EVIDEN

Identity and Access Management

DirX Identity

High Availability

Version 8.10.15, Edition April 2026

Table of Contents

Preface

This document describes how to set up and configure DirX Identity's high availability features. It consists of the following chapters:

- Chapter 1 describes DirX Identity high availability concepts and implementations, administrative fail-over and automatic fail-over with circular monitoring.
- Chapter 2 describes how to install and perform initial configuration.
- Chapter 3 describes how to configure the Java- and C++-based servers.
- Chapter 4 provides information on customizing the supervisor.
- Chapter 5 provides information on the tool for switching between active and passive DirX Identity configurations.

DirX Identity Documentation Set

*Version 8.10.15 | Build 1932 | Date 2026-04-30 *

The DirX Identity document set consists of the following manuals:

- *DirX Identity Introduction*. Use this book to obtain a description of DirX Identity architecture and components.
- *DirX Identity Release Notes*. Use this book to understand the features and limitations of the current release. This document is shipped with the DirX Identity installation as the file **release-notes.pdf**.
- *DirX Identity History of Changes*. Use this book to understand the features of previous releases. This document is shipped with the DirX Identity installation as the file **history-of-changes.pdf**.
- *DirX Identity Tutorial*. Use this book to get familiar quickly with your DirX Identity installation.
- *DirX Identity Provisioning Administration Guide*. Use this book to obtain a description of DirX Identity provisioning architecture and components and to understand the basic tasks of DirX Identity provisioning administration using DirX Identity Manager.
- *DirX Identity Connectivity Administration Guide*. Use this book to obtain a description of DirX Identity connectivity architecture and components and to understand the basic tasks of DirX Identity connectivity administration using DirX Identity Manager.
- *DirX Identity User Interfaces Guide*. Use this book to obtain a description of the user interfaces provided with DirX Identity.
- *DirX Identity Application Development Guide*. Use this book to obtain information how to extend DirX Identity and to use the default applications.
- *DirX Identity Customization Guide*. Use this book to customize your DirX Identity environment.
- *DirX Identity Integration Framework*. Use this book to understand the DirX Identity framework and to obtain a description how to extend DirX Identity.
- *DirX Identity Web Center Reference*. Use this book to obtain reference information about the DirX Identity Web Center.
- *DirX Identity Web Center Customization Guide*. Use this book to obtain information how to customize the DirX Identity Web Center.
- *DirX Identity Meta Controller Reference*. Use this book to obtain reference information about the DirX Identity meta controller and its associated command-line programs and files.
- *DirX Identity Connectivity Reference*. Use this book to obtain reference information about the DirX Identity agent programs, scripts, and files.
- *DirX Identity Troubleshooting Guide*. Use this book to track down and solve problems in your DirX Identity installation.
- *DirX Identity Installation Guide*. Use this book to install DirX Identity.

- *DirX Identity Migration Guide*. Use this book to migrate from previous versions.

Notation Conventions

Boldface type
In command syntax, bold words and characters represent commands or keywords that must be entered exactly as shown.

In examples, bold words and characters represent user input.

Italic type
In command syntax, italic words and characters represent placeholders for information that you must supply.

[ ]
In command syntax, square braces enclose optional items.

{ }
In command syntax, braces enclose a list from which you must choose one item.

In Tcl syntax, you must actually type in the braces, which will appear in boldface type.

|
In command syntax, the vertical bar separates items in a list of choices.

...
In command syntax, ellipses indicate that the previous item can be repeated.

userID_home_directory
The exact name of the home directory. The default home directory is the home directory of the specified UNIX user, who is logged in on UNIX systems. In this manual, the home pathname is represented by the notation *userID_home_directory*.

install_path
The exact name of the root of the directory where DirX Identity programs and files are installed. The default installation directory is *userID_home_directory*/**DirX Identity** on UNIX systems and **C:\Program Files\DirX\Identity** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathnames is represented by the notation *install_path*.

dirx_install_path
The exact name of the root of the directory where DirX programs and files are installed. The default installation directory is *userID_home_directory*/**DirX** on UNIX systems and **C:\Program Files\DirX** on Windows systems. During installation the installation directory can be specified. In this manual, the installation-specific portion of pathname is represented by the notation *dirx_install_path*.

dxi_java_home
The exact name of the root directory of the Java environment for DirX Identity. This location is specified while installing the product. For details see the sections "Installation" and "The Java for DirX Identity".

tmp_path

The exact name of the tmp directory. The default tmp directory is /tmp on UNIX systems. In this manual, the tmp pathname is represented by the notation *tmp_path*.

tomcat_install_path
The exact name of the root of the directory where Apache Tomcat programs and files are installed. This location is defined during product installation.

mount_point
The mount point for DVD device (for example, **/cdrom/cdrom0**).

