompt:

```
rpm -ihv idsldap-cltjava60-6.0.0-0.i386.rpm  
rpm -ihv idsldap-srvproxy32bit60-6.0.0-0.i386.rpm
```

4. Verify that the packages have been installed correctly by typing the following at a command prompt:

```
rpm -qa | grep idsldap
```

If the product has been successfully installed, the following is displayed:

```
idsldap-cltbase60-6.0.0-0  
idsldap-clt32bit60-6.0.0-0  
idsldap-cltjava60-6.0.0-0  
idsldap-srvproxy32bit60-6.0.0-0
```

5. Install the English messages:

```
rpm -ihv idsldap-msg60-en-6.0.0-0.i386.rpm
```

You can install messages in other languages by using the package names for those languages.

6. If you want to include security functions, install GSKit 7.0.3.3 by typing the following at a command prompt:

```
rpm -ihv gsk7bas-7.0-3.3.i386.rpm
```

To configure the proxy server instance, we use the Instance Administration Tool:

1. Start the Instance Administration Tool by typing **idsxinst** at the command line. The IBM Tivoli Directory Server Instance Administration Tool window is displayed.
2. Click **Create**.
3. On the Create a new directory server instance window, click **Create a new directory server instance**.
Click **Next**.
4. On the Instance details window, shown in Figure 5-50 on page 189, complete the following fields:

User name	Type the system user ID of the user who owns the Directory Server instance. The name you enter is also used as the name for the Directory Server instance.
------------------	--

Note: The name of the new Directory Server instance must be unique; if there is already a Directory Server instance on the computer with the same name, you will receive an error message.

Install location	Leave blank, the user home directory is used as a default.
Encryption seed string	Type a string of characters that is used as an encryption seed. Use the same encryption seed string used on the back-end directory server.
Instance description	Type a description of the proxy server instance, for example, Tivoli Directory Server proxy server.

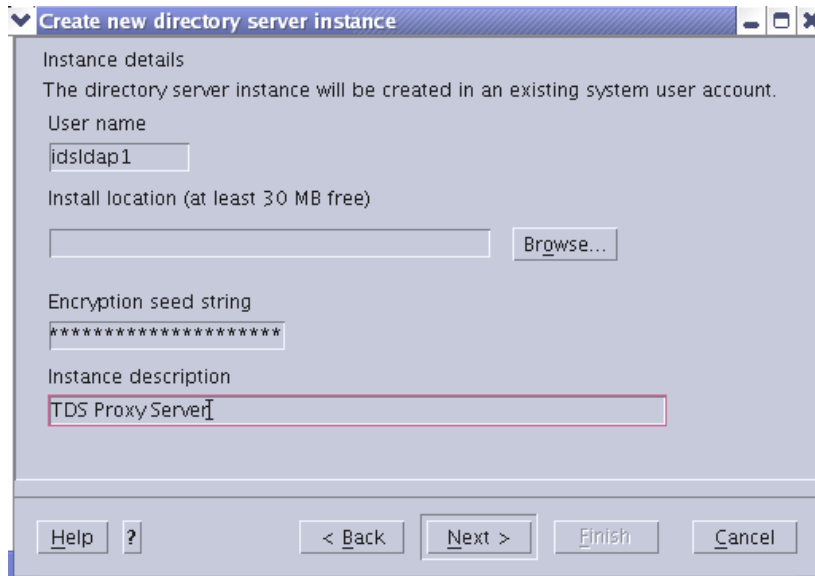


Figure 5-50 Create a new proxy server instance

5. Click **Next**.
6. On the TCP/IP settings for multihomed hosts window, select the **Listen on all configured IP addresses** check box and click **Next** as depicted in Figure 5-51.

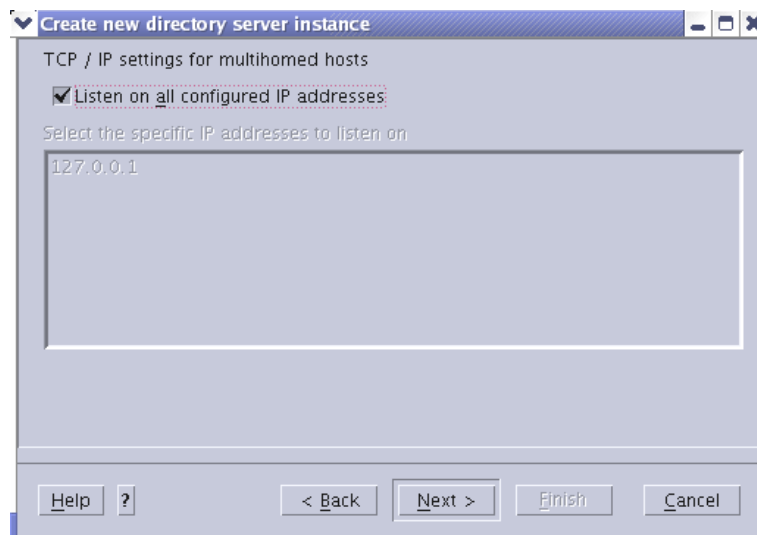


Figure 5-51 Proxy server instance to listen on all configured IP addresses

7. On the TCP/IP port settings window, leave the port numbers prefilled by the Tivoli Directory Server Instance Administration tool as shown in Figure 5-52 and click **Next**.

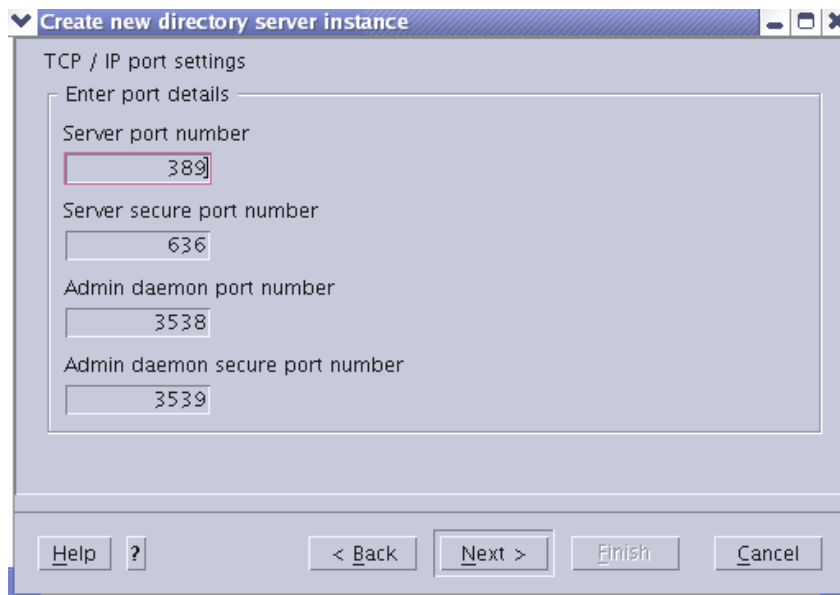


Figure 5-52 Proxy server TCP/IP port settings

Note: If you have two or more Directory Server instances listening on the same IP address (or set of IP addresses), be sure that those Directory Server instances do not use any of the same port numbers.

8. Select the **Configure administrator DN and password** check box and click **Next**:
 - a. Accept the default DN (cn=root).
 - b. Type the password for the administrator DN in the **Administrator Password** field.

Passwords are case sensitive. Double-byte character set (DBCS) characters in the password field are not valid. Record the password in a secure location for future reference.
 - c. Retype the password in the **Confirm password** field.
 - d. Click **Next**.

9. In the Verify settings window, the complete information displays again. To return to an earlier dialog and change information, click **Back**. To begin creating the proxy server instance, click **Finish**.

Before we set up the proxy server and the back-end servers, we need to create an entry on the global administration group. This entry is used as the administrative user that binds the LDAP request from the Identity Manager Server to the proxy server.

The global administration group is a way for the directory administrator to delegate administrative rights in a distributed environment to the database back-end. Global administrative group members are users who have been assigned the same set of privileges as the administrative group with regard to access entries in the database back-end and have complete access to the Directory Server back-end. All global administrative group members have the same set of privileges.

Global administrative group members cannot access schema data. They also do not have access to the audit log. Local administrators, therefore, can use the audit log to monitor global administrative group member activity for security purposes.

Note: The global administration group should be used by applications or administrators to communicate with the proxy server using administrative credentials. For example, the member that will be set up on this implementation sample (*cn=manager,cn=ibmpolicies*) will be used in place of the local administrator (*cn=root*) when directory entries are to be modified through the proxy server. Binding to the proxy server as *cn=root* gives an administrator full access to the proxy server's configuration but only anonymous access to the directory entries.

To set up the proxy server and back-end server, we use the Tivoli Directory Server Web administration console.

Create a user entry for the global administrators group

To create a user entry for the global administrators group, follow these steps and refer to Figure 5-53 on page 192:

1. Log on to the **directoryserver1.taa.com** Web administration console.
2. From the navigation area, expand the **Directory management** link.
3. Click **Add an entry**.
4. From the **Structural object class** drop-down menu, select **person**.
5. Click **Next**.

6. Click **Next** to skip the **Select auxiliary object** classes panel.
7. Type `cn=manager` in the **Relative DN** field.
8. Type `cn=ibmpolicies` in the **Parent DN** field.
9. Type `manager` in the **cn** field.
10. Type `manager` in the **sn** field.

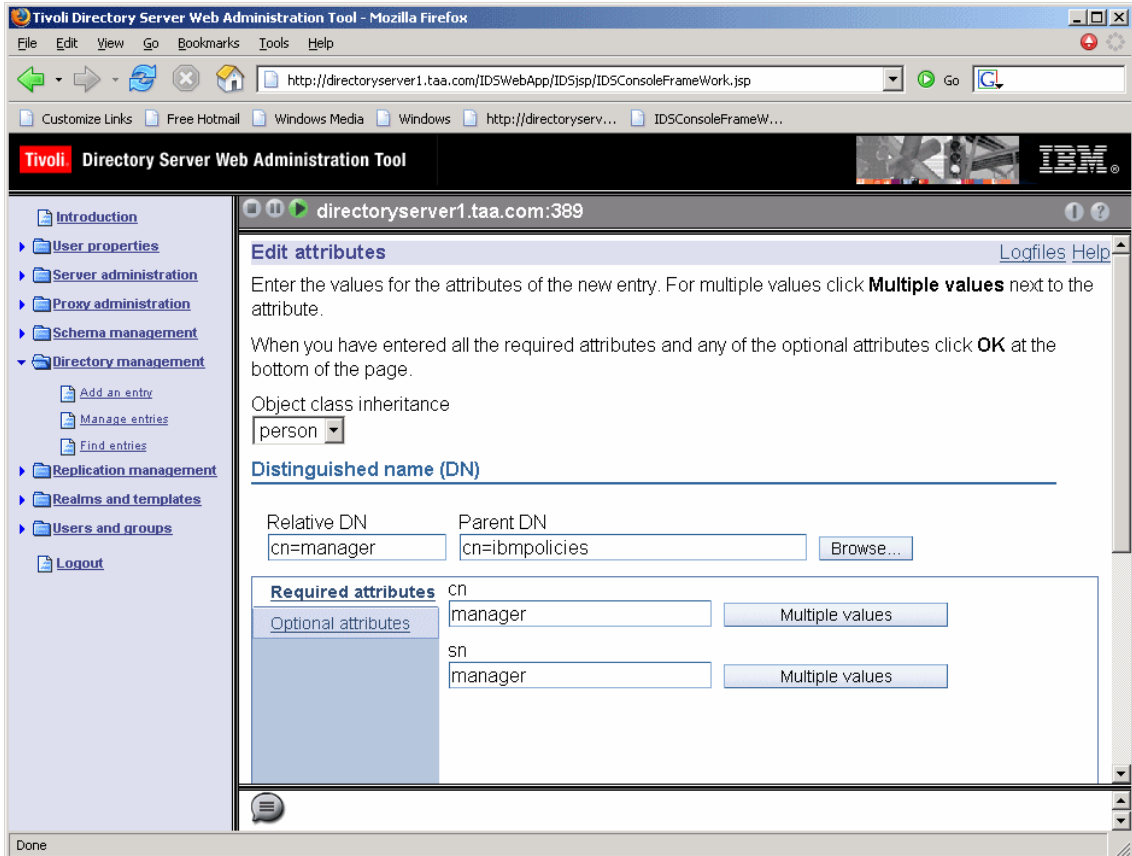


Figure 5-53 User entry for the global administration group

11. Click the **Optional attributes** link.
12. Enter a password in the **userPassword** field.
13. Click **Finish**.
14. Repeat steps 1 through 13 to create the same user entry on the second back-end Directory Server (`direcotryserver2.taa.com`).

Add the user entry to the global administrators group

The following steps add the `cn=manager` to the global administration group as depicted in Figure 5-54:

1. Log on to **directoryserver1.taa.com** Web administration console.
2. From the navigation area, expand the **Directory management** link.
3. Click **Manage entries**.
4. Select the radio button for **cn=ibmpolicies** and click **Expand**.
5. Select the radio button for **globalGroupName=GlobalAdminGroup** and from the Select action drop-down menu, select **Manage members** and click **Go**.
6. Type `cn=manager,cn=ibmpolicies` in the member field and click **Add**.

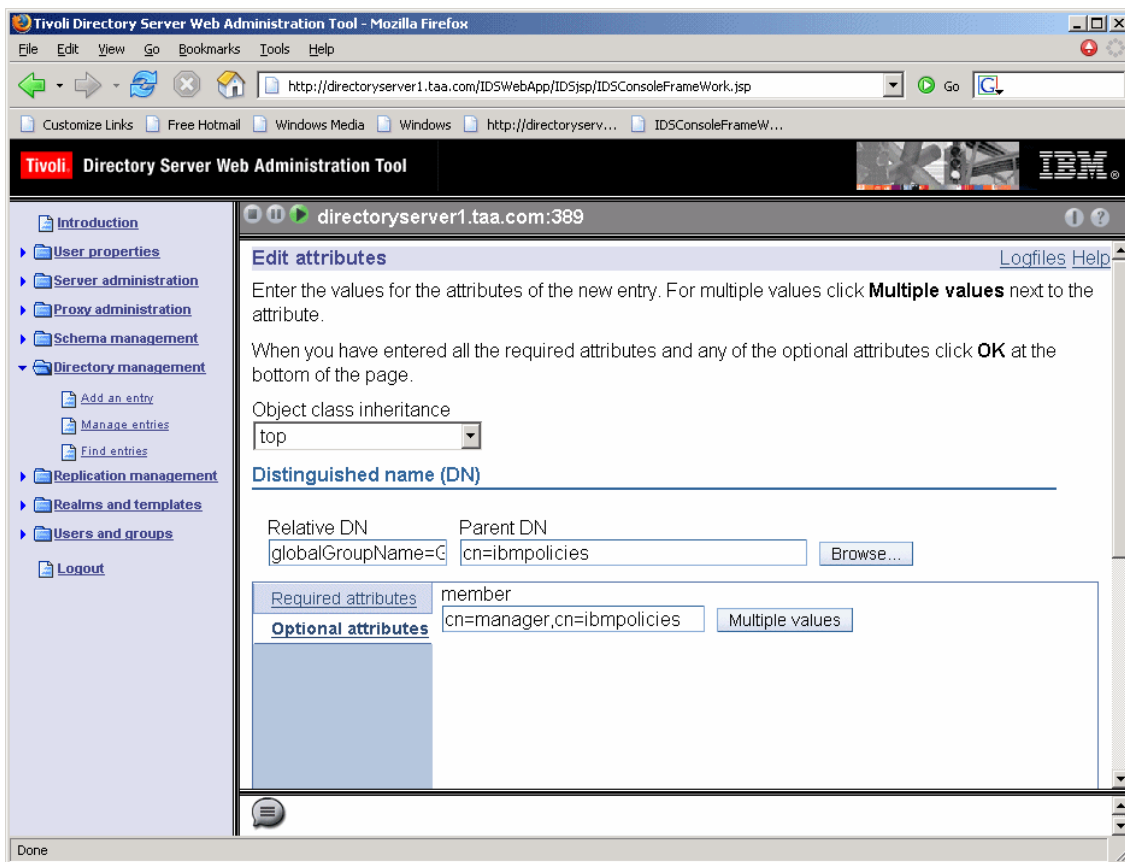


Figure 5-54 Add the `cn=manager` user entry into the global administration group

7. A message displays, “You have not loaded entries from the server”. Only your changes will be displayed in the table. Answer the question “Do you want to continue?” by clicking **OK**.
8. **cn=manager** is displayed in the table. Click **OK**.
The user **cn=manager** is now a member of the global administration group.
9. Repeat the steps 1 through 8 to add the **cn=manager** user entry on the global administration group on the second back-end directory server (direcotryserver2.taa.com).

Setting up the proxy server

In the following section, we configure the proxy server instance and use both the Web administration console and the LDAP command line to define the server group necessary to manage the Directory Server peer-to-peer topology.

Note: In this release, the Web Administration Tool does not support the management of these server groups. The directory administrator must define these server groups using the **idsldapadd** and **idsldapmodify** commands to add and modify the required entries.

To set up the proxy server:

1. Log on to the server that you are going to use as the proxy server, **proxyserver.taa.com**, as **root** user.
2. Start the server in configuration-only mode using the command:

```
idsslapd -I <instance_name> -a
```


In this example, the instance name is **idsldap1**.
3. Log on to the **proxyserver.taa.com** Web administration console as **cn=root**.
4. From the navigation area, expand the **Proxy administration** link as shown in Figure 5-55 on page 195.
5. Click **Manage proxy properties**.
6. Click the **Configure as proxy server** check box.
7. In the **Suffix DN** field, enter **cn=ibmpolicies** and click **Add**.
8. In the **Suffix DN** field, enter **cn=pwdpolicy** and click **Add**.
9. In the **Suffix DN** field, enter **dc=com** and click **Add**.

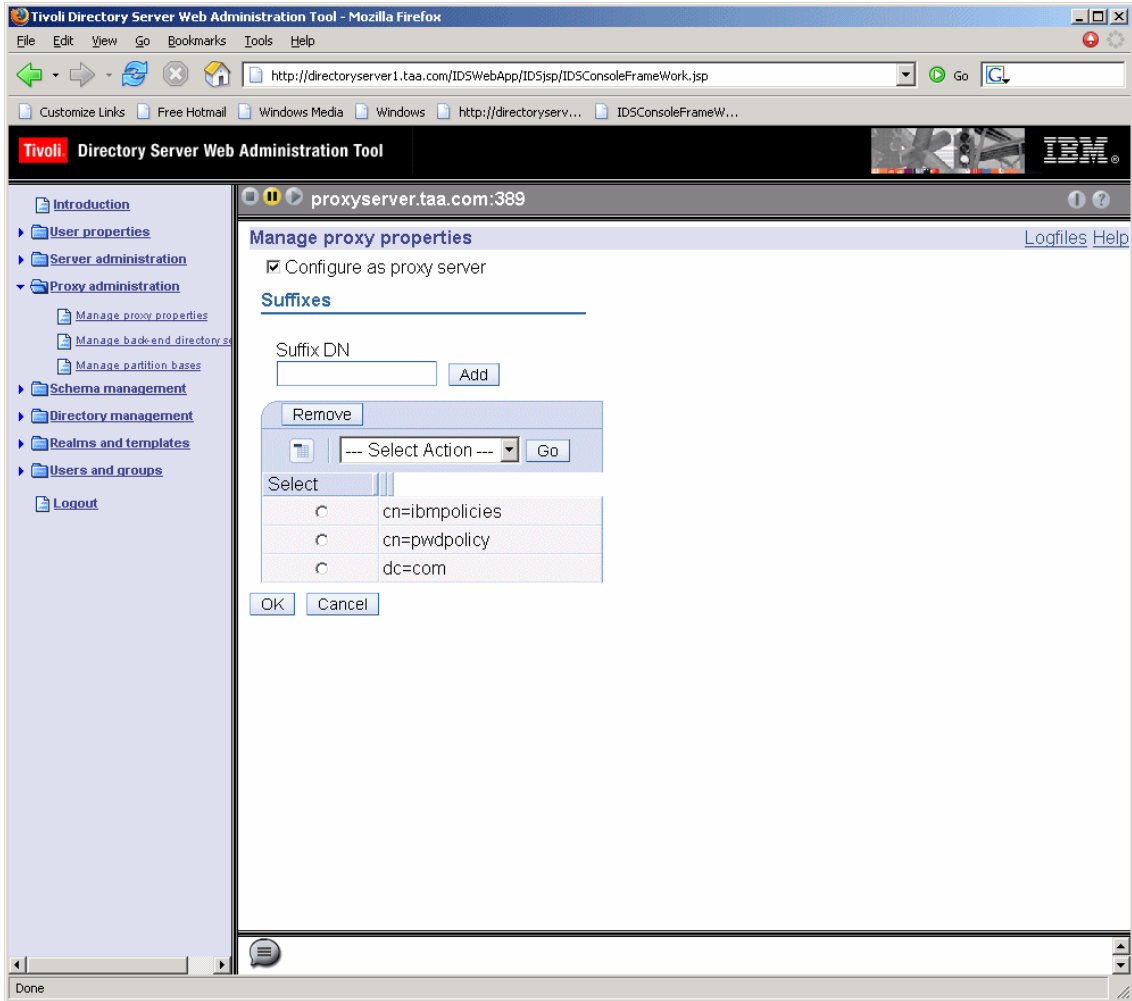


Figure 5-55 Manage proxy properties

10. Click **OK** to save your changes and return to the Introduction panel.

Note: You must log off the Web Administration, and log in again. Doing so updates the navigation area. If you do not log off and then log on again, the navigation area is not updated to correctly to display the proxy server.

11. From the navigation area, click **Manage back-end directory servers** as shown in Figure 5-56 on page 197.

12. Click **Add**.

13. Enter the host name for **directoryserver1.taa.com** in the **Hostname** field.
14. Enter the port number for **directoryserver1.taa.com** (for this example, all servers use 389).
15. Enter the number of connections that the proxy server can have with the back-end server in the **Connection pool size** field.

The minimum value for this Tivoli Directory Server release is 5 and the maximum value is 100.

Note: For this release, do not set the value in the **Connection pool size** field to be less than 5.

16. Specify **Simple** in the **Authentication method** field.

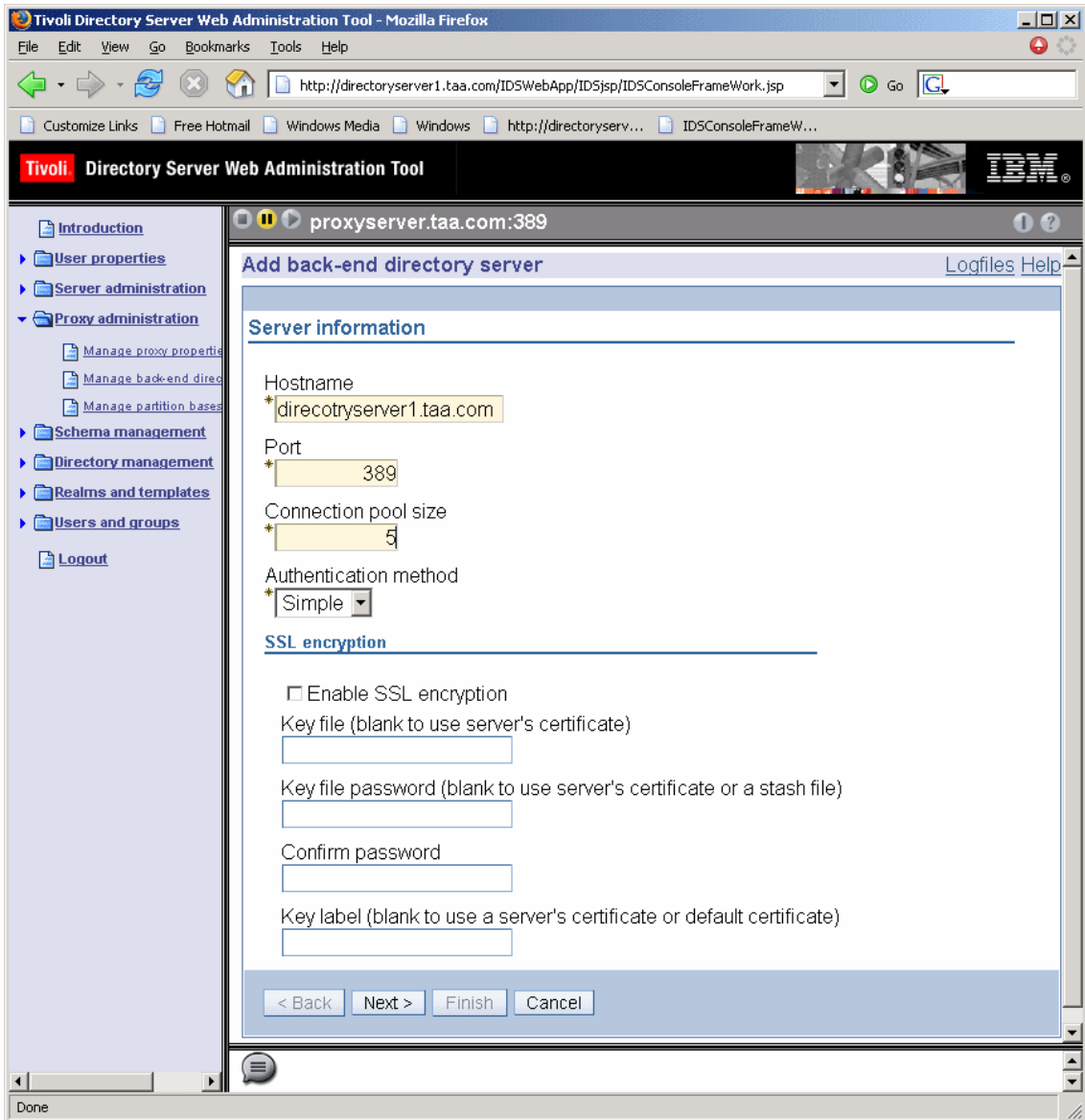


Figure 5-56 Add back-end Directory Server to proxy server

17. Click **Next**.

18. Specify the administration DN or the DN of a member of the local administrator in the **Bind DN** field. For this example, **cn=root**.

19. Specify and confirm the administration password, in the **Bind password** fields.
20. Click **Finish**.
21. Repeat steps 11 through 20 for directoryserver2.taa.com.

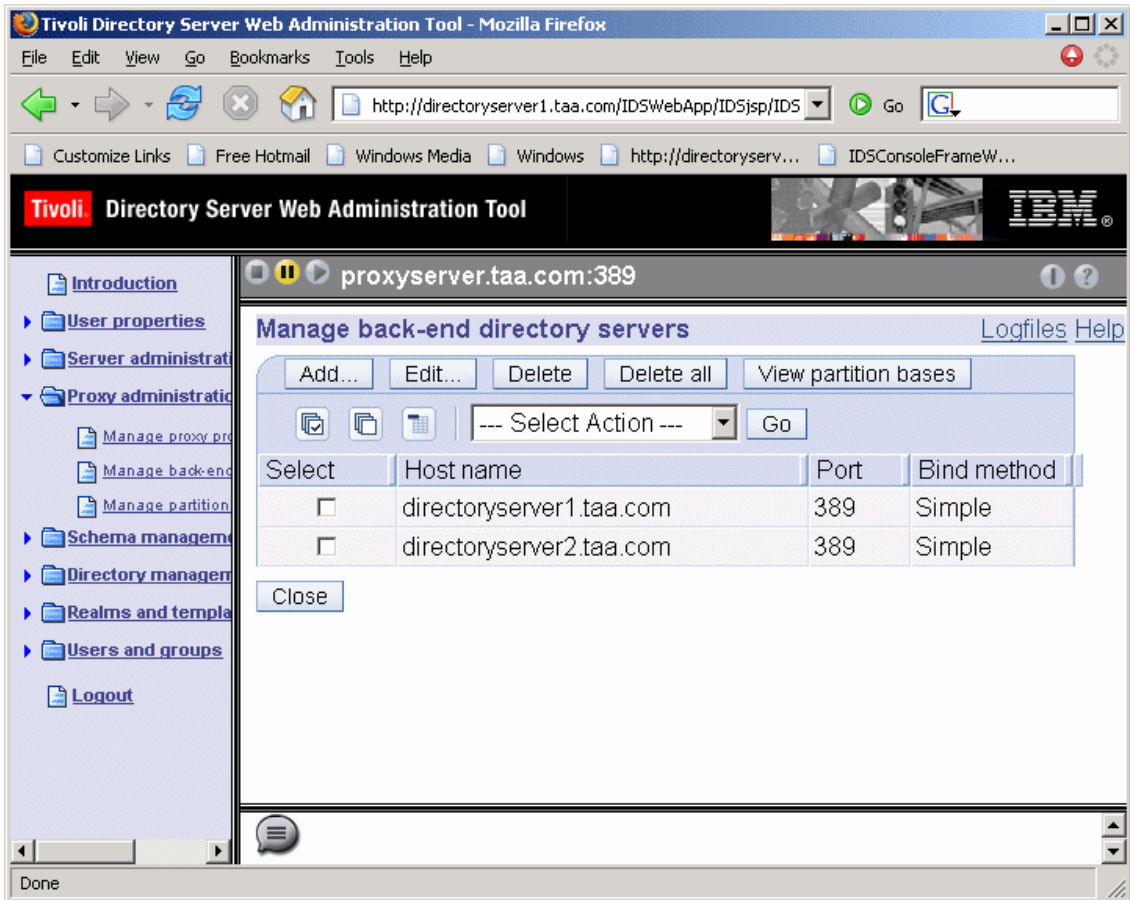


Figure 5-57 Manage back-end directory server summary view

22. When you finish, click **Close** to save your changes and return to the Introduction panel.
23. Ensure that all the back-end servers are started and can be displayed as shown in Figure 5-57.

Note: If the proxy server cannot connect with one or more of the back-end servers at startup, the proxy starts in configuration mode only. This is true unless you set up server groups. We explain how to configure a server group in section “Server groups” on page 200.

Synchronizing global policies

The following steps configure *cn=ibmpolicies* as a single partition. This is necessary to enable you to synchronize the global policies on all of the servers.

Note: Schema modifications are not replicated by the proxy server or to the proxy server. You need to enter any schema updates on each proxy server manually.

To configure *cn=ibmpolicies* as a single partition:

1. Log on to the proxyserver.taa.com Web administration console as *cn=root*.
2. From the navigation area, click **Manage partition bases** as depicted in Figure 5-58 on page 200.
3. On the Partition bases table, click **Add**.
4. Enter *cn=ibmpolicies* in the **Partition DN** field.
5. Enter 1 in the **Number of partitions** field.

Note: A value greater than 1 for *cn=ibmpolicies* and *cn=pwdpolicy* is not supported.

6. Click **OK**.
7. Select the radio button for **cn=ibmpolicies** and click **View servers**.
8. Verify that **cn=ibmpolicies** is displayed in the **Partition base DN** field.
9. In the **Back-end directory servers for partition base** table, click **Add**.
10. From the Back-end directory server menu, select **directoryserver1.taa.com**.
11. Enter 1 in the **Partition index** field.
12. Repeat steps 9 through 11 for **directoryserver2.taa.com**.

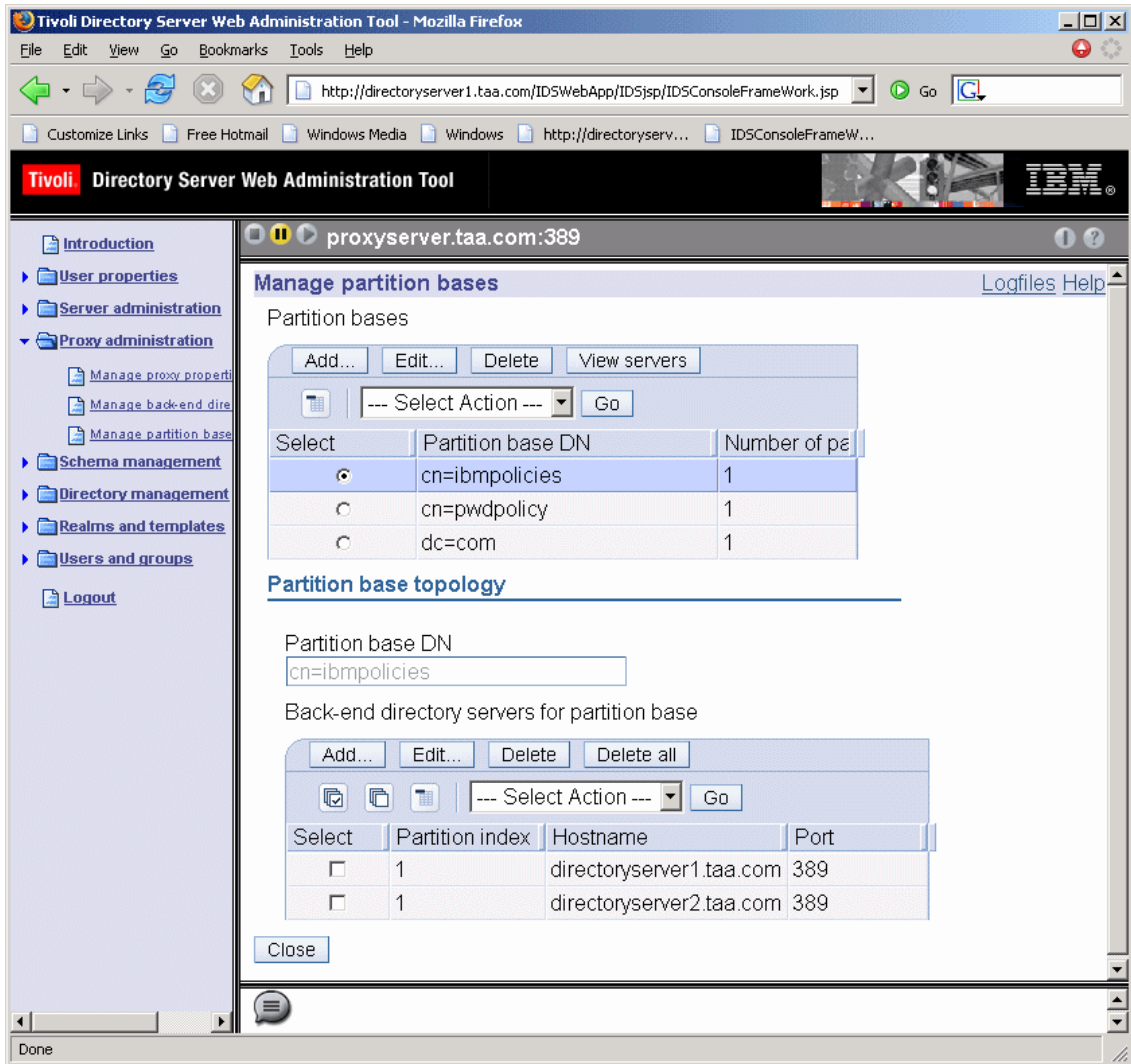


Figure 5-58 Manage partition bases for proxy server

13. Click **OK**.

14. Repeat steps 2 through 12 for *cn=pwdpolicy* and *dc=com*.

Server groups

If the proxy server is unable to contact a back-end server, or if the authentication fails, then the proxy server startup fails. Then, the proxy server starts in configuration only mode by default, unless server groupings have been defined in the configuration file.

Server groups enable the user to state that several back-end servers are mirrors of each other, and that proxy server processing can continue even if one or more back-end servers in the group are down, assuming that at least one back-end server is online. Connections are restarted periodically if the connections are closed for some reason, such as the remote server is stopped or restarted.

The proxy server configuration file supports a special set of entries that enables a directory administrator to define server groups in the configuration file. Each group contains a list of back-end servers. As long as at least one back-end server in each group can be contacted, the proxy server starts successfully and services client requests, although performance might be degraded. Each back-end server in the entry is defined to have an OR relationship, and all the entries have an AND relationship.

Note: With Tivoli Directory Server Version 6.1, the Web Administration Tool does not support the management of these server groups. The directory administrator must define these server groups using the `idsldapadd` command and the `idsldapmodify` command to add and modify the required entries.

The directory administrator must ensure that each of the back-end servers is placed in a server group and that the back-end servers in each server group contain the same partition of the directory database.

Example 5-2 shows typical user-defined server groups.

Example 5-2 Server group definition for two back-end Directory Servers

```
dn: cn=serverGroup, cn=ProxyDB, cn=Proxy Backends, cn=IBM Directory,
cn=Schemas, cn=Configuration
cn: serverGroup
ibm-slapdProxyBackendServerDN: cn=Server1,cn=ProxyDB,cn=Proxy Backends, cn=IBM
Directory, cn=Schemas,cn=Configuration
ibm-slapdProxyBackendServerDN: cn=Server2,cn=ProxyDB,cn=Proxy Backends, cn=IBM
Directory, cn=Schemas,cn=Configuration
objectclass: top
objectclass: ibm-slapdConfigEntry
objectclass: ibm-slapdProxyBackendServerGroup
```

In addition to the server grouping, the administrator must add the serverID of each back-end server in the server group entry. If the server is down, no root DSE information can be gained, and the serverID is needed for determining the supplier/consumer relationships throughout the topology.

Note: In each entry pointed to by *ibm-slapdProxyBackendServerDn*, the attribute *ibm-slapdServerId* must be added, with its value identical to the value on the corresponding back-end server.

Example 5-3 shows the definition of the two back-end Directory Servers on the proxy server configuration file *ibmslapd.conf* for TAA.

Example 5-3 TAA back-end Directory Server definition

```
dn: cn=Server1, cn=ProxyDB, cn=Proxy Backends, cn=IBM Directory, cn=Schemas,
cn=Configuration
cn: Server1
ibm-slapdProxyBindMethod: Simple
ibm-slapdproxyconnectionpoolsize: 5
ibm-slapdProxyDN: cn=root
ibm-slapdProxyPW: {AES256}Xie9CxheU8HGipCOE448gA==
ibm-slapdProxyTargetURL: ldap://directoryserver1.taa.com:389
ibm-slapdServerId: adc88840-2ee0-102a-8c1e-9d0bd4943fe3
objectClass: top
objectClass: ibm-slapdProxyBackendServer
objectClass: ibm-slapdConfigEntry
```

```
dn: cn=Server2, cn=ProxyDB, cn=Proxy Backends, cn=IBM Directory, cn=Schemas,
cn=Configuration
cn: Server2
ibm-slapdProxyBindMethod: Simple
ibm-slapdProxyConnectionPoolSize: 5
ibm-slapdProxyDN: cn=root
ibm-slapdProxyPW: {AES256}Xie9CxheU8HGipCOE448gA==
ibm-slapdProxyTargetURL: ldap://directoryserver2.taa.com:389
ibm-slapdServerId: 955ab540-31e1-102a-8f6e-d2ea857d8100
objectClass: top
objectClass: ibm-slapdProxyBackendServer
objectClass: ibm-slapdConfigEntry
```

Note: The *ibm-slapdServerId* value shown in Example 5-3 cannot be added in entries by using the Web Administration console.

When you are finished with the proxy server configuration, you have to stop and restart the proxy server:

1. Stop the proxy server using the command:

```
idsslapd -I <instance_name> -k
```

In this example, the instance name is `idsldap1`.

2. Start the proxy server using the command:

```
idsslapd -I <instance_name>
```

If the two back-end Directory Servers are up and running, an output similar to Example 5-4 displays. An information message displays the communication established with the back-end server and the server group startup.

Example 5-4 Proxy server starting output

```
[root@proxydemo.taa.com /]# idsslapd -I idsldap1
GLPSRV041I Server starting.
...
GLPCOM003I Non-SSL port initialized to 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver1.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver2.taa.com on port 389.
The server group cn=serverGroup, cn=ProxyDB, cn=Proxy Backends, cn=IBM
Directory, cn=Schemas, cn=Configuration is available. Proxy Server startup
continuing.
GLPPXY003I Successfully established communication with backend server
directoryserver2.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver2.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver2.taa.com on port 389.
GLPPXY003I Successfully established communication with backend server
directoryserver2.taa.com on port 389.
...
GLPSRV009I IBM Tivoli Directory (SSL), 6.0      Server started.
```

Configure Tivoli Identity Manager Server to use proxy server

Once all looks good with the proxy configuration, it is time to change the administrator ID used by our Identity Manager Server to bind to the user registry through the proxy server. For example, the ID we have configured, *cn=manager,cn=ibmpolicies*, will be used in place of the local administrator, *cn=root*, in the Identity Manager configuration. Binding to the proxy server as *cn=root* gives an administrator full access to the proxy server's configuration, but only anonymous access to the back-end directory entries.

In the cluster configuration we used in the TAA implementation, we have to update the Identity Manager Server configuration on each WebSphere Application Server cluster member where the Identity Manager Server is installed.

Note: Before updating the directory connection information about each cluster member, verify that the proxy server and the back-end Directory Servers are running.

Log on to each cluster member machine as root user and follow these steps:

1. Run the Identity Manager system configuration tool with the **install** option by executing the following command in a command line window:

```
ITIM_HOME/bin/runConfig install
```
2. As shown in Figure 5-59 on page 205, click the **Directory** tab.
3. Update the **Principal DN** field with the global administrator user ID that the Tivoli Identity Manager Server will use to log on to the proxy server, in this example, *cn=manager,cn=ibmpolicies*.
4. Update the Directory Server **Host Name** field with the proxy server host name, in our example, *proxyserver.taa.com*.

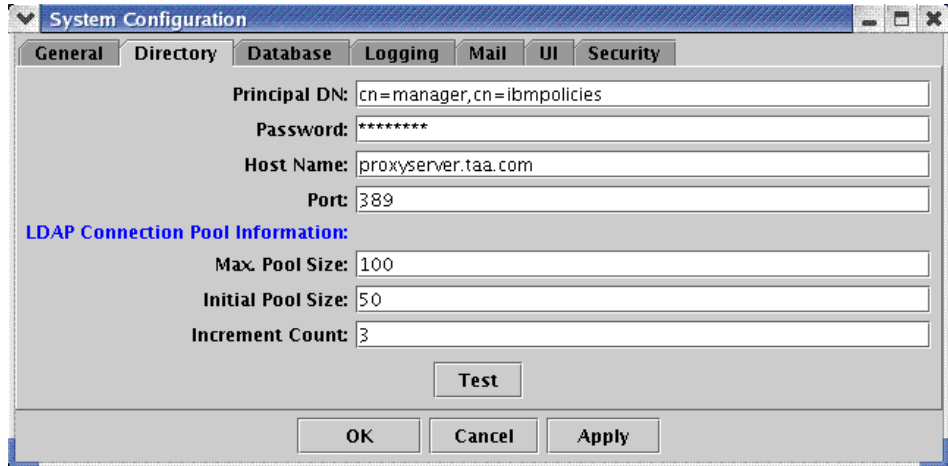


Figure 5-59 Proxy server directory connection configuration

5. Click **Test** to check the connection with the proxy server.
6. Click **OK**.
7. Repeat steps 2 through 6 for each cluster member.
8. Restart the Tivoli Identity Manager Server application using the WebSphere Deployment Manager administration console.

Running the system configuration tool writes log data to the ITIM_HOME/install_logs/runConfig.stdout log file.

Test Identity Manager with proxy server connection

Once all looks good with the Identity Manager proxy server connection, it is time to test. To properly test the proxy server configuration in conjunction with Identity Manager, we use the following two scenarios:

► Scenario A

With all the back-end servers up and running, we add an object using the Tivoli Identity Manager console and verify that this object is created on all the back-end Directory Servers.