1. Overview

DirX Identity provides significant extensions to its load balancing and thus also to its high availability features. As of V8.3, the dynamic load balancing features for Java-based workflows are improved once more and so are the high availability features. As a downside, the recovery features for Tcl-based workflows are slightly reduced.

DirX Identity high availability still focuses on high availability within one site. The implemented solution requires file-based repositories to be accessible from the message brokers, which is usually accomplished with highly-available storage systems in one site. However, this configuration can be a significant cost and performance factor for remote sites, and thus may not always be available.

Workflow implementations that may limit the deployment of high availability include:

- Workflows that import from a file or export to a file, including provisioning workflows, report producers, history record exporters and others.
- Tcl-based workflows with intermediate files, where the activities are distributed across systems.

Administrators can use the Web application Server Admin to get an overview of the state of all Java- and C++-based servers and move functionality between them manually as necessary. Using Server Admin, administrators can:

- Move the scheduler for Java-based workflows to another IdS-J server.
- Move request workflow processing to another IdS-J server.
- Recover the messages of a crashed Java-based server to another IdS-J server.
- Move the configuration handler for forwarding certification changes to another IdS-J server.

The Server Admin functionality comprises **administrative fail-over**.

For **automatic fail-over** DirX Identity supports **Circular monitoring**. In **Circular monitoring** each IdS-J server monitors the state of another server, altogether building a circle. If a monitored server is no longer available, the monitoring server takes over its functionality and the messages not yet fully processed. One of the IdS-J servers monitors all the IdS-C servers. If an IdS-C server is no longer available, it moves the Tcl-based workflows to another IdS-C server.

Note that using DirX Identity's high availability features requires an add-on license that requires the business or the professional suite as a pre-requisite.

Note, too, that the Tcl-based supervisor provided in previous DirX Identity versions cannot be deployed with the new Java-based supervisor, because it also monitors the IdS-C servers and moves the messaging service and Tcl workflows and thus conflicts with these operations in the Java-based supervisor. However, if you have deployed the Tcl-based supervisor, you can continue to run it as long as you don't activate the Java-based supervisor.

The following chapters describe in more detail how to install the high availability features as a whole and then how to configure them.

1.1. Relevant Server Components

The following diagram gives an overview of the Java server components that are important for understanding High Availability:

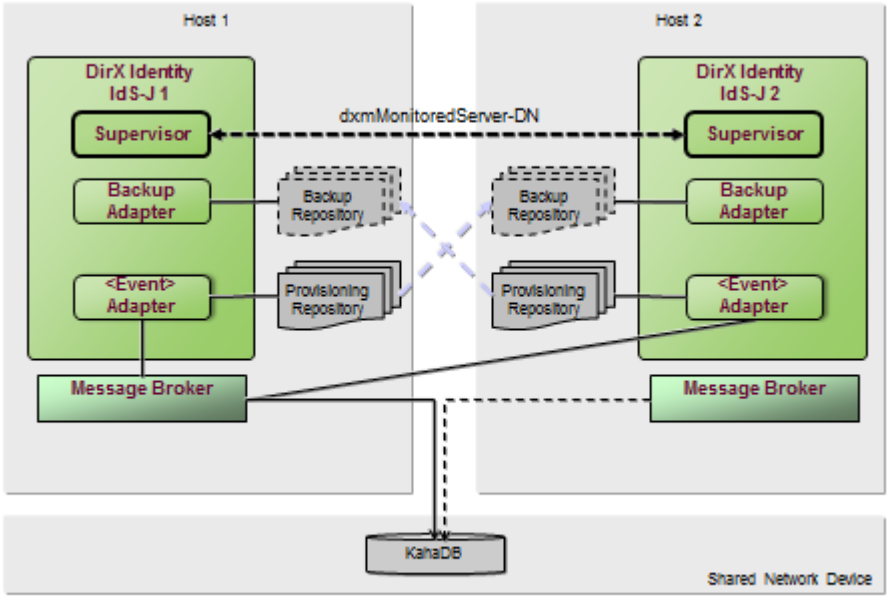

Figure 1. Java-based Server Components

Each Java server is connected to the message broker, realized by Apache ActiveMQ. All JMS clients send their messages to this broker and receive their messages from it. The broker stores the messages in his (shared) repository, implemented by the Apache component KahaDB. For High Availability the repository folder should be located on a shared network device.

The JMS adaptors (for provisioning requests, entry change or password change events) read messages from the message broker and store them in their own local file repository. The adaptors delete a message from their repository only when it is completely processed by the corresponding workflow. The reason for the separate repository is a JMS standard feature: when an adaptor acknowledges a message to the broker, the broker deletes this message and all that were received before. But DirX Identity cannot guarantee that message processing is finished in the order they are obtained from the broker. Processing for some messages takes longer than others. Sometimes errors occur and processing has to be repeated.

If High Availability is activated, each Java server starts its Backup Adaptor. This Backup Adaptor receives messages from the normal JMS adaptors on the monitored Java server and stores them in its local backup repository. When a provisioning or password adaptor on IdS-J2 receives a message from the broker, it immediately sends them to the Backup Adaptor on IdS-J1. When the message has been processed, the JMS adaptor removes it

from its local repository and also instructs the Backup Adaptor to remove it from the backup repository on IdS-J1.

When automatic fail-over is configured, each Java server starts its local supervisor. The supervisor monitors the Java server identified by the Monitored Server link. In the diagram above, IdS-J1 monitors IdS-J2 and vice versa IdS-J2 monitors IdS-J1.

A second message broker can be deployed on any host with a DirX Identity Java server or on any other external server. Only one message broker has exclusive access to the message repository, all other message brokers are locked out and haven't started their connectors for the client. In case the message broker crashes, the database lock is removed, and the next message broker gets the exclusive access to the database (and starts his connectors). There is no algorithm of who is the next broker to take over; it's simply the fastest one. The failover time is about 20 seconds.

1.2. Administrative Fail-over

This section describes how to move functionality from a failed server to a working server manually.

As a pre-requisite, you should have deployed and configured at least two Java-based and two C++-based servers. The message repositories should be located on a shared network device, which is accessible from all the Java servers.

The tool to use here is the Server Admin Web application. It gives an overview of all Java- and C++-based servers and allows you to:

- Recover the messages from the local Backup Adaptor.
- Move the request workflow Timeout Checker processing to another IdS-J server.
- Move the Scheduler for Java workflows to another IdS-J server.
- Move the Configuration Handler to another IdS-J server.
- Disable / enable permanent JMS adaptors.

Server Admin is available if you checked the appropriate selections during installation and initial configuration and is deployed with each IdS-J server. To access it, use the following URL:

http://*your_host*:_port_**/serverAdmin**

By default, the admin port for the first installed IdS-J server is 40000 and for https it is 40001. For each subsequent IdS-J server on the same system, you must choose a different port; for example, 40100, 40200 and so on.

Log in as a user of the DirX Identity domain. Only users in the **ServerAdmins** group in the DirXmetaRole target system are allowed to use Server Admin.

The overview page shows the message brokers, the Java- and the C++-based servers, where you can view the state of each server. For a Java-based server, you can see the set of active adaptors and check boxes that indicate which server is responsible for processing request

workflows and the scheduler.

You can click the **Details** icon to view the details of a selected Java-based server in an extra page. You can click the **Update** button to get an updated server state. For more details on Server Admin, see the *DirX Identity User Interfaces Guide*.

When you notice that an IdS-J server's state has degraded or that the server has dropped out completely, you can move all of its functionality to other IdS-J servers, including:

- The request workflow timeout checker.

The Timeout Checker component actively searches for timeouts of request workflows and their activities and then starts new activities when necessary. Clients - especially Web Center - address their requests to create a new workflow or modify an existing one (for example, in case of an approval) to the IdS-J server, which is currently responsible for the request workflows. If they lose the connection to the request workflow web service, they look up the IdS-J server that is currently configured to host the request workflows and set up a new connection to this server.

- The scheduler for Java workflows.

The scheduler must only be running on one Java server per domain. It is responsible for all schedules of that domain.

- Move JMS adaptors.

Most of the adaptors can be deployed to all servers in parallel, especially those that drive real-time workflows that process events for provisioning, password changes, (user) entry changes, and for mails. Typically, there is no need of moving them.

Especially you should leave the Backup Slave Listener, if high availability is active. They backup the events for the Java server this server is monitoring.

Only the Configuration handler should be deployed at most on one server. It is responsible for distributing changed certificates and message broker configurations – mainly for the Windows Password Listener. This is the only adaptor you can move.

Note that any pending messages cannot be pushed back to the message server. This feature is only available for automatic fail-over.

When a C++-based server fails, the associated workflows and activities have to be moved and – if it was the primary server – also the Status Tracker. Note that the configuration changes are persistently stored in the Connectivity database and thus survive re-starts of all the Java- and C++-based servers. If you want to return to the previous configuration after the failed server is up again, you must do it manually: Perform the move using Server Admin in the same way as previously described. This is the only way to make this change. Changing the configuration with DirX Identity Manager is NOT sufficient, because Manager does not inform the affected servers. They will (de-)activate the corresponding functionality only when they start the next time.

1.2.1. Documentation

To understand this issue, we recommend reading the following chapters:

- *DirX Identity User Interfaces Guide*, especially the chapter on Server Admin.
- *DirX Identity Connectivity Admin Guide*, the chapter on managing Servers.

1.3. Automatic Fail-over with Circular Monitoring

This section describes how to configure the Java-based servers so that they monitor each other as well as the C++-based servers and automatically move functionality from a failed server to an active one.

The message broker setup is independent of this and is used like a black box. Failover of the message broker is done automatically by means of ActiveMQ.