► Scenario B

We stop the first peer back-end Directory Server, in this example, `directoryserver1.taa.com`. We then delete the object created in Scenario A using the Tivoli Identity Manager console. After restarting the back-end Directory Server, we verify that this object has been deleted on all back-end Directory Servers.

Scenario A

To test:

9. Follow steps 1 through 8 that we detailed in “Test replication from Directory Server master to replica” on page 176 to create a Tivoli Identity Manager object.
10. Log on to the `directoryserver2.taa.com` Web administration console as `cn=root` user.
11. Follow steps 14 through 18 that we detailed in “Test replication from Directory Server master to replica” on page 176, to verify that the entry object has been created on the `directoryserver2.taa.com`.

Scenario B

To test:

1. Log on to the `directoryserver1.taa.com` as root user.
2. Stop the server using the command:

```
idsslapd -I <instance_name> -k
```

In this example, the instance name is `idsldap1`.
3. Log in to Tivoli Identity Manager as an administrator (for example, `itim manager`) and delete the object created in Scenario A.
4. Follow steps 14 through 18 that we detailed in “Test replication from Directory Server master to replica” on page 176 to verify that the entry object has been deleted on the `directoryserver2.taa.com`.
5. Restart the server instance, `idsldap1`, using the command:

```
idsslapd -I <instance_name> -k
```
6. Follow steps 14 through 18 that we detailed in “Test replication from Directory Server master to replica” on page 176 to verify that the entry object has been deleted on the `directoryserver1.taa.com`.

5.5 Conclusion

This chapter has focused on providing a complete solution for TAA’s business and functional requirements concerning high availability and scalability.

The necessary project steps we addressed were the configuration of the WebSphere Application Server clustering for the Tivoli Identity Manager application, the setup of the HADR clustering for the IBM DB2 back-end database, and the automatic failover and load balancing solution for the IBM Tivoli Directory Server.

This solution guarantees a maximum of uptime with a minimum of additional hardware and software investments. Now, TAA can fulfill their service level agreements towards their business partners and customers regarding their identity management solution.



Technical implementation phase II

In this chapter, we cover how the functional requirements in Chapter 4, “Project design” on page 113 relate to the actual deployment and how Tivoli Austin Airlines (TAA) physically meets the deployment requirements.

Table 6-1 shows the relationships between the functional requirements and the deployment requirements in phase II.

Table 6-1 Mapping functional requirements to deployment requirements

Functional requirement	TAA's deployment requirement
B. Provide self-care application to empower employees, customers, and business partners to administer their own account needs.	1. Web application deployed within the company's intranet and extranet.
C. Provide self-registration capabilities for customers and business partners.	1. Web application deployed within the company's intranet and extranet.
E. Create an Identity Manager recertification process for all TAA applications.	4. Implement a recertification process to capture necessary metrics for reducing software licensing and maintenance.
F. Make recertification process available company wide and extranet wide.	4. Recertification process.
H. Mitigate error-prone administrative functions.	3. Advanced custom report design.
I. Reduce repetitive administrative duties.	3. Advanced custom report design.
M. Delegate administration of business partner users to business partner administrators.	2. Delegated administration design.

Let us look at the detailed implementation of each of these requirements now.

6.1 Self-care

TAA will provide a self-care application for its users and business partners to manage their accounts, user identities, and passwords. The following sections discuss the design considerations and implementation of the self-care application within TAA's intranet and extranet topologies.

6.1.1 Requirements

In the first portion of phase II, TAA will deploy functionality to relieve the security administration team of many of the user identity-related tasks by shifting the burden to the users themselves. In order to implement this functionality, TAA plans the following steps:

- ▶ A self-care application is developed to provide employees, business partners, and customers the following capabilities:
 - Users are able to request the creation, modification, and deletion of the accounts to which they are authorized by policy and role.
 - Users are able to access and complete the recertification process.
 - New users are able to access and register for a TAA account.
 - An approval process is available for account creation or modification requiring approval by a member of an application administration or management team.
- ▶ The self-care application is deployed in a phased approach to better control and mitigate any issues, which might arise by implementing the following phases:
 - Phase 1: Host the self-care application and test in production using administration and test team member accounts.
 - Phase 2: Open the self-care application to internal employees by region. TAA will do this through notification of employees by region of the new URL and capabilities during a simple training process. The training process consists of an informational e-mail describing the simple interface, capabilities, and a Web site they can go to for more information.
 - Phase 3: Open the self-care application to extranet members by notifying them of the URL and the informational e-mail we just described.
 - Phase 4: Incorporate the remaining business partners that were not covered in phase 3 and customers.

6.1.2 Design considerations

Prior to implementing the requirements, there are design aspects TAA must consider to ensure the correct implementation:

- ▶ The application needs to be lightweight and accessible anywhere, thus, a thick client is precluded as a possible solution and a Web-based application is required.
- ▶ The application needs to be highly available; hence, the application leverages the high availability of the Identity Manager installation as much as possible.
- ▶ The capability of the application needs to reflect the access rights of the user who accesses it. Therefore, a capability has to exist in order to restrict access to certain portions of the application, depending upon the type of user accessing it.
- ▶ Since the business partner administrators need elevated privileges, TAA has decided to create a Tivoli Access Manager-secured junction for these users to be able to access the standard Identity Manager interface for this functionality.

While these represent major considerations, they are not an exhaustive list.

6.1.3 TAA's implementation

The implementation of the self-care application consists of two distinct high-level tasks: The first is the development and test of the self-care application and the second is the deployment of the application. The development task has several subtasks as well, which are the creation of the overall application, and the creation of individual capabilities to offer to the user. In the following sections, we discuss each of these tasks.

Development

TAA will leverage software reuse to enable a faster and more efficient development cycle by starting the development of its self-care application with the self-care application that ships with Identity Manager in the <ITIM HOME>/extensions/examples/self_care directory. Since TAA uses a Windows-based Eclipse¹ development environment for all of its development activities, TAA will modify the existing application using the same development environment.

The existing self-care application includes most of the functionality TAA needs, specifically, password editing, forgotten password, challenge/response,

¹ Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. More information can be obtained at: <http://www.eclipse.org/>

self-registration, and, to some degree, application subscription; thus, there is not a lot of development work needed to include what is missing. The functionality TAA needs to add is the ability to complete To Do list items for account recertification and application subscription support to request application accounts. However, because of the elevated privileges required for the delegated administration requirement, TAA has decided that the users needing this functionality should log into Identity Manager directly and utilize the full-featured Identity Manager interface for these duties. Therefore, there will not be any development requirements associated with the self-care application for the delegated administration requirement. Lastly, there is the modification of how the self-care user interface looks to match the TAA requirements.

The self-care application provided with the Identity Manager installation is a Java 2 Platform Extended Edition (J2EE)² Web application that uses servlets, JavaServer™ Pages™³ (JSP), and property files to present and retrieve information from the user and interface with Identity Manager. The servlets control the dynamic information provided to the JSP as well as control the JSPs themselves. The following list is a high-level description of the different modules contained in the self-care application listed by functional area. Each functional area describes the servlet that controls the processing and the subsequent JSP, which it calls to display and capture information to and from the user:

► Account subscription

– AccountDataBean.java

Account data processing, for example, return getting and setting account DN and status.

– ApplicationServlet.java

Contains the doGet and doPost methods and controls input and output to applications.jsp.

– applications.jsp

Contains logic to display applications available for request and to gather user input of selected applications the user requests.

– application_sub.jsp

Contains the logic used to display the result returned from an application request submission.

² Java 2 Platform Enterprise Edition (J2EE) is an environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, application programming interfaces (APIs), and protocols that provide the functionality for developing multitiered, Web-based applications. For more information, see <http://java.sun.com/javaee/index.jsp>

³ JavaServer Pages (JSP™) technology provides a simplified, fast way to create dynamic web content. For more information, see <http://java.sun.com/products/jsp/>

- ▶ Challenge/response
 - ChangeChallengeResponseServlet.java
 - Logic to create, display, and validate challenge/response answers. Contains the doGet and doPost methods and controls the different JSPs used to display and request information from the user.
 - cr_warning_msg_box.jsp
 - Used to display errors or warnings within an error box on the current JSP from the processing of challenge response answers.
 - cranswers.jsp
 - Used to display and capture user answers to given challenges.
 - cranswersinfo.jsp
 - Used to display an information box on the current JSP generated from the processing of challenge response answers.
 - crforgotpwd.jsp
 - Displays the forgotten password information within the current JSP, which prompts for initial user information.
 - forgotpwd.jsp
 - Generates the initial display page of the challenges for the user to respond to with correct responses in order to log in via a forgotten password.
 - forgotpwdinfo.jsp
 - Displays a results page from answering the challenges.
 - newchangepwd.jsp
 - Displays and captures information to and from the user for a forgotten password forced password change.
- ▶ Change password
 - ChangePasswordServlet.java
 - Logic to create, display, and validate user password changes. Contains the doGet, and doPost methods, and controls the different JSPs used to display and request information from the user.
 - changepwd.jsp
 - Used to display information to the user and capture the required input to submit a change password request.
 - changepwdinfo.jsp
 - Displays results information from the submission of a change password request.

- selfchangepwd.jsp
Displays and captures information required to change a user’s password proactively.
- selfchangepwdinfo.jsp
Displays the result of a user request for a proactive password change.
- pwdrulesinfo.jsp
Displays password rules applicable to a given account password change or creation.
- ▶ Logon
 - logonServlet.java
Contains the doGet and doPost methods and controls input and output to logon and logout JSPs. Used to process the logging in and logging out of the user.
 - logon.jsp
Used to display and capture the information to the user necessary for them to log in.
 - logout.jsp
Displays informational details to the user about being logged off and a link to log back in.
- ▶ Self-registration
 - registerServlet.java
Logic to create, display, and validate the request to create a new person object. Contains the doGet and doPost methods and controls the different JSPs used to display and request information from and to the user.
 - selfregister.jsp
Displays information prompting the user for required information, which it then captures and returns to the servlet for processing.
 - selfregsub.jsp
Displays post processing information from the submission of a self-registration request.
 - welcome.jsp
Used to greet a user who wants to self-register.
- ▶ Self-care
 - selfCareServlet.java

Logic to create, display, and validate the request to change a person objects information, such as common name, surname, address, and so on. Contains the doGet and doPost methods and controls the different JSPs used to display and request information from the user.

- selfcare.jsp

Displays the current user information and captures the changes.

- selfcaresub.jsp

Displays the post processing messages from a user information change request submission.

► To Do List

- todolistServlet.java

Logic to create, display, and validate the completion request for a user's workflow assignments. Contains the doGet and doPost methods and controls the different JSPs used to display and request information from and to the user.

- todolist.jsp

Displays the current assignments and captures the user's decisions and reasons.

► View accounts

- ViewMyAccountsServlet.java

Main control processing gathering a user's current account subscriptions. Contains the doGet and doPost methods for controlling the display of a given user's accounts.

- AccountDataBean.java

Contains centralized methods for displaying a user's current account information.

- myaccounts.jsp

Displays a user's account information to the user.

► Utility and main process

- expiUtil.java

Contains abstracted logic for processing various common functions within the different modules. Includes methods such as Platform Context creation and property loading.

- mainServlet.java

Central process control. Initiates calls to subsequent modules and controls the main menu. Contains a doGet method, which is used to process a

user's selection and subsequent control transfer to the corresponding servlet module.

- main.jsp
Used to display the main menu and subsequent error and informational message boxes.
- index.html
Used for forward control to logonServlet.java.
- expi_header, expi_footer
Used to display a consistent top and bottom banner on all the display modules.
- expiProlog.jsp
This module contains prologue code for all JSPs that require the user to be authenticated prior to allowing them access to a page. If the user is NOT authenticated, the Subject will not exist in the header.
- expired_pwd_warning_msg_box.jsp
Used to display a warning box on the current JSP informing the user that their password has expired.
- ssoerror.jsp
Displays an error box within the current JSP notifying the user of a single sign-on error or warning.

To Do list

The To Do list is a required functionality for TAA in order to complete the recertification process discussed below in 6.5, "Recertification process" on page 290. Since the business requirement driving this functionality is to complete the recertification process, TAA has not included all of the features available from the standard Identity Manager user interface for assignment tasks such as grouping and locking within this phase. Furthermore, the user does not have the capability to complete only certain To Do List items and ignore others; they will have to complete all of their assignments or cancel out and come back to the assignments when they can complete them all. TAA feels that the users who are using the self-care application do not need this functionality in this phase of the deployment, but TAA will consider it for future phases depending on feedback.

In order for TAA to develop this functionality, the following modules were created or modified (see Appendix D, “Additional material” on page 389 about how to obtain the actual source code and details):

► **todolistServlet.java**

The *todolistServlet.java* servlet contains the processing and control logic for the completion of workflow participant assignments. The *todolistServlet* is first called from *main.jsp* when the user selects the *View and Complete My To Do List Assignments* link. See Figure 6-1 on page 220 for details.

The first time through the *todolistServlet*, the *init* method is called and initializes all the JavaServer Pages from the properties file that could be called. Once the *init* method completes, the *doGet* method is called and the session is validated by checking whether a new *session* object is created. This verification is done each time the *doGet* method is called and checks whether the session has timed out, and if so, forwards the user session back to the login page to input their login information again. With the session validated, the servlet requests the subject of the *HttpSession* object, *session*, with a call to the *session.getAttribute* method and uses it to instantiate a Human Resource Managed Object, *HumanResourceMO*. The Human Resource Managed Object represents a human workflow resource (workflow participant) and the object is used to obtain the Workflow Assignment Managed Objects of the user.

A collection of Workflow Assignment Managed Objects, *WorkflowAssignmentMO*, is returned from calling the method *HumanResourceMO.getAssignments*. This collection is saved to the session object for later usage so that the processing does not have to be repeated. Now that the collection of *WorkflowAssignmentMO*s has been retrieved for the user, the different activities that the user must complete are saved to another collection by iterating through the *WorkflowAssignmentMO* collection and calling the *WorkflowAssignmentMO.getActivity* method. This collection is also saved to the session object via the *session.setAttribute* method. By setting the collection to the session, it can be retrieved from the *todolist* JavaServer Page by use of the *session.getAttribute* method, and the objects within the collection can be processed. Upon completion of these tasks, the session is redirected to the *todolist.jsp* for the user to input information.

When the *todolist.jsp* completes, the *doPost* method on the *todolistServlet* is called to process the information input by the user. Here once again before any processing occurs, the session is validated to see if it has timed out and the user redirected back to the login page if necessary. The first processing task is to retrieve the *WorkflowAssignmentMO* collection saved to the session object from within the *doGet* method. This collection is then iterated through to return the Workflow Assignment Managed Objects in order to complete them with the information gleaned from the user’s input into the *todolist.jsp*. With the *WorkflowAssignmentMO*s retrieved from the collection, an

ActivityResult object is instantiated with the user information from the `todolist.jsp`. Once these two objects are available, then the Workflow Assignment Managed Object is completed and the user session is redirected back to the main servlet JSP, `main.jsp`.

► **todolist.jsp**

The `todolist` JSP, `todolist.jsp`, is called with the data generated from the `todolistServlet` in order for the user to input the necessary information for completion of their assignments. The data is passed between the servlet and the JSP by setting variables within the session object using a tag-object tuple. The data is set to the session object by calling the `session.set(TagName, Object)` method, and the data is retrieved by using the same tag used to save the data to the session via `session.getAttribute(TagName)` method.

The first retrieves the collection saved to the session from within the servlet and then iterates through the collection and displays the assignments along with a decision menu and reason text dialog combination. If the collection is empty, then the user is told that they have no pending assignments. Depending upon whether or not the user has any assignments to complete determines the button set that is displayed. If there are tasks to be completed, then the user is presented with both a submit (**OK**) button and a cancel (**Cancel**) button, and if not, then simply an **OK** button. Once the user inputs their decision information, they select **OK** and the form is submitted, or if the user wants to come back at a later time to complete the assignments, they select **Cancel**. If the user selects to submit the form, `todoForm`, by selecting **OK**, then the control is passed back to the servlet `todolistServlet.java` for processing and completion. If the user decides to cancel and to select the **Cancel** button, then control is sent to the main servlet and the session is redirected back to `main.jsp`.

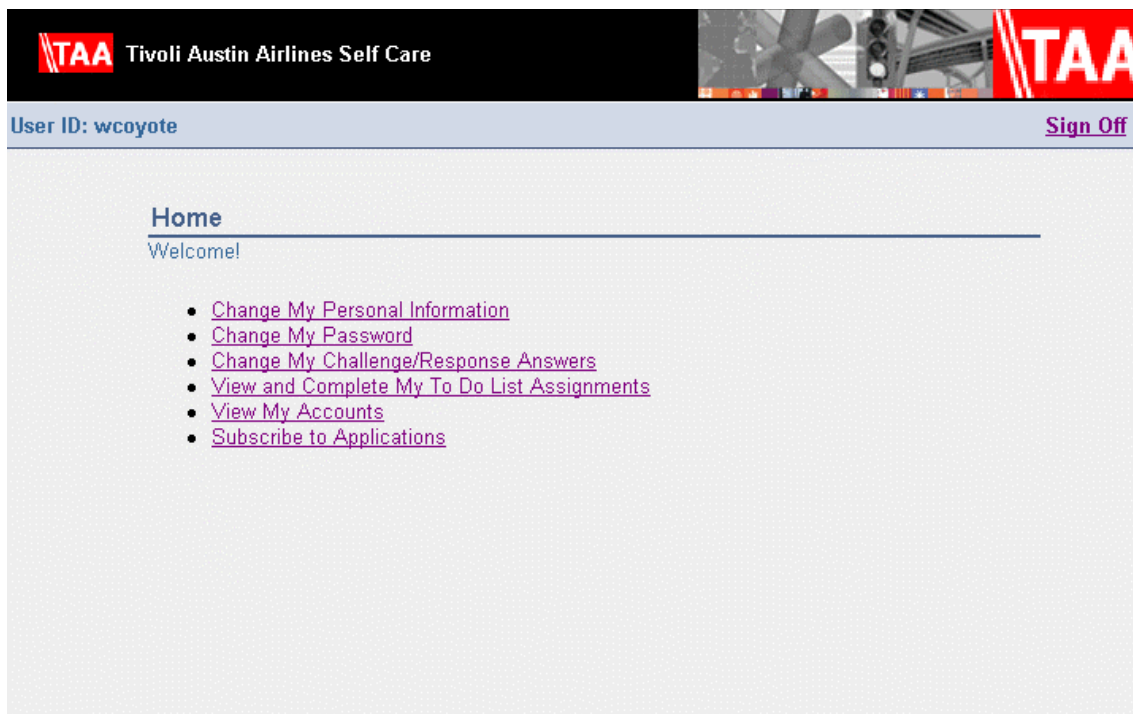


Figure 6-1 Home task view

Account subscription

TAA has decided that for this phase of the self-care application deployment, they simply allow request-based self-requesting of applications, which a user is entitled to by policy. TAA is very aware of the business needs to move the onus of *every day* tasks from the security team to the user to remove budgetary overhead and lessen the task load on the security team.

TAA has modified the self-care package that ships with the Identity Manager application to include the following capabilities:

- ▶ Add a check if Tivoli Access Manager is in use. TAA wants to have the capability to develop and test the self-care application features without having Access Manager installed if necessary, but to keep the Access Manager capabilities for deployment.
- ▶ Find all the services that the current user is entitled to have by policy. Account requests are based upon the services a user is authorized to access via provisioning policy, hence, the user is presented with a list of services which role memberships and policies allow them to request.

- ▶ Display those services to which the user is entitled and enable the user to request an account for these services.

In order for TAA to develop this functionality, the following modules were created or modified (see Appendix D, “Additional material” on page 389 about how to obtain the actual source code and details):

- ▶ **applicationServlet.java**

The *applicationServlet.java* servlet holds all the controlling logic for the application subscription and modification capabilities. It is originally called from *main.jsp* when the user selects the *Subscribe to and maintain applications* link. See Figure 6-1 on page 220 for details.

The first time through the *applicationServlet*, the *init* method is called and initializes all the JavaServer Pages used within the servlet from the properties file. Once the *init* method completes, the *doGet* method is called, and the session is validated by checking whether a new session object is created, redirecting the user back to the login page to input their login information if required. With the session validated, the process enters the *doGet* method in the *applicationServlet*; here the required information for upcoming calls is gleaned from the session, such as subject, platform, and user ID. This information is used to obtain an instance of a Person Managed Object, *personMO*.

The Person Managed Object represents the person object in Identity Manager, providing an interface to search and modify information relating to the person value object. The Person Managed Object (*personMO*) is required to find the services to which the user has authorization to request access. Before any more is done with the *personMO*, a check is done to find out whether Tivoli Access Manager is in use. The check is a branch in the logic, which decides how the service list is to be defined. However, in both branches of logic, the *personMO* is needed to instantiate an Account Managed Object or *AccountMO*.

The Account Managed Object represents an account object in Identity Manager, providing an interface to search and modify information relating to the account value object. If Access Manager is in use, then a call is made to the utility class method *expiUtil.lookupAccounts* to instantiate the *accountMO* and another call to the *expiUtil.account* method to instantiate an account object. Both objects pertain to the user’s Access Manager account. The *setRequestAttributes* method is used to gather those services to which the user is authorized to request access as defined by the Identity Manager services and Access Manager groups in the properties file. See Figure 6-2 on page 222 below for details.

```

#-----
# Application Subscriptions (TAM group)
# The name must correspond to an installed TAM Service. In addition
# the dn must represent the fully qualified ITIM DN for this
# service. (obtained from LDAP).
#-----

application.service.name=TAM Service
application.service.dn=
application.service.attribute=ertam4groupmember

#-----
# Groups are defined in a application.list attribute that contains
# a logical name the references each defined group.
# Each specified group item must provide a verbose text description
# (Application Name) and the defined TAM group respectively.
# The name will be used in the application.jsp and the group.dn
# is the internal reference to the actual TAM group.
#-----
application.list= group1,group2,group3,group4
application.group1.name=Expi Application 1
application.group2.name=Expi Application 2
application.group3.name=Expi Application 3
application.group4.name=Expi Application 4
application.group5.name=Expi Application 5
application.group6.name=Expi Application 6
application.group1.dn=tamgrp1
application.group2.dn=tamgrp2
application.group3.dn=tamgrp3
application.group4.dn=tamgrp4
application.group5.dn=tamgrp5
application.group6.dn=tamgrp6

```

Figure 6-2 Properties file Access Manager stanzas

If Access Manager is not in use, then a call is made to the *expiUtil.lookupAuthAccounts* to obtain a collection of Service Managed Objects, *serviceMO*.

The Service Managed Object represents a service object in Identity Manager, providing an interface to search and modify information relating to the service value object. The *expiUtil.lookupAuthAccounts* method instantiates an *AccountManager* object with the user and platform as arguments and is used to return the authorized services for the user from a call to its *getAuthorizedServices* object method using the personMO and locale as arguments. In both cases, that of Access Manager in use or not, the collection containing the user's authorized services is saved to the request object via the *request.setAttribute(TagName, Object)* method using the same tag-object combination as discussed earlier with the session set method. Just as with the session object methods, the request object methods give the capability to get the saved object back by calling the *request.getAttribute(TagName)* method.

Note that the object is saved as an Object class type and, therefore, must be cast back to the original data structure type. This, as with the session

example above, gives the capability to pass data back and forth from the JavaServer Page used to display the information and grab the user's input.

At this point, the session is ready to be redirected to the *applications.jsp* for the user's input, which is accomplished with a call to the request objects *sendRedirect* method using the name of the page to be redirected to as an argument.

Once the users have finished with their input, the session is passed back to the controlling servlet from the JavaServer Page and enters the *doPost* method of the application servlet. Again, since the session is stateless, it must be validated. Here TAA uses a different method of redirection to the login page if the session is found to be invalid. TAA makes a check to see if single sign-on is in use to find out how to redirect the user back to the login page. With the session validated, the form returned from the *applications.jsp* is checked to see if the user actually input any information or not. In the case of the latter, the session is redirected back to the *main.jsp* for the user to decide what to do next.

If the user did indeed input data, then several pieces of information are gathered for upcoming calls, such as the subject, platform, and personMO. At this point, a check is done to see whether Access Manager is in use or not in order to decide how to continue processing the information gathered to this point.

If Access Manager is in use, then a call is made to obtain the parameters set during the user interaction with the *applications.jsp*. This is done by calling the *request.getParameterNames*, which returns the names of the variables used in the *applications.jsp* in the form of an enumeration. Now the accountMO and account value object are retrieved from the session object. Using the account value object, the attribute values are pulled off the object with the *account.getAttributes* method. These attribute values are used to call the utility object class method *getTamGroups* to obtain a list of Access Manager groups that are currently defined for the user's Access Manager account. Then, a list of the possible Access Manager groups are gathered from the properties file with a method call to *getDefinedGroups*.

With the two lists, a single list is created of only those groups, which are to be added to the user. This is done by adding those that do not currently exist in the user's group definition and removing the groups from the list that were not selected to be added. Once the list is complete, the account value object can be updated with a call to the utility object's *updateAccount* method, supplying the account value object and the Account Managed Object as parameters.

In the case where Access Manager is not in use, a different thread is taken through the processing. In order to provision an account on a service without using Access Manager, there are a number of things that are required. An Account Manager Object, *AccountManager*, is required to actually create the

account creation request. Then a Person Managed Object, a Service Managed Object, an account value object, and finally a schedule date are all required in order to submit an account creation request.

As in the case where Access Manager is in use, the parameter names are gathered with a call to the request object's *getParameterNames* method. Next the collection of Service Managed Objects, which the user is authorized to request and have access to is gathered with a call to the *lookupAuthAccounts* method on the utility object. You can see here that TAA could have optimized processing by saving this collection value when it was processed during the *doGet* method call.

Now, with the list of service names that the user has selected from the *applications.jsp* and the list of Service Managed Objects the user is authorized to have access to, the Service Managed Objects needed to provision an account are selected by matching the service names. Once the Service Managed Objects are collected, the *AccountManager* object is instantiated.

With the Person Managed Object already gathered, the account value object needs to be created next. In order to create an account object, the correct account profile name is required as input to the instantiation call. However, this cannot be obtained from a remote system to the Identity Manager data store and therefore must be maintained in the properties file. The properties file contains a service-account profile mapping that will return the name of the account profile if given the service profile name, which can be obtained dynamically. This is done with a call to the service value object's *getProfileName* method. Once the service profile name is obtained, the account profile name can be retrieved with a *getProperty* method call from the utility object using the service profile name prepended with "profile" as the key. The key is created by concatenating *profile.* and the service profile name. See Figure 6-3 below for details.

```
-----  
# Application Subscriptions (No TAM Env)  
# The profile name must correspond to an installed service and account profile (Obtained from LDAP).  
# In addition the service and account profile tuple must of the form  
# profile.<service profile name>=<account profile name>  
#-----  
profile.adprofile=ADAccount
```

Figure 6-3 Properties file Account Profile stanza

Now that the account profile name is available, the account value object can be created. With this last piece, the account creation request can be submitted by calling the *createAccount* method on the *AccountManager* object with the *PersonMO*, *ServiceMO*, *Account*, and *date* arguments. Once

the account create request is submitted, the return value is captured and the session is redirected to the application-submitted JavaServer Page, *application_sub.jsp*.

► **applications.jsp**

The applications JSP, *applications.jsp*, is called with the data generated from the applicationServlet in order for the user to select the applications to which they want access. The data is passed between the servlet and the JSP by setting variables within the request object using a tag-object tuple. The data is set to the request object by calling the *request.setAttribute(TagName, Object)* method, and the data is retrieved by using the same tag used to save the data to the request object via *request.getAttribute(TagName)* method.

The applications JSP, much like the applicationServlet, has two logic threads. Here TAA left the sample code in place for the case where Access Manager was available, therefore, not having to write it themselves, and then added the thread for the case where Access Manager was not available. For the case where Access Manager is not in use, the collection of authorized Service Managed Objects is retrieved from the request object using the tag name that it was stored with. Then the collection is iterated through and the necessary attributes displayed in order for the user to select the services which they want to request. For the case where Access Manager is in use, the applications available to the user are gathered from the properties file and displayed for the user to select those which they want to request.

Upon finishing choosing the application selection, the user then selects **OK** and control is redirected along with the form to the applicationServlet to process the request.

► **applications_sub.jsp**

When the final processing of the application request is completed by the applicationServlet, control is passed to the applications submitted page to display the result codes and status of the request. The result value is saved to and retrieved from the session object using the method previously described. The result code is then matched and its corresponding message is displayed to the user. From here, control is passed back to the mainServlet and JSP when the user selects **OK**.

► **web.xml**

This is the Web application deployment descriptor that contains all the access control information for the Web application. The deployment descriptor, shown in Figure 6-4 on page 226, needs to be updated with the servlets and JSPs that have been created so that access is correctly granted.

xml	version="1.0" encoding="UTF-8"
DOCTYPE	web-app PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd"
web-app	{icon?, display-name?, description?, distributable?, context-param*, filter*, filter-mapping*, listener*, servlet*, servlet-mapping*, sess*
id	WebApp
display-name	expi
description	External Portfolio Integration Samples (ITIM)
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
id	Servlet_1065208949128
servlet-name	loginServlet
display-name	login
description	EXPI Logon Servlet
servlet-class	examples.expi.loginServlet
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet	{icon?, servlet-name, display-name?, description?, {servlet-class jsp-file}, init-param*, load-on-startup?, run-as?, security-role-ref*}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
servlet-mapping	{servlet-name, url-pattern}
welcome-file-list	{welcome-file+}

Figure 6-4 Self-care deployment descriptor

► **itim_expi.properties**

This is the properties file that controls the self-care application properties. The properties file contains JavaServer Page mappings that need to be updated with the new pages that have been added, and, in the case of a Tivoli Access Manager environment such as that of TAA's production environment, the Access Manager groups and applications also need to be updated. Furthermore, account profiles need to be kept up to date by adding the corresponding service profile → account profile mapping when a new service is added to the portfolio of service offerings. See the `itim_expi.properties` file below in Figure 6-5 on page 227.


```

#-----
# The mappings for the respective JSP pages
#-----
page.logon=logon.jsp
page.logout=logout.jsp
page.home=main.jsp
page.selfchangepwd=selfchangepwd.jsp
page.selfchangepwdinfo=selfchangepwdinfo.jsp
page.changepwd=changepwd.jsp
page.changepwdinfo=changepwdinfo.jsp
page.pwdrulesinfo=pwdrulesinfo.jsp
page.forgotpwd=forgotpwd.jsp
page.forgotpwdinfo=forgotpwdinfo.jsp
page.cranswers=cranswers.jsp
page.cranswersinfo=cranswersinfo.jsp
page.crforgotpwd=crforgotpwd.jsp
page.selfcare=selfcare.jsp
page.selfcare.submitted=selfcaresub.jsp
page.selfcare_footer=selfcare_footer.jsp
page.selfregister=selfregister.jsp
page.selfregister.submitted=selfregsub.jsp
page.applications=applications.jsp
page.applications.submitted=applications_sub.jsp
page.ssoerror=ssoerror.jsp
page.todolist=todolist.jsp
page.myaccounts=myaccounts.jsp

```

Figure 6-5 Self-care properties file update

Self-care user interface

TAA wants the user interface to look like its current Web site's user interface; therefore, the user interface of the self-care application that was used as a starting point needs to be modified according to the requirements of TAA. This is accomplished by modifying the header and footer html files that are provided as well as modifying the accompanying JavaServer Pages. See Figure 6-6 on page 228 below for more information.

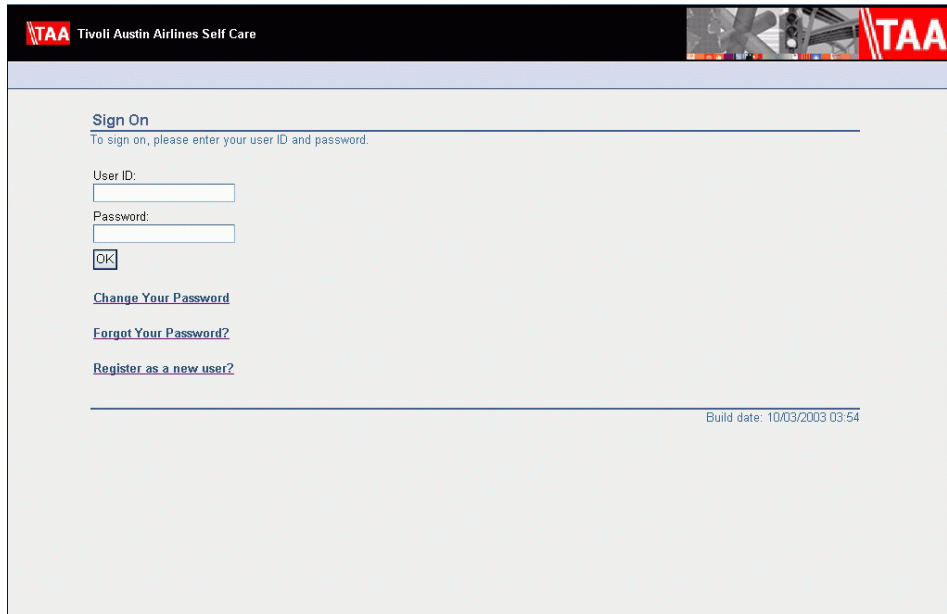


Figure 6-6 TAA self-care sign-on

Access Control Item (ACI)

In order for the TAA employees and business partners to take advantage of the application subscription capabilities of the self-care application, ACIs have to be created to allow it. Specifically, an organizational ACI for accounts needs to be created, with Object Type *erAccountItem*, granting Search, Modify, and Add operations. Attribute permissions set to grant read and write on password, and simply read for owner, password last change date, service, user id, and others.

Since all employees will have an Identity Manager account, no ACI needs to be created allowing the Identity Manager account to be seen or requested. See Figure 6-7 on page 229 for details.

Details		Values			
ACI Name	<input type="text" value="ACI for Account: Grant Search Add and Change Password to Self"/>				
Category	Account				
Scope	<input type="radio"/> Single <input checked="" type="radio"/> SubTree				
Object Type	erAccountItem				
Filter	All Objects				
Attributes					
Attribute Permissions					
Operation	Grant	Deny	None		
Search	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		
Modify	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		
Suspend	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		
Add	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		
Restore	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		
Remove	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		
ACI Principals					
Apply permissions to user's own information (Allow Self)					<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow Supervisor					<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Domain Administrator					<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Sponsor					<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Access for the following ITIM groups					
<input type="button" value="Add"/>					
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>					

Figure 6-7 Self-care ACI Setup for principal Self

Deployment

Before the self-care application can be deployed, it must be fully tested according to TAA's User Acceptance Test criteria, or UAT. Along with UAT criteria, there are documentation and training required by TAA prior to the application deployment. Since the self-care application is fairly trivial as far as user complexity, TAA has decided that there is not a requirement for classroom training, but, instead, there will be an online tutorial for users in order to gain required knowledge, along with the standard online help.

Steps toward deployment include the following:

- ▶ Install self-care Web archive file onto extranet application server.
- ▶ Configure Tivoli Access Manager single sign-on for new application.
- ▶ Configure self-care application.
- ▶ Phased deployment.

The idea is to divide the deployment into small manageable groups that can be managed more easily by a small number of people and, therefore, not commit the entire team to the deployment. This way, standard day to day tasks are still taken care of within TAA's service level agreement timeline.

- Phase 1 incorporates the internal deployment team. This enables the team to work out any big issues prior to phase 2.
- Phase 2 incorporates internal TAA employees by region.
- Phase 3 incorporates the business partners by groups.
- Phase 4 incorporates the remaining business partners not covered in phase 3 and customers.

6.2 Delegated administration

This section discusses the requirements, design considerations, and implementation of a delegated administration design for TAA's identity management environment.

6.2.1 Requirements

Currently TAA's business partners do not have accounts; however, they require certain resources that are located in the TAA IT infrastructure. TAA has decided to grant business partners access rights to some of its resources and delegate the administrative operations for people and accounts of the business partners to save help desk costs.

6.2.2 Design considerations

Before discussing the various areas within Identity Manager where delegation of administration might be implemented, we need to consider the delegation approach in general.

Delegation areas

There are various distinct areas to consider when dealing with delegation in Identity Manager:

- ▶ **People and account management:** Includes actions such as account creations, modifications, and password resets. These operations are controlled by Access Control Items (ACIs).
- ▶ **Business processes:** Includes actual auditable business tasks required to ensure the necessary business processes are put in place to assist with meeting the audit and business compliance controls of the organization such as relevant approval tasks and account compliance enforcement.
- ▶ **Service and policy management:** Includes the management of the service definitions in Identity Manager for the managed resources and also the roles, the management of the provisioning, identity, and password policies. These operations are controlled by ACIs.
- ▶ **Access control definition:** Includes the administration of the access controls defined by ACIs and also the management of the organization tree and Identity Manager groups. System administrators and domain administrators are responsible for the management of the organization tree and Identity Manager groups. Refer to the IBM Redpaper, *Organization Chart Design for IBM Tivoli Identity Manager*, REDP-3920, for more detailed information.

In TAA's case, the responsibilities for the management of the people and accounts belonging to the business partners are delegated to the business partner's own administrators. And they are also defined as the participants of the business processes for the business partner's accounts. The responsibilities for the management of services, policies, and access control definitions *are not delegated* to business partners.

Principals

Every ACI has a set of principals. These are the Identity Manager users who are granted or denied some accesses by the ACI. An ACI's principals can be defined by one or more Identity Manager groups or by dynamically calculated relationships. These dynamically calculated relationships are Supervisor, Sponsor, and Domain Administrator:

- ▶ **Identity Manager groups**

Each ACI can be defined with multiple Identity Manager groups as its principals statically, however, because there are no published APIs to set the principals of ACIs, this definition needs to be done manually for each ACI. This can cause an increase in ACI maintenance in the case where TAA has a large number of business partners to be controlled by ACIs.

- ▶ **Supervisor**

Any ACI granting a right to supervisors of an organizational unit will grant that right to all supervisors, but only within their own organizational unit and its sub-units. This assumes that the organizational units themselves are in the scope of the ACI. The key consideration is that only one person can be defined as supervisor within one organizational unit. This means that the administrative work cannot be shared between the business partner's multiple administrators.

- ▶ Sponsor

Basically the same as a supervisor.

- ▶ Domain administrator

As with supervisor, an ACI granting a right to domain administrators will grant that right to all domain administrators within only their own admin domain located under the ACI. Additionally, multiple domain administrators can be defined within one admin domain.

The use of *admin domains* automatically gives the domain administrators the full set of privileges to control within their domain, because they can define their own ACIs within the domain. To prevent certain operations from domain administrators, system administrators need to define additional ACIs denying those operations on the organizational container located above the admin domains.

With the above considerations, TAA has decided to define their ACIs using *domain administrator* as the principal.

Target

ACI target is defined with a category or a profile. Practically, it is related to the objectclass of the target entry. For example, an ACI whose target is the person category applies to *erPersonItem* entities, an ACI whose target is the person profile applies to *inetOrgPerson* entities, and so on. This means, an ACI applying to a profile also applies to profiles whose objectclass is in subordination to the one of the former. Careful consideration about this relationship should be required when designing custom person profiles.

Participants

Delegation of business processes is based on the definition of the participants within workflow actions, such as approvals, RFIs, workorders, and compliance alerts. These participants should be given access rights to view the relevant data on their To Do list. Refer to the online *IBM Tivoli Identity Manager Information Center Version 4.6, SC23-5267*, for more information about workflow participants.

Regarding domain administrator's participant type, the *participant* is the domain administrator of the organizational container that is associated with the subject of the workflow. If the subject is a person, service, or policy, the *organizational container* is the container in which the subject resides. For an account type subject, the service is used to determine the organizational container. Because TAA delegates to business partners the management of their own accounts but not of services, domain administrator is not an appropriate participant type for workflows that have an account type subject. Instead, we create organizational roles for each business partner's administrators and define them as a participant in the account management workflows by custom participant scripts.

Based on the above considerations, TAA designed their delegated administration scenario as follows:

- ▶ Create an admin domain and organizational role in Identity Manager for each business partner.
- ▶ Customize business processes for business partner's accounts.
- ▶ Define ACIs granting people and account management to the administrators of each business partner.
- ▶ Define ACIs denying other operations to the administrators of each business partner.

6.2.3 TAA's implementation

In this section, we document the implementation for the delegated administration of TAA's business partners.

Designing admin domains and domain administrators

First, let us look at the organization tree design and how to set up the business partner administrators.

Organization tree design

TAA has a lot of business partners and each business partner should be registered as an admin domain with delegated administrators. Based on the region where the partner is located, TAA divided them into three groups and placed them under the special organizational containers holding the group for each region. The resulting organizational tree is shown in Figure 6-8 on page 234.



Figure 6-8 Placement of the admin domains and the domain hold containers

Each *domain hold container* is defined as an organizational unit with its location code. This code is used for bulk feeding of admin domains in a later implementation step. The attributes for a domain hold container are depicted in Figure 6-9 on page 235.

Organizational Unit Name *	<input type="text" value="Central Region BPs"/>	
Location Name	<input type="text" value="1000"/>	
Description	<input type="text"/>	
Supervisor	<input type="text"/>	<input type="button" value="Search"/> <input type="button" value="Clear"/>

Figure 6-9 The attribute of a domain hold container

As shown in Figure 6-10, each admin domain has a name and its partner code. This code is later used for the self-registration of the business partner employees to locate them in the proper domain.

Admin Domain Name *	<input type="text" value="Metric Rent-A-Car"/>	
Description	<input type="text" value="1004"/>	
Administrator	<input type="text"/>	<input type="button" value="Search"/> <input type="button" value="Delete"/>

Figure 6-10 The admin domain for a business partner

Setting up business partner administrators

Each of the business partner's administrators, who are entitled to manage their own employees, must be a member of:

- ▶ Administrator of the own business partner domain.
For the management of people and accounts, acting as participant type within the person subject type workflow.
- ▶ Organizational role of the own business partner administrator.
For acting as participant type within the account subject type workflow.
- ▶ Identity Manager group that can access the Identity Manager organizational tree.

For accessing the people located in their own domain.

To automate assigning business partner administrators, TAA made some customizations within Identity Manager. These are based on the *title* attribute of the business partner people.

As for domain administrators, TAA customized the *add* workflow with a script node as depicted in Figure 6-11. This script is extended by a FESI extension that automates the add person process for the domain administrators. Refer to Appendix D, “Additional material” on page 389, about how to obtain the source code.

In Example 6-1, the actual line which needs to be placed in the `<ITIM_HOME>/data/fesiextensions.properties` file is given.

Example 6-1 fesiextensions.properties

```
fesi.extension.Workflow.AdminDomainModel=com.ibm.itim.custom.fesiextensions.AdminDomainModelExtension
```

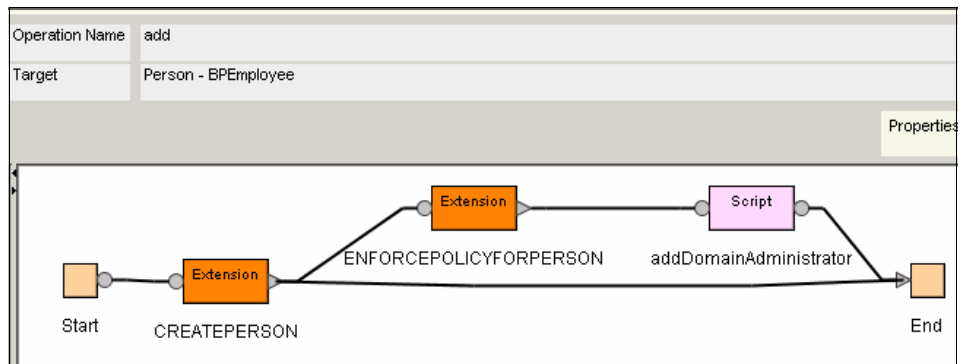


Figure 6-11 Operation workflow for adding a business partner employee

The properties of the script node, including the required JavaScript, are shown in Figure 6-12 on page 237.