The following diagram illustrates this deployment:

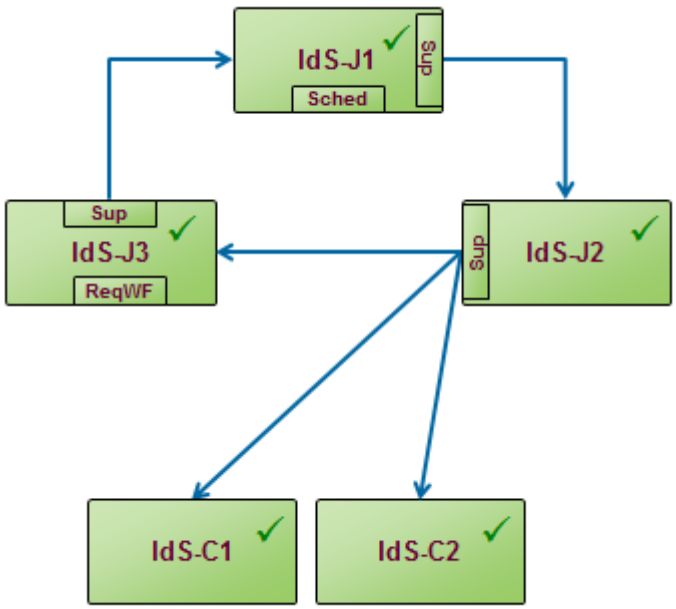

Figure 2. Automatic Fail-over with Circular Monitoring

The deployment comprises several Java-based servers and two C++-based servers. The Java-based servers monitor each other in a circle: IdS-J1 monitors IdS-J2, IdS-J2 monitors IdS-J3 and IdS-J3 monitors IdS-J1. IdS-J1 hosts the scheduler for the Java workflows, IdS-J2 monitors all C++-based servers and IdS-J3 processes the request workflows.

Use DirX Identity Manager to configure this scenario as follows:

- For each of the Java-based server entries in the Connectivity database:
- Activate Automatic Monitoring.
- Enter the monitored Java-based server.
- Enter the supervisor configuration and reference it from each Java-based server. The

supervisor configuration entries are Configuration → Java Supervisors (see DirX Identity Manager's Connectivity View → Expert View). Create your own folder – preferably one per domain – and a new configuration entry. The important fields to be entered are the **Monitoring Interval**, the **Retry Count** and the fields for defining the **mail**. The supervisor sends an e-mail whenever it considers a server to be unavailable and moves functions to another one.

We recommend using the same supervisor configuration for all Java-based servers.

- For exactly one Java-based server, check **Monitor C++-based Servers**.
- For exactly one Java-based server set the flag for the scheduler.
- For exactly one Java-based server set the flag for request workflow Timeout checker.

No special configuration is needed for the C++-based servers: just distribute the Tcl-based workflows and their activities according to your needs.

A supervisor considers a monitored server to be down when it does not respond to a JMX monitor operation (getState) after several (retryCount) repetitions or when the returned state is below a certain limit (4 in a range of 0 to 10). Note that the supervisor recognizes when a server has been intentionally stopped and does not consider this to be a failure. In other words, when a server is intentionally stopped, the supervisor does not automatically take over its services. However, you can perform the move using Server Admin as described in the chapter above. The following diagram illustrates an example.

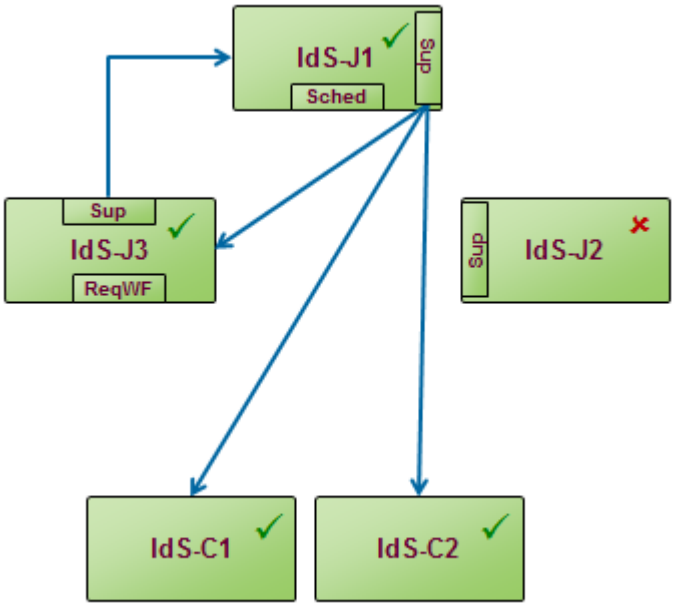

Figure 3. Automatic Fail-over with Circular Monitoring - Java-based Server Down

In this example, let's assume that IdS-J2 is no longer responding. IdS-J1 takes over the monitoring tasks of the IdS-J2 supervisor: it monitors IdS-J3 and all adaptors that are active on IdS-J2, but not on IdS-J1.