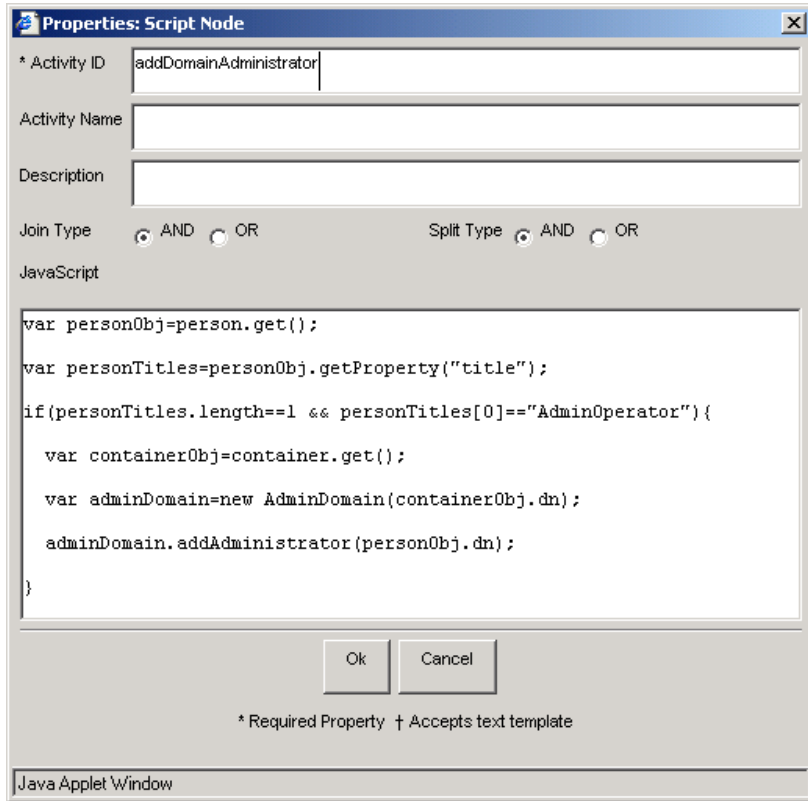


Figure 6-12 JavaScript for adding a domain administrator

TAA created an organizational role on each of the business partner domains. These organizational roles dynamically evaluate the administrative members from the domain with *title* attribute filter. The definition of the organizational role is depicted in Figure 6-13 on page 238.

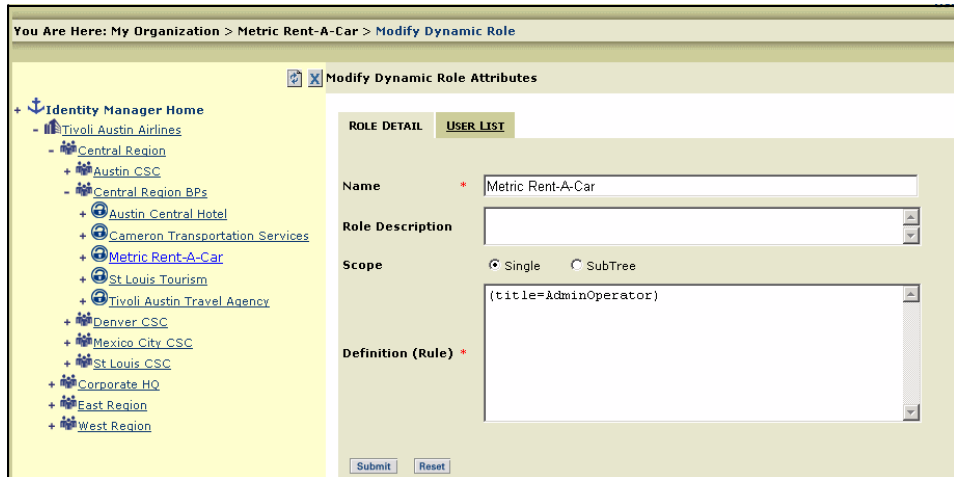


Figure 6-13 Domain specific organizational role

In order to control access to the organizational tree, TAA defined an Identity Manager group, shown in Figure 6-14, on the top container of the organization to enable its members to access the tree.



Figure 6-14 Identity Manager group for organizational tree access

To automate the group member assignment, TAA created an organizational role that is common for all domain administrators and a provisioning policy assigning the Identity Manager accounts owned by domain administrators to the group. The common organizational role has a filter using the *title* attribute as seen in Figure 6-15 on page 239. Figure 6-16 on page 239 shows the provisioning policy mapping for the common organizational role to the tree access group.

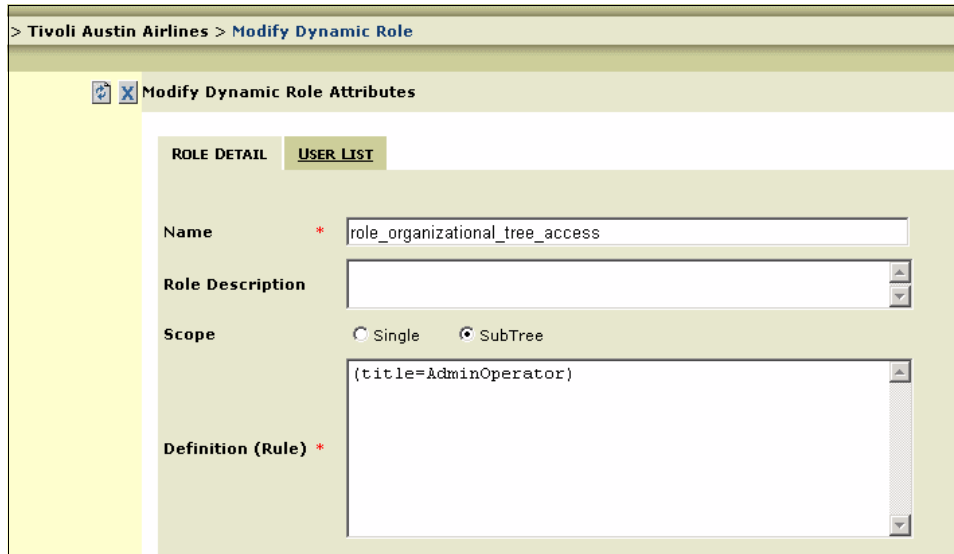


Figure 6-15 Common domain administrator role for organizational tree access



Figure 6-16 Provisioning policy mapping the common organizational role to the tree access group: Membership tab

Figure 6-17 on page 240 shows the entitlement information for this provisioning policy and Figure 6-18 on page 240 shows the provisioning parameter dialog.



Figure 6-17 Provisioning policy mapping the common organizational role to the tree access group: Entitlements tab

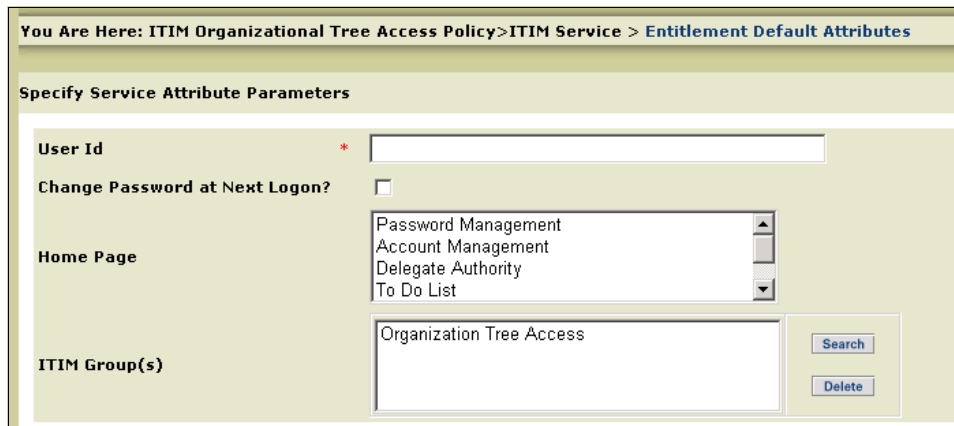


Figure 6-18 Provisioning policy mapping the common organizational role to the tree access group: Provisioning parameters

Bulk feed admin domains and organizational roles

TAA has a lot of business partners to register. To save the cost of registering them, TAA has created a tool that imports the business partner list and creates admin domains and organizational roles for each business partner. This tool uses the dataservices API of Identity Manager; it is called BulkFeedAdminDomain.sh.

The input list file consists of multiple records and each record contains three items for the business partner company:

1. The location code of the organizational container that holds the business partner's admin domain. As described in "Organization tree design" on page 233, this is based on the region of the business partner company.

2. The business partner company name.
3. The business partner code. This is used for the self-registration of business partner employees.

The input file content is displayed in Example 6-2.

Example 6-2 Business partners list file

```
1000|Austin Central Hotel|1001
1000|Cameron Transportation Services|1002
1000|Metric Rent-A-Car|1003
1000|St Louis Tourism|1004
1000|Tivoli Austin Travel Agency|1005
2000|Gracy Bus Tour|2001
2000|Hotel New York|2002
3000|Seattle Airport Service|3001
```

Refer to Appendix D, “Additional material” on page 389, about how to obtain the necessary files.

Example 6-3 depicts a sample execution output of the business partners bulk feed.

Example 6-3 Execution of business partners bulk feed

```
[root@itimsrver1 taka]# ./BulkFeedAdminDomain.sh ./BPList.txt
Created Admin Domain: Austin Central Hotel
Created Organizational Role: Austin Central Hotel
Created Admin Domain: Cameron Transportation Services
Created Organizational Role: Cameron Transportation Services
Created Admin Domain: Metric Rent-A-Car
Created Organizational Role: Metric Rent-A-Car
Created Admin Domain: St Louis Tourism
Created Organizational Role: St Louis Tourism
Created Admin Domain: Tivoli Austin Travel Agency
Created Organizational Role: Tivoli Austin Travel Agency
Created Admin Domain: Gracy Bus Tour
Created Organizational Role: Gracy Bus Tour
Created Admin Domain: Hotel New York
Created Organizational Role: Hotel New York
Created Admin Domain: Seattle Airport Service
Created Organizational Role: Seattle Airport Service

-----COMPLETED SUCCESSFULLY-----

Companies imported: 8
Total time(sec): 24
[root@itimsrver1 taka]#
```

Customize business processes

TAA decided to delegate two responsibilities for its business processes. These are:

- ▶ Approval Participant for self-registration of business partner employees.
- ▶ Approval Participant for recertification of the business partner accounts.

Self-registration

TAA decided to delegate the responsibility for approval for self-registration of business partner employees to each domain administrator. The customized *selfRegister* workflow is shown in Figure 6-19.

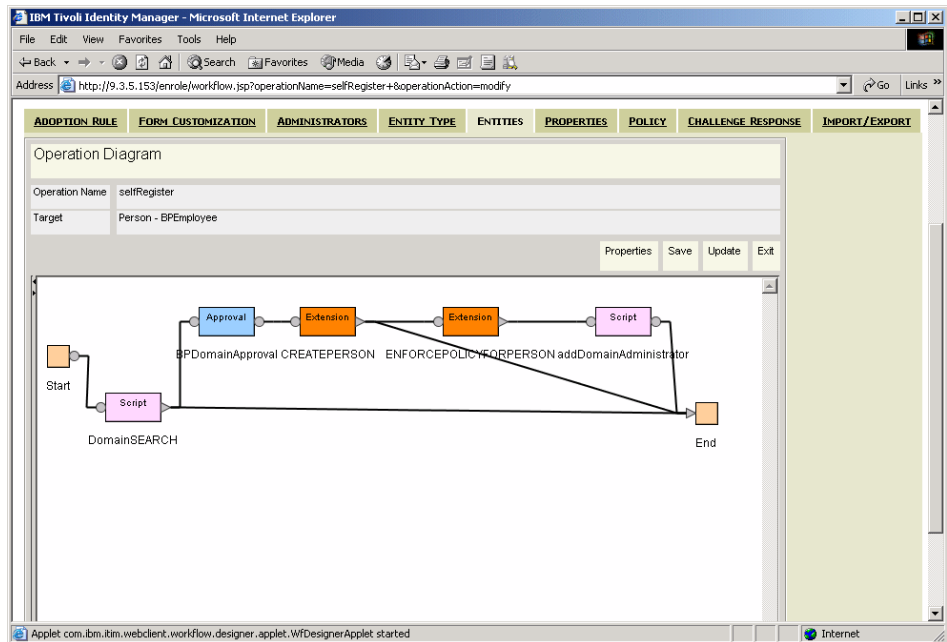


Figure 6-19 Operation workflow for business partners selfRegister

The *DomainSEARCH* script node, with the script displayed in Figure 6-20 on page 243, determines the business partner company domain of the new employee. It is based on the *location* attribute of the employee and business partner code described in the organizational tree design.

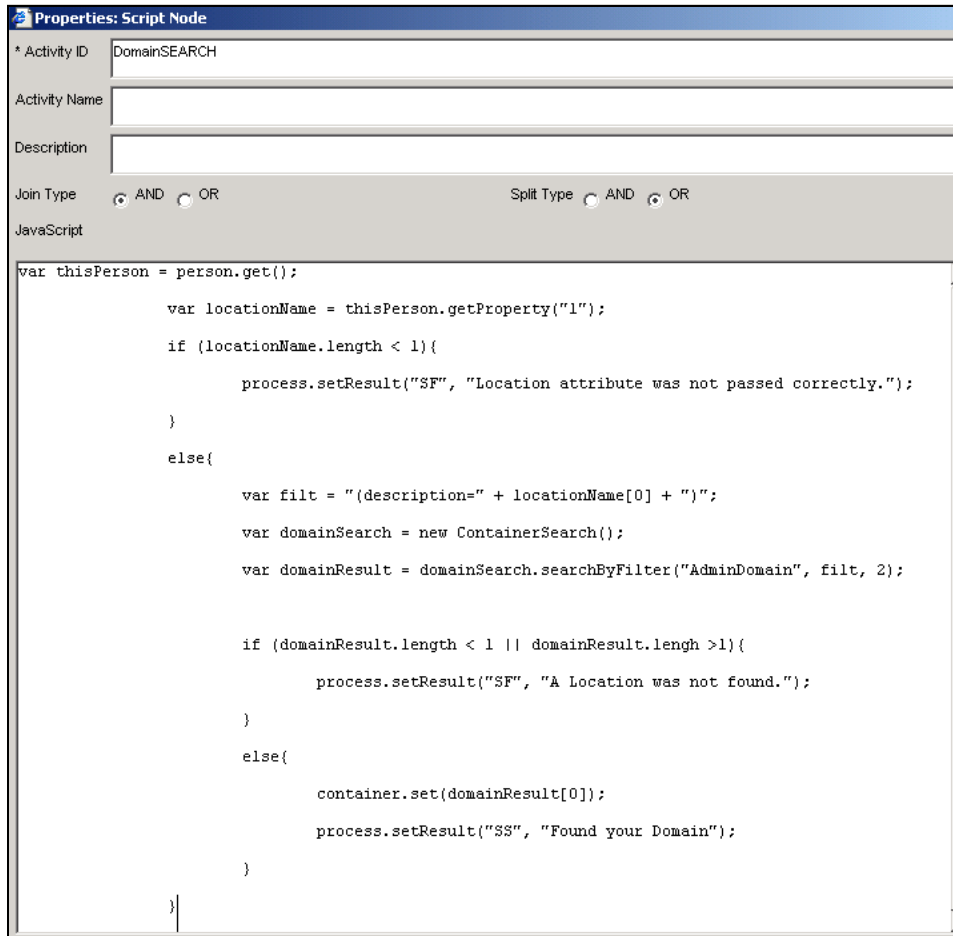


Figure 6-20 JavaScript for resolving the business partner domain

Within the configuration of the BPDomainApproval node, shown in Figure 6-21 on page 244, the Participant is set to Domain Administrator type. Because *selfRegister* is a person subject type workflow, the administrators of the domain where the new employee attempts to register will receive an approval request.

Properties: Approval Node

General | Notification | Action Text | Postscript

* Activity ID: BPDomainApproval

Activity Name:

* Participant: Domain Administrator

Escalation Participant: Participant Type

Escalation Limit: 1 Days 0 Hours 0 Minutes 0 Seconds

Join Type: AND OR Split Type: AND OR

Entity Type: Person

Input Parameters Search Relevant Data

ID	Type	Relevant Data ID
entry	Person	person
container	organizationalcontainer	container

Ok Cancel

* Required Property † Accepts text template

Java Applet Window

Figure 6-21 Configuration of Approval from Domain Administrator

Recertification

Because the account recertification process should be defined as an account subject type workflow, the approval participant is defined in the organizational role for the business company domain resolved by a custom script. For more details, refer to 6.5, “Recertification process” on page 290.

Define granting ACIs for domain administrators

Business partner administrators are delegated to perform people and account management for the employees of their company within the TAA realm. TAA defined the following three ACIs in the top container of the organizational tree to grant the necessary operations for each domain administrator as shown in Figure 6-22 on page 245, Figure 6-23 on page 246, and Figure 6-24 on page 247.

> Tivoli Austin Airlines > Edit ACI > ACI Detail

Access Control Item Details

Details	Values
ACI Name	ACI Granting BPEmployee management
Category	Person
Scope	<input type="radio"/> Single <input checked="" type="radio"/> SubTree
Object Type	taaBPEmployee
Filter	All Objects

Attributes

[Attribute Permissions](#)

Operation	Grant	Deny	None
Search	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Modify	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Suspend	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Add	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Restore	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Transfer	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Remove	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

ACI Principals

Apply permissions to user's own information (Allow Self)	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Supervisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Domain Administrator	<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow Sponsor	<input type="radio"/> Yes <input checked="" type="radio"/> No

Figure 6-22 ACI granting people management to administrators of business partners

> Tivoli Austin Airlines > Edit ACI > ACI Detail

Access Control Item Details

Details	Values
ACI Name	ACI Granting BPAccount management
Category	Account
Scope	<input type="radio"/> Single <input checked="" type="radio"/> SubTree
Object Type	erAccountItem
Filter	All Objects

Attributes

[Attribute Permissions](#)

Operation	Grant	Deny	None
Search	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Modify	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Suspend	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Add	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Restore	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Remove	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

ACI Principals

Apply permissions to user's own information (Allow Self)	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Supervisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Domain Administrator	<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow Sponsor	<input type="radio"/> Yes <input checked="" type="radio"/> No

Figure 6-23 ACI granting account management to administrators of business partners

Tivoli Austin Airlines > Edit ACI > ACI Detail

Access Control Item Details

Details	Values
ACI Name	ACI Granting BPITIMAccount management
Category	Identity Manager User
Scope	<input type="radio"/> Single <input checked="" type="radio"/> SubTree
Object Type	Identity Manager User
Filter	All Objects

Attributes
[Attribute Permissions](#)

Operation	Grant	Deny	None
Search	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Modify	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Suspend	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Add	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Restore	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>
Remove	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>

ACI Principals

Apply permissions to user's own information (Allow Self)	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Supervisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Domain Administrator	<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow Sponsor	<input type="radio"/> Yes <input checked="" type="radio"/> No

Figure 6-24 ACI granting Identity Manager Account management to administrators of business partners

Define denying ACIs for domain administrators

Policy, service, role, and container management are not allowed for business partners. TAA created the necessary ACIs to deny these operations to business partner domain administrators. The Provisioning Policy ACI is depicted in Figure 6-25 on page 248. The ACI definition for service, role, and container management has to execute accordingly.

Tivoli Austin Airlines > Edit ACI > ACI Detail

Access Control Item Details

Details	Values
ACI Name	ACI Denying BPProvisioningPolicy management
Category	Provisioning Policy
Scope	<input type="radio"/> Single <input checked="" type="radio"/> SubTree
Object Type	erProvisioningPolicy
Filter	All Objects

Attributes

[Attribute Permissions](#)

Operation	Grant	Deny	None
Search	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Modify	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Add	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>
Remove	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>

ACI Principals

Allow Supervisor	<input type="radio"/> Yes <input checked="" type="radio"/> No
Allow Domain Administrator	<input checked="" type="radio"/> Yes <input type="radio"/> No
Allow Sponsor	<input type="radio"/> Yes <input checked="" type="radio"/> No

Figure 6-25 ACI denying Provisioning Policy management to administrators of business partners

6.3 Advanced custom report design

In this section, we discuss the requirements, design considerations, and implementation of techniques used to extend the Identity Manager reports built-in designer in order to mitigate error-prone administrative functions and to reduce repetitive administrative duties. The report design techniques we show in this section use the Identity Manager built-in report designer and editing report XML templates in order to modify the search filter criteria and have greater flexibility to design reports.

Note: To design report templates using the Crystal Reports designer tool is beyond the scope of this section. Refer to the chapter titled “Using Crystal Reports with IBM Tivoli Identity Manager” in the online *IBM Tivoli Identity Manager Information Center Version 4.6, SC23-5267*, for detailed information about Crystal Reports template creation and integration with Tivoli Identity Manager.

6.3.1 Requirements

TAA has already addressed the basic requirements relating to the internal control reporting for the identity lifecycle management using the standard (ready to use) reports provided by Identity Manager.

Since some of the internal IT systems are regarded as more and more critical for the business growth of the company, the CIO requires detailed information about all account information (account status, account compliance, and so on) for each regional center.

TAA wants to implement one particular report for each regional center that discloses the non-compliant and disallowed accounts.

To ease the administrative reporting tasks, one of the goals is to have all the information predefined and ready to use for authorized Identity Manager administrators. The administrators only need to run the particular custom report without the need to specify any additional criteria.

Because of cost constraints, TAA does not want to buy new software licenses for an external reporting tool. TAA can accomplish this goal by using the Identity Manager built-in report designer and the customization of the search filter custom report template.

For a complete list of the Identity Manager standard reports, refer to the chapter titled “Reporting identity management data” in the *IBM Tivoli Identity Manager Information Center Version 4.6, SC23-5267*.

6.3.2 Design considerations

The design and implementation of a customized report template for the internal control of identity management data can be approached using the built-in Identity Manager report designer or using an external third-party reporting application, such as Crystal Reports. The decision to use the built-in functions is principally driven by the business requirement for cost reduction.

Before discussing how custom reporting can be implemented with the built-in report designer and XML editing for the search filter, we need to consider the overall report design approach using an identity management solution:

- ▶ What are the operational target systems for the custom reports (IT critical systems, individuals, audit operations, audit information, and so on) that are not covered by standard Identity Manager reporting?
- ▶ What personal, service, or account attributes have to be staged for the custom reports, and what should the prefilled (fixed) values that are defined for the ready to use report generation be?
- ▶ What is the average number of rows that each report will show? And what is the primary choice for the report format, for example, PDF (portable document format) or CSV (comma-separated value text file, sometimes called comma-delimited files)?

In the following sections, we discuss these issues in detail.

Operational targets for custom reporting

The starting point for the implementation of custom reports for the IT environment is the analysis and definition of the critical IT systems that require continuous and more accurate internal control. We want to avoid generating too many reports or reports on non-critical targets and situations. Providing too many predefined reports could actually inhibit the achievement of a business goal. The best practice is to implement the report customization using a staged approach.

Mapping attributes and search criteria definition

Map only the entities and attributes for which you want to generate a custom report. These mappings directly impact the performance of IBM Tivoli Identity Manager, because all of the data from the directory server is copied to the database each time a data synchronization is performed.

In the search filter criteria definition, you can present a lot of multiple choice lists and open fields that can make the report template more flexible, but this could be a very time-consuming task for an administrator who has to run this report a lot of times with the same choice values. A prefilled value for the search can help to consistently reduce execution time for weekly or daily reports.

Custom report format

A data analysis should be performed to verify the average number of records included when a particular report is executed. The result can guide your choice of the custom report format. By default, a PDF report can contain up to 5000 records. You can change this value by using the *enrole.ui.report.maxRecordsInReport* property in the UI.properties file. You do not have to restart the server for the changes to take effect. Changes take place

within 30 seconds. This change can have an impact on execution performance and should be carefully evaluated in a test environment.

Note: To design more sophisticated report templates with sophisticated search filter criteria, we can use the Crystal Reports Designer and then import the templates into the IBM Tivoli Identity Manager console. Refer to the chapter “Using Crystal Reports with IBM Tivoli Identity Manager” in the online *IBM Tivoli Identity Manager Information Center Version 4.6, SC23-5267*.

6.3.3 TAA’s implementation

TAA wants to implement three custom reports in this phase, one for each regional center, that show the account and compliance status of all the employees that work in the regional center. The reports only have to show the non-compliant and disallowed accounts and have to be ready for use by the authorized administrators. The administrators should be able to execute the reports without having to provide further input, only selecting the format of either CSV or PDF.

The implementation of a custom report template involves the execution of the following steps:

- ▶ Map the entities and attributes that can be included in the report.
- ▶ Stage the reporting tables used to generate the report.
- ▶ Create the report templates that determine the content of the custom report.
- ▶ Create the advanced search filter criteria for the custom report.

Optional: If *Incremental Data Synchronizer* is configured within your environment, run the Incremental Data Synchronizer to update any changes to the report data and access control item. If the Incremental Data Synchronizer is not configured, you must select Synchronize Data to perform this step.

The following section discusses the implementation of the three custom reports described in Table 6-2.

Table 6-2 Custom reports overview description

Title	Columns	Description
Account report for Central Region	<ul style="list-style-type: none"> ▶ Employee full name ▶ Employee number ▶ Organizational unit ▶ Account user ID ▶ Account compliance ▶ Service name 	This report shows all non-compliant or disallowed accounts for employees in the Central Region.
Account Report for East Region	<ul style="list-style-type: none"> ▶ Employee full name ▶ Employee number ▶ Organizational unit ▶ Account user ID ▶ Account compliance ▶ Service name 	This report shows all non-compliant or disallowed accounts for employees in the Central Region.
Account Report for West Region	<ul style="list-style-type: none"> ▶ Employee full name ▶ Employee number ▶ Organizational unit ▶ Account user ID ▶ Account compliance ▶ Service name 	This report shows all non-compliant or disallowed accounts for employees in the Central Region.

Before we start with the custom report implementation steps, it is necessary to recall the TAA organization tree and TAA entities structure involved in this report customization.

TAA uses a location-based model for their organization tree. The resulting TAA organization tree is shown in Figure 6-26 on page 253.



Figure 6-26 TAA organization tree

People in TAA are placed in the tree based on their relevant location. For example, a person located in the Denver CSC is placed in the Denver CSC organizational unit. People, who are not part of the CSC, are placed into their region's organizational unit. This includes regional administrators. For example, administrators for the central region are placed in the Central Region organizational unit.

Identity Manager's person objects are based on the LDAPv3 standard *inetOrgPerson* object class. This object class does not provide an attribute appropriate for storing the employee status (*active*, *leave of absence*, or *inactive*) and the regional center in which the employee works (Central Region, East Region, or West Region), so TAA has to define their own custom person type with two new attributes that can hold this data.

The new object class' name is *taaEmployee*. It uses *inetOrgPerson* as its superior class. It adds *taaEmployeeStatus* and *taaEmployeeRegionLoc* as allowed attributes. The new attributes created on the *taaEmployee* object are:

- taaEmployeeStatus** This attribute is not indexed; it does not allow multiple values. It stores case-insensitive strings.
- taaEmployeeRegionLoc** This attribute is not indexed; it does not allow multiple values. It stores case-insensitive strings.

The *taaEmployeeRegionLoc* attribute will be used for the custom report search filter definition.

Refer to Chapter 9, “Technical implementation: Phase I”, in the *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996, for more details about Identity Manager’s custom person object implementation.

Note: Before we begin with the custom report implementation, we have to ensure that the *availableForNonAdministrators* property in the *adhocreporting.properties* file is set to `true`. This property enables the names of custom reports to be displayed in the Target list of reports when you create an ACI definition.

Map the entities and attributes

The type of data that can be included in a custom report is determined by the report schema. To create the custom report schema, you must create an attribute mapping that specifies the entities and entity attributes that can be included in a report. The Employee entity is not a default Identity Manager entity, so its attributes are not available by default for custom reports.

To map the Employee’s attributes needed for the custom report design, complete the following steps:

1. In the Main Navigation Bar, click **Report**.
2. Click **Design Schema** in the task bar.
3. In the Schema Designer page, select an Employee entity from the list of objects.

Both Mapped and Unmapped attributes for the entity selected are displayed in lists labeled Mapped Attributes and Unmapped Attributes, respectively.

4. Select **Employee Number** on the Unmapped Attributes list and click the > (Map) button.
5. Repeat step 4 for Employee First Name, Employee Last Name, Employee Region Location, and Organizational Unit Name. Figure 6-27 on page 255 shows the attributes mapped for the Employee entity.
6. Click **Submit**.

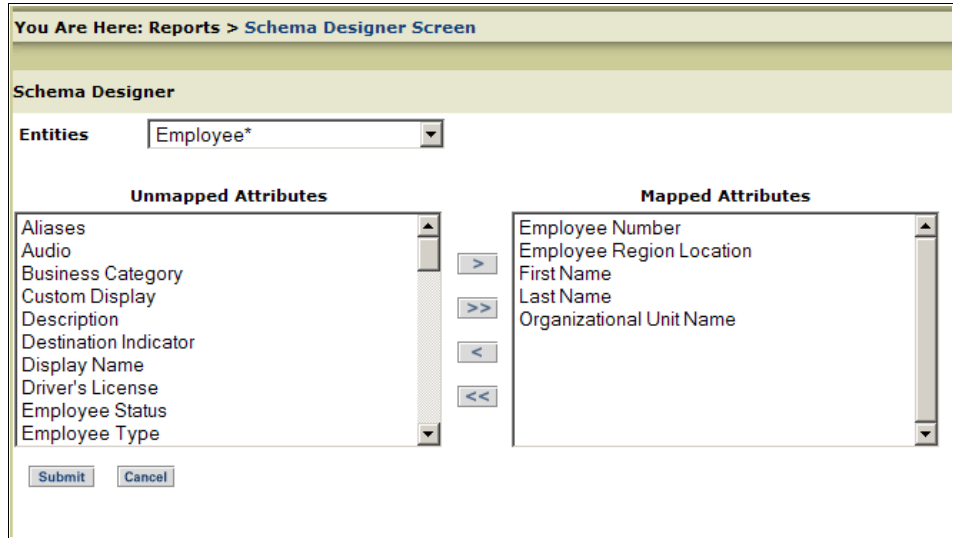


Figure 6-27 Employee entity attribute mapping

Staging data for custom reports

Generally, you do not need to perform a data synchronization task when you modify a report. However, in this implementation, since Employee data has been mapped to be used in the custom report, you must perform a data synchronization for the changes to take effect. The entities and attributes that you mapped using the Design Schema task are made available for the Design Report task only after a data synchronization task is completed.

Note: The Ad hoc reporting functionality is not available during Synchronization. The design report and run report functions will be available when the Synchronization process has completed.

Creating custom report template

All reports implemented in this section are generated using *report templates*. A report template defines the layout of a report and the filter criteria that determines the contents of the report. When you select a report to run in the console, you select the report template used to generate the report.

In the report layout details you specify the report name, sort ordering, and page numbering as well as the entities and attributes that you want to include in the report. The entities that you select are listed as table names in the *Table* field of the report. The attributes that you select are listed as table columns in the

Column field of the report. A sample of a report layout design used for the standard report “Individual Accounts” is shown in Figure 6-28.

The screenshot shows the 'Report Designer' interface for the 'Individual Accounts' report. The breadcrumb trail at the top reads 'You Are Here: Reports > Report Designer'. The main section is titled 'Add/Modify Report Template' and is divided into several sections:

- Report Header:** Contains two checked options: 'Report creation date and time' and 'Report created by'.
- Page Header:** Contains a 'Report Title' field with the value 'Individual Accounts'.
- Page Information:** Contains a table with columns: Function, Table, Column, Width, Sort, and Sort Order. The current values are: Function: Select function, Table: Select entity, Column: Select Column, Width: 5, Sort: None, Sort Order: (empty). Below the table are 'Add' and 'Delete' buttons.
- Added Columns:** A list box containing the following items: Select Column, Person.Last Name, Person.First Name, Organizational Container.Name, AccountUser Id, Service.Service Type, and Service.Service Name. A 'Page m of n' checkbox is checked to the right of this section.

At the bottom of the interface are 'Continue' and 'Cancel' buttons.

Figure 6-28 Individual accounts layout report designer view

The report filters that you specify determine the contents of the report. A sample of a report filter design used for the standard report “Individual Accounts” is shown in Figure 6-29 on page 257.

You Are Here: Report > Report Designer > Create Filter

Create Filter for Report

Function/Operator	Entity/User Input	Column	Condition
Select function ▼	Select entity ▼	Select Column ▼	
Select operator ▼			
Select function ▼	Select entity ▼	Select Column ▼	Select condition ▼

Select filter

- Account.Parent.DN Equals Organizational.Container.DN AND
- Account.Owner Equals Person.DN AND
- Account.Service Equals Service.DN AND
- Person.DN Like '_USERINPUT_' AND
- Organizational.Container.DN Like '_USERINPUT_'

Select stylesheet to be applied :	Standard ▼
Select Report Category	Individual Reports ▼

Figure 6-29 Individual accounts filter report designer view

To create the custom “Account Report for Central Region” template, complete the following steps:

1. In the Main Navigation Bar, click **Report**.
2. Click **Design Report** in the task bar.
3. In the Report List page, click **Add**.
4. Enter Account Report for Central Region in the **Report Title** field.
5. From the Table drop-down list, select **Employee**.
6. From the Column drop-down list, select **Full Name**.
7. Click **Add**. Your window should look like Figure 6-30 on page 258.

You Are Here: Reports > Report Designer

Add/Modify Report Template [?](#)

Report Header
 Report creation date and time
 Report created by

Page Header
 Report Title: Account Report for Central Region

Page Information

Function	Table	Column	Width	Sort	Sort Order
Select function ▼	Employee ▼	Select Column ▼	5 ▼	None ▼	▼

Page m of n

Added Columns

- Select Column
- Employee Full Name

Figure 6-30 Account report for Central Region designer view

8. Repeat steps 5 through 7 to add the other columns listed in Table 6-3.

Table 6-3 Attributes displayed in Account Report for Central Report

Table name	Column name
Employee	Full Name
Employee	Employee Number
Organizational Container	Name
Account	User Id
Account	Account Compliance
Service	Name

At the end of the columns report template design, your window should like Figure 6-31 on page 259.

You Are Here: Reports > Report Designer

Add/Modify Report Template

Report Header	Report creation date and time <input checked="" type="checkbox"/>					
	Report created by <input checked="" type="checkbox"/>					
Page Header	Report Title	Account Report for Central Region				
Page Information	Function	Table	Column	Width	Sort	Sort Order
	Select function ▼	Select entity ▼	Select Column ▼	5 ▼	None ▼	▼
	Add Delete					
	<input checked="" type="checkbox"/> Page m of n					
Added Columns	<div style="border: 1px solid black; padding: 2px;"> Select Column Employee Full Name Employee.Employee Number Organizational Container.Name Account.User Id Account.Account Compliance Service.Service Name </div>					
Continue Cancel						

Figure 6-31 Account report for Central Region columns design view

9. Click **Continue**.
10. From the first Entity/User Input drop-down list, select **Employee**.
11. From the second Function/Operator box, select **Like**.
12. From the second Entity/User Input box, select **_USERINPUT_**.
13. From the condition box, select **AND**.
14. Click **Add**. Your window should look like Figure 6-32 on page 260.

You Are Here: Report > Report Designer > Create Filter

Create Filter for Report

Function/Operator	Entity/User Input	Column	Condition
Select function ▼	Select entity ▼	Select Column ▼	
Select operator ▼			
Select function ▼	Select entity ▼	Select Column ▼	Select condition ▼

Select filter

Employee.Employee Region Location Like '_USERINPUT_' AND

Select stylesheet to be applied :	Standard ▼
Select Report Category	Custom Reports ▼

Figure 6-32 Account report for Central Region search filter design view

15. Repeat steps 10 through 14 to add the other search filter criteria listed in Table 6-4.
16. Click **Submit**.

Table 6-4 Search filter criteria defined for Account Report for Central Region

Entity.Column	Operator	Entity.Column
Employee.Employee Region Location	Like	'_USERINPUT_'
Account.Owner	Equals	Employee.DN
Account.Service	Equals	Service.DN
Account.Parent DN	Equals	Organizational Container.DN
Account.Account Compliance	Greater Than	'_USERINPUT_'

At the end of the search filter design, your window should like Figure 6-33 on page 261.

You Are Here: Report > Report Designer > Create Filter

Create Filter for Report

Function/Operator	Entity/User Input	Column	Condition
Select function ▼	Select entity ▼	Select Column ▼	
Select operator ▼			
Select function ▼	Select entity ▼	Select Column ▼	Select condition ▼

Select filter

- Employee.Employee Region Location Like '_USERINPUT_' AND
- Account.Owner Equals Employee.DN AND
- Account.Service Equals Service.DN AND
- Account.Parent DN Equals Organizational Container.DN AND
- Account.Account Compliance Greater Than '_USERINPUT_'

Select stylesheet to be applied :	Standard ▼
Select Report Category	Custom Reports ▼

Figure 6-33 Account report for Central Region search filter

For detailed information about Tivoli Identity Manager database tables reference, refer to *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6, SC32-1769*.

Create the advanced search filter criteria

To implement a ready to use report for each regional center that only shows non-compliant and disallowed accounts, you need to modify the report search filter criteria with predefined user input values.

To do this, you must edit the XML template of the custom report. This section discusses how to export, modify the XML report template, and import Identity Manager database information using the DB2 command line.

To modify the custom search filter for the “Account report for Central Region” template, complete the following steps:

1. Connect to the Identity Manager database using the command:

```
db2 connect to ITIMDB user enrole using <password>
```

ITIMDB specifies the name of the Identity Manager database and *<password>* is the *enrole* user password.

2. Display the template report list using the command:

```
db2 select ID , TITLE from enrole.REPORT
```

Where *REPORT* is the table that stores details of the reports designed and generated by Tivoli Identity Manager users, and *ID* and *TITLE* are the table columns needed to identify the report template “Account Report for Central Region”.

Let us take a look at the output:

D	TITLE

1	individualAccounts
2	individualAccountsByRole
3	accountsOnService
4	policiesGoverningRole
5	servicesGrantedToIndividual
6	accountOperations
7	accountOperationsPerformedByIndividual
8	approvalAndRejections
9	pendingApprovals
10	suspendedAccounts
11	suspendedIndividuals
12	services
13	policies
15	operationReport
16	rejectedReport
17	userReport
18	accountReport
19	dormantAccounts
22	nonCompliantAccounts
14	accessControlInformation
20	reconciliationStatistics
21	auditEvents
51	Account Report for Central Region

The schema of the table REPORT on the Identity Manager database is shown on Table 6-5.

Table 6-5 Report table

Column name	Description	Data type
ID	Unique ID for the table.	int
TITLE	Report type given to the report.	varchar
TYPE	Indicates whether the report was designed using Tivoli Identity Manager or Crystal Reports.	varchar
AUTHOR	Author of the report (designer).	varchar
REPORT_SIZE	The size of the report template stored in the REPORT_DATA column of this table.	int
REPOR_ DATA	The report template is stored here.	binary large object
STYLESHEET	Name of the style sheet to use for displaying the report.	varchar
REPORTSUBTYPE	Identifies if this report is a user-defined report or an out-of-box report.	varchar
REPORTCATEGORY	Identifies which category the run should be listed under on the Run Reports page.	varchar
EDITABLE	Indicates if this report can be edited or not. The value is N for reconciliation statistics and access control information reports.	char

The “Account Report for Central Region” has a report ID of 51.

3. Switch to the directory where you want to export the report template, in the example, /tmp/reports.

Export the report template using the following command where /tmp/reports/report.del is the output file for the result query and /tmp/reports/ is the path for the report template (binary large object type - BLOB):

```
db2 export to /tmp/reports/report.del of del lobs to /tmp/reports/ modified by lobsinfile select * from REPORT where ID=51
```

The output of the DB2 export command is below:

```
# db2 export to /tmp/reports/report.del of del lobs to /tmp/reports/
modified by lobsinfile select * from REPORT where id=51
SQL3104N The Export utility is beginning to export data to file
"/tmp/reports/report.del".
```

SQL3100W Column number "2" (identified as "TITLE") in the output DEL format file is longer than 254 bytes.

SQL3100W Column number "3" (identified as "TYPE") in the output DEL format file is longer than 254 bytes.

SQL3100W Column number "4" (identified as "AUTHOR") in the output DEL format file is longer than 254 bytes.

SQL3100W Column number "7" (identified as "STYLESHEET_NAME") in the output DEL format file is longer than 254 bytes.

SQL3100W Column number "9" (identified as "REPORTCATEGORY") in the output DEL format file is longer than 254 bytes.

SQL3100W Column number "10" (identified as "REPORTSUBTYPE") in the output DEL format file is longer than 254 bytes.

SQL3105N The Export utility has finished exporting "1" rows.

Number of rows exported: 1

Two files will be created on the directory specified report.del and db2exp.001.

For a complete reference of the DB2 export command, refer to DB2 Universal Database™ Reference Commands in the online DB2 Information Center.

In the /tmp/reports directory, the export command has now created two files:

report.del Result query of the db2 export command

db2exp.001 Report template

The report.del file contains a reference to the binary large object file (BLOB) and its file size (see Example 6-4).

Example 6-4 Content of the report.del file

```
51,"Account Report for Central Region","Designer","itim  
manager",1994,"db2exp.001.0.1994/","standard","Y","customReports","custom"
```

The report template is stored in the db2exp.001 file (see Example 6-5).

Example 6-5 Report template in XML format

```
<?xml version="1.0" encoding="UTF-8"?>  
<AdHocReport Version="1.1">  
  <Created>  
    <Time>2006-03-07 02:21:04.494</Time>  
    <TimeZone>GMT</TimeZone>
```

```

    <Name>itim manager</Name>
</Created>
<LastUpdate>
    <Time>2006-03-07 02:21:04.494</Time>
    <TimeZone>GMT</TimeZone>
    <Name>itim manager</Name>
</LastUpdate>
<Style>standard</Style>
<ReportHeader>
    <Title>dummy</Title>
</ReportHeader>
<Page Lines="45">
    <PageHeader>
        <TimeStamp TimeZone="GMT" show="YES"/>
        <User show="YES"/>
        <Logo show="YES"/>
    </PageHeader>
<Body>
    <Query>
        <Columns>
            <Column Width="5" id="1">
                <Header>Employee.cn</Header>
                <Source>Employee.cn</Source>
                <Sort Order="1" Type="ASC"/>
            </Column>
            <Column Width="5" id="2">
                <Header>Employee.employeenumber</Header>
                <Source>Employee.employeenumber</Source>
                <Sort Order="2" Type="ASC"/>
            </Column>
            <Column Width="5" id="3">
                <Header>OrganizationalContainer.name</Header>
                <Source>OrganizationalContainer.name</Source>
                <Sort Order="3" Type="ASC"/>
            </Column>
            <Column Width="5" id="4">
                <Header>Account.eruid</Header>
                <Source>Account.eruid</Source>
                <Sort Order="4" Type="ASC"/>
            </Column>
            <Column Width="5" id="5">
                <Header>Account.eraccountcompliance</Header>
                <Source>Account.eraccountcompliance</Source>
                <Sort Order="5" Type="ASC"/>
            </Column>
            <Column Width="5" id="6">
                <Header>Service.erservicename</Header>
                <Source>Service.erservicename</Source>
                <Sort Order="6" Type="ASC"/>
            </Column>
        </Columns>
    </Query>
</Body>
</Page Lines="45">
</ReportHeader>
<Style>standard</Style>
</LastUpdate>
</Created>
</Name>itim manager</Name>

```

```

        </Column>
    </Columns>
    <Tables>
        <Table>Service</Table>
        <Table>Account</Table>
        <Table>OrganizationalContainer</Table>
        <Table>Employee</Table>
    </Tables>
    <Filter>
        <![CDATA[Employee.taaemployeeregionloc like '_USERINPUT_'
AND Account.owner = Employee.DN AND Account.erservice = Service.DN AND
Account.erparent = OrganizationalContainer.DN AND
Account.eraccountcompliance > '_USERINPUT_' ]]>
    </Filter>
</Query>
</Body>
<PageFooter>
    <PageNumber Format="10fN" show="YES"/>
</PageFooter>
</Page>
</AdHocReport>

```

In the body of the report XML, we focus on the following tags:

Columns	List of the columns shown on the report.
Tables	List of the Identity Manager database table in which the selected columns are located.
Filter	Search filter criteria used on the template report.

4. Edit the search filter criteria modifying the `_USERINPUT_` values with the following:

```

Employee.taaemployeeregionloc like 'Central Region'
Account.eraccountcompliance > '1'

```

The `Account.eraccountcompliance` uses one of the following values:

```

0 = unknown
1 = Compliant
2 = Disallowed
3 = Non-compliant

```

By defining a search filter criteria with `Account.eraccountcompliance` greater than 1, we are selecting only the Disallowed and non-compliant accounts.

Let us take a look at the details after updating the search filter criteria:

```

<![CDATA[Employee.taaemployeeregionloc like 'Central Region' AND
Account.owner = Employee.DN AND Account.erservice = Service.DN AND
Account.erparent = OrganizationalContainer.DN AND
Account.eraccountcompliance > '1' ]]>

```


5. Update the new file size information in the report.del file.

You have to update not only the REPORT_SIZE value but also the BLOB information reference “db2exp.001.0.1994/” shown in Example 6-4 on page 264.

In our implementation, the db2exp.001 file size after the search filter update is 1986 bytes, so the report.del has to be updated as shown in Example 6-6.

Example 6-6 Report.del file after the search filter update

```
51,"Account Report for Central Region","Designer","itim  
manager",1986,"db2exp.001.0.1986/","standard","Y","customReports","custom"
```

6. Import the report template using the following command:

```
db2 import from /tmp/reports/report.del of del lobs from /tmp/reports/  
modified by lobsinfile insert_update into REPORT
```

Where /tmp/reports/report.del is the output file for the result query updated with the new report file size and /tmp/reports/ is the path for the report template with the search filter modified.

The output of the db2 export command is shown below:

```
# db2 import from /tmp/reports/report.del of del lobs from /tmp/reports/  
modified by lobsinfile insert_update into enrole.report  
SQL3109N The utility is beginning to load data from file  
"/tmp/reports/report.del".
```

```
SQL3110N The utility has completed processing. "1" rows were read from  
the input file.
```

```
SQL3221W ...Begin COMMIT WORK. Input Record Count = "1".
```

```
SQL3222W ...COMMIT of any database changes was successful.
```

```
SQL3149N "1" rows were processed from the input file. "0" rows were  
successfully inserted into the table. "0" rows were rejected.
```

```
Number of rows read           = 1  
Number of rows skipped        = 0  
Number of rows inserted       = 0  
Number of rows updated        = 1  
Number of rows rejected       = 0  
Number of rows committed     = 1
```

The report template “Account Report for Central Region” has been updated and you can see it on the Identity Manager built-in designer.

7. In the Main Navigation Bar, click **Report**.
8. Click **Design Report** in the task bar.

9. Click **Account Report for Central Region**.
10. Click **Continue** and verify the new search filter as shown in Figure 6-34.

You Are Here: Report > Report Designer > Create Filter

Create Filter for Report

Function/Operator	Entity/User Input	Column	Condition
Select function ▼	Select entity ▼	Select Column ▼	
Select operator ▼			
Select function ▼	Select entity ▼	Select Column ▼	Select condition ▼

Select filter


- Employee.Employee Region Location Like 'Central. Region' AND
- Account.Owner Equals Employee.DN AND
- Account.Service Equals Service.DN AND
- Account.Parent DN Equals Organizational Container.DN AND
- Account.Account Compliance Greater Than '1'.

Select stylesheet to be applied :	Standard ▼
Select Report Category	Custom Reports ▼

Figure 6-34 Search filter designer view

11. Click **Cancel** to exit without saving any changes.

The “Account Report for Central Region” is now ready for all authorized administrators, and it can be tested.
12. In the Main Navigation Bar, click **Report**.
13. Click **Run Report** in the task bar.
14. Select **Custom Reports**.
15. Select **Account Report for Central Region** report.
16. Select Report Format, for instance, PDF, and click **Submit**. The report result is shown in Figure 6-35 on page 269.



Account Report for Central Region

Report Criteria

Date Printed	03-07-06
Time Printed	11:58
Time Zone	GMT-06:00

Employee Full Name	Employee Number	Organizational Container Name	Account User ID	Account Compliance	Service Name
Alessandro Faustini	43027	Austin CSC	afaustin	Noncompliant	Central Region Linux
Alma Ferman	35056	Mexico City CSC	aferman	Noncompliant	Central Region Linux
Lenin Sedano	39035	Denver CSC	isedano	Noncompliant	Central Region Linux
Ohmar Martinez	35041	St Louis CSC	omartine	Disallowed	Central Region Linux
Ozzie Smith	46078	Austin CSC	osmith	Disallowed	Central Region Linux

© 1999-2004 IBM. All Rights Reserved. Tivoli Identity Manager 4.6 1 of 1

Figure 6-35 TAA custom report

6.4 Automated operation report delivery

This section discusses the requirements, design considerations, and implementation of an automated operation report delivery customization for TAA's improved audit.

6.4.1 Requirements

TAA has recently come under increased scrutiny from its board members and from governmental agencies as it has become more profitable. Because of this, TAA is now forming a standing audit and compliance team which can field all requests from shareholders, board members, and auditors alike. With this action, TAA realizes that it must change how it currently reports and tracks information to make the team's mission successful.

TAA uses Identity Manager to create and maintain their accounts. Their Identity Manager administrators have performed these operations. And in our plan, these

tasks will also be performed by the delegated administrators to maintain business partner's accounts in the future. Currently, TAA uses Identity Manager predefined reporting functions to audit these administrative operations as you see in Table 6-6.

Table 6-6 Reports for monitoring account creation and password change activities

Purpose	Predefined Identity Manager reports
Identify password changes	Account operations Account operations performed by an individual
Identify operations performed by an Identity Manager user	Operation report

After reviewing the reporting functions, TAA found a few issues for their audit team:

- ▶ The reporting functions predefined in Identity Manager provide no ability to send the reports to members of the audit team automatically. Audit team members need to access Identity Manager periodically to generate the report of the administrative operations in the time period.
- ▶ The audit team members have to log in to Identity Manager as Identity Manager administrators in order to be granted privileged access rights to execute the reports. But, as administrators, they have all other Identity Manager privileges as well, which poses a security liability.

Because of these restrictions, TAA has decided to implement a new reporting function to audit administrative operations. The new reporting function has to meet the following requirements:

- ▶ The new reporting function should gather information about all administrative operations requested on Identity Manager at fixed intervals.
- ▶ The new reporting function should generate a report from the gathered information and send it to TAA's audit team automatically.
- ▶ The audit team members need no additional access rights on Identity Manager to review the reports received from the new reporting function.

6.4.2 Design considerations

Audit records detailing all Identity Manager operations are stored within the relational database. For further details, refer to the *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6*, SC32-1769. To gather information about administrative operations, we use an SQL statement that queries the audit records in the database.

Lifecycle rules give Identity Manager the ability to define an event or events that are triggered based on time intervals or immediately. The rules can also have matching criteria evaluated against an entity or entity type, to reduce the scope of the target entities the lifecycle rules should be performed against. Events in lifecycle rules are constructed with a operation workflow including custom JavaScript nodes or custom Java extension nodes to implement the business logic. To approach this solution, we construct a new operation workflow with custom nodes to generate a proper SQL statement and send it to the database.

A *work order node* in a workflow is used to send a notification to Identity Manager users. We use it in our workflow to notify TAA's audit team about the result of the database query. The content of the notification is formatted using a notification template within the work order node. The audit team members have to be registered in Identity Manager and own their e-mail address in order to receive the notification, but no additional access rights are required.

We also configure a post office mechanism for reducing the number of e-mail notifications the audit team receives. It can be configured to collect similar notifications for a period of time and combine multiple e-mails into one notification that is then sent to a user. You can optionally enable or disable this function in Identity Manager via the Web interface.

6.4.3 TAA's implementation

This section describes how to create the automated periodical operation report to meet TAA's audit requirements. The following sections describe:

- ▶ The workflow extension for sending a query to a database and receiving the results
- ▶ The operation workflow for generating a proper SQL statement and gathering information about operations done in Identity Manager
- ▶ The work order node for sending the operation report to the audit team members
- ▶ The lifecycle rule scheduling the report based on time intervals
- ▶ The post office configuration for aggregating the e-mails of the operation report

Workflow extension for database query

To access and query the database used by Identity Manager we have implemented two Java classes and both of them are packaged in `com/ibm/itim/custom/workflow`. Refer to Appendix D, "Additional material" on page 389 for detailed information about how to extract this code from the deliverable.

AbstractExtension class

The `AbstractExtension` class defines itself as an implementation of the Identity Manager `WorkflowApplication` interface. All of the other extension classes should extend this class. The primary purpose of this class is to provide a common interface that workflow extensions can use to report the failure of an extension activity. In order to simplify and make the code easier to read as well as maintain, all of the extensions have been created here where they can be accessed by all other extensions. Each of the methods comes in two types: one method that takes a `java.lang.Throwable` as an argument, and one method that does not.

DatabaseExtensions class

This class provides an extension activity: `sqlSelect`. It must be called with a string that already contains the complete SQL statement and the JNDI name of a JDBC™ data source as an input parameter. This data source must be configured in the application server, before you can use the extension.

The `sqlSelect` activity's SQL statement must be a single select. The activity creates a List of Lists as an output parameter. Each of the inner lists contains the data from one row of the select statement's result set. Each element of the inner list contains the data from one column of that row. The order of the data in the inner lists is determined by the order of the columns in the select statement. The data is left in whatever format was returned by the JDBC driver, and varies depending on the declared type of the column in the database. If the Java object returned by the JDBC driver was not able to be serialized, for example, if the column had a type of CLOB, then the value in the list will be null.

This extension reads the `enRole.properties` file in order to retrieve data required to connect to the application server and requests a JDBC connection from the data source's connection pool. It does this using the undocumented `PropertiesManager` Identity Manager class.

The JDBC connection is always set to automatically commit changes after each statement. The SQL statements are executed with a five minute timeout.

Any SQL exceptions thrown while obtaining the JDBC connection or while executing the SQL statement cause the workflow engine to throw a Corba transaction roll back exception. This causes the current workflow process to hang forever while the transaction is tried repeatedly.

To use this extension in an Identity Manager workflow, you must perform the following steps:

1. Compile the source files and create a jar archive file.

For brevity, the compiling steps are omitted. Refer to the documents located in the `<ITIM install directory>/extensions/examples` directory. After

successfully compiling the files, use the jar command to create a jar archive file from the compiled codes. We created the custom.jar file.

2. Add the jar file to the classpath.

The custom.jar must be added to Identity Manager's classpath. Repeat the following steps on each server of the WebSphere cluster.

- a. Make sure WebSphere server is stopped.
- b. Copy custom.jar file to the
<WAS_HOME>/installedApps/<CELL_NAME>/enrole.ear folder.
- c. Open the file.
<WAS_HOME>/installedApps/<CELL_NAME>/enrole.ear/app_web.war/
META-INF/MANIFEST.MF with a text editor and add custom.jar to the
Class-Path list.
- d. Start WebSphere server for the changes to take effect.

3. Register the extension with workflowextensions.xml.

Edit the workflowextensions.xml file in the <ITIM install directory>/data directory to register the *sqlSelect* extension. Add the following XML code into the activity list.

```
<ACTIVITY ACTIVITYID="sqlSelect" LIMIT="0">
  <IMPLEMENTATION_TYPE>
    <APPLICATION
CLASS_NAME="com.ibm.itim.custom.workflow.DatabaseExtensions"
METHOD_NAME="sqlSelect"/>
    </IMPLEMENTATION_TYPE>
    <PARAMETERS>
      <IN_PARAMETERS PARAM_ID="dataSourceName" TYPE="String"/>
      <IN_PARAMETERS PARAM_ID="sqlStatement" TYPE="String"/>
      <OUT_PARAMETERS PARAM_ID="results" TYPE="List"/>
    </PARAMETERS>
  </ACTIVITY>
```

After the registration, restart the enRole application.

Operation workflow for audit report

In this step, we create an operation for the ITIMAccount profile. This operation will be called with an ITIMAccount entity whose operations should be audited.

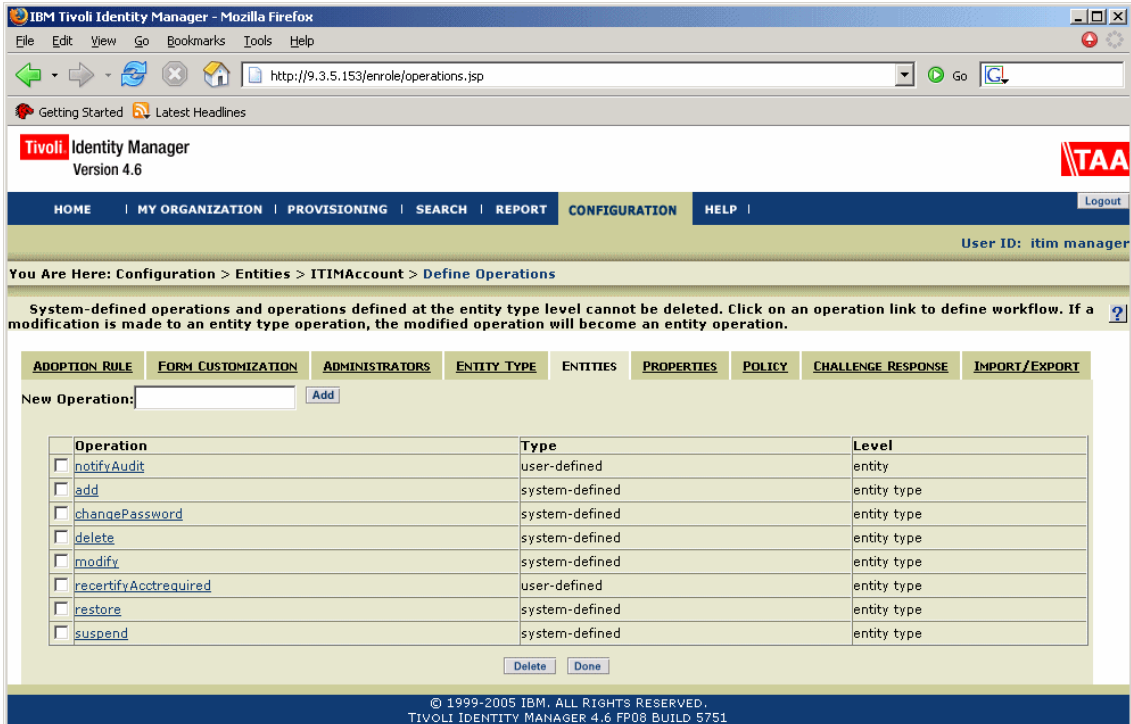


Figure 6-36 Operation *notifyAudit*

We created a *notifyAudit* workflow as shown in Figure 6-36.

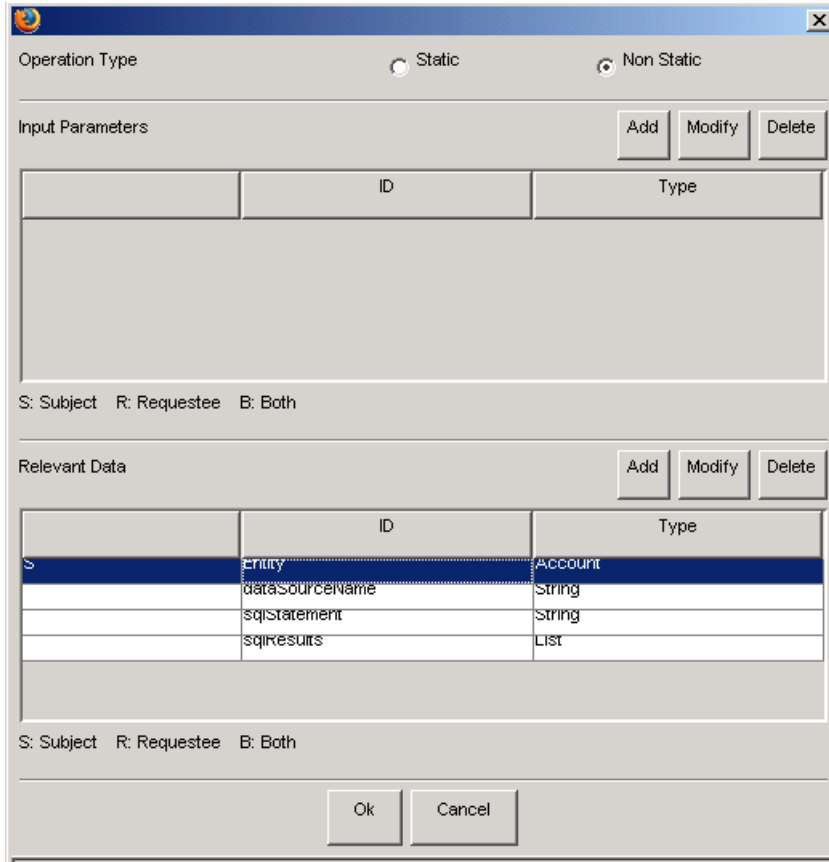


Figure 6-37 Property of notifyAudit workflow

This workflow uses three relevant data entries for its database query. The workflow Operation Type is defined as non-static. Figure 6-37 shows the configuration dialog. You also have to specify the following details:

- ▶ *dataSourceName* is a string relevant data to specify the JNDI name of the Identity Manager database. It must be defined with *enroleDataSource* as default value, which is configured in the application server at the install of Identity Manager. The configuration of the relevant data is shown in Figure 6-38 on page 276.

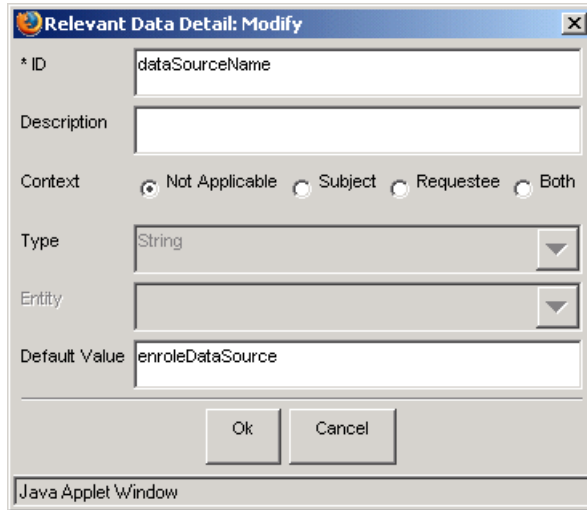


Figure 6-38 *dataSourceName* property

- ▶ *sqlStatement* is another string relevant data to be set to a proper SQL statement. It has no default value.

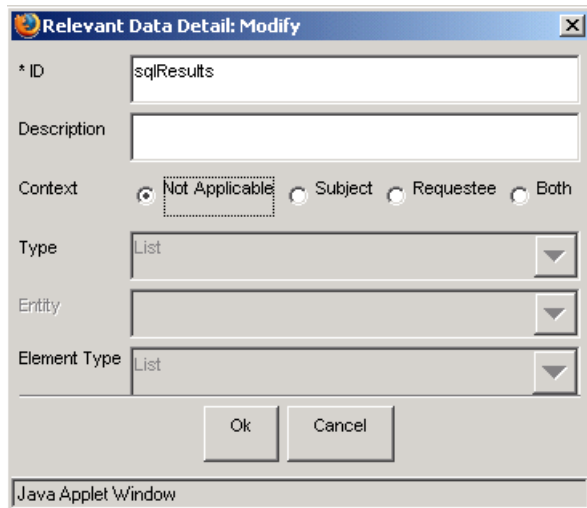


Figure 6-39 *sqlResults* property

- ▶ *sqlResults* is a List data to store the results of the database query. Each list represents an operation record and contains a list of data items for the record. Its configuration is shown in Figure 6-39.

Workflow nodes

The *notifyAudit* workflow has three nodes for its operation, which is shown in Figure 6-40.

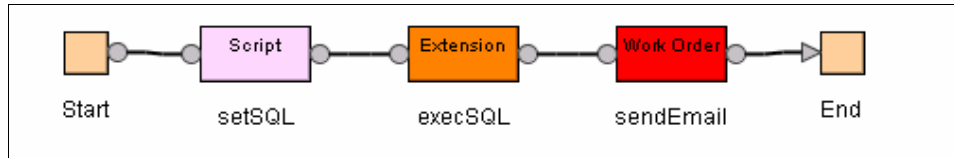


Figure 6-40 Workflow for operation *notifyAudit*

The nodes are:

▶ **setSQL script node**

In this node, the workflow generates a proper SQL statement to query audit records operated by the entity of the workflow execution.

▶ **execSQL extension node**

This node is used to send the query to the Identity Manager database with the *sqlSelect* extension implemented in the previous step.

▶ **sendEmail workorder node**

This node sends the audit report to the audit team using an e-mail.

Figure 6-41 on page 278 shows the configuration of the setSQL script node.

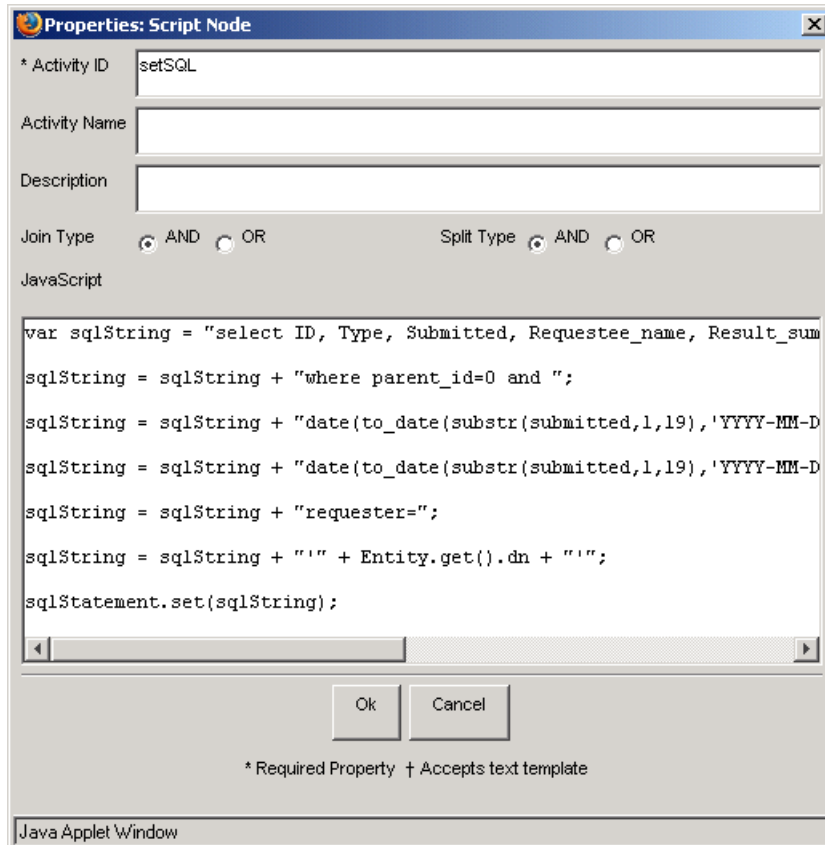


Figure 6-41 setSQL script node

The setSQL script node generates an SQL statement and sets the *sqlStatement* relevant data. For TAA's audit requirements, we defined the following report items:

- ▶ Process ID
- ▶ Process Type
- ▶ Submitted Date
- ▶ Requestee Name
- ▶ Result Summary

We also defined seven days as the interval of the periodical audit.

The following script is used to generate an SQL statement to query the report items and set it to *sqlStatement* relevant data within the setSQL script node.

```
var sqlString = "select ID, Type, Submitted, Requestee_name, Result_summary  
from enrole.process ";  
sqlString = sqlString + "where parent_id=0 and ";  
sqlString = sqlString + "date(to_date(substr(submitted,1,19),'YYYY-MM-DD  
HH24:MI:SS') + current timezone) > date(current timestamp - 8 days) and ";  
sqlString = sqlString + "date(to_date(substr(submitted,1,19),'YYYY-MM-DD  
HH24:MI:SS') + current timezone) < date(current timestamp) and ";  
sqlString = sqlString + "requester=";  
sqlString = sqlString + "'" + Entity.get().dn + "'";  
sqlStatement.set(sqlString);
```

Refer to *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6, SC32-1769*, for further information about the database tables.

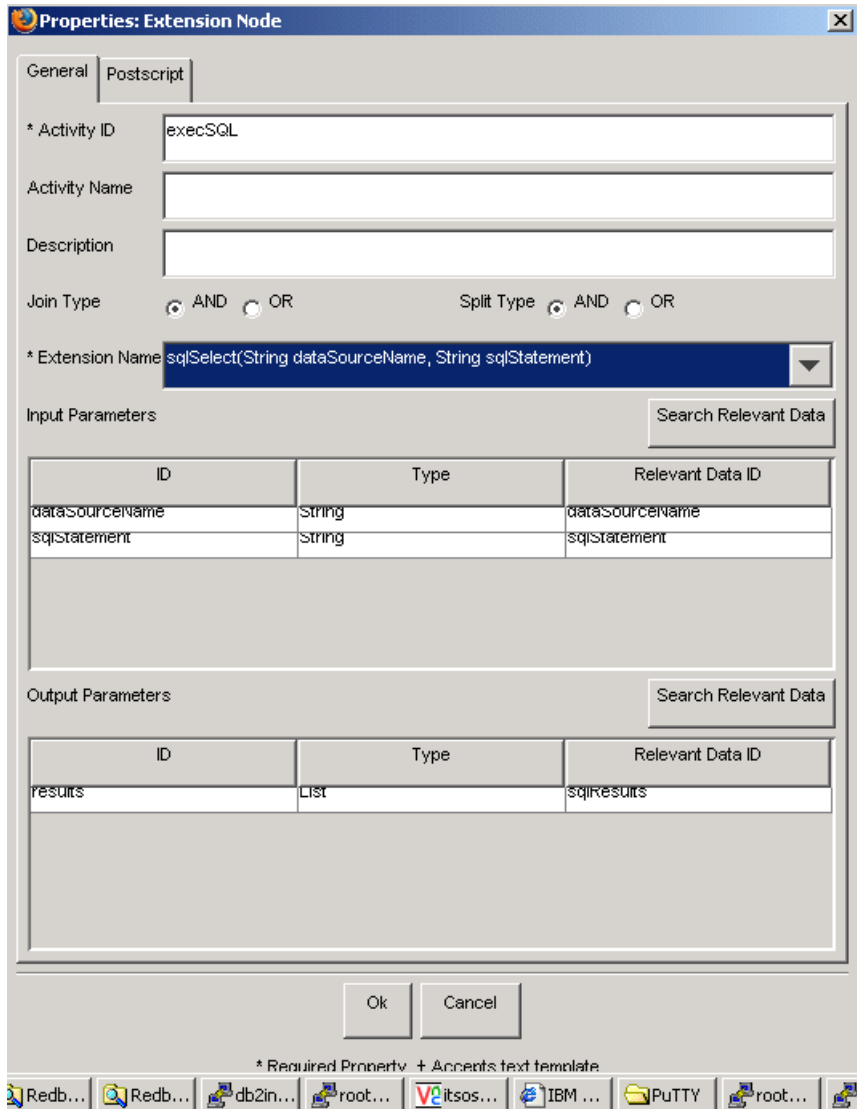


Figure 6-42 execSQL extension node

The configuration of the execSQL extension node is shown in Figure 6-42. We selected the Extension Name *sqlSelect*, which we registered before, and mapped the relevant data to parameters of the extension.

The sendEmail work order node creates a report from the results of the database query and sends it to audit team members. We used the HTML table option as the report format.

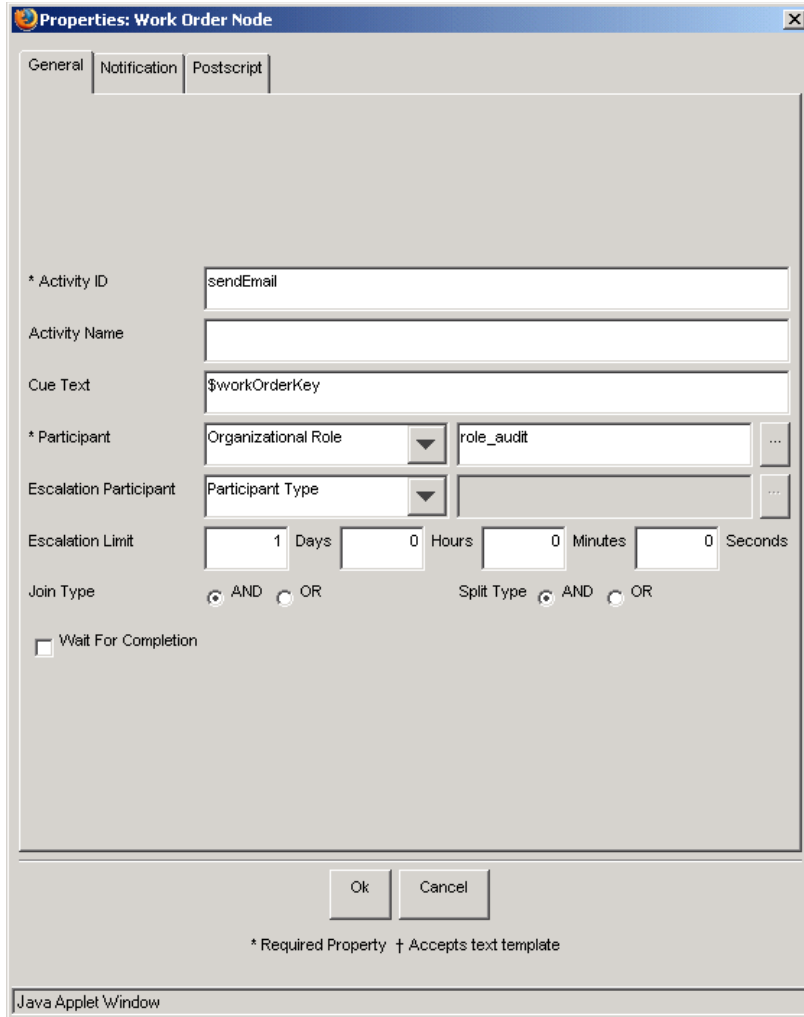


Figure 6-43 General information within sendEmail work order node

Figure 6-43 shows the configuration of the general tab within the sendEmail work order node. In this configuration, we specified audit team members with an Organizational Role *role_audit*. Each member of the audit team must be given the role in Identity Manager for receiving the audit report.

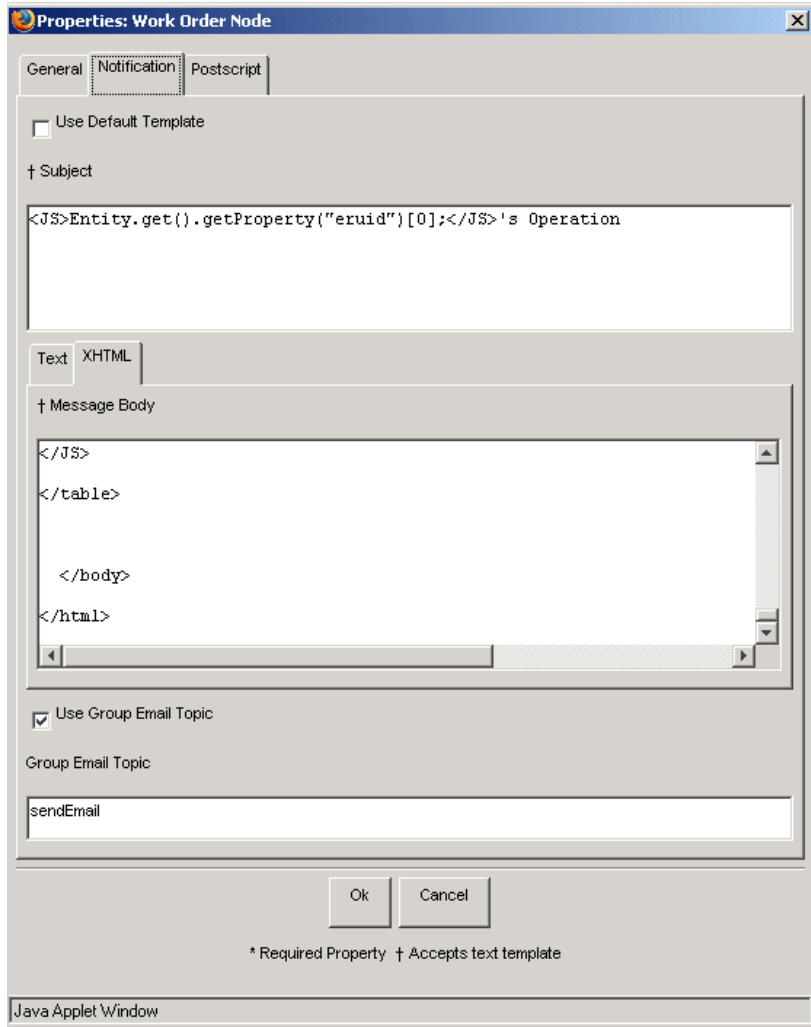


Figure 6-44 Notification template with sendEmail work order node

Figure 6-44 shows the notification template of the sendEmail work order node. We used a customized template for XHTML Message Body to format the audit report. This template is shown in Example 6-7.

Example 6-7 XHTML Message Body within sendEmail work order node

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```



```

<head>
  <title>${TITLE}</title>
  <meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />
</head>
  <body bgcolor="ffffff">
<table border="1" cellspacing="0" cellpadding="1">
<caption><JS>Entity.get().getProperty("eruid")[0];</JS>'s operation</caption>
<tr>
<th><RE key="processID" /></th>
<th><RE key="processType" /></th>
<th><RE key="timeSubmitted" /></th>
<th><RE key="requestedFor" /></th>
<th><RE key="resultSummary" /></th>
</tr>
<JS>
table="";
results=sqlResults.get();
for(j=0;j<results.length-1;j++){
result=results[j];
table= table + '<tr>';

table= table + '<td>';
table= table + result[0];
table= table + '</td>';

table= table + '<td>';
table= table + '<RE><KEY>' + 'processType.' + result[1] + '</KEY></RE>';
table= table + '</td>';

subTime=result[2].substring(0,4)+result[2].substring(5,7)+result[2].substring(8
,10);
subTime=subTime+result[2].substring(11,13)+result[2].substring(14,16)+"Z";
table= table + '<td>';
table= table + '<RE><KEY>readOnlyDateFormat</KEY><PARM>'+
Enrole.toMilliseconds(subTime) + '</PARM></RE>';
table= table + '</td>';

table= table + '<td>';
table= table + result[3];
table= table + '</td>';

table= table + '<td>';
table= table + '<RE><KEY>' + 'processState.' + result[4] + '</KEY></RE>';
table= table + '</td>';

table= table + '</tr>';
}
return table;
</JS>

```

```
</table>
</body>
</html>
```

Scheduling the lifecycle rule

Next, a lifecycle rule is constructed for the ITIMAccount entity. This rule is named *weeklyAuditReport* and it uses the *notifyAudit* operation defined earlier, as shown in Figure 6-45.

The screenshot shows a web-based configuration interface. At the top, a breadcrumb trail reads: "You Are Here: Configuration > Entities > ITIMAccount > Modify Lifecycle Rule". Below this is a header bar with the text "Complete both tabs and submit". The main content area has two tabs: "GENERAL" and "EVENT", with "EVENT" being the active tab. There are three main fields: "Lifecycle Rule Name" with a text input containing "weeklyAuditReport", "Lifecycle Rule Description" with a large empty text area, and "Operation" with a dropdown menu showing "notifyAudit". Each field has a red asterisk indicating it is required. At the bottom right, there are "Done" and "Cancel" buttons.

Figure 6-45 General information within the lifecycle rule for *notifyAudit*

This lifecycle rule is limited to administrative Identity Manager users by filtering the title attribute of the owner person. The owner person, who has the AdminOperator title, will be audited.

You Are Here: Configuration > Entities > ITIMAccount > Modify Lifecycle Rule

Provide filter, schedule and submit

GENERAL **EVENT**

Entity Filter

```
(${owner.title}=AdminOperator)
```

Schedule List

Weekly at 00:00 on Sunday

Add

Delete

Done **Cancel**

Figure 6-46 Event information within the lifecycle rule for notifyAudit

Figure 6-46 shows the configuration of the filter and schedule within the lifecycle rule. The execution of the lifecycle rule was scheduled at 00:00 every Sunday.

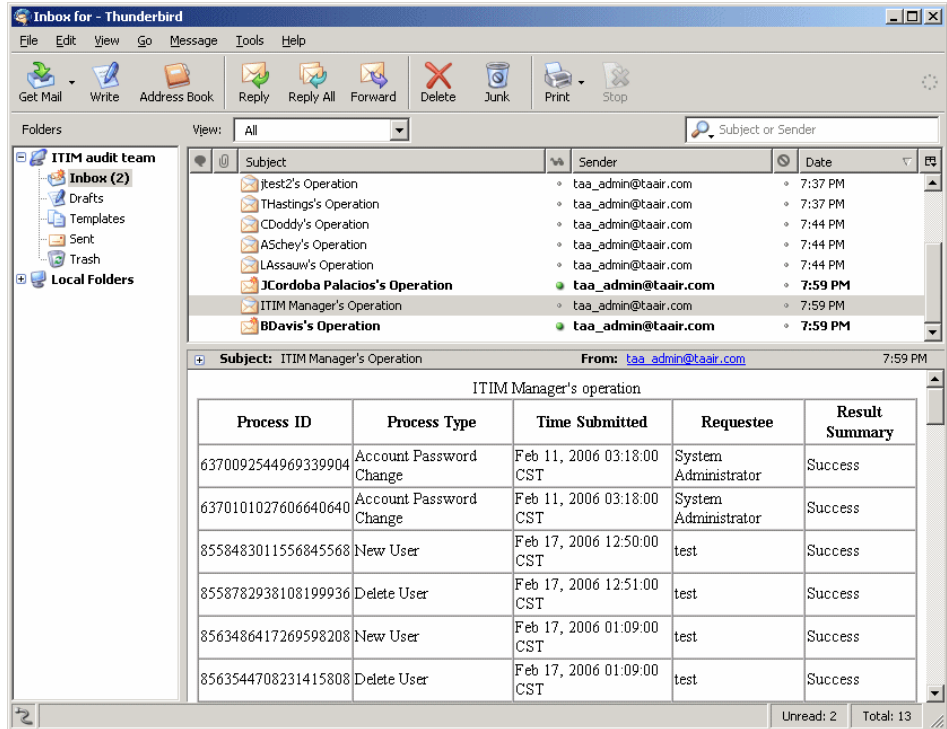


Figure 6-47 Audit report for each of the administrative operators

Every audit team member will receive the audit report e-mails as shown in Figure 6-47.

Post office configuration

In order to reduce the number of e-mail notifications an audit team member receives, we enable the post office configuration as shown in Figure 6-48 on page 287.

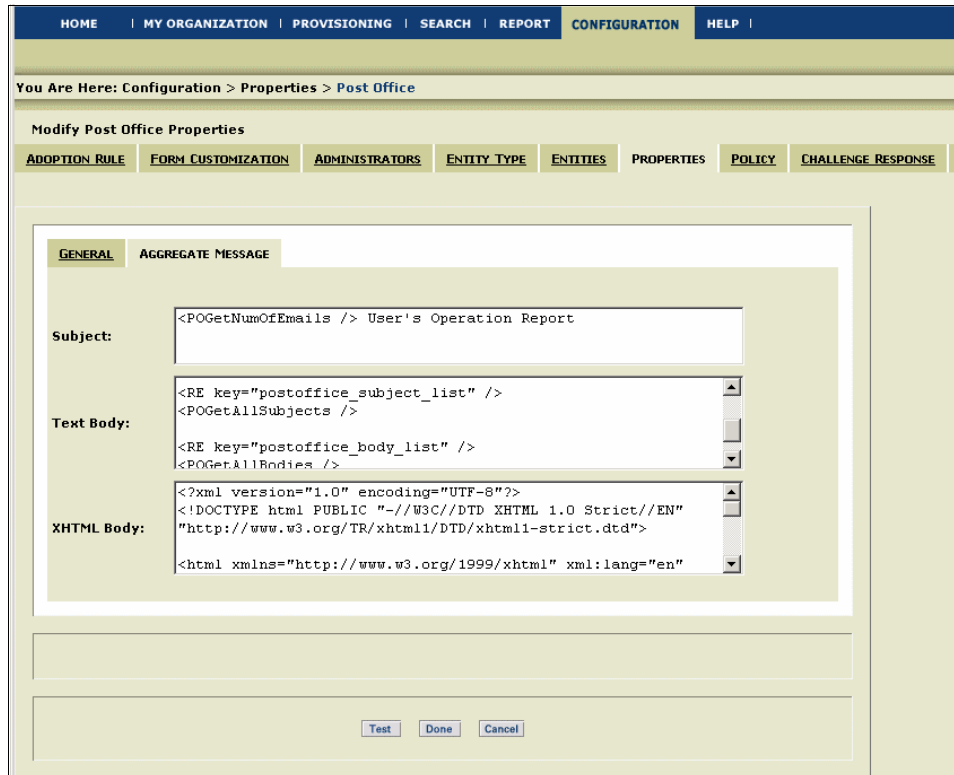


Figure 6-48 PostOffice configuration for aggregation of audit report

The post office can aggregate all the Text Message Bodies of the notifications and combine them into one Text or XHTML Message Body. We reconfigured the Text Message Body of the notification template within the sendEmail work order node in Figure 6-44 on page 282.

The configured notification template is shown in Example 6-8.