The supervisor changes the configuration accordingly in the Connectivity database and requests its hosting IdS-J server to start the additional adaptors.

If IdS-J1 would fail, then IdS-J3 would take over especially the scheduler. Analogous, if IdS-J3 fails, then IdS-J2 would take the responsibility for the request workflows.

When IdS-J2 comes up again, the previous configuration is not automatically restored. The administrator must move the adaptors, the scheduler and/or the request workflow service back to IdS-J2. This is not so for the monitoring tasks, because the supervisor does not change the configuration regarding monitoring. Therefore, IdS-J2 will again monitor IdS-J3 and the IdS-C servers. IdS-J1 continues to monitor IdS-J2 and stops monitoring the others as soon as it considers IdS-J2 to be up and running.

When IdS-C1 fails to respond to the JMX getState() operation, IdS-J2 moves the schedules, workflows and activities to IdS-C2: it changes the configuration in the connectivity database accordingly and requests IdS-C2 to re-start and evaluate the configuration again.

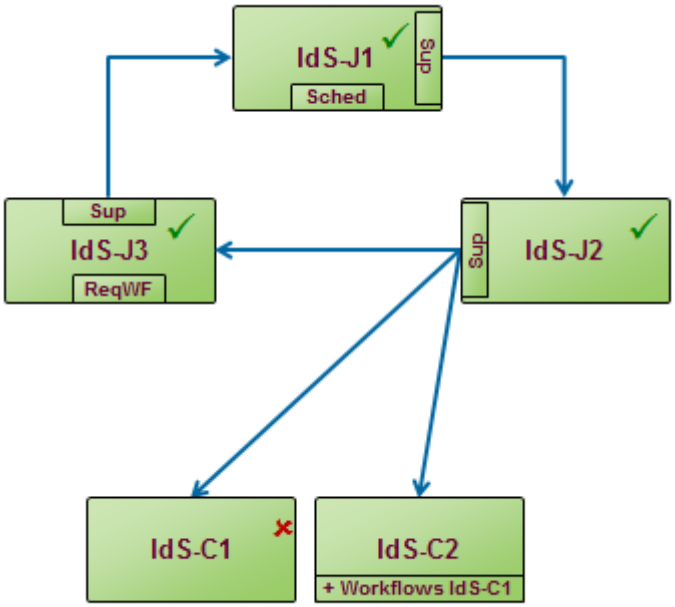

Figure 4. Automatic Fail-over with Circular Monitoring – C++-based Server Down

1.3.1. Documentation

To understand this issue, we recommend reading the following chapters:

- *DirX Identity Connectivity Administration Guide*: the chapters on Java-based server configuration, messaging service configuration and on Java Supervisor configuration in the context-sensitive help.

2. Installation and Initial Configuration

This chapter describes the installation and initial configuration for high availability.

2.1. Installation

For a detailed description of the installation procedure, see the *DirX Identity Installation Guide*. This chapter lists just the items you should address when installing the features for high availability.

Choose Licensed Feature Set dialog:

Make sure that you select **High Availability** in addition to **Business Suite**. You can also select **Professional Suite** and / or **Password Management**, but neither one of these packages is needed for High Availability.

Choose Install Set dialog:

If you selected **High Availability** in the previous step, you can now select **High Availability** here, which allows you to select from this item:

- **Server Admin** – select this item to install the Server Admin Web application (and Java-based supervisor) that can be deployed with each IdS-J server. You should always select this item because it gives you an overview of the state of all Java- and C++-based servers.

2.2. Initial Configuration

After installation, you need to configure your deployment using the Initial Configuration wizard. This section just mentions the parts relevant for High Availability. For complete details on this task, see the *DirX Identity Installation Guide,* chapter *Configuration*.

System-wide Configuration dialog:

Enable High Availability for the system by checking the appropriate check box.

Configuration Options dialog:

Server Admin (including Supervisor-J) Configuration – select this item to deploy the Server Admin Web application to the IdS-J server you select in a subsequent step. Make sure that the Server Admin is deployed for each deployed IdS-J server.

Java-based Server:

For High Availability, set the **message repository** for the embedded message broker to a network device that can be accessed by the other Java-based servers – at least the one that is monitoring it.

In this case, the server typically needs to run as a domain account that has enough access rights to create files. The local system account is typically not sufficient. For Windows

platforms you have to set that account in the Windows Services application.

As an alternative, you can set the repository with DirX Identity Manager: select the server in the Configuration folder and enter the correct path. Then re-start the services of both this server and the monitoring one to activate this.

To deploy additional Java-based servers, run the Configuration wizard again.

In the dialog **Java based Server** create a new server by selecting <Create a new Java server> in the field **Server to update or create**.