Example 6-8 Text Message Body within sendEmail work order node

```
&lt;table border="1" cellspacing="0" cellpadding="1"&gt;
&lt;caption&gt;<JS>Entity.get().getProperty("eruid")[0];</JS>'s
operation&lt;/caption&gt;
&lt;tr&gt;
&lt;th&gt;<RE key="processID" /&gt;&lt;/th&gt;
&lt;th&gt;<RE key="processType" /&gt;&lt;/th&gt;
&lt;th&gt;<RE key="timeSubmitted" /&gt;&lt;/th&gt;
&lt;th&gt;<RE key="requestedFor" /&gt;&lt;/th&gt;
&lt;th&gt;<RE key="resultSummary" /&gt;&lt;/th&gt;
&lt;/tr&gt;
```

```

<JS>
table="";
results=sqlResults.get();
for(j=0;j<results.length-1;j++){
result=results[j];
table= table + '<tr>';

table= table + '<td>';
table= table + result[0];
table= table + '</td>';

table= table + '<td>';
table= table + '<RE><KEY>' + 'processType.' + result[1] + '</KEY></RE>';
table= table + '</td>';

subTime=result[2].substring(0,4)+result[2].substring(5,7)+result[2].substring(8
,10);
subTime=subTime+result[2].substring(11,13)+result[2].substring(14,16)+"Z";
table= table + '<td>';
table= table + '<RE><KEY>readOnlyDateFormat</KEY><PARM>'+
Enrole.toMilliseconds(subTime) + '</PARM></RE>';
table= table + '</td>';

table= table + '<td>';
table= table + result[3];
table= table + '</td>';

table= table + '<td>';
table= table + '<RE><KEY>' + 'processState.' + result[4] + '</KEY></RE>';
table= table + '</td>';

table= table + '</tr>';
}
return table;
</JS>
</table>

```

In the post office configuration, we edited the XHTML Body template to combine the administrative operation report as in Example 6-9 on page 289:

Example 6-9 XHTML Message Body within Post Office Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>${TITLE}</title>
  <meta content="text/html; charset=UTF-8" http-equiv="Content-Type" />
</head>
<body bgcolor="ffffff">
<POGetAllBodies />
</body>
</html>
```

A sample aggregated audit report is shown in Figure 6-49.

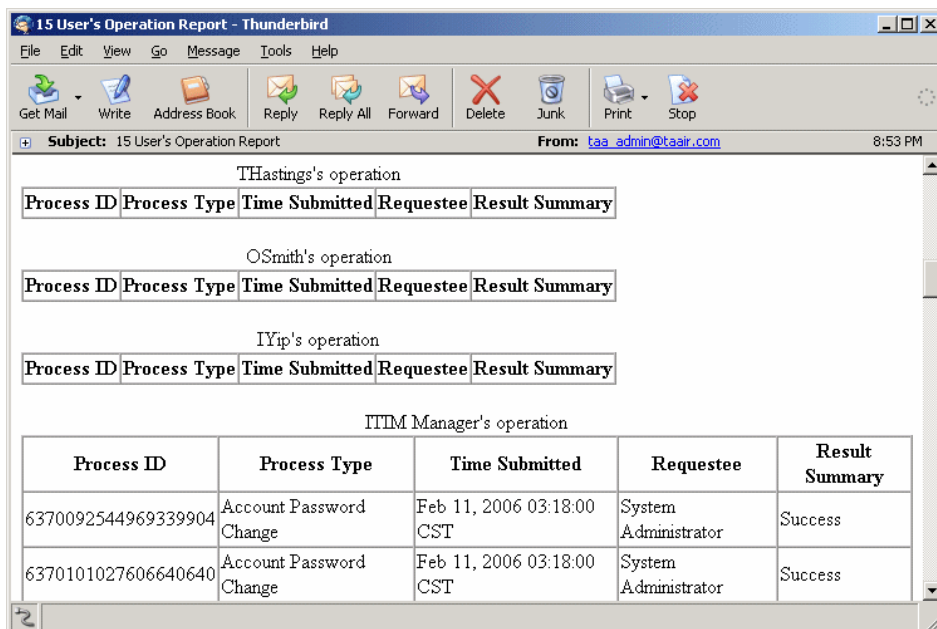


Figure 6-49 Aggregated audit report for the admin operators

This concludes the implementation of the audit reports.

6.5 Recertification process

To assist with managing the lifecycle for people, internal employees (TAA employees), and business partner employees, a recertification for accounts is required every three months in order to disable or remove accounts that do not have a business need to exist.

This recertification process helps TAA to strictly control the accounts needed for the company business and to reduce the annual software license expenses.

6.5.1 Requirements

TAA has decided to implement this certification process for internal employees and business partner employees owning accounts on the most critical systems and the most expensive software license applications. In TAA's case, the software license is charged on a per-user fee based on Linux accounts. If temporary access for this application is granted without a particular business reason, it can result in overlooked accounts for which TAA has to pay license fees. In addition to paying unnecessary license fees, an account that exists without a business reason on a critical system, or an account that is no longer required, can be a security risk and poses a problem that needs to be solved. As a result, a business reason has to be verified for every account every three months.

Initially, the account owner has to certify the need for access. The second approver is either the business partner domain administrator for each business partner domain or the person's supervisor for all TAA employees.

At the end of this particular TAA recertification process, a notification of the approval or rejection is sent to both the account owner and the Linux service.

6.5.2 Design considerations

Lifecycle rules give Identity Manager administrators the ability to define events to occur immediately or be triggered based on time intervals. The rules can also have matching criteria evaluated against an entity or entity type to reduce the scope of the target entities the lifecycle rules should be performed against.

Lifecycle rules can be defined as *global*, associated with an *entity type*, or associated with an *entity*.

In order to differentiate the lifecycle rule schedule for each regional center we define one lifecycle rule for each TAA regional center. These lifecycles rules will be associated with the account entity type.

6.5.3 TAA's implementation

All Linux user account holders must have their business need for an account periodically revalidated. In order to accomplish this, a lifecycle rule executes to allow the account owner to recertify the need to have Linux accounts. Three lifecycle rules will be implemented to maintain different schedules for each Linux service as shown in Table 6-7.

Table 6-7 Lifecycle rules and workflow for operation

Service name	Lifecycle rule schedule
West Region Linux	8 a.m. on the fifteenth day for each end of business quarter
Central Region Linux	8 a.m. on the sixteenth day for each end of business quarter
East Region Linux	8 a.m. on the seventeenth day for each end of business quarter

The approval for the second step is provided by the person's supervisor for TAA employees and the BP domain administrators for each business partner. The `certifyWestAccountNeeded`, `certifyCentralAccountNeeded`, and `certifyEastAccountNeeded` lifecycle rules are created for this purpose.

Table 6-8 shows the criteria used to define the workflow for operation and lifecycle rules.

Table 6-8 Lifecycle rules and workflow for operation

Workflow operation name	Lifecycle operation name	Target account	First approval	Final approval
<code>recertifyAcctRequired</code>	<code>certifyWestAccountNeeded</code>	TAA employee Business partner person	Account owner	Account owner supervisor or BP domain administrator
<code>recertifyAcctRequired</code>	<code>certifyCentralAccountNeeded</code>	TAA employee Business partner person	Account owner	Account owner supervisor or BP domain administrator
<code>recertifyAcctRequired</code>	<code>certifyEastAccountNeeded</code>	TAA employee Business partner person	Account owner	Account owner supervisor or BP domain administrator

In the following section, we describe the steps needed to implement the entity type operational workflow and the lifecycle rules.

Certification operation workflow

First, a lifecycle operation is constructed for the account entity type called *recertifyAcctRequired*, as shown in Figure 6-50.

The screenshot shows the 'Define Operations' page for the 'Account' entity type. The breadcrumb trail is 'You Are Here: Configuration > Entity Type > Account > Define Operations'. A message states: 'System-defined operations cannot be deleted. Click on an operation link to define workflow.' The page has several tabs: ADOPTION RULE, FORM CUSTOMIZATION, ADMINISTRATORS, ENTITY TYPE, ENTITIES, PROPERTIES, POLICY, CHALLENGE RESPONSE, and IMPORT/EXPORT. Below the tabs is a 'New Operation:' field with an 'Add' button. The main content is a table with three columns: Operation, Type, and Level. The table lists several operations, with 'recertifyAcctRequired' being user-defined and having an 'entity type' level.

Operation	Type	Level
<input type="checkbox"/> add	system-defined	entity type
<input type="checkbox"/> changePassword	system-defined	entity type
<input type="checkbox"/> delete	system-defined	entity type
<input type="checkbox"/> modify	system-defined	entity type
<input type="checkbox"/> recertifyAcctRequired	user-defined	entity type
<input type="checkbox"/> restore	system-defined	entity type
<input type="checkbox"/> suspend	system-defined	entity type

At the bottom of the table are 'Delete' and 'Done' buttons.

Figure 6-50 Operation *recertifyAcctRequired*

It is defined as an instance-based (non-static) operation, and so it has the account instance itself as an input parameter. The owner, the business partner domain administrators, or the account owner supervisor are derived in our custom lifecycle operation workflow itself.

The business logic of the operation is defined with one Work Order activity that e-mails either an affirmation or rejection of the Linux account in question. The affirmation or rejection of the account is sent both to the account owner and to the service's supervisor. If there are multiple accounts, then the post office feature, which aggregates e-mails of the same topic (requests for recertification in this case), will ensure that e-mail volumes aggregate the e-mails into one, and provide workflow request details in the same e-mail. The e-mail includes the account details and request information is sent to the approver as a To Do approval. The certification operation workflow is shown in Figure 6-51 on page 293. The decision has also been made to default all escalations to Identity Manager administrators; hence, notice that escalation participants on the relevant nodes have not been specified.

Note: Only *non-static operations* can be used by lifecycle rules. Static operations require input to them that is unavailable if used in the context of a lifecycle rule; hence, this has been disallowed. Refer to the online *IBM Tivoli Identity Manager Information Center Version 4.6*, SC23-5267, for more details about static versus non-static operations.

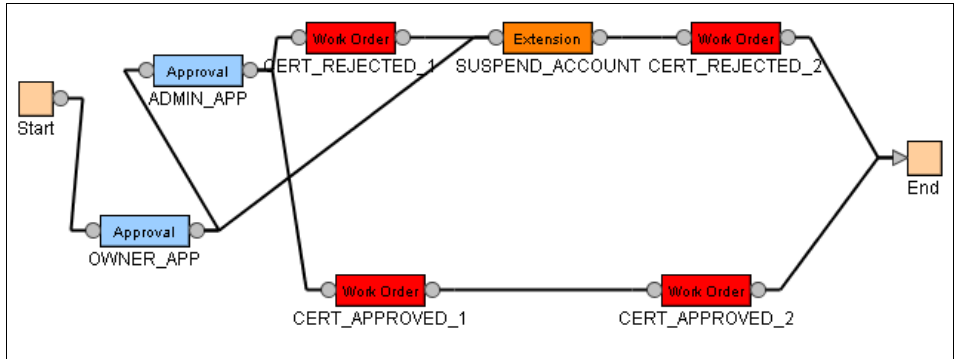


Figure 6-51 Workflow for operation recertifyAcctRequired

Figure 6-52 on page 294 shows the operation workflow properties, which are shown by clicking **Properties** in the workflow designer applet. Notice that the operation type is *non-static*, and, hence, by default, the Entity-relevant data entry will be present. The other relevant data entries need to be added using **Add** in the *relevant data* section of the window. Also, note that the Entity-relevant data entry is set to *Subject*. This is the default and cannot be changed within the context of the operation workflow applet. That is, it cannot be set to *Requestee*, *Both*, or *Not Applicable*. This is important to note for the purposes of the supervisor approval. We explain this when we discuss the ADMIN_APP business partner administrator domain approval node.

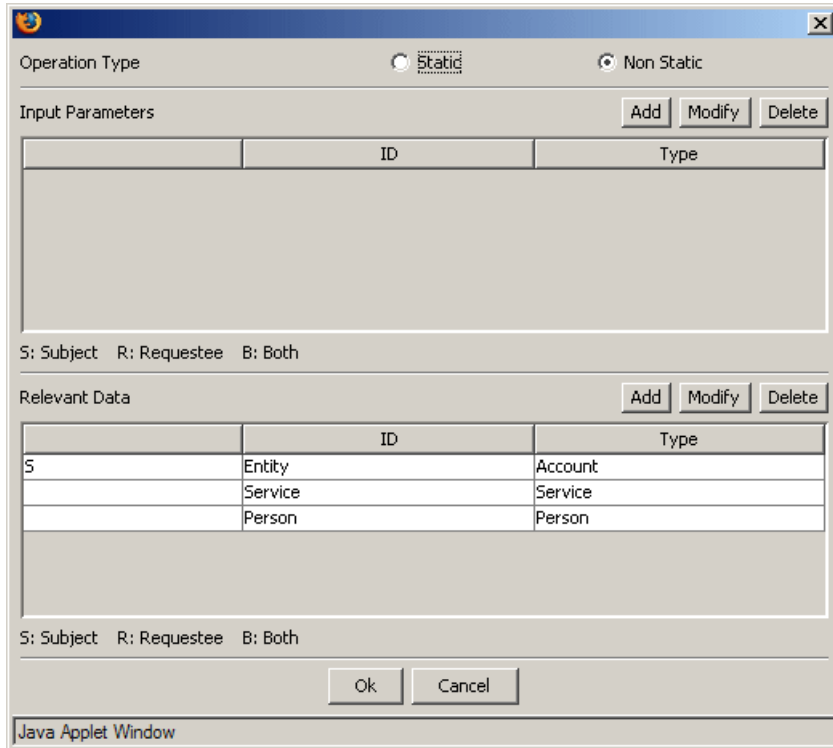


Figure 6-52 *recertifyAcctRequired operation workflow properties*

Figure 6-53 on page 295 shows the General tab of the OWNER_APP approval workflow node.

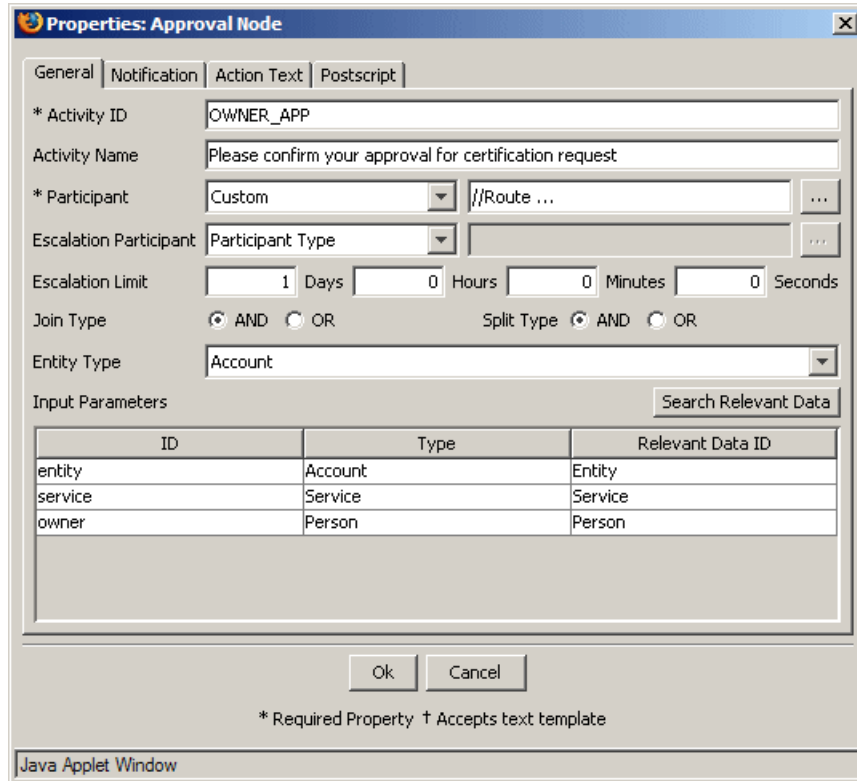


Figure 6-53 OWNER_APP approval workflow node General tab

The OWNER_APP approval workflow node specifies a custom participant to use as the approver. This uses the following script:

```
var account = Entity.get();
process.setRequesteeData(account);
var acctOwnerDN = account.getProperty ("owner")[0];
return new Participant(ParticipantType.USER, acctOwnerDN);
```

This is shown in Figure 6-54 on page 296. Essentially, this script sets the participant to be the person's account owner. The person's account owner participant is not available in operation workflows as an item in the participant drop-down list, but can be scripted as shown here. The first two lines of the script are needed to set the requestee object within the workflow to be the Entity account object. Identity Manager uses the requestee object to extract the person's account owner; hence, it is not able to resolve the participant without a value existing for the requestee. A simpler way is to set the Entity-relevant data object to *Both* instead of *Subject*, as shown and described in Figure 6-51 on page 293; but, as previously mentioned, this is impossible through the workflow

designer applet. Being able to set the Entity relevant data object to Both implies that it is both the subject and the requestee. This would negate the need to script the assignment of the requestee object.

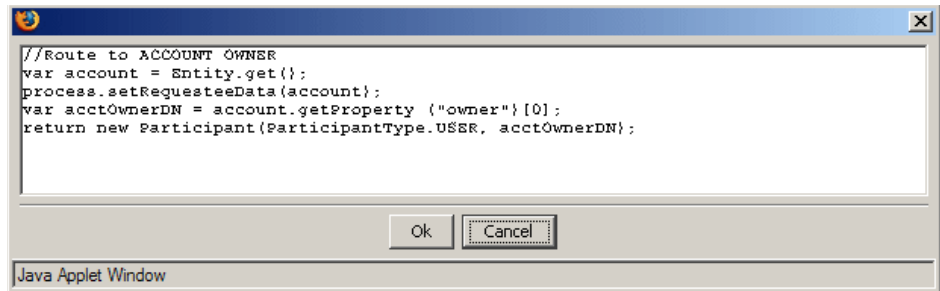


Figure 6-54 OWNER_APP approval workflow node custom participant script

Figure 6-55 on page 297 shows the Action Text tab of the same approval node. Note the nonstandard approval code, rejection code, cue text, and action text. This is for display purposes on the window seen by the approval participant (as shown in Figure 6-56 on page 298), as well as the notification e-mail sent to the participant. Note that the action text on the window refers to certification activities instead of the default approval and rejection activities shown by default for approvals. The text seen on the window is defined in the `CustomsLabels.properties` file in the data directory of the Identity Manager installation machine. The following lines have been added to this file:

```
cerAA=Re-certify account
cerAR=Do not re-certify account
cer_reqApprovalCue=Re-certify/Do not re-certify account
```

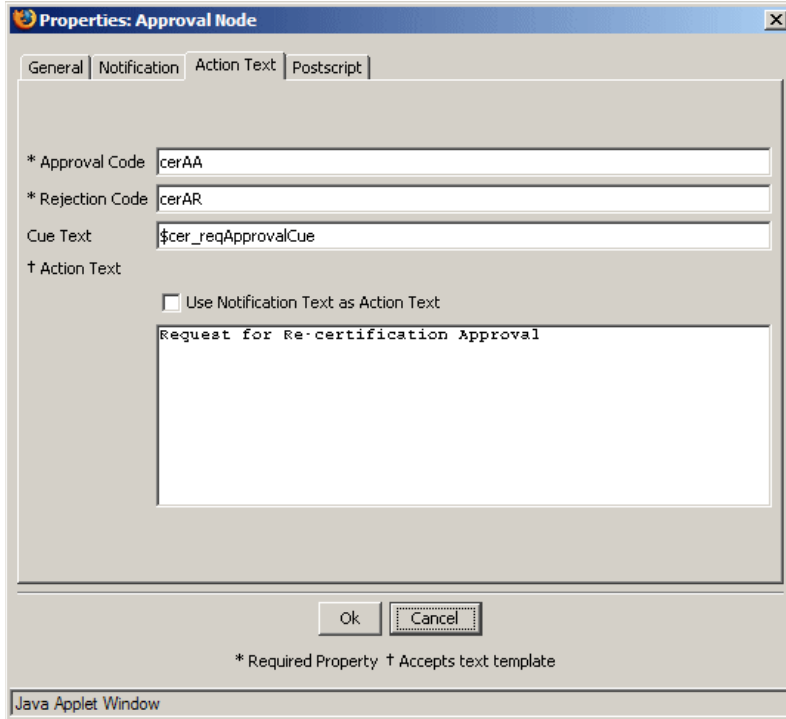


Figure 6-55 OWNER_APP approval workflow node Action Text tab

You Are Here: Home > To Do List > Approval

Re-certify/Do not re-certify account

Description	Request for Re-certification Approval
Request Id	9112986375782196421
Requestor	taa
Requestee	Alessandro Faustini
Subject	afaustin
Time Submitted	Mar 15, 2006 9:13 AM

[View Request Data](#)

Explanation

[Re-certify account](#) [Do not re-certify account](#) [Cancel](#)

Figure 6-56 Approval window for recertification of Linux account

Because of the nonstandard approval and rejection codes used by TAA in the approval nodes, the transition lines between the nodes have to cater for the different codes. This is shown in Figure 6-57 on page 299, which shows the properties of the transition line between the OWNER_APP approval node and the ADMIN_APP approval node. This transition is followed in the event that the service owner approves the recertification of the account in question.

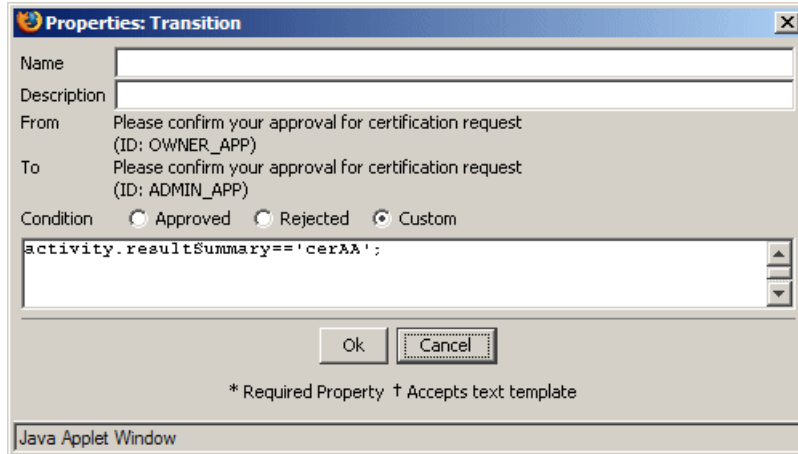


Figure 6-57 OWNER_APP approval workflow node to ADMIN_APP approval workflow node transition line

Figure 6-58 shows the properties for the transition line between the OWNER_APP approval node and the SUSPEND_ACCOUNT extension workflow node. In the event the service owner rejects the recertification request, this transition line will be followed.

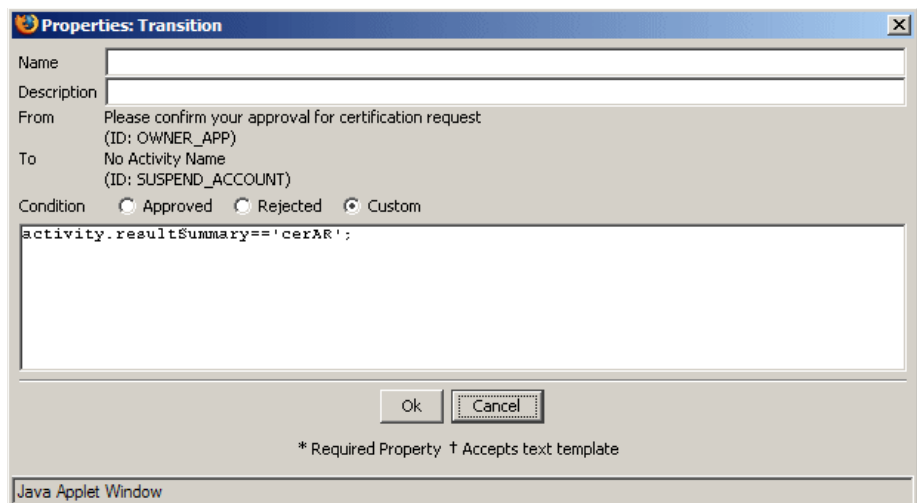


Figure 6-58 OWNER_APP approval workflow node to SUSPEND_ACCOUNT extension workflow node transition line

Figure 6-59 on page 300 shows the General tab of the ADMIN_APP approval workflow node.

Properties: Approval Node

General | Notification | Action Text | Postscript

* Activity ID: ADMIN_APP

Activity Name: Please confirm your approval for certification request

* Participant: Custom //Route ...

Escalation Participant: Participant Type

Escalation Limit: 1 Days 0 Hours 0 Minutes 0 Seconds

Join Type: AND OR Split Type: AND OR

Entity Type: Account

Input Parameters Search Relevant Data

ID	Type	Relevant Data ID
entity	Account	Entity
service	Service	Service
owner	Person	Person

Ok Cancel

* Required Property † Accepts text template

Java Applet Window

Figure 6-59 ADMIN_APP approval workflow node General tab

The ADMIN_APP approval workflow node specifies a custom participant to be used as the approver. This uses the script shown in Example 6-10.

Example 6-10 ADMIN_APP approval workflow node custom participant script

```
//Route to ADMIN DOMAIN Administrator
var account = Entity.get();
process.setRequesteeData(account);
var acctOwnerDN = account.getProperty("owner")[0];
var acctOwnerfilter = acctOwnerDN.substring(0,30);
var search = new PersonSearch();
var employeeObj =search.searchByFilter("taaBPEmployee", "(" + acctOwnerfilter +
")", 2 );
if (employeeObj.length != 0) {
    var domainDN = employeeObj[0].getProperty("erparent")[0];
    var domainfilter = domainDN.substring(0,30);
    var domainsearch = new ContainerSearch();
    var currentDomain = domainsearch.searchByFilter("AdminDomain", "(" +
domainfilter + ")", 2);
    var domainName = currentDomain[0].getProperty("ou")[0];
    var roles = (new RoleSearch()).searchByName(domainName);
```

```
    var roleObj = roles[0];
    return new Participant(ParticipantType.ROLE, roleObj.dn);
} else
return new Participant(ParticipantType.SUPERVISOR);
```

Essentially, this script defines the custom participant to be either the *business partner administrator* (if the account owner is an employee of a business partner) or the account owner's *supervisor* (if the account owner is an employee of TAA). Using *admin domains* allows TAA to create and administer different business partners as separate entities with their own policies, services, ACIs, and so on. In this implementation, each business partner is defined as an admin domain and can have its own administrator that cannot administer or view other admin domains' persons, accounts, services, ACIs, and so on. In this implementation, we have defined a dynamic role for each admin domain with a one-to-one mapping between the dynamic role name and the admin domain name. For instance, if the admin domain name is BP Admin Domain 1, the dynamic role name is BP Admin Domain 1. The approval request is sent to all members of the role mapped with the admin domain.

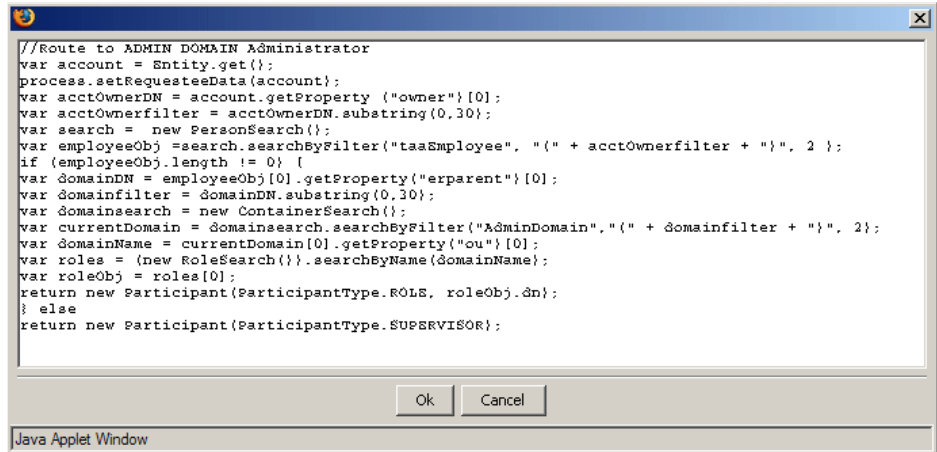
The reason the first two lines of the script, shown in Example 6-10 on page 300, need to be present is to set the requestee object within the workflow to be the Entity account object. Identity Manager uses the requestee object to extract the account owner (acctOwnerDN). The account owner DN is used to search the person's account owner object. The account owner DN is used to search for a person using a filter PersonSearch.searchByFilter. The arguments used for PersonSearch.searchByFilter are:

profileName	The name of the person profile to use, in the sample, this is taaBPEmployee.
filter	LDAP search filter that defines the criteria for returned containers to meet, in the sample, this is a substring of the account owner DN.
scope	Optional search scope, in the sample, 2 for SubTree Scope.

If the account owner is a TAA internal employee, then the person's supervisor is used as custom participant.

The person object returned by the person search is used to search the dynamic role mapped with the business partner admin domain. To search the role, we used the RoleSearch.searchByName() JavaScript extension with the ou value of the admin domain name as the argument.

See Figure 6-60 on page 302.



```
//Route to ADMIN DOMAIN Administrator
var account = Entity.get();
process.setRequesteeData(account);
var acctOwnerDN = account.getProperty("owner")[0];
var acctOwnerfilter = acctOwnerDN.substring(0,30);
var search = new PersonSearch();
var employeeObj = search.searchByFilter("taaEmployee", "(" + acctOwnerfilter + ")", 2);
if (employeeObj.length != 0) {
var domainDN = employeeObj[0].getProperty("exparent")[0];
var domainfilter = domainDN.substring(0,30);
var domainsearch = new ContainerSearch();
var currentDomain = domainsearch.searchByFilter("AdminDomain", "(" + domainfilter + ")", 2);
var domainName = currentDomain[0].getProperty("ou")[0];
var roles = (new RoleSearch()).searchByName(domainName);
var roleObj = roles[0];
return new Participant(ParticipantType.ROLS, roleObj.dn);
} else
return new Participant(ParticipantType.SUPERVISOR);
```

Ok Cancel

Java Applet Window

Figure 6-60 ADMIN_APP approval workflow node custom participant script

Figure 6-61 on page 303 shows the CERT_REJECTED_1 work order workflow node. This sends an e-mail to the account owner to notify him that the recertification process has completed, and that the account in question has not been recertified and, hence, has been suspended. Note that the Wait For Completion check box has not been enabled. This allows the process to complete without having to wait for the account owner to perform any action to the work order in their To Do list. The CERT_REJECTED_1 work order workflow node specifies the account owner as the custom participant to use as the approver.

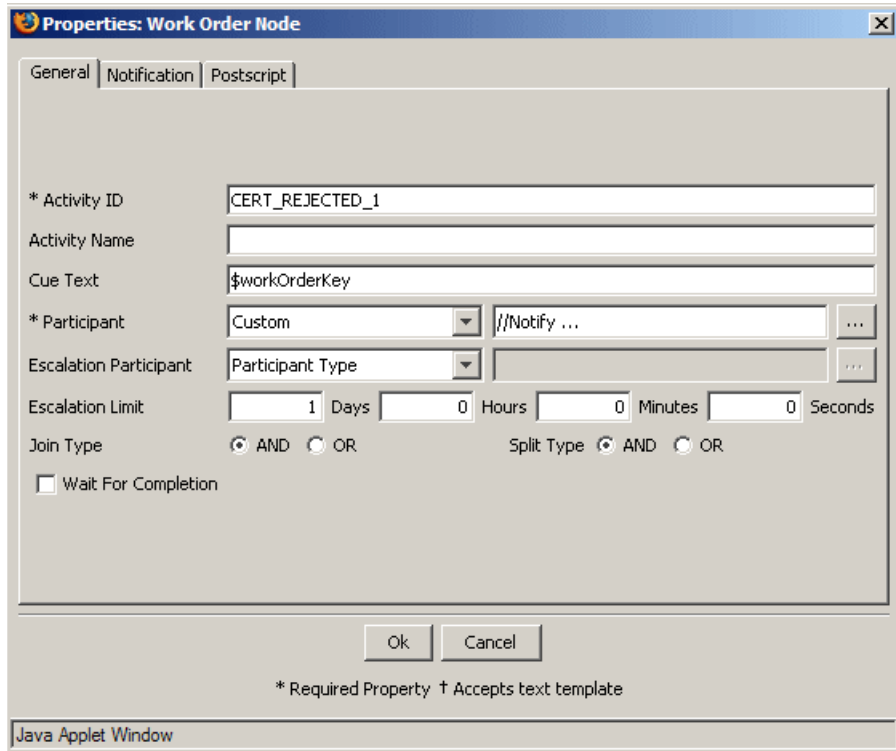


Figure 6-61 CERT_REJECTED_1 work order workflow node

Figure 6-62 on page 304 shows the CERT_APPROVED_1 work order workflow node. This sends an e-mail to the account owner to notify the account owner that the recertification process has completed, and that the account in question has been recertified. As in the CERT_APPROVED_1 work order workflow, note that the Wait For Completion check box has not been enabled. The CERT_APPROVED_1 work order workflow node specifies the account owner as the custom participant to use as the approver.

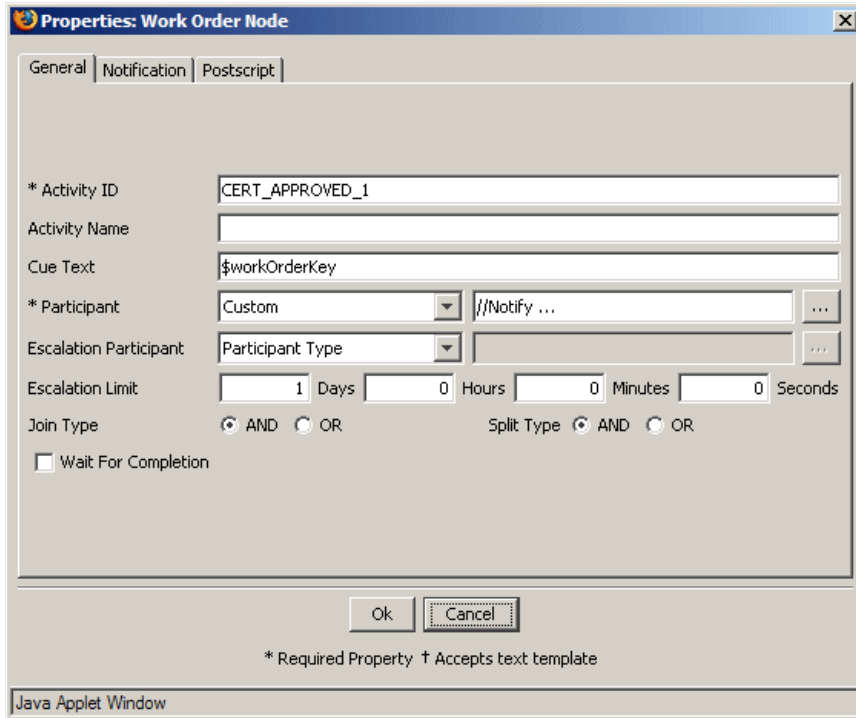


Figure 6-62 CERT_APPROVED_1 work order workflow node

Figure 6-63 on page 305 shows the CERT_APPROVED_2 work order workflow node. This sends an e-mail to the service owner to notify the service owner that the recertification process has completed and that the account in question has been recertified. Note that the Wait For Completion check box has not been enabled.

Properties: Work Order Node

General | Notification | Postscript

* Activity ID: CERT_APPROVED_2

Activity Name:

Cue Text: \$workOrderKey

* Participant: Service Owner

Escalation Participant: Participant Type

Escalation Limit: 1 Days 0 Hours 0 Minutes 0 Seconds

Join Type: AND OR

Split Type: AND OR

Wait For Completion

Ok Cancel

* Required Property † Accepts text template

Java Applet Window

Figure 6-63 CERT_APPROVED_2 work order workflow node

Figure 6-64 on page 306 shows the CERT_REJECTED_2 Notification tab of the work order workflow node. This sends an e-mail to the service owner to notify the service owner that the recertification process has completed and that the account in question has not been recertified and, hence, has been suspended.

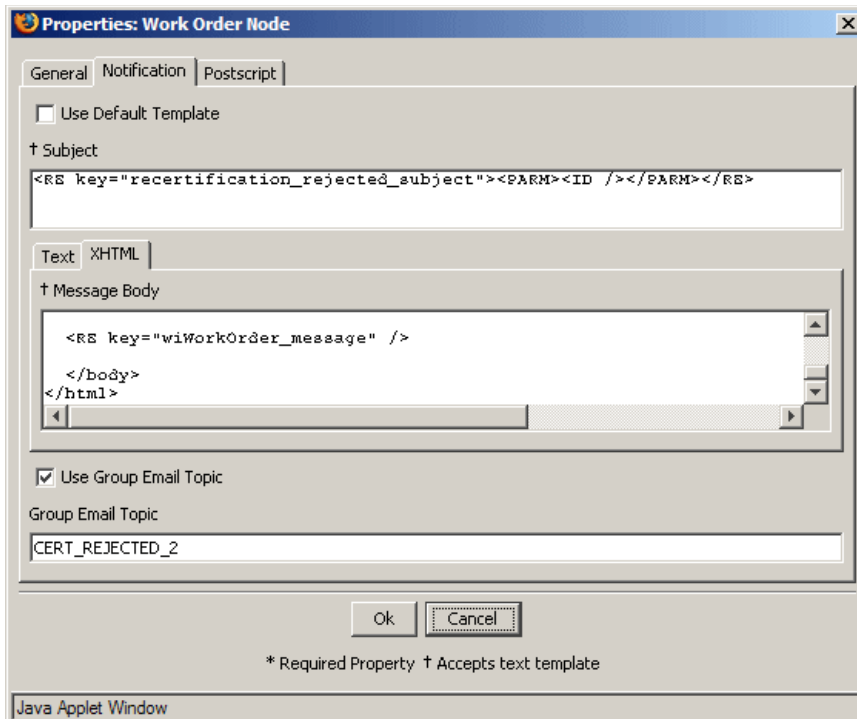


Figure 6-64 CERT_REJECTED_2 work order workflow node

Note that any transition lines shown in the workflow in Figure 6-51 on page 293 but not described above are standard transition lines generated by joining nodes within the workflow applet.

Certification lifecycle rules

A lifecycle rule for each TAA regional center has been constructed for the account Entity type. For the central region, it is named `certifyCentralAccountNeeded` and uses the operation `recertifyAcctRequired` defined earlier, as shown in Figure 6-65 on page 307.

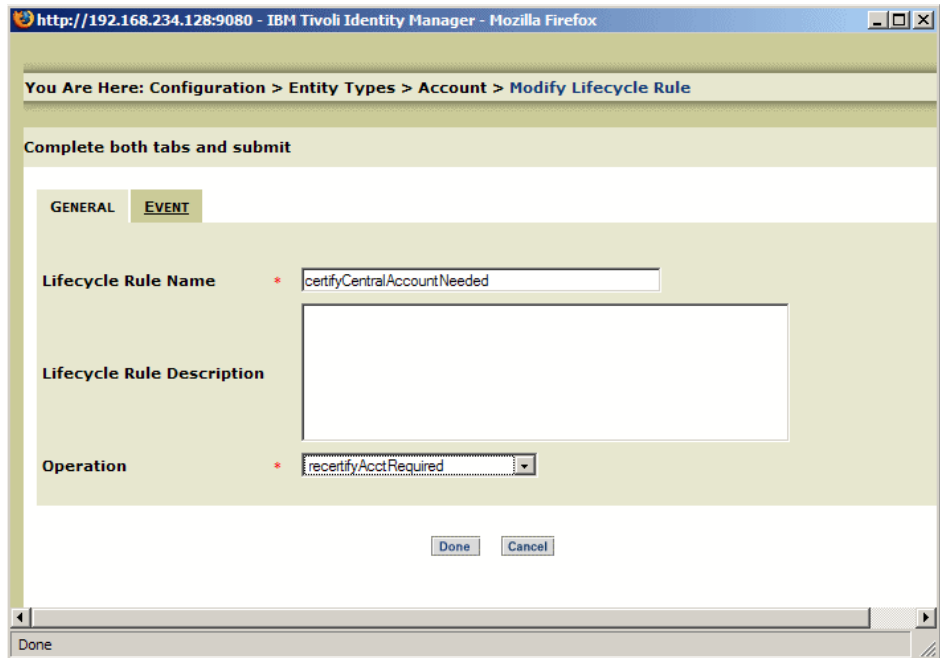


Figure 6-65 General tab within the lifecycle rule definition for Linux central region recertification

This lifecycle rule has been limited to the TAA employee and business partner owners of a Linux account on the central region center by virtue of the filter shown in Figure 6-66 on page 309.

Since the filter is based on attributes, only the attributes associated with the schema of the Entity, or Entity Type are legal. Tivoli Identity Manager provides two custom extensions to the filter syntax:

- ▶ Relationship expressions
 - Identity Manager relationship LDAP filter expressions, for example, Account Owner → owner relationship, or host service account → service relationship.
- ▶ System expressions
 - System LDAP filter expressions refer to a system object and a date keyword, which resolve to the current date and time.

These two LDAP expression types are evaluated at run time; note, however, a lifecycle (LDAP filter) rule will be syntactically checked on definition.

The relationship syntax is the following:

```
(${relationship.attribute}=value)
```

Relationship, in this syntax, can be:

- ▶ Parent
- ▶ Owner
- ▶ Organization
- ▶ Supervisor
- ▶ Sponsor
- ▶ Administrator
- ▶ Role
- ▶ Account
- ▶ Service

In the current implementation, we used a relationship expression that selects all Linux accounts on the Linux service “Central Region Linux”. The lifecycle filter is shown below:

```
(${service.erservicename}=Central Region Linux)
```

Note: The evaluation steps are important to keep in mind while composing relationship expressions. Most importantly, the related object type must be known in order to refer to a valid attribute name after the dot (.) operator to ensure the expressions are well formed, valid, and will produce a match. In the account entity and entity type, the relationship expression types permitted are only owner and service.

Refer to the online *IBM Tivoli Identity Manager Information Center Version 4.6*, SC23-5267, for more details about relationship expression evaluation.

Notice in Figure 6-66 on page 309 that the lifecycle rule also contains the schedule definition.

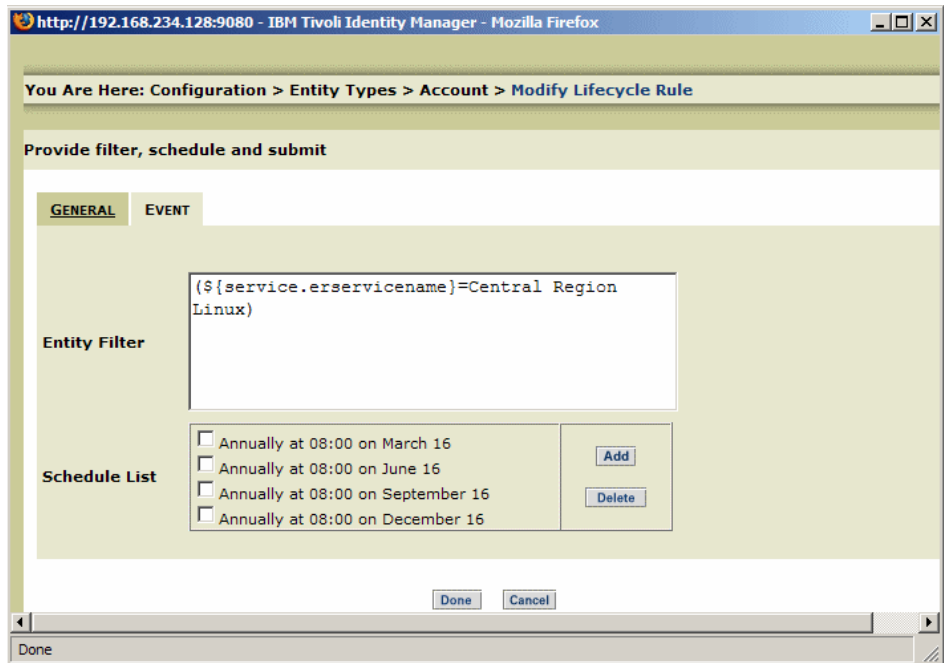


Figure 6-66 Event tab within the lifecycle rule definition for recertification

With this configuration, TAA will certify the need for Linux accounts each business quarter. This process will start at 8 a.m. on the sixteenth day of each quarter.

Similar lifecycle rules have been defined for each regional Linux service.

6.6 Conclusion

In conclusion of our advanced design business scenario at Tivoli Austin Airlines, you can see that the business needs were best served by using a phased approach to the different aspects of the project requirements. By utilizing an incremental approach to deploying capabilities, feedback on the deployed functionality could be used to drive future deployment activities, and the needs of the business and users can be best served.

To summarize:

▶ Phase I: High Availability (HA)

In phase I, the infrastructure was created for the required capabilities for a powerful and robust service offering. The components of this phase were:

- Application Server HA
- Relational Database Management System HA
- Directory Server HA

▶ Phase II: Application and processes

In phase II of the advanced design project, we capitalized on the Highly Available Identity Manager environment by offering applications and processes that helped satisfy the business requirements outlined for the project.

– Self-care application

Provided an application interface that greatly simplified the Identity Manager standard interface and abstracted all unnecessary tasks from the user

– Delegated administration (extranet)

Enabled business partners to take control of the administrative duties involved in day to day maintenance of their employees' identity management and, therefore, removed some of the overhead and responsibilities of TAA's security team

– Advanced custom report design

Enabled the capability to extend the standard reports included with Identity Manager in order to mitigate the need for a third-party reporting system

– Automated report delivery

Provided the capability to automatically run and deliver operational reports to the audit and reporting team

– Recertification process

Provided the capability to capture needed metrics on what application licensing was required and how many seats (licenses), and, therefore, enabled reducing licensing to only what is required



Part 3

Appendixes



Corporate policy and standards

Technology should not drive the corporate policy; it should be the other way around. Once you know what you need to protect and the potential threats and risks to those assets, you can start protecting them. First, all the threats and risks are classified in a study based on certain elements, such as:

- ▶ Direct financial loss
- ▶ Indirect financial loss (such as investigation, recovery, and so on)
- ▶ Loss of confidential information
- ▶ Liability
- ▶ Image impact (loss of goodwill, customer loyalty, and so on)
- ▶ Cost of risk mitigation or transfer
- ▶ Accepting residual risk

This study can process the same threats and risks applied to different assets, but concludes at a different level of liability, based on your particular business environment. Then, the decision has to be made: accept, mitigate, or transfer the risk. This process can be handled by external consultants, such as IBM Global Services, or by an internally appointed team. The process can use both formal and informal methods, but the result is usually a blend of these approaches. The threat identification, as well as this severity study, using a formal approach is done in conjunction with the organization by applying a standard and a proven methodology.

It is tempting to directly translate the threat analysis into a technical solution, but it should first lead to the corporate policy and standards. These documents will highlight the risks and present how they must be handled enterprise-wide.

The first document that you must write is, therefore, the *corporate policy document*. It must outline the high-level directions to be applied enterprise-wide. It is absolutely not technical; it is derived from the business of the enterprise and should be as static as possible, as seen in Figure A-1.

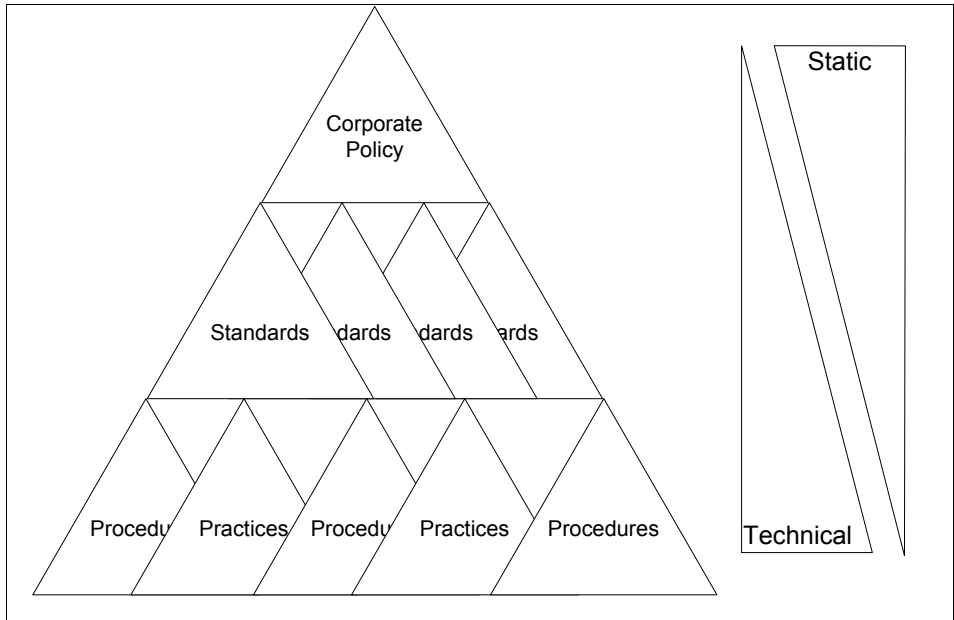


Figure A-1 Dynamics for policy, standards, practices, and procedures

Note: *Policies* is a very common term, and in many products you I find specific policy sections. These are the product-related policies that are covered in the practice or procedure documents. The corporate policy is unrelated to products and is a high-level document.

Standards, practices, and procedures

Standards are derived from the corporate policy. They are documents explaining how to apply the policy details in terms of *authentication*, *access control*, and so on. They explain how the policy must be applied. Changes in threats or major technology changes can impact them.

The standards are then mapped to *practices* or *procedures*.

The practices are descriptions of practical implementations of the standard on an operating system, application, or any other endpoint. They detail precise configurations, such as the services to be installed, the way to set up user accounts, or how to securely install software.

The procedures document the single steps to apply to requests, the approval flow, and the implementation flow. Such a procedure could be the request to access a specific set of sensitive data, where the approval path (system owner, application owners, and so on) and conditions (Virtual Private Network (VPN), strong authentication, and so on) are explained in detail.

Tip: Approval procedures are often implemented by sending e-mails or paperwork. The efficiency can be improved by using a computer to handle these repetitive tasks and ensure that changes within the company are applied quickly to the procedures. As we explain later, this can reduce human errors.

Practical example

Here is an example of how to define and implement a policy with procedures and practices.

The operations manager has reported an increased workload on the help desk due to problems caused by employees downloading non-business-related programs onto their systems.

The problems range from the introduction of viruses to disruption of business processes, with a real financial impact. To address this problem, upper management incorporated, in the corporate policy, the following directive, “The corporate assets may be used only to perform enterprise-related tasks”.

First, the policy must be communicated to all employees in the enterprise.

The standards for the networking part explain which services can be allowed on the employee’s computer. The practice then explains how to set up the Windows or

Linux clients according to the standards, and the procedures explain how to perform a request, the requirements, and the approval paths to get special services installed on your computer.

The existing clients are updated and controls are performed to verify the compliance, in addition to further auditing of the environment.

We summarize the five steps we went through in Figure A-2. It is a common approach adopted in many methodologies.

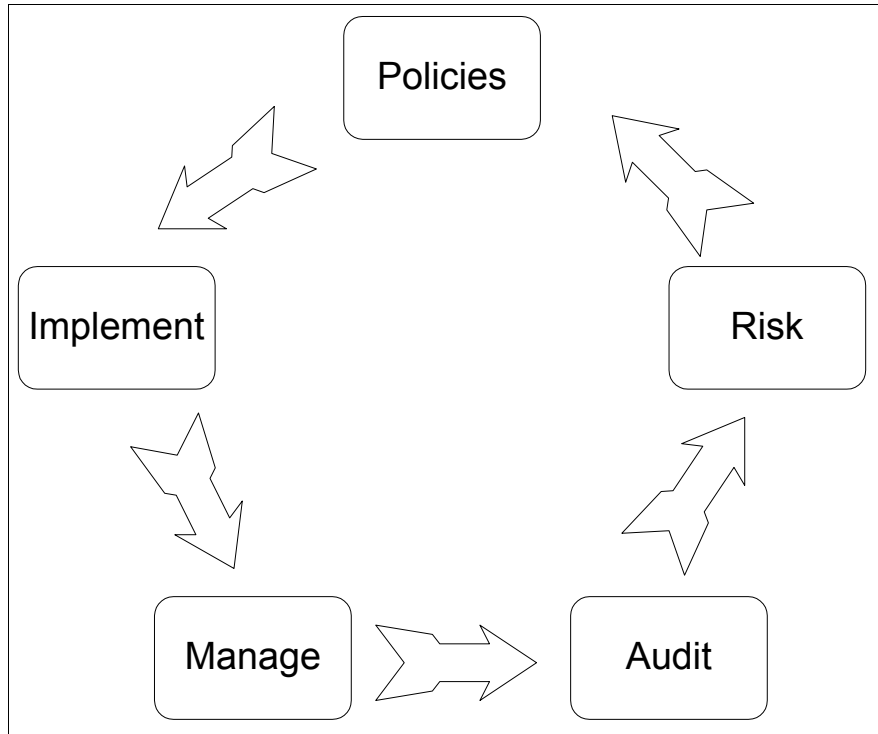


Figure A-2 The five steps in defining your IT security

External standards and certifications

The discussion about corporate policies suggests that internal business needs are the drivers for designing corporate policies. While this is true, there are a number of external factors that can change these business needs and policies. Some of these external pressures can be detailed enough to specify not only policies, but also standards and procedures.

We show examples of these external drivers in this section. The list is not exhaustive, nor is each description complete. We provide this list as a guide to the type of standards that might (or might not) apply to your organization, and, therefore, some of the external factors you must consider when creating policies.

Many organizations use these external standards as a guide to help them formulate their own corporate policies. It is not uncommon to find organizations using the ISO 17799 standards, but without having them externally audited and certified. These standards are seen as a good foundation for security.

Industry specific requirements

Some industry sectors have standards that are specific to that industry sector. Two examples are:

- ▶ **Identrus**

The Identrus standards are based upon standard PKI technologies for authenticating secure transactions. In addition to the technology layers, Identrus provides a complete infrastructure to help companies operate effectively and safely on the Internet, across economic and political boundaries, and with familiar business partners and new ones.

In addition to the technology standards and processes, Identrus describes an all important set of business rules, contracts, and liabilities that creates a trusted environment particularly for use in the banking and finance sectors.

- ▶ **CFR 21 Part 11**

CFR 21 Part 11 applies to electronic records that are created, modified, maintained, archived, retrieved, or transmitted under any record requirements covered by Food and Drug Administration regulations.

Any pharmaceutical company that wants to sell or market its products in America needs to abide by these rules. Corporate policies, standards, and processes need to reflect this requirement.

Product or solution certifications

Some products or solutions can be certified before use so that a potential purchaser has an understanding that the product or solution will fit the role for which it is needed.

Common Criteria

This is a set of tests originally based upon the US Orange book and European/Australian ITSEC evaluations. It is currently recognized by 14 countries. There are seven levels of tests. Evaluation Assurance Levels (EALs)

1–4 are typically used in the commercial areas, while the tests representing the higher EALs 5–7 are reserved for the security testing of highly secure environments.

CAPS UK

In addition to internationally recognized evaluations, there can be local evaluations that impact an organization. The UK Government's Communications-Electronic Security Group (CESG) have produced the Assisted Products Scheme in effort to help commercial product vendors produce cryptographic products suitable for use by the British government. It is called CAPS (CESG Assisted Product Scheme). CAPS is similar in purpose to the FIPS 140 (for the US and Canadian governments) and the Cryptographic Advisory Note (CAN) (for the Australian and New Zealand governments).

Nationally and internationally recognized standards

Some standards bodies publish broad general sets of standards that an organization can implement. These standards can be audited, and, hence, the organization can be sure they are complying.

BS7799

The most widely known standard. British standard (BS) 7799 and its international cousin ISO17799 are intended to serve as a single reference point for identifying a range of security controls, needed for most situations, where information systems are used in industry and commerce within large, medium, and small organizations. BS7799 was written in February 1995 and was updated in May 1999.

BS 7858

BS 7858 is just one example of some of the other less, well known standards that could affect security policy. Specifically, BS 7858 gives recommendations for the security screening of personnel to be employed in an environment where the security of people, goods, or property is a significant feature of the employing organization's operations.

Legal requirements

The laws of the country in which an organization operates are many and diverse. The application of the laws is variable from geography to geography, and it is good to be aware of the impact of them upon corporate security policies. Modern democracies are often fond of creating freedom of information laws. One of the problems with these laws is that the laws are directly contrary to the same democracies' wish to maintain the privacy of individual information.

Privacy law is, therefore, a growing area. Some examples are:

- ▶ UK Data Protection Act 1998

An act to make new provisions for the regulation of the processing of information relating to individuals, including the obtaining, holding, use, or disclosure of such information.

- ▶ European Data Directive 95/46/EC

This directive and others give direction to issues surrounding the protection of individuals with regard to the processing of personal data and the free movement of such data. The way they interact with national law must also be considered.

- ▶ US Health Insurance Portability and Accountability Act 1996

The Health Insurance Portability and Accountability Act 1996 (HIPAA) was passed by the United States Congress to ensure the privacy of an individual's private medical data.

Summary

Corporate policies must be thought of as business level requirements. They are primarily internal business drivers, but they can be impacted by external factors, so corporate policies I have to take these factors into account. Subsidiary standards and the procedures and practices that result are also produced.

Corporate policies should be relatively static and technology free, while standards, practices, and procedures can be more fluid and technology specific.



B

Source code

This appendix provides the complete JAVA source code for the following four classes:

- ▶ BulkFeedAdminDomain
- ▶ AdminDomainModelExtension
- ▶ AdminModelExtension
- ▶ AbstractExtension

Appendix D, “Additional material” on page 389 describes how to download an electronic jar file of these classes and how to use them in the context of our Tivoli Austin Airlines business case example.

BulkFeedAdminDomain.java

```
package com.ibm.itim.custom.dataservices;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.LineNumberReader;
import java.io.UnsupportedEncodingException;
import java.util.StringTokenizer;

import com.ibm.itim.common.AttributeValue;
import com.ibm.itim.common.AttributeValues;
import com.ibm.itim.dataservices.model.ModelCommunicationException;
import com.ibm.itim.dataservices.model.ModelCreationException;
import com.ibm.itim.dataservices.model.ObjectNotFoundException;
import com.ibm.itim.dataservices.model.PartialResultsException;
import com.ibm.itim.dataservices.model.CompoundDN;
import com.ibm.itim.dataservices.model.SearchParameters;
import com.ibm.itim.dataservices.model.SearchResults;
import com.ibm.itim.dataservices.model.domain.AdminDomain;
import com.ibm.itim.dataservices.model.domain.AdminDomainEntity;
import com.ibm.itim.dataservices.model.domain.AdminDomainFactory;
import com.ibm.itim.dataservices.model.domain.DynamicRole;
import com.ibm.itim.dataservices.model.domain.DynamicRoleFactory;
import com.ibm.itim.dataservices.model.domain.OrganizationalContainerEntity;
import com.ibm.itim.dataservices.model.domain.OrganizationalContainerSearch;
import com.ibm.itim.dataservices.model.domain.DirectorySystemSearch;

/**
 * @author tsanui
 *
 * BulkFeedAdminDomain will import a list file of Business Partner Companies
 * and create a new Admin Domain for each Business Partner under the specified
 * organizational unit. This class will also create a dynamic role with ldap
 * filter "(title=AdminOperator)" on each of the Admin Domains.
 */
public class BulkFeedAdminDomain {

    public static final String DELIMITER = "|";

    /**
     * Command line interface.
     *
     * @param argv
     *         Arguments of the command.
     */
}
```



```

* First argument is required and it provides a name of the file to be
* imported.
*
* Second argument is optional and it provides a character for delimiter in
* the import file.
*
*/
public static void main(String[] argv) {

    if (argv.length < 1 || argv.length > 2) {
        System.out
            .println("Usage: java BulkFeedAdminDomain <importFileName>
[<delimiter>] \n");
        System.out
            .println("Usage: Where <importFileName> is required and refers
to a file with company information to import\n");
        System.out
            .println("Usage: and <delimiter> is optional with default of
\"|\"and is the delimiter for fields within the text file.\n");
        System.out
            .println("Usage: Example - java BulkFeedAdminDomain
./companies.txt | ");
        System.exit(0);
    }

    String delimiter;
    if (argv.length == 2) {
        delimiter = argv[1];
    } else {
        delimiter = DELIMITER;
    }

    int total = 0;
    long beginTime = System.currentTimeMillis();

    BufferedReader in;
    try {
        /*
        * Setting up File for Reading
        */
        FileReader importFileReader = new FileReader(argv[0]);
        in = new LineNumberReader(importFileReader);
        String tmpString;
        StringTokenizer strTokens;

        while ((tmpString = in.readLine()) != null) {
            strTokens = new StringTokenizer(tmpString, delimiter);
            int count = strTokens.countTokens();

```

```

        if (count < 3) {
            System.out
                .println("----ERROR---- Importing line (wrong token
size) Line: "
                    + tmpString + "\n Tokens: " + count);
            System.out
                .println("File format is: Parent OrgUnit Location
Code|Business Partner Company Name|Business Partner Company Code\n");
            System.exit(0);
        }

        /*
         * Begin setting variables with tokens.
         *
         * SAMPLE
         *
         * "1000|Tivoli Austin Travel Agency|1001"
         *
         * First Token is Location of the parent organizational unit for
         * the new Admin Domain.
         */
        String parentOULocation = strTokens.nextToken().trim();

        /*
         * Second Token is Business Partner Company Name.
         */
        String companyName = strTokens.nextToken().trim();

        /*
         * Third Token is Business Partner Company Code.
         */
        String companyCode = strTokens.nextToken().trim();

        /*
         * Create AttributeValue pairs for Admin Domain needs.
         */
        AttributeValues attrs = new AttributeValues();
        AttributeValue attrOU = new AttributeValue("ou", companyName);
        attrs.put(attrOU);
        AttributeValue attrCode = new AttributeValue("description",
            companyCode);
        attrs.put(attrCode);

        AdminDomain myAdminDomain = new AdminDomain(attrs);

        /*
         * Look up the parent organizational unit.
         */

```

```

        OrganizationalContainerSearch myOrgContainerSearch = new
OrganizationalContainerSearch();
        CompoundDN searchContext = new DirectorySystemSearch()
            .lookupDefault().getLogicalNameContext();
        SearchResults containerSearchResults = myOrgContainerSearch
            .searchByFilter(searchContext, "(l=" + parentOULocation
                + ")", new SearchParameters());
        if (containerSearchResults.size() != 1) {
            System.out.println("Failed to search a parent OU for "
                + companyName);
            containerSearchResults.close();
            continue;
        }
        OrganizationalContainerEntity myOrgContainerEntity =
(OrganizationalContainerEntity) containerSearchResults
            .iterator().next();
        containerSearchResults.close();

        /*
         * Create the new Admin Domain.
         */
        AdminDomainFactory myAdDomFac = new AdminDomainFactory();
        AdminDomainEntity myAdminDomainEntity = myAdDomFac.create(
            myOrgContainerEntity, myAdminDomain);
        System.out.println("Created Admin Domain: " + companyName);

        /*
         * Create the new dynamic role with ldap filter
         * "(title=AdminOperator)"
         */
        AttributeValues roleAttrs = new AttributeValues();
        AttributeValue roleNameAttr = new AttributeValue("errolename",
            companyName);
        roleAttrs.put(roleNameAttr);
        DynamicRole myDynamicRole = new DynamicRole(roleAttrs);
        myDynamicRole.setDefinition("(title=AdminOperator)");
        myDynamicRole.setScope(DynamicRole.SINGLE_LEVEL_SCOPE);
        DynamicRoleFactory myDynRoleFac = new DynamicRoleFactory();
        myDynRoleFac.create(myAdminDomainEntity, myDynamicRole);
        System.out.println("Created Organizational Role: "
            + companyName);

        total++;
    }
} catch (UnsupportedEncodingException e) {
    System.out.println("----ERROR---- " + e.getMessage());
} catch (FileNotFoundException e) {
    System.out.println("----ERROR---- " + e.getMessage());
} catch (IOException e) {

```

```

        System.out.println("----ERROR---- " + e.getMessage());
    } catch (ObjectNotFoundException e) {
        System.out.println("----ERROR---- " + e.getMessage());
    } catch (PartialResultsException e) {
        System.out.println("----ERROR---- " + e.getMessage());
    } catch (ModelCommunicationException e) {
        System.out.println("----ERROR---- " + e.getMessage());
    } catch (ModelCreationException e) {
        System.out.println("----ERROR---- " + e.getMessage());
    }
}

/*
 * Finished, print statistics
 */
System.out.println("\n-----COMPLETED SUCCESSFULLY-----\n");
System.out.println("Companies imported: " + total);
System.out.println("Total time(sec): "
    + ((System.currentTimeMillis() - beginTime) / 1000));
return;
}
}

```

AdminDomainModelExtension.java

```
package com.ibm.itim.custom.fesiextensions;

import com.ibm.itim.dataservices.model.DistinguishedName;
import com.ibm.itim.dataservices.model.ModelCommunicationException;
import com.ibm.itim.dataservices.model.ObjectNotFoundException;
import com.ibm.itim.dataservices.model.domain.AdminDomain;
import com.ibm.itim.dataservices.model.domain.AdminDomainEntity;
import com.ibm.itim.dataservices.model.domain.AdminDomainSearch;
import com.ibm.itim.script.ScriptEvaluatorException;

import FESI.jslib.JSEException;
import FESI.jslib.JSEExtension;
import FESI.jslib.JSFunctionAdapter;
import FESI.jslib.JSGlobalObject;
import FESI.jslib.JSObject;

/**
 * @author tsanui
 *
 * A FESI extension that allows scripts to add an administrator to an Admin
 * Domain on ITIM.
 *
 */
public class AdminDomainModelExtension implements JSEExtension {

    private static String DOMAIN_OBJECT = "domainEntity";

    private static String RE_INVALID_ARGUMENTS =
"com.ibm.itim.script.ScriptEvaluator.CUSTOM_ERROR_INVALID_FUNCTION_ARGUMENTS";

    /**
     * (non-Javadoc)
     *
     * @see
     FESI.jslib.JSEExtension#initializeExtension(FESI.jslib.JSGlobalObject)
     */
    public void initializeExtension(JSGlobalObject go) throws JSEException {

        /**
         * Create an object in the root level JavaScript namespace.
         * "AdminDomain" is a constructor that creates a new instance of the
         * AdminDomainWrapper class.
         */
        go.setMember("AdminDomain", new AdminDomainWrapper());
    }
}
```

```

/**
 * @author tsanui
 *
 * Provide "addAdministrator" function to AdminDomainWrapper object.
 *
 */
private class AddAdministratorFunction extends JSFunctionAdapter {

    /**
     * (non-Javadoc)
     *
     * @see FESI.jslib.JSFunction#doCall(FESI.jslib.JSObject,
     *      java.lang.Object[])
     */
    public Object doCall(JSObject thisObject, Object[] args)
        throws JSEException {
        if (args.length != 1 || !DistinguishedName.isDN(args[0].toString()))
        {
            Object errorValues[] = { "addAdministrator(PersonDN)" };
            throw new JSEException("Argument error",
                new ScriptEvaluatorException(RE_INVALID_ARGUMENTS,
                    errorValues));
        }

        AdminDomainEntity domainEntity = (AdminDomainEntity) thisObject
            .getMember(DOMAIN_OBJECT);
        DistinguishedName personDN = new DistinguishedName(args[0]
            .toString());
        try {
            AdminDomain domain = (AdminDomain) domainEntity
                .getDirectoryObject();
            domain.addAdministrator(personDN);
            domainEntity.update();
        } catch (ModelCommunicationException mce) {
            throw new JSEException("Failed to add domain administrator "
                + personDN.getAsString(), mce);
        } catch (ObjectNotFoundException onfe) {
            throw new JSEException("Failed to add domain administrator "
                + personDN.getAsString(), onfe);
        }

        return null;
    }
}

/**
 * @author tsanui
 *
 * Provide a wrapper feature for AdminDomainEntity class in the ITIM

```

```

* JavaScript environment.
*/
private class AdminDomainWrapper extends JSFunctionAdapter {

    /*
    * (non-Javadoc)
    *
    * @see FESI.jslib.JSFunction#doNew(FESI.jslib.JSObject,
    *     java.lang.Object[])
    */
    public Object doNew(JSObject thisObject, Object[] args)
        throws JSEException {
        if (args.length != 1 || !DistinguishedName.isDN(args[0].toString()))
        {

            Object errorValues[] = { "AdminDomain(dn)" };
            throw new JSEException("Argument error",
                new ScriptEvaluatorException(RE_INVALID_ARGUMENTS,
                    errorValues));
        }
        DistinguishedName dn = new DistinguishedName((String) args[0]);
        try {
            AdminDomainEntity domainEntity = (new AdminDomainSearch())
                .lookup(dn);
            JSObject domainObject = thisObject.getGlobalObject()
                .makeJSObject();
            domainObject.setMember(DOMAIN_OBJECT, domainEntity);
            domainObject.setMember("addAdministrator",
                new AddAdministratorFunction());
            return domainObject;

        } catch (ModelCommunicationException mce) {
            throw new JSEException("Failed to lookup AdminDomain "
                + dn.getAsString(), mce);
        } catch (ObjectNotFoundException onfe) {
            throw new JSEException("Failed to lookup AdminDomain "
                + dn.getAsString(), onfe);
        }
    }
}
}

```

AbstractExtension.java

```
/*
 * Created on Jan 17, 2006
 */
package com.ibm.itim.custom.workflow;

import com.ibm.itim.logging.JLogUtil;
import com.ibm.itim.workflow.application.WorkflowApplication;
import com.ibm.itim.workflow.application.WorkflowExecutionContext;
import com.ibm.itim.workflow.model.Activity;
import com.ibm.itim.workflow.model.ActivityResult;
import com.ibm.log.Level;
import com.ibm.log.PDLogger;

/**
 * @author davis, brian
 *
 * This class acts as a base for other workflow extension classes. It provides
 * a set of common error handlers for methods that implement workflow extension
 * activities. The error handlers all write either an error or warning to the
 * ITIM message log. If an exception is provided they will also write a stack
 * trace in the ITIM trace file. They will then return an ActivityResult
 * announcing that an activity has returned a failure or warning. This
 * ActivityResult can be returned by the method that is implementing the
 * failing workflow activity.
 */
public abstract class AbstractExtension implements WorkflowApplication {

    private WorkflowExecutionContext workflowContext;

    /**
     * (non-Javadoc)
     *
     * @see
     com.ibm.itim.workflow.application.WorkflowApplication#setContext(com.ibm.itim.w
     orkflow.application.WorkflowExecutionContext)
     */
    public void setContext(WorkflowExecutionContext workflowContext) {
        this.workflowContext = workflowContext;
    }

    /**
     * @return Returns the workflowContext.
     */
    protected WorkflowExecutionContext getWorkflowContext() {
        return workflowContext;
    }
}
```



```

/**
 * Log an error and stack trace.
 *
 * @param error
 *         The exception to trace
 * @param msg
 *         The message to write to the log file
 * @param loggingMethod
 *         The name of the method that is reporting the error.
 * @return An ActivityResult reporting a failure with the same message
 *         written to the log, plus the exception message.
 */
protected ActivityResult extensionError(Throwable error, String msg, String
loggingMethod) {
    PDLogger logger = JLogUtil.getTraceLogger(this);
    if (logger.isLoggable(Level.DEBUG_MIN)) {
        Activity activity = this.workflowContext.getActivityVO();
        logger.text(Level.DEBUG_MIN, this, loggingMethod, msg + ": Workflow
context process="
            + this.workflowContext.getProcessEO().getId() + ", activity=" +
activity.getId() + "["
            + activity.getIndex() + "]"");
    }
    if (error != null) {
        logger.exception(Level.ERROR, this, loggingMethod, error, msg);
        msg += (": " + error.getClass().getName() + ": " +
error.getMessage());
    }
    return new ActivityResult(ActivityResult.FAILED, msg, null);
}

/**
 * Log an error.
 *
 * @param msg
 *         The message to write to the log file
 * @param loggingMethod
 *         The name of the method that is reporting the error.
 * @return An ActivityResult reporting a failure with the same message
 *         written to the log.
 */
protected ActivityResult extensionError(String msg, String loggingMethod) {
    return extensionError(null, msg, loggingMethod);
}

/**
 * Log a warning and stack trace.
 *

```

```

    * @param error
    *         The exception to trace
    * @param msg
    *         The message to write to the log file
    * @param loggingMethod
    *         The name of the method that is reporting the warning.
    * @return An ActivityResult reporting a warning with the same message
    *         written to the log, plus the exception message.
    */
    protected ActivityResult extensionWarning(Throwable error, String msg,
String loggingMethod) {
        PDLogger logger = JLogUtil.getTraceLogger(this);
        if (logger.isLoggable(Level.DEBUG_MIN)) {
            Activity activity = this.workflowContext.getActivityVO();
            logger.text(Level.DEBUG_MIN, this, loggingMethod, msg + ": Workflow
context process="
                + this.workflowContext.getProcessEO().getId() + ", activity=" +
activity.getId() + "["
                + activity.getIndex() + "]"");
        }
        if (error != null) {
            logger.exception(Level.WARN, this, loggingMethod, error, msg);
            msg += (": " + error.getClass().getName() + ": " +
error.getMessage());
        }
        return new ActivityResult(ActivityResult.WARNING, msg, null);
    }

    /**
    * Log a warning.
    *
    * @param msg
    *         The message to write to the log file
    * @param loggingMethod
    *         The name of the method that is reporting the warning.
    * @return An ActivityResult reporting a warning with the same message
    *         written to the log.
    */
    protected ActivityResult extensionWarning(String msg, String loggingMethod)
    {
        return extensionWarning(null, msg, loggingMethod);
    }
}

```

AbstractExtension.java

```
/*
 * Created on Jan 16, 2006
 */
package com.ibm.itim.custom.workflow;

import java.io.IOException;
import java.io.Serializable;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Properties;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import javax.sql.DataSource;

import com.ibm.itim.common.properties.PropertiesManager;
import com.ibm.itim.dataservices.model.DirectoryObject;
import com.ibm.itim.dataservices.model.domain.Account;
import com.ibm.itim.dataservices.model.domain.Person;
import com.ibm.itim.exception.ITIMEException;
import com.ibm.itim.logging.JLogUtil;
import com.ibm.itim.logging.SystemLog;
import com.ibm.itim.util.EncryptionManager;
import com.ibm.itim.workflow.model.ActivityResult;
import com.ibm.log.Level;
import com.ibm.log.PDLogger;

/**
 * @author davis, brian
 *
 * Workflow extensions that do database searches and updates. The database must
 * be reachable via a DataSource defined in the application server. The SQL
 * statements may be hard coded.
 */
public class DatabaseExtensions extends AbstractExtension {

    private static final String PROP_FILE = "enrole";

    private static final String FACTORY_PROP =
"enrole.appServer.contextFactory";
```

```

private static final String URL_PROP = "enrole.appServer.url";

private static final String PRINCIPAL_PROP = "enrole.appServer.systemUser";

private static final String CREDENTIALS_PROP =
"enrole.appServer.systemUser.credentials";

/**
 * Execute a SQL select statement. The statement must return a single
 * result set.
 *
 * @param dataSourceName
 *         The name of the datasource without the leading "jdbc/".
 * @param sqlStatement
 *         The SQL statement.
 * @return A list of lists as an output parameter. Each element of the
 *         outer list will be a row from the result set. Each element of
 *         the inner lists will be a column of a result row.
 */
public ActivityResult sqlSelect(String dataSourceName, String sqlStatement)
{
    /**
     * Get the database connection, and execute the command.
     */
    ArrayList result = new ArrayList();
    Statement stmt = null;
    ResultSet resultSet = null;
    PDLogger logger = JLogUtil.getTraceLogger(this);
    if (logger.isLoggable(Level.DEBUG_MID))
        logger.entry(Level.DEBUG_MID, this, "sqlSelect", dataSourceName,
sqlStatement);
    boolean maxDebug = logger.isLoggable(Level.DEBUG_MAX);
    try {
        stmt = getStatement(dataSourceName);
        resultSet = stmt.executeQuery(sqlStatement);
        int columnCount = resultSet.getMetaData().getColumnCount();
        while (resultSet.next()) {
            /**
             * For each row in the result set, create a new list, populate
             * it with the columns in the row, and add it to the result
             * list. Enter a null for columns with non-serializable data
             * (such as CLOB).
             */
            ArrayList row = new ArrayList();
            for (int i = 1; i <= columnCount; i++) {
                Object o = resultSet.getObject(i);
                if (o == null || Serializable.class.isInstance(o)) {
                    row.add(o);
                }
            }
        }
    }
}

```

```

        if (maxDebug)
            logger.text(Level.DEBUG_MAX, this, "sqlSelect", "Column
" + i + ": " + o);
        } else {
            row.add(null);
            if (maxDebug)
                logger.text(Level.DEBUG_MAX, this, "sqlSelect", "Column
" + i
                    + ": not serializable");
        }
    }
    result.add(row);
}
} catch (SQLException e) {
    return extensionError(e, "Error executing statement " + sqlStatement,
"sqlSelect");
} catch (DataSourceRetrievalException e) {
    return extensionError(e.getCause(), "Error getting connection",
"sqlSelect");
} finally {
    closeDown(stmt);
}

    result.add(new ArrayList(Collections.singletonList("returned value")));
    ArrayList outputParams = new ArrayList();
    outputParams.add(result);
    return new ActivityResult(ActivityResult.SUCCESS, null, outputParams);
}

/**
 * Get a connection to the database from the DataSource's connection pool.
 *
 * @param dataSourceName
 *         The name of the DataSource without the leading "jdbc/".
 * @return a JDBC Connection
 * @throws DataSourceRetrievalException
 *         as a wrapper for any exception encountered. This may be an
 *         IOException if the ITIM properties files are not readable.
 *         Or it may be a NamingException if the DataSource does not
 *         exist. Or it may be a SQLException if a database connection
 *         can't be created.
 */
private Connection getConnection(String dataSourceName) throws
DataSourceRetrievalException {
    /**
     * Get the context info from the enRole.properties file.
     */
    PropertiesManager propMgr;

```

```

try {
    propMgr = PropertiesManager.gInstance();
} catch (IOException e) {
    throw new DataSourceRetrievalException(e);
}
Properties p = new Properties();
p.put(Context.INITIAL_CONTEXT_FACTORY, propMgr.getProperty(PROP_FILE,
FACTORY_PROP));

p.put(Context.PROVIDER_URL, propMgr.getProperty(PROP_FILE, URL_PROP));
p.put(Context.SECURITY_PRINCIPAL, propMgr.getProperty(PROP_FILE,
PRINCIPAL_PROP));
String credentials = propMgr.getProperty(PROP_FILE, CREDENTIALS_PROP);
if (credentials.length() > 0 && credentials.charAt(credentials.length()
- 1) == '=')
    /*
     * This ITIM installation encrypts the passwords stored in its
     * properties files.
     */
    credentials = EncryptionManager.getInstance().decrypt(credentials);
p.put(Context.SECURITY_CREDENTIALS, credentials);

/*
 * The app server's context factory will return an instance of a
 * DataSource.
 */
DataSource dataSource;
try {
    Context ctx = new InitialContext(p);
    Object o = ctx.lookup(dataSourceName);
    dataSource = (DataSource) PortableRemoteObject.narrow(o,
DataSource.class);
} catch (ClassCastException e) {
    throw new DataSourceRetrievalException(e);
} catch (NamingException e) {
    throw new DataSourceRetrievalException(e);
}

/*
 * Get the database connection.
 */
Connection con = null;
try {
    con = dataSource.getConnection();
} catch (SQLException e) {
    throw new DataSourceRetrievalException(e);
}
try {
    /*

```

```

        * Automatically commit after each SQL statement.
        */
        con.setAutoCommit(true);
    } catch (SQLException e) {
        try {
            con.close();
        } catch (SQLException ignored) {
        }
        throw new DataSourceRetrievalException(e);
    }
}

return con;
}

/**
 * Get a database connection, and use it to create a JDBC Statement object.
 *
 * @param dataSourceName
 *         The name of the DataSource without the leading "jdbc/".
 * @return a JDBC Statement object
 * @throws DataSourceRetrievalException
 *         as a wrapper for any exceptions encountered. This may be any
 *         one of the exceptions that can be encountered by the
 *         getConnection method, or a SQLException if a Statement could
 *         not be created using the connection.
 */
private Statement getStatement(String dataSourceName) throws
DataSourceRetrievalException {
    Connection con = getConnection(dataSourceName);
    Statement stmt;
    try {
        stmt = con.createStatement();
    } catch (SQLException e) {
        try {
            con.close();
        } catch (SQLException ignored) {
        }
        throw new DataSourceRetrievalException(e);
    }
    try {
        /*
         * Configure the statement to timeout if any SQL command runs for
         * more than 5 minutes. That should be enough for most purposes. I
         * don't want a workflow thread to hang forever waiting for a dead
         * database.
         */
        stmt.setQueryTimeout(300);
    } catch (SQLException e) {
        closeDown(stmt);
    }
}

```

```

        throw new DataSourceRetrievalException(e);
    }
    return stmt;
}

/**
 * Close a JDBC Statement and its Connection.
 *
 * @param stmt
 *         the Statement
 */
private void closeDown(Statement stmt) {
    if (stmt != null) {
        Connection con = null;
        try {
            /*
             * Get the statement's connection.
             */
            con = stmt.getConnection();
        } catch (SQLException ignored) {
            SystemLog.getInstance().logError(this, "Error getting con " +
ignored.toString());
        }
        try {
            stmt.close();
        } catch (SQLException ignored) {
            SystemLog.getInstance().logError(this, "Error closing stmt " +
ignored.toString());
        }
        if (con != null)
            try {
                con.close();
            } catch (SQLException ignored) {
                SystemLog.getInstance().logError(this, "Error closing con " +
ignored.toString());
            }
        else
            SystemLog.getInstance().logError(this,
"Lost a reference to a statement's connection. Possible
connection leak.");
    }
}

/**
 * @author davisbri
 *
 * This class acts as a wrapper for any exceptions that are encountered
 * while creating JDBC connections and statements. The handlers that catch
 * this exception are actually interested in the "cause" exception.

```



```
*/
private static class DataSourceRetrievalException extends Exception {
    public DataSourceRetrievalException(Throwable cause) {
        super(cause);
    }
}
}
```

applicationServlet.java

```
package examples.expi;
import java.io.IOException;
import java.util.Collection;
import java.util.Date;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.Vector;
import java.rmi.RemoteException;

import javax.security.auth.Subject;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import com.ibm.itim.apps.ApplicationException;
import com.ibm.itim.apps.PlatformContext;
import com.ibm.itim.apps.Request;
import com.ibm.itim.apps.identity.PersonMO;
import com.ibm.itim.apps.provisioning.AccountMO;
import com.ibm.itim.apps.provisioning.AccountManager;
import com.ibm.itim.apps.provisioning.ServiceMO;
import com.ibm.itim.common.AttributeValue;
import com.ibm.itim.common.AttributeValues;
import com.ibm.itim.dataservices.model.domain.Account;
import com.ibm.itim.dataservices.model.domain.Service;

/**
 * @version 2.0
 * @author aannas
 */

public class applicationServlet extends HttpServlet {
    private HttpSession session;
    private static expiUtil utilObject = null;
    private static String LOGON_PAGE = "logon.jsp";

    // These get loaded at init
    private static String ERROR_MESSAGE;
    private static String MAIN;
    private static String LOGON;
    private static String APPLICATIONS;
    private static String APPLICATIONS_SUB;
```

```

/**
 * Method init
 * Load up defines!
 * @see javax.servlet.GenericServlet#init()
 */

public void init() throws ServletException{
    log("init(): start");

    try {
        utilObject = new expiUtil();
    } catch (Exception e) {
        e.printStackTrace();
    }

    LOGON = utilObject.getPropertySSOCheck(utilObject.LOGON_PAGE);
    log("init(): Logon Page = " + LOGON);
    log("applicationServlet:init(): end");

    // Application Subscription page
    APPLICATIONS = utilObject.getProperty(utilObject.APPLICATIONS_PAGE);
    log("init(): Applications page: " + APPLICATIONS);
    if (isNullOrEmpty(APPLICATIONS))
        throw new ServletException("Could not load Applications page location
from properties file");
    APPLICATIONS.trim();

    // Application Subscription Submitted page
    APPLICATIONS_SUB =
utilObject.getProperty(utilObject.APPLICATIONSSUB_PAGE);
    log("init(): Applications Submitted page: " + APPLICATIONS_SUB);
    if (isNullOrEmpty(APPLICATIONS_SUB))
        throw new ServletException("Could not load Applications Submitted
page location from properties file");
    APPLICATIONS_SUB.trim();
    log("init()complete: Applications Submitted page: " + APPLICATIONS_SUB);

}

/**
 * Method doGet
 * Obtains the necessary attributes from the Request (user must be
authenticated) and all the required
 * data items provided in the session. If all items are provided, if TAM
is in use then a list of applications (TAM groups) is
 * processed and control is forwarded to the applications.jsp page. If TAM
is not in use, then the list of

```

```

    * applications the authenticated user is authorized to have access to is
    processed and the applications.jsp page.
    * The jsp page builds a table of Applications that the user may pick
    (select or un-select).
    *
    * @see javax.servlet.http.HttpServlet#void
    (javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
    */
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        log("doGet()");

        session = req.getSession(false);
        if (session == null) {
            log("Session is not valid.");
            req.setAttribute("message", "The Session is no longer valid");
            resp.sendRedirect("logon.jsp");
            return;
        }

        // do we have a valid subject to work with - this will be the case
        // if the logon was successful, otherwise not.

        if (!isSubjectAssigned(req, resp))
            return;

        Subject subject = (Subject) session.getAttribute(expiUtil.SUBJECT);

        if (subject == null) {
            log(
                "applicationServlet:doGet() - no subject - redirecting to logon
                page");
            resp.sendRedirect(LOGON_PAGE);
        }

        PlatformContext platform =
            (PlatformContext) session.getAttribute(expiUtil.PLATFORM_CONTEXT);

        if (platform == null) {
            log(
                "applicationServlet:doGet() - no Platform Context object -
                redirecting to logon page");
            resp.sendRedirect(LOGON_PAGE);
        }

        String userID = (String) session.getAttribute(expiUtil.LOGON_ID);
        if (userID == null) {
            log(

```

```

        "applicationServlet:doGet() - no userID found - redirecting to
logon.jsp");
        resp.sendRedirect(LOGON_PAGE);
    }

    PersonMO personMo = (PersonMO) session.getAttribute(expiUtil.PERSONMO);

    if (personMo == null) {
        log(
            "applicationServlet:doGet() - no PersonMO object found -
redirecting to logon page");
        resp.sendRedirect(LOGON_PAGE);
    }

    // load the Account information respective to the person
    Account account = null;
    AccountMO acctMO = null;
    if (utilObject.isTAMService()){
        log("isTAMService: doGet");
        acctMO =
            utilObject.lookupAccounts(
                platform,
                subject,
                personMo,
                utilObject.getProperty(expiUtil.APP_SERVICE_DN));
        account = utilObject.account;
    }else{
        log("applicationServlet:doGet() - TAM Not in use: setting account to
null");
        account = null;
    }
    if (account != null) {
        // save off the AccountMO and respective account for this service in
order to reduce
        // processing in the doPost...

        session.setAttribute(expiUtil.ACCOUNTMO, acctMO);
        session.setAttribute(expiUtil.ACCOUNT, account);

        // add the group names from the property file to the request header.
Those placed in the
        // request header are currently selected groups. Those left off are
configured in the
        // property file but are not selected.

        setRequestAttributes(req, account.getAttributes());

        utilObject.forward(
            req,

```

```

        resp,
        "",
        utilObject.getProperty(expiUtil.APPLICATIONS_PAGE));
    } else {
        log(
            "applicationServlet:doGet() - no Account found for Service " +
            utilObject.getProperty(expiUtil.APP_SERVICE_NAME));
        Collection authAcctNames =
            utilObject.lookupAuthAccounts(platform,subject,personMo);
        if (!authAcctNames.isEmpty()){
            log("doGet() - AuthAccounts found ");
            session.setAttribute("authAccountNames",authAcctNames);
            resp.sendRedirect(APPLICATIONS);
        }else{
            log("doGet() - No AuthAccounts found ");
            session.setAttribute("Result",new Integer(-1));
            utilObject.forward(
                req,
                resp,
                "There are No Authorized Services to select",
                APPLICATIONS_SUB);
        }
    }
}
} //doGet

/**
 * Method doPost
 * Processes the selections made in the applications.jsp. A delta of the
 * selected and unselected
 * groups is made an any changes are submitted to the account update
 * process.
 * @see javax.servlet.http.HttpServlet#void
 (javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {
    String sGroup;

    session = req.getSession(false);
    if (session == null) {
        log(
            "applicationServlet:doPost() - Session is not valid.");
        req.setAttribute("message", "The Session is no longer valid");

        // under SSO we can't error out to the logon servlet either.
        // In this case we use the mapped LOGON_PAGE (which should be the
        // ssoerror.jsp

        if (utilObject.isSSOEnabled())

```

```

        resp.sendRedirect(LOGON_PAGE);
    else
        resp.sendRedirect(expiUtil.LOGON_SERVLET);
    return;
}

// Make user the user input some information
if (validateSubmittedAnswers(req, resp, session)){

    PlatformContext platform =
        (PlatformContext) session.getAttribute(expiUtil.PLATFORM_CONTEXT);

    if (platform == null) {
        log(
            "applicationServlet:doPost() - no Platform Context object -
redirecting to logon page");
        resp.sendRedirect(LOGON_PAGE);
    }

    Subject subject = (Subject) session.getAttribute(expiUtil.SUBJECT);

    if (subject == null) {
        log(
            "applicationServlet:doGet() - no subject - redirecting to logon
page");
        resp.sendRedirect(LOGON_PAGE);
    }

    PersonMO personMo = (PersonMO)
session.getAttribute(expiUtil.PERSONMO);

    if (personMo == null) {
        log(
            "applicationServlet:doGet() - no PersonMO object found -
redirecting to logon page");
        resp.sendRedirect(LOGON_PAGE);
        return;
    }

    log("applicationServlet:doPost()");
    // If TAM is in use
    if (utilObject.isTAMService()){
        Enumeration pname = req.getParameterNames();

        // Simply list the values in the enumeration
        while (pname.hasMoreElements()) {
            String element = (String) pname.nextElement();
            String sValue = (String)req.getParameter(element);

```

```

        log("Parameter: " + element + ": " + sValue);
    }

    if (!isSubjectAssigned(req, resp))
        return;

    AccountMO acctMO = (AccountMO)
session.getAttribute(expiUtil.ACCOUNTMO);

    if (acctMO == null) {
        log(
            "applicationServlet:doPost() - no accountMO - redirecting
to logon page");
        resp.sendRedirect(LOGON_PAGE);
        return;
    }

    Account account = (Account)
session.getAttribute(expiUtil.ACCOUNT);
    if (account == null) {
        log(
            "applicationServlet:doPost() - no account - redirecting to
logon page");
        resp.sendRedirect(LOGON_PAGE);
        return;
    }

    // build the consolidated lists of new groups and those to be
removed.