Select the host on which you run the wizard and in the **Java-based Server Ports and SSL** dialog especially set appropriate values for the ports and port ranges: Make sure they do not overlap with the settings of the other servers on the same host.

In case you need to connect using SSL, provide the necessary key and trust stores. For simplicity, you can re-use the certificate and key of the other servers: In the configuration wizard set the location to the sub-folder *private* of this server and copy them afterwards.

2.3. Documentation

To understand this issue, we recommend reading the following chapters:

- *DirX Identity Installation Guide*, especially chapters 3 (installation) and 4 (configuration).

3. Configuration

As a pre-requisite for high availability, you need to deploy at least two Java-based servers.

3.1. Java-based Servers

To set up a new Java-based server, use the Configuration Wizard as described in the chapter "Installation and Initial Configuration".

3.1.1. Assign Scheduler, Request WorkflowTimeout Check and Adaptors

As soon as you have deployed more than one Java server, you have a choice as to where run scheduler, request workflows and selected adaptors. Scheduler and request workflows can only be deployed on one server per domain. Most of the adaptors can run on all Java servers in parallel. If you want, you can disable one or more on selected servers. But if you run the associated type of workflows, make sure that at least one adaptor of this type is active. Using Server Admin you can move these components at runtime without re-starting the servers. See the chapter "Supervisor Customization" for instructions.

To support running all workflows on all IdS-J servers, make sure you configure all IdS-J servers so that they provide the same resource families. Keep in mind that workflow activities – in both provisioning and request workflows – require a resource family that the hosting IdS-J server must provide.

DirX Identity Manager can give you an overview of the adaptors, the scheduler and the timeout check deployment: open the Connectivity view and go to the Expert View. In the Connectivity Configuration Data tree, go to Configuration → DirX Identity Servers → Java Servers. Right-click on a Java-based server and then select **Manage IdS-J Configuration** from the context menu. This opens a dialog that gives you an overview of the adaptors, the scheduler, the request workflow timeout check responsibility and the supervisors in respective tabs.

The **Adaptors** tab presents a two-dimensional list: each Java-based server is listed horizontally, and each adaptor is listed vertically. Click the appropriate radio or check button(s) to deactivate adaptors that you don’t need. This action instructs the corresponding Java-based server not to start the adaptor, which saves threads and sockets resources in both the Java-based server and the messaging service.

The **Request Workflow Timeout Check** tab also presents the list of Java servers and radio buttons for the request workflow timeout check. This component should run on only one server.

The **Scheduler** tab lets you assign the scheduler to one of the Java servers of the domain.

For changing most of the settings however, you have to open the Java server configuration objects individually. To activate your changes, you must stop and then start all the Java-based servers in the proper sequence.

3.1.2. Configure Monitoring Circle

For **Automatic Fail-over** you have to configure a monitoring circle: each Java server should monitor another one and should in turn also be monitored by another one. See the section "Automatic Fail-over with Circular Monitoring" for an overview.

You must use the Identity Manager for the configuration task. In the **HA** tab of each Java-based server activate Automatic Monitoring, select the server to be monitored and reference the link to the supervisor configuration.

For checking your circle and obtain an overview, right-click a Java-based server object and then select **Manage IdS-J Configuration** from the context menu.

The **Supervision** tab shows whether automatic monitoring is set for all servers and how Java servers monitor each other.

3.1.3. Backup Adaptors

For enabling recovery of adaptor messages, you don't have to configure anything special. The backup adaptors start automatically when the Automatic Monitoring Flag is set.

3.1.4. Secure Connections – SSL/TLS

If you use SSL to secure the connections to the Java-based servers, you need to generate a private key for the Java-based server and then put the corresponding certificate into the trust store of each client. To keep things simple, we recommend using the same private key for all Java-based servers. For more details, see the section "Establishing Secure Connections with SSL" in the chapter "Managing the Connectivity System" in the *DirX Identity Connectivity Administration Guide*.

If you operate Web Center in single sign-on mode, you must generate a private key for Web Center and then put the corresponding certificate into each Java-based server's trust store. For details, see the section "Deployment Descriptor web.xml" in the chapter "Web Center Configuration" of the *DirX Identity Web Center Reference*. We recommend using the same private key for all deployed Web Center instances.

3.2. C++-based Servers

With regard to High Availability nothing special has to be configured for the C++-based servers except for the Status path.

It is recommended to have the Status path on a shared network (for Windows: Should be an UNC path and the service must run under an account which has write access to this UNC path). In that case, status files can be accessed even if the machine where the IdS-C is located is down.

Just keep in mind that the Status Tracker is automatically started in the IdS-C with attribute dxmRunStatusTracker set to true. The Configuration Wizard on initial installation sets this value for the first installed C++-based server.

3.3. Supervisor Configuration

The configuration entries for the Java-based supervisor are located in the Connectivity database (see DirX Identity Manager's Expert View) in Configuration → Java Supervisors. Each subfolder here defines a supervisor configuration.

In the "circular monitoring" automatic fail-over scenario, the configuration for the supervisor embedded in an IdS-J server is identified by a reference from the IdS-J configuration to the subfolder in Java Supervisors.

To configure the supervisor for the circular monitoring scenario, follow these guidelines:

- In the Supervisor section of the Java Server configuration entry (see the Java-based Server tab) of each Java-based server that should be part of the monitoring process:
- Check **Automatic Monitoring** to enable it.
- Select the Java-based server to be monitored.
- Select the supervisor configuration entry. We recommend using the same supervisor configuration for all supervisors; that is, reference the same supervisor entry from all Java server configuration entries.
- Make sure that **Automatic Monitoring** is enabled for all the Java-based servers in the monitoring cycle and that the server-to-server monitoring circle is closed. Verify this by viewing the **Supervision** tab in the **Manage IdS-J Configuration** dialog.
- In exactly one Java-based server configuration entry, check **Monitor C++-based Servers** (see the Java-based Server tab, Supervisor section).

3.4. Documentation

To understand this issue, we recommend reading the following chapters:

- *DirX Identity Web Center Reference*, chapter "Web Center Configuration".
- *DirX Identity Connectivity Administration Guide*, chapter "Managing the Connectivity System".

4. Supervisor Customization

The Java-based supervisor is deployed with the Web application Server Admin in the embedded Tomcat Web container of each IdS-J.

The main component in the supervisor is a Groovy script. The script contains the control logic for monitoring the server(s), moving functionality and sending email notifications. It uses Java classes in the module Admin Client. The most important class is AdminClientController. It provides methods to

- Read configuration data from and change in the Connectivity database.
- Obtain the state of Java-based and C++-based servers.
- Move adaptors between Java-based servers.
- Move request workflow processing and scheduler between Java-based servers.
- Move workflows between C++-based servers.

The Sendmail class allows for sending an e-mail.

Adapting the Groovy script requires Java knowledge. Only a few simple Groovy-specific syntaxes are used within the script. Editing and running the script is possible even with a normal text editor. You don't need a compiler. For more convenience, we recommend using Eclipse with the Groovy plug-in. For more details on Groovy, check the Groovy web page: http://groovy.codehaus.org/.

The following sections help you to understand the logic of the supervisor script to help you adapt it to your needs. The first sections describe the script, a special section explains how to adapt e-mail texts and the last sections describe the most important Java utility classes.

Find the script and properties files in the following sub-folders of each respective Java server:

- Start script and properties for mails in subfolder **tomcat/webapps/serverAdmin/WEB-INF/scripts/sv**.
- Other common scripts in subfolder **tomcat/webapps/serverAdmin/WEB-INF/classes/sv**.

4.1. Supervisor in Server Admin

The control logic of the supervisor deployed in Server Admin is contained in the **InsideGroovySupervisor.groovy** script. This script is called from the Supervisor servlet on start-up only when the configuration in the Connectivity database requests automatic monitoring.

The **startMonitoring** method contains the script's control logic. After reading its configuration it starts the monitoring loop. The method **isJavaServerUp** returns false if the Java server with the given name is not considered active. If this supervisor is also the one to monitor the C++ servers, it performs this in the method **monitorCServer**.

If a Java server is considered to be down, the supervisor, in its **handleInactiveServer** method, takes over all the features of the failed servers. These are especially the following items:

- The scheduler, if the failed server hosted it.
- The request workflow processing, if the failed server was responsible for request workflows.
- The server's movable adaptors. If one of the other JMS adaptors was active in the failed server, but not in the local one, then the supervisor activates it also on its embedding server.
- The server's monitoring tasks; that is, it also checks the state of those servers that should be monitored by the failed one.

The **monitorCServer** method monitors the registered C++-based servers. It calls the **isCServerUp** method to check the state of a given C++-based server. If the C++-based server is registered as active but fails to respond, the supervisor:

- Sends an e-mail informing about the fail state.
- Moves the Tcl-based workflows controlled by the failed server to another running C++-based server.
- Sends an email informing about the move.
- Requests the affected C++-based servers to reload.

The **sendmail** method sends an e-mail using the Sendmail class of the ClientAdminController component. The method obtains the mail configuration from the supervisor configuration, especially: from, to, subject, body and the name of the mail server. If a subject is passed to the method, it adds it to the configured subject. If a body is passed to the method, it adds it to the configured body.

4.2. Changing Mails in Supervisor Scripts

The supervisor scripts take the basic mail parameters from the supervisor configuration in the Connectivity database: see the section Supervisor Configuration above. These parameters specify mail parameters such as the recipients (to, cc, bcc), the subject and the body of the mail as well as the mail service.

The scripts send e-mails in various situations, especially when they move functionality from a failed server to an active one. All of these mails use the same mail configuration. To identify the actual situation, the scripts add specific information to the subject and the body:

- Subject: the configured subject is extended with ": " followed by a string.
- Body: the configured body is extended with a new line followed by a string.

The strings are taken from the file **mail.properties** in the sub-folder **tomcat/webapps/serverAdmin/WEB-INF/scripts/sv** of each affected Java server. Note that is has to be deployed for each server. They can be customized by adapting this file. The

supervisor scripts find the appropriate texts via a key that identifies the situation. The currently evaluated keys are:

start: subject of the mail immediately before starting the monitor loop.

startbody: body of the mail immediately before starting the monitor loop.

j_down: subject of the mail after a Java-based server becomes inactive.

j_down_body: body of the mail after a Java-based server becomes inactive.

c_down: subject of the mail after a C++-based server becomes inactive.

c_down_body: body of the mail after a C++-based server becomes inactive.

INF_MOVE_ATS: body of the mail after workflows have been moved between C++-based servers.

c_ ats_move: subject of the mail after the messaging service has been moved.

c_ ats_move_body: body of the mail after the messaging service has been moved.

The texts contain string placeholders of the form **%*n***$s* where *n* is an integer that indicates the *n*th passed parameter; for example, **%1$s**. These parameters are the dynamic values depending on the context, typically identifying a server name.

If you want to suppress mails, find the location by scanning for the string "sendmail" in the appropriate script. Just uncomment the lines by prefixing them with "//" or remove them.

If you want to send additional mails: copy the lines for generating the subject and body strings and for sending the mail and paste them into the appropriate location. Replace the keys with your own names and extend the **mail.properties** file with the appropriate lines.

You can also replace the pre-configured mail parameters and set your own ones by using your own sendmail method. Just copy and paste the default sendmail method to define a new method (for example, myCustomSendmail) and adapt it to your needs.

4.3. Java Classes

The supervisor script uses a set of Java classes. The following sections give an overview on them.

4.3.1. AdminClientController

The **net.atos.dirx.dxi.admin.client.controller.AdminClientController** class provides a number of useful methods to read and update the configuration entries of the Java- and C++-based servers, to obtain the state of these servers and to move functionality between servers. The most important methods are:

getSupervisionConfigurationByName

Reads the supervisor configuration with the given name from the Connectivity database

and returns it as a map with the option names as the index.

getServers

Reads the configuration of all Java-based servers from the Connectivity database and returns them in a map that is indexed by the display name of the server. The configuration for a server is provided as a Java bean with the configuration options available as a map.

getCServers

Reads the configuration of all C++-based servers from the Connectivity database and returns them in a map that is indexed by the display name of the server. The configuration for a server is provided as a Java bean with the configuration options available as a map.

getStateOfJavaServer

Sends a JMX operation to obtain the state of the indicated Java-based server. The state is given as an integer in the range from 0 (bad) to 10 (good).

getStateOfCServer

Sends a JMX/SOAP operation to obtain the state of the indicated C++-based server. The state is given as an integer in the range from 0 (bad) to 10 (good).

moveAdaptors

Moves the JMS adaptors and also the responsibility for request workflows from one server to another one.

moveReqWFType

Moves the responsibility for processing the request workflows from one server to another. It first changes the settings in the connectivity database and then requests the affected Java-based servers to reload their configuration and thus start the processing.

moveTclWorkflowAndActivities

Moves the workflows and activities that are controlled by a given C++-based server to another one given by name. It first changes the settings in the connectivity database and then requests the affected Java-based servers to reload their configuration and thus start the processing.

4.3.2. Sendmail

The **net.atos.dirx.dxi.admin.client.controller.Sendmail** class can send a mail. It sets the mail content type to either "text/html", if the body matches the HTML pattern or to "UTF-8" otherwise.

5. Switching between Active and Passive Configurations

DirX Identity high availability supports an “active/passive” scenario (sometimes called a “warm standby” configuration), in which a secondary node or system—the “passive” configuration—acts as the backup for an identical primary system—the “active” configuration. The secondary system is completely installed and configured, but the software components are not running. If a failure occurs on the primary node, the software components are started on the secondary node. The switch is handled manually or is automated by using a failover component. Data is regularly replicated to the secondary system or stored on a shared disk.

DirX Identity high availability provides a tool for switching between active and passive DirX Identity configurations. If you have selected **High Availability** in the **Choose Licensed Feature Set** dialog during installation, you will find this tool installed in the directory:

*dxi_install_path***/ha/tools/switchConfiguration/**

This directory contains the following files:

- **activatePassiveConfiguration.bat** – the tool for activating the passive configuration.
- **activatePassiveConfigurationOnSample01.bat** - an example of how to call the tool.
- Sample tool configuration files

The **activatePassiveConfiguration.bat** tool performs the following tasks:

- Sets the Scheduler flag in LDAP for the target Java-based Server (active/not active)
- Sets