    Collection colExistingRoles;
    Collection colNewRoles;
    Collection colDefinedRoles;
    AttributeValues attrValues = account.getAttributes();

    colExistingRoles = utilObject.getTamGroups(attrValues);
    colNewRoles = new Vector(0);

    // get the set of applications (groups) that can are configured
// in the properties file...

    colDefinedRoles = getDefinedGroups();

    // get the roles selected in the JSP
    pname = req.getParameterNames();
    while (pname.hasMoreElements()) {
        String element = (String) pname.nextElement();
        sGroup = req.getParameter(element);
        if (!sGroup.equals("")) {

```



```

        if (!colExistingRoles.contains(sGroup)) {
            colNewRoles.add(sGroup);
        }
        colDefinedRoles.remove(sGroup);
    }
    // now run through the collection and consolidate the groups
that are to be
    // added and deleted

    boolean bChanges = false;

    // build collection on new groups
    Iterator it = colNewRoles.iterator();
    while (it.hasNext()) {
        sGroup = (String) it.next();
        colExistingRoles.add(sGroup);
        bChanges = true;
    }

    // update the list for those that are to be deleted (these were
not part of the parameters
    // that were selected...thus are deleted...

    it = colDefinedRoles.iterator();
    while (it.hasNext()) {
        sGroup = (String) it.next();
        colExistingRoles.remove(sGroup);
        bChanges = true;
    }

    // if any changes need to be made, set the attribute.
    // NOTE: only the groups in the property files get
manipulated...
    // other groups are left untouched.

    if (bChanges) {
        log("New Group Attr = " + colExistingRoles.toString());

        // add the new roles to the attribute list
        AttributeValue modAttrVal =
            new
AttributeValue(utilObject.getProperty(expiUtil.APP_SERVICE_ATTR),
colExistingRoles);

        account.setAttribute(modAttrVal);

        // update the account object with the new attributes...and
if successful,

```

```

        // forward to the self care submitted page...otherwise
forward back to the
        // self-care page

        if (utilObject.updateAccount(acctMO, account)) {
            utilObject.forward(
                req,
                resp,
                "",
                utilObject.getProperty(expiUtil.SELFCARESUB_PAGE));
            return;
        }
    }

    utilObject.forward(
        req,
        resp,
        "Application Subscription failed (please check the server
logs).",
        utilObject.getProperty(expiUtil.SELFCARE_PAGE));
    }
// TAM is not in use
}else{
    log("Application Servlet TAM Not in use...");
    Enumeration peters = req.getParameterNames();
    Collection authAccts = utilObject.lookupAuthAccounts(platform,
subject, personMo);
    Iterator authItr = authAccts.iterator();
    while ( peters.hasMoreElements()){
        String pmeterName = (String)peters.nextElement();
        log("Parameter element: " + pmeterName);
        if (!pmeterName.equals("OK")){
            log("Parameter element != OK");
            while (authItr.hasNext()){
                ServiceMO authSvcMO = (ServiceMO)authItr.next();
                try {
                    Service authSvc = authSvcMO.getData();
                    String authSvcName = authSvc.getName();
                    log("pmeterName: " + pmeterName + " == authSvcName: "
+ authSvcName);

                    if (pmeterName.equals(authSvcName)){
                        log("TRUE : pmeterName: " + pmeterName + " ==
authSvcName: " + authSvcName);
                        AccountManager acctMgr = new
AccountManager(platform, subject);
                        if (acctMgr == null){
                            log("AccountManager instantiation failed");
                            session.setAttribute("Result",new Integer(-1));
                            utilObject.forward(

```

```

        req,
        resp,
        "Application Subscription failed (please
check the server logs).",
utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
        return;
    }
    AttributeValues newAcctParms = null;
    String svcProfileName = "";
    try{
        newAcctParms =
acctMgr.getAccountParameters(personMo,authSvcM0);
        log("New Account Attrs: " + newAcctParms);
        svcProfileName =
authSvcM0.getData().getProfileName();
        log("svcProfileName: " + svcProfileName);
    }catch (ApplicationException e){
        e.printStackTrace();
        log("returning NULL Application Exception");
        session.setAttribute("Result",new Integer(-1));
        utilObject.forward(
            req,
            resp,
            "Application Subscription failed (please
check the server logs).",
utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
        return;
    }catch (RemoteException e){
        e.printStackTrace();
        log("returning NULL Remote Exception");
        session.setAttribute("Result",new Integer(-1));
        utilObject.forward(
            req,
            resp,
            "Application Subscription failed (please
check the server logs).",
utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
        return;
    }
    String propName = expiUtil.APP_PROF.toString() +
"." + svcProfileName.toLowerCase();

    // get the attribute name and value from the
properties file
    log("propName: " + propName);

```

```

        String acctProfile =
utilObject.getProperty(propName);
        if (acctProfile == null) {
            log("returning NULL Profile not defined in
properties file");
            session.setAttribute("Result",new Integer(-1));
            utilObject.forward(
                req,
                resp,
                "Application Subscription failed (please
check the server logs).",
            utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
            return;
        }
        log("Application profile: " + acctProfile);
        Account acct = new Account(acctProfile);
        acct.setAttributes(newAcctParms);
        log("New Account Object Created : " +
acct.getName());
        Request acctReq = null;
        try{
            acctReq =
acctMgr.createAccount(personMo,authSvcMO, acct, new Date());
        }catch (ApplicationException e){
            e.printStackTrace();
            log("Account Create failed Application
Exception");
            session.setAttribute("Result",new Integer(-1));
            utilObject.forward(
                req,
                resp,
                "Application Subscription failed (please
check the server logs).",
            utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
            return;
        }catch (RemoteException e){
            e.printStackTrace();
            log("Account Create failed Remote Exception");
            session.setAttribute("Result",new Integer(-1));
            utilObject.forward(
                req,
                resp,
                "Application Subscription failed (please
check the server logs).",
            utilObject.getProperty(expiUtil.APPLICATIONSSUB_PAGE));
            return;
        }
    }
}

```



```

} // doPost

/**
 * Method isSubjectAssigned.
 * Verifies if a Subject is provided in the request.
 * @param req
 * @param resp
 * @return boolean - true if Subject was found, false otherwise
 * @throws IOException
 */
private boolean isSubjectAssigned(
    HttpServletRequest req,
    HttpServletResponse resp)
    throws IOException {
    Subject subject = (Subject) session.getAttribute("subject");
    if (subject == null) {
        log("Session is not valid (no subject).");

        session.invalidate();

        req.setAttribute("message", "The Session is no longer valid.");
        resp.sendRedirect(LOGON_PAGE);
        return false;
    }
    return true;
}

private boolean validateSubmittedAnswers(
    HttpServletRequest req,
    HttpServletResponse resp,
    HttpSession session)
    throws ServletException, IOException {

    Enumeration pmeters = req.getParameterNames();
    while (pmeters.hasMoreElements()){
        if (pmeters.nextElement().equals(null)){
            return false;
        }
    }

    return true;
}

/**
 * Method setRequestAttributes.
 * @param req
 * @param selfcareAttrs
 * @param person

```

```

*/
public void setRequestAttributes(
    HttpServletRequest req,
    AttributeValues attrValues) {
    log("setRequestAttributes - Groups");

    Collection roles = utilObject.getTamGroups(attrValues);

    log("Account contains:: ");
    Iterator iter = roles.iterator();
    while (iter.hasNext()) {
        log("Group: " + iter.next());
    }

    String groups = utilObject.getProperty(expiUtil.APP_LIST);

    StringTokenizer st = new StringTokenizer(groups, ",");
    while (st.hasMoreTokens()) {
        String sAttr = st.nextToken();
        String appPropertyName =
            new String("application." + sAttr + ".name");
        String appPropertyDN = new String("application." + sAttr + ".dn");

        if (appPropertyName != null) {
            //log("sAttr=" + sAttr + " App Property: " + appPropertyName);

            String appName =
                new String(utilObject.getProperty(appPropertyName));
            if (appName != null) {
                String appNameDN =
                    new String(utilObject.getProperty(appPropertyDN));

                if (roles.contains(appNameDN)) {
                    req.setAttribute(sAttr, appName);
                    log(
                        "sAttr: (dn=" + appNameDN + ") " + sAttr + " - " +
                        appName);
                } else {
                    log(
                        "sAttr: (dn=" + appNameDN + ") " + sAttr + " -
                        (blank)");
                    req.setAttribute(sAttr, "");
                }
            } else
                req.setAttribute(sAttr, "");
        } else
            req.setAttribute(sAttr, "");
    }
}

```

```

/**
 * Method getDefinedApplications.
 * Returns a collection of defined application names (DN's of the Groups)
from the property file.
 * @return Collection
 */
public Collection getDefinedGroups() {
    Collection colRoles = new Vector(0);

    log("getDefinedApplications - Groups: ");

    String appString = utilObject.getProperty(expiUtil.APP_LIST);
    log("Application List: " + appString);

    StringTokenizer st = new StringTokenizer(appString, ",");
    while (st.hasMoreTokens()) {
        String sAttr = st.nextToken();
        String appPropertyName =
            new String("application." + sAttr + ".name");
        String appPropertyDN = new String("application." + sAttr + ".dn");

        if (appPropertyName != null) {
            log("App Property: " + appPropertyName);

            String appName =
                new String(utilObject.getProperty(appPropertyName));
            if (appName != null) {
                log("App Name: " + appName);

                String appNameDNStr =
                    new String(utilObject.getProperty(appPropertyDN));

                colRoles.add(new String(appNameDNStr));
            }
        }
    }

    return colRoles;
}

private boolean isNullOrEmpty(Object o) {

    if ((o == null) || (o.equals(""))
        return true;
    else
        return false;
}

```


}

applications.jsp

```
<!--
* Licensed Materials - Property of IBM
* @version 2.0
* @author aannas
*
* Description:
*
* JSP to display application subscription
*
*****/
-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page language="java" contentType="text/html; charset=WINDOWS-1252"%>
<%@ page import="examples.expi.*" %>
<%@ page import="java.util.*" %>
<%@ page import="com.ibm.itim.apps.provisioning.ServiceMO"%>
<%@ page import="com.ibm.itim.dataservices.model.domain.Service"%>
<%@ include file="expiProlog.jsp" %>

<TITLE>Tivoli Austin Airlines Self Care- Application Subscription</TITLE>

<SCRIPT language="JavaScript">
    function getLogoutURL()
    {
        document.location= "./logon";
    }

    function getHomeURL()
    {
        document.location= "./logon";
    }

    function getChangePwdURL()
    {
        document.location= "./changepwd.jsp";
    }

    function submitApplications()
    {
        document.appForm.action = "./applicationServlet";
        document.appForm.submit();
    }

```

```

        function goToMainPage()
        {
            document.location="./main.jsp";
        }
</SCRIPT>
</HEAD>

<%@ include file="expi_header.html" %>
<BODY>
<table border=0 cellpadding=0 cellspacing=0 width=100%>
    <tr>
        <td width="80"></td>
        <td width="100%">

            <table border=0 cellpadding=0 cellspacing=0 width=100%>
                <tr>
                    <td class="heading-text">Applications Subscription</td>
                </tr>
                <tr>
                    <td class="heading-line"></td>
                </tr>
                <tr>
                    <td class="text-description">Please select from the following
services
                    and press OK when completed</td>
                </tr>
            </table>
<FORM NAME="appForm" METHOD="POST">
<TABLE border="0" width="377">
<TBODY>
    <%
        log("applications.jsp start");
        expiUtil utilObject = null;

        try {
            utilObject = new expiUtil();
        } catch (Exception e) {
            e.printStackTrace();
        }
        // If TAM is Not in Use
        if (!utilObject.isTAMService()){
            String sAttr, sAttrValue, sAttrValueDN, sAttrText = "";
            String isChecked = null;
            Collection authNames =
(Collection)session.getAttribute("authAccountNames");
            Iterator appItr = authNames.iterator();
            while (appItr.hasNext()){

```

```

        ServiceMO authSvcMO = (ServiceMO)appItr.next();
        Service authSvc = authSvcMO.getData();
        sAttr = authSvc.getName();
        sAttrValueDN = authSvc.getDistinguishedName().toString();
        sAttrText = authSvc.getName();
        sAttrValue = (String)request.getAttribute(sAttr);
        if (sAttrValue == null || sAttrValue.equals(""))
            isChecked = "";
        else {
            isChecked = "checked";
        }
    }
    %>
<TR>
    <TH><INPUT type="checkbox" name="<%= sAttr %>" value="<%=
sAttrValueDN %>" <%= isChecked %>></TH>
    <TD><%= sAttrText %></TD>
</TR>
<%
    }
}else{
// If TAM Is in Use
String appAttrs = utilObject.getProperty(expiUtil.APP_LIST);

if (appAttrs != null) {
    session.setAttribute("appuserActivity","active");
    StringTokenizer st = new StringTokenizer(appAttrs, ",");
    while (st.hasMoreTokens()) {
        String sAttr = st.nextToken();
        String isChecked = null;

        String sPropName = "application." + sAttr + ".name";
        String sPropNameDN = "application." + sAttr + ".dn";

        // get the attribute name and value from the properties
file

        String sAttrText = utilObject.getProperty(sPropName);
        if (sAttrText == null) {
            sAttrText = "(Application name not defined in properties
file)";
        }

        String sAttrValueDN = utilObject.getProperty(sPropNameDN);
        if (sAttrValueDN == null) {
            sAttrValueDN = "(Application group name not defined in
properties file)";
        }
    }
}
}

```

```

// if this attribute was on the request header - that means
it's selected
// otherwise not.

String sAttrValue = (String)request.getAttribute(sAttr);
if (sAttrValue == null || sAttrValue.equals(""))
    isChecked = "";
else {
    isChecked = "checked";
}

%>
<TR>
    <TH><INPUT type="checkbox" name="<%= sAttr %>" value="<%=
sAttrValueDN %>" <%= isChecked %>></TH>
    <TD><%= sAttrText %></TD>
</TR>
<%
    }
    }else{
        session.setAttribute("appuserActivity","none");
    }
}
%>
</TBODY>
</TABLE>

<br>
<input type="submit" name="OK" id="OK" value="OK" class="button"
onMouseOver="pviiClassNew(this,'buttonover')"
onMouseOut="pviiClassNew(this,'button')" onClick="submitApplications()">
    <input type="button" name="Cancel" id="Cancel" value="Cancel"
class="button" onMouseOver="pviiClassNew(this,'buttonover')"
onMouseOut="pviiClassNew(this,'button')" onClick="goToMainPage()">
</form>
</td>

</tr>
</table>
</body>
</html>

```

application_sub.jsp

```
<!--
* Licensed Materials - Property of IBM
* @version 2.0
* @author aannas
*
* Description:
*
* JSP to display application submission.
*
*****/
-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<%@ page language="java" contentType="text/html; charset=WINDOWS-1252"%>
<HEAD>
<TITLE>Tivoli Austin Airlines Self Care - Application Request Submitted</TITLE>
</HEAD>

<%@ include file="expj_header.html" %>

<script language="JavaScript">
function goToMainPage() {
    document.location="main.jsp";
}
</script>

<BODY>
<table border=0 cellpadding=0 cellspacing=0 width=90%>
    <tr>
        <td width="80"></td>
        <td width="100%">

            <table border=0 cellpadding=0 cellspacing=0 width=100%>
                <tr>
                    <td></td>
                </tr>
                <tr>
                    <td class="heading-text">Application Request Submission</td>
                </tr>
                <tr>
                    <td class="heading-line"></td>
                </tr>
            </table>
        </td>
    </tr>
</table>
```

```

<TD></TD>
</TR>
<tr>
<%Integer submissionRslt = (Integer)session.getAttribute("Result");
  if (submissionRslt.intValue()== 2){
    log("Request Submission result code: " +
submissionRslt.intValue());
    %>
    <td class="text-description">Your request was successfully
submitted.</td>
    <%
      }else if (submissionRslt.intValue()== 0){
        log("Request Submission result code: " +
submissionRslt.intValue());
        %>
        <td class="text-description">Your request failed to start on
submission. Please check the logs for details.</td>
        <%
          }else if (submissionRslt.intValue()== 1){
            log("Request Submission result code: " +
submissionRslt.intValue());
            %>
            <td class="text-description">Your request is in process. Please
review accounts status.</td>
            <%
              }else if (submissionRslt.intValue()== 3){
                log("Request Submission result code: " +
submissionRslt.intValue());
                %>
                <td class="text-description">Your request failed on submission.
Please check the logs for details.</td>
                <%
                  }else if (submissionRslt.intValue()== 4){
                    log("Request Submission result code: " +
submissionRslt.intValue());
                    %>
                    <td class="text-description">Your request produced a warning
during submission. Please check the logs for details.</td>
                    <%
                      }else{
                        log("Request Submission result code: " +
submissionRslt.intValue());
                        %>
                        <td class="text-description">Your request failed prior to
submission. Please check the logs for details.</td>
                        <%
                          }
                        %>

```

```

        </tr>
    </table>

    <!-- If there's an error message, display it! -->
    <table width="90%" border="0" cellspacing="0" cellpadding="0">
        <tr>
            <td><%@ include file="error_msg_box.jsp" %></td>
        </tr>
    </table>
    <br>

    <p>
    <table>

        <tr>
            <td><br>
                <input type="button" name="OK" id="Cancel" value="OK"
                class="button" onMouseOver="pviiClassNew(this,'buttonover')"
                onMouseOut="pviiClassNew(this,'button')" onClick="goToMainPage()">
            </td>
        </tr>
    </table>
    </td>
</tr>
</table>
</body>
</html>

```


todolistServlet.java

```

/*****
 *
 * Licensed Materials - Property of IBM
 *
 * (C) COPYRIGHT IBM Corp. 2005
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 *
 *****/

/*****
 * FILE: %Z%M% %I% %W% %G% %U%
 *
 * This servlet contains methods that control the self registration
 * process for the Servlet Portfolio Example.
 *
 * The user must provide a set of attributes that are considered mandatory
 * for the self registration process. The attributes used are imbedded in the
 * selfregister.jsp. This example shows the method of using variables directly
 in the
 * JSP. This contrasts to the method used in the selfCareServlet which uses the
 * <b>itim_expi.properties<b> file to define and autoconfigure the input table.
 *
 * One the user provides the necessary data, it is validated and if it
 * appears correct, the ITIM APIs are used to create the Person object.
 * Note: Account provisioning must be configured for at least the ITIM Accounts
 for
 * the samples to function correctly and allow the user to log on after
 * creating an account. In additional, TAM accounts can be provisioned to allow
 * control of Applications to which the user may subscribe. (Refer to the
 * ITIM documentation for Provisioning information).
 *
 *****/

package examples.expi;

import java.io.IOException;
import java.rmi.RemoteException;
import java.util.Collection;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Vector;
import java.util.Map;
import java.util.HashMap;
```

```

import javax.servlet.*;
import javax.servlet.http.*;
import javax.security.auth.Subject;

import com.ibm.itim.apps.ApplicationException;
import com.ibm.itim.apps.PlatformContext;
import com.ibm.itim.apps.SchemaViolationException;
import com.ibm.itim.apps.workflow.HumanResourceMO;
import com.ibm.itim.apps.workflow.WorkflowActivityMO;
import com.ibm.itim.apps.workflow.WorkflowAssignmentMO;
import com.ibm.itim.apps.workflow.WorkflowProcessMO;
import com.ibm.itim.workflow.model.WorkflowProcess;
import com.ibm.itim.workflow.model.ActivityResult;
import com.ibm.itim.workflow.model.Activity;
import com.ibm.itim.workflow.model.Assignment;

/**
 * @version 2.0
 * @author aannas
 *
 * todoServlet:
 * This servlet works in conjunction with the todoList.jsp that provides
 * a subset of data for a user to complete their assignments in ITIM.
 *
 */
public class todolistServlet extends HttpServlet {

    public static final String PLATFORM_CONTEXT = "platform";
    private String LOGON_PAGE = "logon.jsp";

    // Current Assigned Activities in TIM
    private static final String ASSIGNMENTS = "todoListAssignments";

    // These get loaded at init
    private static String ERROR_MESSAGE;
    private static String MAIN;
    private static String LOGON;
    private static String TDL_ASSIGNMENTS;

    /**
     * Method init
     * Load up defines!
     * @see javax.servlet.GenericServlet#init()
     */

    public void init() throws ServletException {

```

```

    log("tolodlistServlet init()");
    expiUtil utilObject = new expiUtil();

    /**
     * Defines
     */
    ERROR_MESSAGE = utilObject.ERR_LABEL;

    // Logon page
    LOGON = utilObject.getPropertySSOCheck(utilObject.LOGON_PAGE);
    log("init(): logon page: " + LOGON);
    if (isNullOrEmpty(LOGON))
        throw new ServletException("Could not load logon page location from
properties file");
    LOGON.trim();

    // Main page
    MAIN = utilObject.getProperty(utilObject.HOME_PAGE);
    log("init(): Home page: " + this.MAIN);
    if (isNullOrEmpty(MAIN))
        throw new ServletException("Could not load home page location from
properties file");
    MAIN.trim();

    // To Do List page
    TDL_ASSIGNMENTS =
utilObject.getProperty(utilObject.TDLASSIGNMENTS_PAGE);
    log("init(): To Do List Assignments page: " + TDL_ASSIGNMENTS);
    if (isNullOrEmpty(TDL_ASSIGNMENTS))
        throw new ServletException("Could not load To Do List Assignments
page location from properties file");
    TDL_ASSIGNMENTS.trim();

} // init

/**
 * Method destroy
 * Right now this does nothing.
 * @see javax.servlet.Servlet#destroy()
 */

public void destroy() {

} // destroy

```

```

/**
 * @see javax.servlet.http.HttpServlet#void
 (javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
 */
public void doGet(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    // Get the session
    HttpSession session = req.getSession(true);

    //Check if it has expired
    if (session.isNew()) {
        log("doPost(): No previous session; forwarding to logon page");
        gotoPage(
            LOGON,
            "Your session has expired or is invalid. Please sign on again.",
            session,
            req,
            resp);
        return;
    }

    Collection assignCol = new Vector();
    Collection wkflMOCOL = new Vector();

    try {
        Subject subject = (Subject) session.getAttribute(expiUtil.SUBJECT);
        log("TodolistServlet : Subject = " + subject);

        log("call to HumanResourceMO()");
        HumanResourceMO hrMO = new
HumanResourceMO((PlatformContext)session.getAttribute(PLATFORM_CONTEXT),
subject);
        log("After hrMO new()");
        Collection hrMOassnmts = hrMO.getAssignments();
        log("After Collection hrMOassnmts");
        if (hrMOassnmts.size() == 0) {
            assignCol = null;
            session.setAttribute(ASSIGNMENTS,assignCol);
            log("No Pending Activities");
            resp.sendRedirect(TDL_ASSIGNMENTS);
        }
        log("executetodosearch() - after hrMOassnmts");
        Iterator it = hrMOassnmts.iterator();
        while (it.hasNext()) {
            log("it.hasNext()");
            WorkflowAssignmentMO wkflwAssgnMO =
(WorkflowAssignmentMO)it.next();
            wkflMOCOL.add(wkflwAssgnMO);

```

```

        log("Assignment added: " + wkflwAssgnMO);
        WorkflowActivityMO wkflwActMO = wkflwAssgnMO.getActivity();
        log("Parameter List Created");
        assignCol.add(wkflwActMO);
        session.setAttribute("activities",assignCol);
        //Save the collection so not repeat the processing later
        session.setAttribute("assignments",wkflwMOCol);
    }
    session.setAttribute(ASSIGNMENTS, assignCol);
    resp.sendRedirect(TDL_ASSIGNMENTS);

    } catch (RemoteException e) {
        e.printStackTrace();
        gotoPage(TDL_ASSIGNMENTS,"Error: Failed on Remote
assignmentCompletion",session,req, resp);
    } catch (SchemaViolationException e) {
        e.printStackTrace();
        gotoPage(TDL_ASSIGNMENTS,"Error:
SchemaViolationException",session,req, resp);
    } catch (ApplicationException e) {
        e.printStackTrace();
    }
}

} //doGet

/**
 * Enables a user to view and complete the user's assignments using the
external ITIM API HumanResource.getAssingments().
 * Minor validation is carried out to for passwords
(attribute=ersharedsecret).
 *
 *
 * @param req Provides access to the form data parametes (data fields).
 * @exception ServletException
 * @exception IOException
 */
public void doPost(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {

    log("doPost()");

    // Get the session
    HttpSession session = req.getSession(true);

    //Check if it has expired
    if (session.isNew()) {
        log("doPost(): No previous session; forwarding to logon page");
        gotoPage(
            LOGON,

```

```

        "Your session has expired or is invalid. Please sign on again.",
        session,
        req,
        resp);
    return;
}
Collection wfMOcl = (Collection)session.getAttribute("assignments");
Iterator it = wfMOcl.iterator();
WorkflowActivityMO wkflwActMO;
WorkflowAssignmentMO wkflwAssgnMO;
Activity actvty;
while (it.hasNext()) {
    log("it.hasNext()");
    wkflwAssgnMO = (WorkflowAssignmentMO)it.next();
    log("WorkflowAssignmentMO object instantiated: " +
wkflwAssgnMO.getID());
    try{
        log("About to retrieve Activity");
        wkflwActMO = wkflwAssgnMO.getActivity();
        actvty = wkflwActMO.getData();
        ActivityResult actRslt = null;
        if (req.getParameter(actvty.getDesignId() +
"decision").equals("SUCCESS")){
            log("Retrieving req.getParameter(actvty.getDesignId() +
reason): " + req.getParameter(actvty.getDesignId() + "reason"));
            actRslt = new ActivityResult(actRslt.STATUS_COMPLETE,
actRslt.SUCCESS,
                req.getParameter(actvty.getDesignId() + "reason"),new
Vector());
            log("actRslt.getAttribute Description Retrieved and
actRslt.setDescription set: "
                + actRslt.getDescription());
            log("Activity Result set to Success");
            log("Completing wkflwAssgnMO");
            wkflwAssgnMO.complete(actRslt);
            log("wkflwAssgnMO Completed");
        }else{
            log("Retrieving req.getParameter(actvty.getDesignId() +
reason): " + req.getParameter(actvty.getDesignId() + "reason"));
            actRslt = new ActivityResult(actRslt.STATUS_COMPLETE,
actRslt.REJECTED,
                req.getParameter(actvty.getDesignId() + "reason"),new
Vector());
            log("actRslt.getAttribute Description Retrieved and
actRslt.setDescription set: "
                + actRslt.getDescription());
            log("Activity Result set to Rejected");
            log("Completing wkflwAssgnMO");
            wkflwAssgnMO.complete(actRslt);

```

```

        log("wkflwAssgnMO Completed");
    }
} catch (ApplicationException e) {
    e.printStackTrace();
    log("ApplicationException in doPost");
} catch (RemoteException e) {
    e.printStackTrace();
    log("RemoteException in doPost");
}
}
log("done doPost");
resp.sendRedirect(MAIN);
} //doPost

private boolean isEmpty(Object o) {

    if ((o == null) || (o.equals("")) )
        return true;
    else
        return false;
}

/**
 * Method goToPage.
 * Set the message in the request and forward it on to the specified page.
 * @param page
 * @param errorMessage
 * @param session
 * @param request
 * @param response
 * @throws ServletException
 * @throws IOException
 */
private void goToPage(
    String page,
    String errorMessage,
    HttpSession session,
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {

    if (errorMessage != null) {
        log("Error Message : " + errorMessage);
        session.setAttribute(ERROR_MESSAGE, errorMessage);
    }

    RequestDispatcher dispatcher =
        getServletContext().getRequestDispatcher("/" + page);

```

```
        dispatcher.forward(request, response);  
    } // goToPage  
}
```


todolist.jsp

```
<!--
* Licensed Materials - Property of IBM
* @version 2.0
* @authoraannas
*
* Description:
*
* JSP to display To Do List.
*
*****/
-->

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<%@ page
import="java.util.*,com.ibm.itim.workflow.model.*,com.ibm.itim.apps.workflow.*"
%>

<title>Tivoli Austin Airlines Self Care - To Do List</title>

<SCRIPT language="JavaScript">
    function getLogoutURL()
    {
        document.location= "./logon";
    }

    function getHomeURL()
    {
        document.location= "./logon";
    }

    function getChangePwdURL()
    {
        document.location= "./changepwd.jsp";
    }

    function submitTodolist()
    {
        document.todoForm.action = "./todolistServlet";
        document.todoForm.submit();
    }

    function goToMainPage()
    {
```

```

        document.location="./main.jsp";
    }
</SCRIPT>
</head>

<%@ include file="expi_header.html" %>
<BODY>
<BR>

<table width="100%" border="0" cellspacing="0" cellpadding="3">
  <tr>
    <td width="80">
      
    </td>

    <td width="100%">
      <table width="90%" border="0" cellspacing="0" cellpadding="0">
        <tr>
          <td class="heading-text">To Do List Activities</td>
        </tr>

        <tr>
          <td class="heading-line"></td>
        </tr>

        <!-- Description -->
        <tr>
          <td class="text-description"> &nbsp;&nbsp;&nbsp;Complete the activities and
click OK. <br>
          </td>
        </tr>
      </table>

      <!-- If there's an error message, display it! -->
      <table width="90%" border="0" cellspacing="0" cellpadding="0">
        <tr>
          <td><%@ include file="error_msg_box.jsp" %></td>
        </tr>
      </table>
      <br>

      <FORM NAME="todoForm" METHOD="POST">

      <table width="100%" border="0" cellspacing="0" cellpadding="3">
        <%
          Collection wkflwProcCol =
(Collection)session.getAttribute("todoListAssignments");
          if (wkflwProcCol != null) {

```

```

        Iterator wkasnit = wkflwProcCol.iterator();
        while (wkasnit.hasNext()) {
            log("todolist.jsp hasNext");
            WorkflowActivityMO wkflwActMO =
                (WorkflowActivityMO)wkasnit.next();
            log("todolist.jsp getLstItm wkflwActMO" + wkflwActMO.toString());
        }

        <tr>
    </tr>
    <tr>
        <%
            if (!wkflwActMO.getData().getDescription().equals(null)){
                log("wkflwActMO.name(): " + wkflwActMO.getData().getName());
                log("wkflwActMO.getDescription(): " +
                    wkflwActMO.getData().getDescription());
                log("wkflwActMO.getDesignId(): " +
                    wkflwActMO.getData().getDesignId());
            }
            <td nowrap
                class="text-normal"><%=wkflwActMO.getData().getDescription() %> &nbsp;
                <%=wkflwActMO.getContainer().getData().getSubjectService() %>
                &nbsp;
                <%=wkflwActMO.getContainer().getData().getSubject() %> &nbsp;
            </td>
            <%
            }else if ( !wkflwActMO.getData().getName().equals(null)){
                log("wkflwActMO.name(): " + wkflwActMO.getData().getName());
                log("wkflwActMO.getDescription(): " +
                    wkflwActMO.getData().getDescription());
                log("wkflwActMO.getDesignId(): " +
                    wkflwActMO.getData().getDesignId());
            }
            <td nowrap class="text-normal"><%=wkflwActMO.getData().getName()
        %></td>
            <td nowrap
                class="text-normal"><%=wkflwActMO.getContainer().getData().getSubject()%><br></td>
            <%
            }else{
                log("wkflwActMO.getDesignId(): " +
                    wkflwActMO.getData().getDesignId());
            }
            <td nowrap
                class="text-normal"><%=wkflwActMO.getData().getDesignId() %><br></td>
        <%
        }
    %>

```

```

</tr>
<tr>
</tr>

<tr>
  <td class="text-bold">Decision</td>
</tr>

<tr>
  <td>
    <SELECT name="<%=wkflwActMO.getData().getDesignId() +
"decision" %>" >
      <OPTION value="REJECT">
        Reject
      </option>
      <OPTION value="SUCCESS">
        Approve
      </option>
    </SELECT>
  </td>
</tr>
<tr>
  <td class="text-bold">Reason</td>
</tr>
<tr>
  <td>
    <TEXTAREA name="<%=wkflwActMO.getData().getDesignId() + "reason" %>"
rows="5" cols="66" title="" id=""></TEXTAREA>
  </td>
</tr>

<tr>
  <td nowrap class="text-normal"><br></td>
</tr>

<%
}
}else{
%>
<TR>
  <td class="text-bold"> You have no pending activities at this
time </td>
</TR>
<%
}
%>

<!-- buttons -->

```

```

        <tr>
            <td>
                <%
                    if (wkflwProcCol != null) {
                        %>
                            <input type="submit" name="OK" id="OK" value="OK" class="button"
onMouseOver="pviiClassNew(this,'buttonover')"
onMouseOut="pviiClassNew(this,'button')" onClick="submitTodolist()">
                                <input type="button" name="Cancel" id="Cancel" value="Cancel"
class="button" onMouseOver="pviiClassNew(this,'buttonover')"
onMouseOut="pviiClassNew(this,'button')" onClick="goToMainPage()">
                                    <%
                                        }else{
                                            %>
                                                <input type="button" name="OK" id="OK" value="OK" class="button"
onMouseOver="pviiClassNew(this,'buttonover')"
onMouseOut="pviiClassNew(this,'button')" onClick="goToMainPage()">
                                                    <%
                                                        }
                                                    %>
                                                </td>
                                            </tr>
                                        </table>
                                    </td>
                                </tr>
                            </table>
                        </BODY>
                    </html>

```

mainServlet.java

```

/*****
 *
 * Licensed Materials - Property of IBM
 *
 * (C) COPYRIGHT IBM Corp. 2005
 *
 * US Government Users Restricted Rights - Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 *
 *****/

/*****
 * FILE: %Z%M% %I% %W% %G% %U%
 *
 * This servlet contains an initialization method that must be executed
 * before the main.jsp page gains control. Its responsibility is to check
 * if the TAM Service (as configured in the itim_expi.properties file)
 * is active on the targetted ITIM Server.
 *****/
package examples.expi;

import java.io.IOException;
import java.util.Collection;
import java.util.Enumeration;
import java.util.Iterator;
import java.util.StringTokenizer;
import java.util.Vector;

import javax.security.auth.Subject;
import javax.servlet.*;

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

import com.ibm.itim.apps.PlatformContext;
import com.ibm.itim.apps.identity.PersonMO;
import com.ibm.itim.apps.provisioning.AccountMO;
import com.ibm.itim.common.AttributeValue;
import com.ibm.itim.common.AttributeValues;
import com.ibm.itim.dataservices.model.DistinguishedName;
import com.ibm.itim.dataservices.model.domain.Account;

/**

```

```

* @version 2.0
* @authoraannas
*/
public class mainServlet extends HttpServlet {
    private HttpSession session;
    private static expiUtil utilObject = null;
    private String LOGON_PAGE = "logon.jsp";

    /**
     * Method init
     * @see javax.servlet.GenericServlet#init()
     */
    public void init() {
        log("mainServlet:init(): start");

        try {
            utilObject = new expiUtil();
        } catch (Exception e) {
            e.printStackTrace();
        }

        LOGON_PAGE = utilObject.getPropertySSOCheck(utilObject.LOGON_PAGE);
        log("EXPI:init(): LOGON_PAGE = " + LOGON_PAGE);
        log("mainServlet:init(): end");
    }

    /**
     * Method doGet
     * Obtains the necessary attributes from the Request (user must be
     authenticated) and all the required
     * data items provided in the session. If all items are provided, the list
     of applications (TAM groups) is
     * processed and control is forwarded to the applications.jsp page. The
     jsp page builds a table of
     * Applications that the user may pick (select or un-select).
     *
     * @see javax.servlet.http.HttpServlet#void
     (javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse)
     */
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {

        log("doGet(): start");

        session = req.getSession(false);
        if (session == null) {
            log("No session");
            // See if we got here via SSO through WebSEAL
            String userID = (String) req.getHeader("iv-user");

```

```

        if ( (userID != null) && (! userID.equals("")) ) {
            log("iv-user = " + userID);
            log("Forwarding to loginServlet.doPost()");
            RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/" + "logonServlet");
            dispatcher.forward(req, resp);

        } else {
            req.setAttribute("message", "The Session is no longer valid");
            resp.sendRedirect("logon.jsp");
        }
        return;
    }

    // do we have a valid subject to work with - this will be the case
    // if the logon was successful, otherwise not.

    if (!isSubjectAssigned(req, resp))
        return;

    HttpSession session = (HttpSession) req.getSession(false);
    if (session == null) {
        log("mainServlet:doGet() - no session - redirecting to logo page");
        resp.sendRedirect(LOGON_PAGE);
    }

    Subject subject = (Subject) session.getAttribute(expiUtil.SUBJECT);

    if (subject == null) {
        log("mainServlet:doGet() - no subject - redirecting to logon page");
        resp.sendRedirect(LOGON_PAGE);
    }

    PlatformContext platform =
        (PlatformContext) session.getAttribute(expiUtil.PLATFORM_CONTEXT);

    if (platform == null) {
        log("mainServlet:doGet() - no Platform Context object - redirecting
to logon page");
        resp.sendRedirect(LOGON_PAGE);
    }

    String userID = (String) session.getAttribute(expiUtil.LOGON_ID);
    if (userID == null) {
        log("mainServlet:doGet() - no userID found - redirecting to
logon.jsp");
        resp.sendRedirect(LOGON_PAGE);
    }
}

```



```

        PersonMO personMo = (PersonMO) session.getAttribute(expiUtil.PERSONMO);

        if (personMo == null) {
            log("mainServlet:doGet() - no PersonMO object found - redirecting to
logon page");
            resp.sendRedirect(LOGON_PAGE);
        }

        // load the Account information respective to the person
        String serviceDN = utilObject.getProperty(expiUtil.APP_SERVICE_DN);

        if (serviceDN == null || serviceDN.equals("")) {
            log("mainServlet:doGet() - TAM Service DN not specified in property
file");
            session.setAttribute(expiUtil.TAM_SERVICE_ACTIVE, "false");
        }else {

            AccountMO acctMO = utilObject.lookupAccounts(
                platform,
                subject,
                personMo,

utilObject.getProperty(expiUtil.APP_SERVICE_DN));
            Account account = utilObject.account;

            if (account != null) {
                log("mainServlet:doGet() - TAM Service is configured...forwarding
to main.jsp");

                // save off the AccountMO and respective account for this service
in order to reduce
                // processing in the doPost...

                session.setAttribute(expiUtil.ACCOUNTMO, acctMO);
                session.setAttribute(expiUtil.ACCOUNT, account);
                session.setAttribute(expiUtil.TAM_SERVICE_ACTIVE, "true");
            } else {
                log("mainServlet:doGet() - "
                    + utilObject.getProperty(expiUtil.APP_SERVICE_NAME)
                    + " is NOT configured...forwarding to main.jsp");
                session.setAttribute(expiUtil.TAM_SERVICE_ACTIVE, "false");
            }
        }

        // always forward to the Home page (main.jsp by default)

        utilObject.forward(
            req,
            resp,

```

```

        "",
        utilObject.getProperty(expiUtil.HOME_PAGE));
    }

    /**
     * Method isSubjectAssigned.
     * Verifies if a Subject is provided in the request.
     * @param req
     * @param resp
     * @return boolean - true if Subject was found, false otherwise
     * @throws IOException
     */
    private boolean isSubjectAssigned(
        HttpServletRequest req,
        HttpServletResponse resp)
        throws IOException {
        Subject subject = (Subject) session.getAttribute("subject");
        if (subject == null) {
            log("Session is not valid (no subject).");

            session.invalidate();

            req.setAttribute("message", "The Session is no longer valid.");
            resp.sendRedirect(LOGON_PAGE);
            return false;
        }
        return true;
    }
}

```

main.jsp

```
<!--
/*****
 * FILE: %Z%M% %I% %W% %G% %U%
 *
 * Description:
 *
 * JSP that handles the "main" or "home" page.
 * This is the page the user is forwarded to after a successful login.
 *
 * Licensed Materials - Property of IBM
 * @version 2.0
 * @author aannas
 *****/
-->

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<%@ page language="java" contentType="text/html; charset=UTF-8"%>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=UTF-8" >
<link href="css/imperative.css" rel="stylesheet" type="text/css">
<title>Tivoli Austin Airlines Self Care - Home Page</title>
<%@ page import="examples.expi.*" %>
<%@ include file="expiProlog.jsp" %>

</HEAD>

<%@ include file="expi_header.html" %>

<table border=0 cellpadding=3 cellspacing=0 width=100%>
  <tr>
    <td width="80"></td>
    <td width="100%">
      <table border=0 cellpadding=0 cellspacing=0 width=90%>
        <tr>
          <td></td>
        </tr>
        <tr>
          <td class="heading-text">Home</td>
        </tr>
        <tr>
          <td class="heading-line"></td>
        </tr>
        <tr>

```

```

        <td class="text-description">Welcome!<br></td>
    </tr>

    <!-- Display any errors -->
    <tr>
        <td>
            <%@ include file="error_msg_box.jsp" %>
        </td>
    </tr>

    <!-- Warn about needed updates to challenge/response answers -->
    <tr>
        <td>
            <%@ include file="cr_warning_msg_box.jsp" %>
        </td>
    </tr>

    <tr>
        <td class="text-normal"><br>
        <ul>
            <li><A href="selfCareServlet">Change My Personal
Information</A>
            <li><A href="changepwd.jsp">Change My Password</A>
            <li><A href="ChangeChallengeResponseServlet">
                Change My Challenge/Response Answers</A>
            <li><A href="todolistServlet">
                View and Complete My To Do List Assignments</A>
            <li><A href="ViewMyAccountsServlet">
                View My Accounts</A>

            <!-- If TAM is installed, put up subscribe link -->
            <%
                String sTAMConfigured =
                (String)session.getAttribute(expiUtil.TAM_SERVICE_ACTIVE);
                if (sTAMConfigured != null &&
                sTAMConfigured.equalsIgnoreCase("true")) {
                    <%
                        <li><A href="applicationServlet">Subscribe to Applications
with Tivoli Access Manager</A>
                    <%
                        }else{
                    <%
                        <li><A href="applicationServlet">Subscribe to
Applications</A>
                    <%
                        }
                    <%
                }
            <%
            </ul>
        </td>
    </tr>

```

```
        </td>
    </tr>

</table>
</td>
</tr>

</table>

</html>
```




Tivoli Directory Server proxy server

In this appendix we provide a general short introduction to the Tivoli Directory Server proxy server component.

The *proxy server* is a special type of IBM Tivoli Directory Server that can provide request routing, load balancing, failover, distributed authentication, and support for distributed/membership groups and partitioning of containers. Most of these functions are provided in a new back-end, the proxy back-end. The proxy server does not have an RDBM back-end and cannot take part in replication. A directory proxy server sits at the front end of a distributed directory and provides efficient routing of user requests, therefore, improving performance in certain situations and providing a unified directory view to the client. The proxy server also provides data support for groups and ACLs that are not affected by partitioning and support for partitioning of flat namespaces. It can also be used at the front end of a server cluster for providing failover and load balancing. For high availability, it is not necessary to partition the data and distribute it across multiple servers, but instead, each server can be a complete replica of the other, as shown in Figure C-1 on page 386.

The proxy server is configured with connection information to connect to each of the back-end servers for which it is proxying. The connection information consists of host address, port number, bind DN, credentials, and a connection pool size. Each of the back-end servers is configured with the DN and

credentials that the proxy server uses to connect to it. The DN must be a member of the back-end server's (local) administration group or local administrator. Finally, the proxy server is configured with its own schema. You need to ensure that the proxy server is configured with the same schema as the back-end servers for which it is proxying. The proxy server must also be configured with partition information.

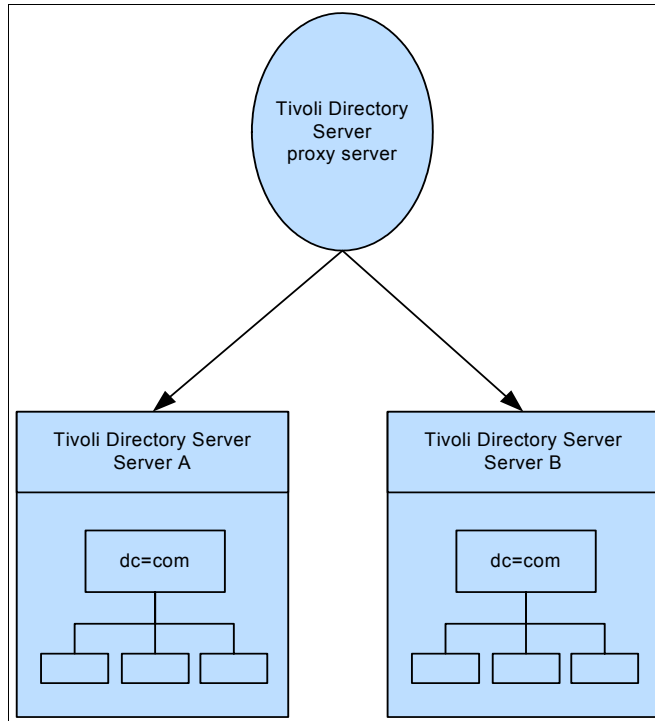


Figure C-1 Distributed directory with a Directory Server proxy server

The proxy server is aware of all the replicas for a given partition, and load balances read requests between the online replicas. The proxy server is aware of all of the masters for a given partition, and must use one of these as the primary master. The first master found in the partition is the primary master. If the primary master server is down, the proxy server is capable of failing over to a backup server (one of the other master servers). If the requested operation cannot be performed by the servers that are currently online, the proxy server returns an operations error. Note: For better performance, all back-end servers and the proxy server should share the same stash files. The proxy server performs load balancing for read requests, and failover for update requests. If a back-end server is unavailable, the operation will error out. All subsequent operations will fail over to the next available server.

In a proxied directory, failover support between proxies is provided by creating an additional proxy server that is identical to the first proxy server. These are not the same as peer masters, the proxy servers have no knowledge of each other and must be managed through a load balancer. A Tivoli Directory Server proxy server failover configuration is shown in Figure C-2 on page 387.

A load balancer, such as the IBM WebSphere Edge Server, has a virtual host name that applications use when sending updates to the directory. The load balancer is configured to send those updates to only one server. If that server is down, or unavailable because of a network failure, the load balancer sends the updates to the next available proxy server until the first server is back online and available. Refer to your load balancer product documentation for information about how to install and configure the load balancing server.

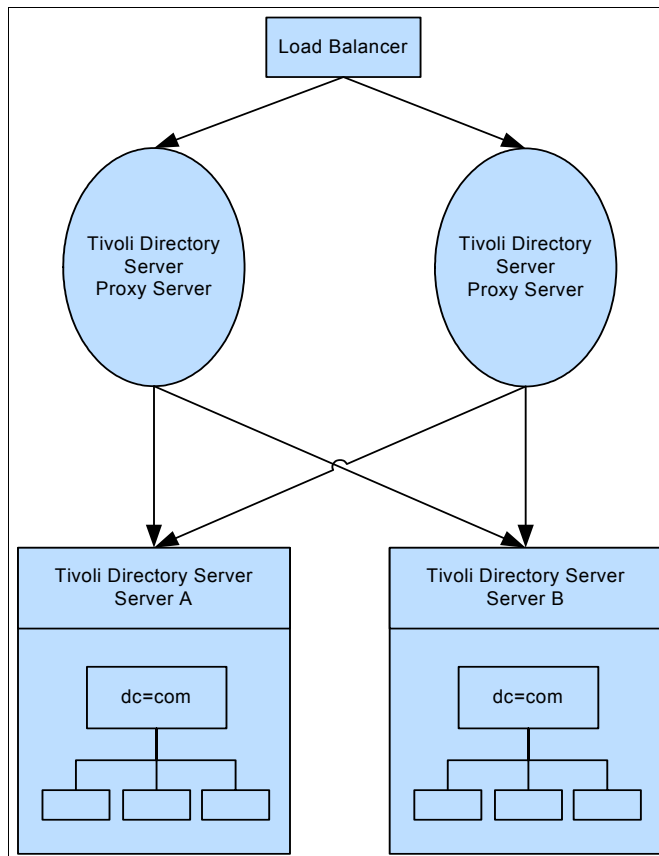


Figure C-2 Failover configuration for Tivoli Directory Server proxy server

Note: In a load-balanced proxy environment, if a proxy server fails, the first operation sent to it fails and returns an error. All subsequent operations are sent to the failover proxy server. The first operation that failed can be retried. It is not automatically sent to the failover server.

For more comprehensive information about configuring the Tivoli Directory Server proxy server, such as using the command line interfaces, see Chapter 15, “Distributed directories”, in the *IBM Tivoli Directory Server Version 6.1 Administration Guide*, SC32-1564.



Additional material

This redbook refers to additional material that you can download from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247242>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG247242.

Using the Web material

Note: The Identity Manager updates (Fixpacks and Intrim Fixes) should be reviewed for items which might affect the materials listed within this guide. Specifically, Fixpack 14 includes changes to the Self-Care application shipped with the Identity Manager product, for example.

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
itim_expi.war	Self-care Web Archive
itim_expi.properties	Properties file for self-care war
BPList.txt	Delegated Admin input list
BulkFeedAdminDomain.sh	Delegated Admin Load Script
custom.jar	Delegated Admin custom classes

System requirements for downloading the Web material

We recommend the following system configuration:

Important: For the system sizing, CPU, Operating System version and type, and memory configuration should be taken from the Deployment plan created for the deployment of the product. The values below are simply minimum values and might not work for all installations.

Hard disk space:	/tmp: 512 MB, ITIM_HOME: 500 MB
Operating System:	W2k3 Server, RH3EL, Solaris9, AIX 5.2, 5.3
Processor:	1 Ghz, 500 Mhz, 440 Mhz, 375 Mhz
Memory:	2GB

How to use the Web material

This section explains how to deploy the self-care application and how you can install the delegated administration custom.jar file.

Self-care application

Note: The self-care application modifications were developed and tested solely on Windows and, therefore, might need some modifications to work properly if used on a different platform.

WebSphere installation

Copy the Web archive file to the system where you are installing the self-care application. Open the WebSphere administrative console running on the system where you are installing the self-care application.

Follow these steps:

1. Under **Applications** → **Enterprise Applications**, select **Install**.
2. For **Local Path**, specify the location of the itim_expi.war file.
3. For **Context Root**, type /itim_expi
4. Repeatedly click **Next** until you arrive at the Summary window.
5. Once at the Summary window, click **Finished**.
6. Save your settings.
7. Copy the itim_expi.properties file to the \$WEBSPPHERE_HOME/properties directory (where \$WEBSPPHERE_HOME is the directory where WebSphere Application Server is installed) on the system where the self-care application is being installed.
8. Append the following line to the wsjaas.conf file in the \$WEBSPPHERE_HOME/properties directory (where \$WEBSPPHERE_HOME is the directory where WebSphere Application Server is installed) on the system where the self-care application is being installed:

```
ITIM {  
  com.ibm.ws.security.common.auth.module.proxy.WSLoginModuleProxy required  
  delegate=com.ibm.itim.apps.jaas.spi.PlatformLoginModule;  
};
```

Restart the WebSphere Application Server on the system where you are installing the self-care application.

SSO configuration

In order for the TAA personnel as well as business partners to use the self-care application, the current single sign-on environment must be configured for the new application. The following discusses how to set up the self-care application with TAA's Tivoli Access Manager SSO environment.

1. Create a public junction from WebSEAL to the self-care application.

A WebSEAL junction to the self-care application can be created using either the Access Manager Web Portal Manager, or by using the pdadmin command line interface. The example below shows setting up a WebSEAL junction using pdadmin. Note that this example is based on the:

- WebSEAL server named default-webseald-w2kalpha
- IBM HTTP server host named w2kalpha

Use the `pdadmin server list` command to list your WebSEAL servers, and use the correct IBM HTTP server host name when you create the WebSEAL junction.

```
C:\>pdadmin -a sec_master -p password
pdadmin sec_master> server list (can be abbreviated to 's l')
    amwpm-w2kalpha
    default-webseald-w2kalpha
pdadmin sec_master> s t default-webseald-w2kalpha list
/
pdadmin sec_master> s t default-webseald-w2kalpha create -t tcp -h w2kalpha
-c iv-user -j -e utf8_bin /ihs
Created junction at /ihs
pdadmin sec_master> s t default-webseald-w2kalpha list
/
/ihs
pdadmin sec_master> s t default-webseald-w2kalpha show /ihs
    Junction point: /ihs
    Type: TCP
    Basic authentication mode: filter
    Authentication HTTP header: insert - iv_user
    Scripting support: yes
    Preserve cookie names: no
    Delegation support: no
    Mutually authenticated: no
    Insert WebSphere LTPA cookies: no
    Insert WebSEAL session cookies: no
    Request Encoding: UTF-8, Binary
    Server 1:
        Server State: running
        Hostname: w2kalpha
```

- **-c iv-user**: Insert the Tivoli Access Manager user ID into requests sent to the junctioned server (sent in HTTP header *iv-user*).
- **-j**: Store a cookie on the browser that is used for junction resolution (this is required for some applications and is also required for Identity Manager to function correctly).
- **-e utf8_bin**: Option for utf8_bin encoding of requests. utf8_bin required only if user IDs containing a blank need to be sent to Identity Manager (for example, the user ID *itim manager*).

2. Enable SSO for Identity Manager.

Edit {ITIM_InstallDir}/data/ui.properties file and set `ssoEnable=true`

```
# Single Sign On enabled (true|false)
enrole.ui.ssoEnabled=true
```

Edit {ITIM_InstallDir}/data/enRoleAuthentication.properties and set `idsEqual=true`

```
# THE FOLLOWING IS FOR WEBSEAL AUTHENTICATION PROVIDER ONLY
# Set to true if all ITAM IDs equal ITIM IDs
# Set to false if any ITAM ID does not equal an ITIM ID
enrole.authentication.idsEqual=true
```

3. Create a test user common to Access Manager and Identity Manager for testing.

```
pdadmin sec_master> user create "itim manager" "cn=itim manager,o=ibm,c=us"
"itim manager" manager password
pdadmin sec_master> user modify "itim manager" account-valid yes
```

4. Configure Access Manager link for forgotten passwords.

Edit the Access Manager login.html files:

```
/opt/pdweb/html.tivoli/lib/html/C/login.html
```

```
/opt/pdweb/www-default/lib/html/C/login.html
```

Alter the URL for forgotten passwords, so that it uses the junction created earlier.

5. Access the Access Manager login page and click on the forgotten password link.

Surf to the regular Access Manager Login page.

Click on the forgotten password link. You should be redirected to the self-care application.

6. Reset the user's password.

Use the self-care application to reset the user's password. Test the new password, once it has been successfully changed.

Delegated administration custom.jar installation

1. Copy custom.jar to the `<WAS_HOME>/AppServer/installedApps/<server name>/enRole.ear/`

2. Update the fesiextensions.properties file by adding the line:

```
fesi.extension.Workflow.AdminDomainModel=com.ibm.itim.custom.fesiextensions
.AdminDomainModelExtension
```

3. Restart Identity Manager.

Glossary

access (1) The ability to read, update, delete, or otherwise use a resource. Access to protected resources is usually controlled by system software. (2) The ability to use data that is stored and protected on a computer system.

access control In computer security, the process of ensuring that the resources of a computer system can be accessed only by principals in authorized ways.

access control item (ACI) Data that (a) identifies the permissions of principals and (b) is assigned to a resource.

access control list In computer security, a list that is associated with a resource that identifies all the principals that can access the resource and the permissions for those principals.

account An entity that contains a set of parameters that define the application-specific attributes of a principal, which include the identity, user profile, and credentials.

ACI target The resource for which you define the access control items. For example, an ACI target can be a service.

adapter (1) A set of software components that communicate with an integration broker and with applications or technologies in order to perform tasks, such as executing application logic or exchanging data. (2) A transparent, intermediary software component that allows different software components with different interfaces to work together.

administrative domain A logical collection of resources that is used to separate responsibilities and manage permissions.

adopt To assign an orphan account to the appropriate owner.

adoption rules The set of rules that determine which orphan accounts belong to which owners.

agent A process that manages target resources on behalf of a system in order to respond to requests.

aggregate message A collection of notification messages that are combined into a single e-mail, along with optional user-defined text.

alias In identity management, an identity for a user, which might match the user ID. The alias is used during reconciliation to determine who owns the account. A person can have several aliases, for example, GSmith, GWSmith, and SmithG.

approval A type of workflow activity that allows someone to approve or reject a request. See also workflow.

audit trail A chronological record of events or transactions. You can use audit trails for examining or reconstructing a sequence of events or transactions, managing security, and for recovering lost transactions.

authentication The process of verifying that an entity is the entity that it claims to be, often by verifying a user ID and password combination. Authentication does not identify the permissions that a person has in the system.

authorization The process of granting a user either complete or restricted access to an object, resource, or function.

Certificate Authority (CA) An organization that issues certificates. The CA authenticates the certificate owner's identity and the services that the owner is authorized to use, issues new certificates, renews existing certificates, and revokes certificates that belong to users who are no longer authorized to use them.

challenge-response authentication An authentication method that requires users to respond to a prompt by providing information to verify their identity when they log in to the system. For example, when users forget their password, they are prompted (challenged) with a question to which they must provide an answer (response) in order to either receive a new password or receive a hint for specifying the correct password.

Common Criteria A standardized method, which is used by international governments, the United States federal government, and other organizations, for expressing security requirements in order to assess the security and assurance of technology products.

connector A plug-in that is used to access and update data sources. A connector accesses the data and separates out the details of data manipulations and relationships.

credentials Authentication information that is associated with a principal.

CSV In computers, a CSV (comma-separated values) file contains the values in a table as a series of ASCII text lines organized so that each column value is separated by a comma from the next column's value and each row starts a new line. Here's an example:

```
Doe,John,944-7077
Johnson,Mary,370-3920
Smith,Abigail,299-3958
```

A CSV file is a way to collect the data from any table so that it can be conveyed as input to another table-oriented application. Spreadsheet programs or relational database applications can read CSV files. A CSV file is sometimes referred to as a flat file.

DAC Discretionary access control (DAC) is used to control access by restricting a subject's access to an object. It is generally used to limit a user's access to a file. In this type of access control, it is the owner of the file who controls other users' accesses to the file. Using a DAC mechanism allows users control over access rights to their files. When these rights are managed correctly, only those users specified by the owner may have some combination of read, write, execute, and so on permissions to the file.

DAML Directory Access Markup Language. An XML specification that extends the functions of Directory Services Markup Language (DSML) 1.0 in order to represent directory operations. In Tivoli Identity Manager, DAML is mainly used for server to agent communications. See also Directory Services Markup Language v2.0.

Directory server A server that can add, delete, change, or search directory information on behalf of a client.

Directory Services Markup Language v1.0 (DSMLv1) An XML implementation that describes the structure of data in a directory and the state of the directory. DSML can be used to locate data into a directory. DSMLv1 is an open standard defined by OASIS. Contrast with Directory Services Markup Language v2.0.

Directory Services Markup Language v2.0

(DSMLv2) An XML implementation that describes the operations that a directory can perform (such as how to create, modify, and delete data) as well as the results of those operations. While DSMLv1 can be used to describe the structure of data in a directory, DSMLv2 can be used to communicate with other products about that data. DSMLv2 is an open standard defined by OASIS. Contrast with DSMLv1.

distinguished name (DN) The name that uniquely identifies an entry in a directory. A distinguished name is made up of name-component pairs. For example, CN=John Doe, O=My Organization, C=US.

domain administrator The owner of an administrative domain.

dynamic content tags A set of XML tags (based on the XML Text Template Language (XTTL) schema) that allows the administrator to provide customized information in a message, notification, or report.

dynamic organizational role An organizational role that is assigned to a person by using an LDAP filter. When a user is added to the system and the LDAP filter parameters are met, the user is automatically added to the dynamic organizational role.

Eclipse Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. More information can be obtained at: <http://www.eclipse.org/>

entitlement In security management, a data structure, service, or list of attributes that contains externalized security policy information.

entitlement workflow A workflow that defines the business logic that is used when provisioning a policy. For example, an entitlement workflow is used to define approvals for managing accounts.

entity A person or object about which you want to store information or manage. For example, a person and an organization are both entities.

entity type Categories of managed objects. See also entity.

escalation The process that defines what happens and who acts when an activity has not been completed in the specified amount of time.

escalation limit The amount of time, for example, hours or days, that a participant has to respond to a request, before an escalation occurs.

event The encapsulated data that is sent as a result of an occurrence, or situation, in the system.

failover An operation that switches a system to a redundant or standby system when services fail.

FESI extension A Java extension that can be used to enhance JavaScript code and then be embedded within a FESI script.

Free EcmaScript Interpreter (FESI) An implementation of the EcmaScript scripting language, which is an ISO standard scripting language that is similar to the JavaScript scripting language.

group A collection of Tivoli Identity Manager users.

identity The subset of profile data that uniquely represents a person or entity and that is stored in one or more repositories.

identity feed The automated process of creating one or more identities from one or more common sources of identity data.

identity policy The policy that defines the user ID to be used when creating an account for a user.

IIOB (Internet Inter-ORB Protocol) A protocol that is used for communication between Common Object Request Broker Architecture (CORBA) object request brokers (ORBs).

JDBC Java Database Connectivity is an application program interface (API) specification for connecting programs written in Java to the data in popular databases. The application program interface lets you encode access request statements in SQL that are then passed to the program that manages the database. It returns the results through a similar interface.

JMS Java Message Service is an application program interface from Sun Microsystems™ that supports the formal communication known as messaging between computers in a network. Sun's JMS provides a common interface to standard messaging protocols and also to special messaging services in support of Java programs. Sun advocates the use of the Java Message Service for anyone developing Java applications, which can be run from any major operating system platform.

JNDI Java Naming and Directory Interface™ enables Java platform-based applications to access multiple naming and directory services. Part of the Java Enterprise application programming interface (API) set, JNDI makes it possible for developers to create portable applications that are enabled for a number of different naming and directory services, including file systems, directory services, such as Lightweight Directory Access Protocol (LDAP), Novell Directory Services, and Network Information System (NIS), and distributed object systems, such as the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), and Enterprise JavaBeans™ (EJB).

join directive The set of rules that defines how to handle attributes when two or more provisioning policies are applied. Two or more policies might have overlapping scope, so the join directive specifies what actions to take when this overlap occurs.

JSP JavaServer Page is a technology for controlling the content or appearance of Web pages through the use of servlets, which are small programs that are specified in the Web page and run on the Web server to modify the Web page before it is sent to the user who requested it.

Kerberos Kerberos is a secure method for authenticating a request for a service in a computer network. Kerberos was developed in the Athena Project at the Massachusetts Institute of Technology (MIT). The name is taken from Greek mythology; Kerberos was a three-headed dog who guarded the gates of Hades. Kerberos lets a user request an encrypted "ticket" from an authentication process that can then be used to request a particular service from a server. The user's password does not have to pass through the network.

LDAP Lightweight Directory Access Protocol is a software protocol for enabling anyone to locate organizations, individuals, and other resources, such as files and devices, in a network, whether on the public Internet or on a corporate intranet. LDAP is a "lightweight" (smaller amount of code) version of Directory Access Protocol (DAP), which is part of X.500, a standard for directory services in a network.

LDIF (LDAP Data Interchange Format) A file format that is used to describe directory information as well as changes that need to be applied to a directory, such that directory information can be exchanged between directory servers that are using LDAP.

lifecycle Passage or transformation through different stages over time. For example, markets, brands, and offerings have life cycles.

lifecycle rules A set of rules in a policy that determines which operations to use when automatically handling commonly occurring events, such as suspending an account that has been inactive for a period of time.

location An entity that is a subdivision of an organization, usually based on geographical area.

MAC The need for a mandatory access control (MAC) mechanism arises when the security policy of a system dictates that protection decisions must not be decided by the object owner and the system must enforce the protection decisions (for example, the system enforces the security policy over the wishes or intentions of the object owner).

The POSIX.6 standard provides support for a mandatory access control policy by providing a labeling mechanism and a set of interfaces that can be used to determine access based on the MAC policy.

managed resource An entity that exists in the run-time environment of an IT system and that can be managed.

MASS IBM Method for Architecting Secure Solutions.

operation An action that can be performed against an object; for example, add, modify, or delete.

operational workflow A workflow that defines the life cycle process for accounts, persons, and other entities.

organization A hierarchical arrangement of organizational units, such that each user is included once and only once.

organization tree A hierarchical structure of an organization that provides a logical place to create, access, and store organizational information.

organizational container An organization, organizational unit, location, business partner unit, or administration domain.

organizational role In identity management, a list of account owners that is used to determine which entitlements are provisioned to them.

organizational unit A type of organizational container that represents a department or similar grouping of people.

orphan account On a managed resource, an account whose owner cannot be automatically determined by the provisioning system.

password retrieval The method of retrieving a new or changed password by accessing a designated Web site and specifying a shared secret.

password strength policy A policy that defines the password strength rules. A password strength policy is applied whenever a password is set or modified.

password strength rules The set of rules that a password must conform to, such as the length of the password and the type of characters that are allowed (or not allowed) in the password.

password synchronization The process of coordinating passwords across services and systems such that only a single password is needed to access those multiple services and systems.

person An individual in the system that has a person record in one or more corporate directories.

post office A component that collects notifications from the appropriate workflow activities and distributes those notifications to the appropriate workflow participants.

provisioning The process of providing, deploying, and tracking a service or component.

provisioning policy A policy that defines the access to various managed resources, such as applications or operating systems. Access is granted to all users, users with a specific role, or users who are not members of a specific role.

RBAC With RBAC (Role-Based Access Control), security is managed at a level that corresponds closely to the organization's structure. Each user is assigned one or more roles, and each role is assigned one or more privileges that are permitted to users in that role. Security administration with RBAC consists of determining the operations that must be executed by persons in particular jobs, and assigning employees to the proper roles. Complexities introduced by mutually exclusive roles or role hierarchies are handled by the RBAC software, making security administration easier.

reconciliation The process of synchronizing data in a central data repository with data on a managed resource.

request for information (RFI) A workflow activity that requests additional information from the specified participant.

ROI For a given use of money in an enterprise, the ROI (return on investment) is how much "return," usually profit or cost saving, results. An ROI calculation is sometimes used along with other approaches to develop a business case for a given proposal. The overall ROI for an enterprise is sometimes used as a way to grade how well a company is managed. If an enterprise has the immediate objectives of getting market revenue share, building infrastructure, positioning itself for sale, or other objectives, a return on investment might be measured in terms of meeting one or more of these objectives rather than in immediate profit or cost saving.

rule A set of conditional statements that enable computer systems to identify relationships and execute automated responses accordingly.

schema The fields and rules in a repository that comprise a profile.

scope In identity management, the set of entities that a policy or an access control item (ACI) can affect.

service A representation of a managed resource, application, database, or system.

service owner A role that identifies the person who owns and maintains a particular service in Tivoli Identity Manager. See also service.

service selection policy A policy that determines which service to use in a provisioning policy. See also provisioning policy.

service type A category of related services that share the same schemas. See also service.

SOAP Simple Object Access Protocol is a way for a program running in one kind of operating system to communicate with a program in the same or another kind of an operating system by using the HTTP Protocol and XML as the mechanisms for information exchange.

SSL The Secure Sockets Layer is a commonly-used protocol for managing the security of a message transmission on the Internet. SSL has recently been succeeded by Transport Layer Security (TLS), which is based on SSL.

static organizational role An organizational role that is manually assigned to a person. See also organizational role.

supervisor A role that identifies the person who supervises another set of users and who is often responsible for approving or rejecting requests that are made by those users.

suspend To deactivate an account so that the account owner cannot access the service.

system administrator A role that identifies the person who is responsible for the configuration, administration, and maintenance of Tivoli Identity Manager.

universally unique identifier (UUID) The 128-bit numerical identifier that is used to ensure that two entities do not have the same identifier. The identifier is unique for all space and time.

work order A workflow activity that requires a participant to perform an activity outside of the scope of the system.

workflow The sequence of activities performed in accordance with the business processes of an enterprise.

XML Extensible Markup Language is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. For example, computer makers might agree on a standard or common way to describe the information about a computer product (processor speed, memory size, and so forth) and then describe the product information format with XML. Such a standard way of describing data would enable a user to send an intelligent agent (a program) to each computer maker's Web site, gather data, and then make a valid comparison. XML can be used by any individual or group of individuals or companies that want to share information in a consistent way.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 404. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014
- ▶ *Identity Management Design Guide with IBM Tivoli Identity Manager*, SG24-6996
- ▶ *Deployment Guide Series: IBM Tivoli Identity Manager*, SG24-6477
- ▶ *IBM WebSphere V5.1 Performance, Scalability, and High Availability WebSphere Handbook Series*, SG24-6198
- ▶ *WebSphere Scalability: WLM and Clustering Using WebSphere Application Server Advanced Edition*, SG24-6153

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Tivoli Identity Manager Database and Schema Reference Version 4.6*, SC32-1769
- ▶ *IBM Tivoli Identity Manager Information Center Version 4.6*, SC23-5267
- ▶ *IBM Tivoli Identity Manager Planning for Deployment Guide Version 4.6*, SC32-1708
- ▶ *IBM Tivoli Identity Manager Problem Determination Guide Version 4.6*, SC32-1491-01
- ▶ *IBM Tivoli Identity Manager Version 4.6: Release Notes*, GI11-4212-03
- ▶ *IBM Tivoli Identity Manager Version 4.6: Manager Server Installation and Configuration Guide for WebSphere Environments Version 4.6*, SC32-1750-01

- ▶ *IBM Tivoli Directory Integrator 6.0: Getting Started Guide*, SC32-1716
- ▶ *IBM Tivoli Directory Integrator 6.0: Users Guide*, SC32-1718
- ▶ *IBM Tivoli Directory Integrator 6.0: Reference Guide*, SC32-1720
- ▶ *IBM Tivoli Directory Server Administration Guide Version 6.1*, SC32-1564
- ▶ *IBM Tivoli Directory Server Installation and Configuration Guide Version 6.1*, SC32-1560

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ The online product documentation *IBM Tivoli Identity Manager Information Center Version 4.6*, SC23-5267 can be obtained at the following address:
<http://publib.boulder.ibm.com/tividd/td/IdentityManager4.6.html>
- ▶ The online product documentation *IBM DB2 UDB Version 8.2 Information Center* can be obtained at the following address:
<http://publib.boulder.ibm.com/infocenter/db2luw/v8//index.jsp>
- ▶ National Institute of Standards and Technologies homepage.
<http://www.nist.gov/>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

- Access Control Item
 - see ACI
- Access Manager 84
 - Policy Server 105
 - single sign-on 230
- account 19, 35
 - certification process 290
 - management 7, 10, 231
 - management module 8
 - recertification 115, 213
 - subscription 213, 220
 - suspension 102
- AccountMO 221, 223
- ACI 21, 228, 231, 251
 - principal 231
 - target 232
- adapter
 - connectivity 12
 - event notification 81
 - high availability 76
 - SSL communication 89
- admin domain 232–233
 - bulk feed 240
- administration
 - API 12
 - delegation 84, 120, 210, 230
- administrator
 - business partner 235
- AES256 encryption 160
- Analysis
 - API package 42
- API 13
 - Analysis package 42
 - Application package 15, 53
 - Authentication package 17
 - Data Services package 19
 - Domain package 19, 26
 - JAAS 14
 - Logging package 39
 - Mail package 40
 - Model package 19–20, 48
 - Password rules package 43

- persistent storage 19
- Policy package 19, 41
- Provider package 46
- Provisioning package 52
- Query package 51
- Remote Services package 46
- Schema package 20
- System package 18–19, 37
- Workflow package 19, 47

- application
 - API package 53
 - interface module 7
 - subscription 213
- Application Programming Interface
 - see API
- application server
 - high availability 63, 125
- applications JSP 225
- applications.jsp 223
- applicationServlet.java 221
- approval 120
 - request 243
- asynchronous messaging 10
- audit 114, 116, 120, 231
 - trail 11
- authentication
 - API package 17
 - module 9
- authorization
 - module 10
- availability
 - monitoring 132

B

- backup 117
- business partner
 - administrator 235
- business process 231
 - customization 242
- business requirements 113

C

- cascading replication 68

- certification
 - lifecycle rules 306
 - process 290
- challenge/response 27, 212, 214
- change
 - management 117
- circular logging 140
- cluster
 - configuration 125
 - topology 128
- compliance 114, 116, 231
 - requirement 56
- component
 - placement 83
- configuration
 - ... of ports 86
- connectivity 12
- continuous operation 122, 125
- controlled network 84
- CORBA 398
- credential 105
- Crystal Reports 58, 249
- custom
 - person type 253
 - report design 210, 248
 - operational targets 250
 - reports 58
 - service provider 57

D

- DAC 396
- DAML 12, 396
- data
 - management zones 108
 - services module 11
 - store
 - schema 20
- Data Services
 - API package 19
- database 11
 - high availability 73, 132
 - stand-by 133
- dataservices API 240
- DB2
 - encrypted communications 89
 - HADR 132
 - configuration 138–139
 - takeover 150
 - HADR security 90
 - high availability 132
 - High Availability Disaster Recovery 75
 - mutual takeover multiple partition 74
 - delegated administration 84, 120, 230
 - delegation
 - management 39
 - deployment
 - descriptor 225
 - manager 63
 - requirement 210
 - deprovisioning 102
 - design
 - custom reporting 249
 - delegated administration 230
 - objectives 117
 - organization tree 233
 - self-care 212
 - desktop module 6
 - development cycle 212
 - directory
 - proxy server 385
 - Directory Information Tree 68
 - Directory Integrator 13
 - Directory Server
 - AES256 encryption 160
 - global administration group 191
 - high availability 67, 152
 - Master-Slave replication 161
 - Multiple-Master replication 161
 - peer-to-peer replication 161, 182
 - peer-to-peer topology 153
 - proxy server 72, 153, 185
 - load balancing 186
 - server group 200
 - secondary instance 154
 - suffix 158
 - Discretionary Access Control 396
 - DMZ 84
 - Domain
 - API package 19, 26
 - domain
 - administrator 232
 - granting ACIs 244
 - hold container 234
 - DSML 12–13
 - dynamic role 8, 37

E

- Eclipse 212
- EcmaScript 55
- EJB 398
- entitlement workflow 56
- entity
 - management module 8
- erPersonItem 232
- event notification 81
- Extensible Markup Language 401
- extension API 14
- extranet 84
 - project phase 120
 - requirements 115

F

- FESI
 - extensions 55–56
 - JavaScript interpreter 55
- FESI extension 236
- Firefox viii
- firewall
 - port configuration 86
- form
 - creation 5
 - design module 6
 - rendering module 6
- Free ECMA Script Interpreter
 - see FESI
- functional requirement 114, 210

G

- global administration group 191
- group 231
 - management 39
- GUI server 84

H

- HADR 132
 - circular logging 140
 - configuration 138–139
 - takeover 150
- help desk
 - costs 114
- high availability 61, 115, 117, 122
 - adapter 76
 - application server 63, 125

- database 73, 132
- Directory Server 152
- directory server 67
- horizontal scaling 64
- LDAP 67
- physical component architecture 83
- project phase 119
- security configuration 87
- self-care application 212
- topology 123
- historical
 - information 11
- horizontal cluster configuration 125
- HTTPS 12
- HumanResourceMO 218

I

- IBM DB2 Universal Database
 - see DB2
- IBM Tivoli Directory Integrator
 - see Directory Integrator
- IBM Tivoli Directory Server
 - see Directory Server
- IBM WebSphere Application Server
 - see WebSphere Application Server
- identity
 - API package 15
 - management 7
 - module 8
 - policy 8, 13
- inactivity code 102
- Incremental Data Synchronizer 251
- inetOrgPerson 232, 253
- interface API 14
- itim_expi.properties 226

J

- J2EE Web application 213
- JAAS
 - API 14
 - API package 16
- Java
 - API 14
 - Mail API 40
- Java Database Connectivity 398
- Java Message Service 10, 398
- Java Naming and Directory Interface 398
- JavaScript

- ... in workflow script 236
- API 13
- extensible framework 55
- JavaServer Pages 398
- JDBC 398
- JMS 10, 398
- JNDI 398
- JSP 398
 - applications 225
 - applications.jsp 223
 - todolist 219

K

- Kerberos 398

L

- LDAP 398
 - automated failover 71
 - directory 11
 - high availability 67
 - manual failover 69
 - master-replica replication 68
 - peer-to-peer replication 68
 - replica server 105
 - replication 68
 - SSL communication 88
- liability 313
- license
 - costs 115
 - fee reduction 114
- lifecycle 8, 398
 - characteristics 22
 - management 55
 - reporting 249
 - recertification 290
 - rule 290
 - rule events 271
 - rule scheduling 284
 - rules 306
- Lightweight Directory Access Protocol 398
- logging 39
 - API package 39
 - module 11
- logical component architecture 4
- logical component design
 - service layer 9
 - Web User Interface 5

M

- MAC 399
- mail
 - API package 40
 - module 11
- maintenance
 - costs 115
 - fee reduction 114
- managed resource 231
- managed service 57
 - provisioning 46
- Mandatory Access Control 399
- master-replica
 - replication model 68
- Master-Slave replication 161
- menu system module 6
- messaging
 - module 10
- Model
 - API package 19–20, 48
- monitoring 132
- Mozilla viii
- Multi-Master replication 161

N

- network diagram 102
- network zone 84
 - controlled 84
 - restricted 85
 - secure 85
- node agent 63
- non-functional requirements 117–118

O

- operation workflow 56, 271, 292
- orchestration
 - module 10
- organization tree 231, 252
 - design 233
 - module 6
- organizational role 237–238, 400
 - bulk feed 240

P

- participant 232
- password
 - change 214

- editing 212
- policy 8
- reset 102
- rule 43, 215
- rules API package 43
- peer-to-peer
 - replication 68, 182
 - topology 153
- performance 117
- persistent storage 19
- personMO 221–222
- personnel management 101
- placement
 - ... of components 83
- policy
 - API package 19, 41
 - corporate 313
 - identity 13
 - management 7, 231
 - management module 8
 - module 9
 - provisioning 13, 238
- port
 - configuration 86
- post office 40, 271, 292, 399
 - configuration 286
- practices 315
- principal 231
- procedures 315
- Provider
 - API package 46
- provisioning 10, 36, 52
 - API package 15, 52
 - policy 8, 13, 42, 56, 238
 - time reduction 114
- proxy server 185, 385
 - load balancing 186
 - server group 200

Q

- Query
 - API package 51
- queue topology 65

R

- RBAC 400
- recertification 115, 210, 213, 217, 244, 290
- reconciliation 12, 400

- recovery 117
- Redbooks Web site 404
 - Contact us xi
- relational database 11
- relationship 23, 231
- remote services
 - API package 46
 - module 11
- replication
 - LDAP 68
- report
 - attribute mapping 254
 - automated delivery 269
 - custom template 255
 - data staging 255
 - design 210
 - filter criteria 255
 - operation workflow creation 273
 - schema 254
 - search filter criteria 261
 - template (XML) 264
- reporting 58
 - custom design 248
 - module 8
- requirements
 - custom report design 248
 - delegated administration 230
 - recertification 290
 - self-care 211
- restricted network 85
- return on investment 400
- reverse proxy 105
- ROI 400
- role 8, 36, 231, 237
 - module 9
- Role Based Access Control 400

S

- schedule information 11
- scheduling
 - module 10
- schema 400
 - API package 20
- script node 56, 236
- search
 - API package 15
 - filter criteria 261
 - module 6

- secure network 85
- Secure Sockets Layer 400
- security
 - configuration 87
 - policy 85
- self-care 84, 115, 210–211, 215
 - application 220
 - application ACI 228
 - application deployment 229, 390
 - application properties 226
 - design 212
 - project phase 120
 - requirements 211
 - user interface 227
- self-registration 210, 213, 215
 - workflow customization 242
- service
 - ... selection policy 8
 - layer 9
 - management 231
 - provider 57
 - provisioning 46
- service level agreement 122
- serviceMO 222
- servlet 213
 - applicationServlet.java 221
 - todolistServlet.java 218
- Simple Object Access Protocol 400
- single sign-on 230
 - configuration 391
- snapshot database 81
- SOAP 400
- sponsor 232
- SSL 400
 - communication design 88
- staging data
 - ... for custom reports 255
- stand-by database 133
- status change 10
- subscription 220
- suffix 158
- supervisor 232
- system
 - API package 18–19, 37
 - configuration module 7
- systems management zone 108

T

- takeover 150
- target 232
- Thunderbird viii
- To Do
 - approval 292
- To Do list 217
- todolist JSP 219
- todolistServlet.java 218
- training 211
- transactional information 11

U

- user management 109

W

- Web User Interface Layer 5
- web.xml 225
- WebSEAL 84, 105
- WebSphere Application Server
 - cluster 63, 125, 127
 - cluster topology 128
 - deployment manager 63
 - horizontal scaling 64
- workflow 401
 - API 13
 - API package 15, 19, 47
 - customization 236
 - customization for self-registration 242
 - design 5
 - extension for database query 271
 - extensions 56
 - management capabilities 48
 - management module 8
 - module 10
 - operation 292
 - operation workflow 271
 - operation workflow for audit report 273
 - participant 232
 - queue topology 65
 - resource 218
- WorkflowAssignmentMO 218

X

- X.500 398
- X.509 certificate authentication 9
- XML 12, 401

IBM
Tivoli
Identity
Manager



Redbooks

Identity Management Advanced Design for IBM Tivoli Identity Manager

(0.5" spine)

0.475" <-> 0.875"

250 <-> 459 pages



Identity Management Advanced Design for IBM Tivoli Identity Manager

Complete self-care scenario using workflow, lifecycle rules, and certification

Identity and user lifecycle management projects are being deployed more and more frequently - and demand is growing. By demonstrating how IBM Tivoli Identity Manager can be made resilient and adapted to special functional requirements, this IBM Redbook creates or enhances confidence in the IBM Tivoli Identity Manager-based solution for senior management, architects, and security administrators.

High availability scenario for WebSphere, DB2, and LDAP

Advanced design topics may start with infrastructure availability for all involved components, Web application, and database server clustering as well as LDAP multi-master setups, continuing with compliance challenges addressing enhanced auditing and reporting, and designing and creating your own self-care/self-registration application environment that embraces external users and business partners offering fine-tuned workflow options and lifecycle management capabilities.

Addressing compliance with audit and reporting

The powerful features and extensions of IBM Tivoli Identity Manager are opening doors into a world of advanced design and customization for every identity management challenge you may encounter.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks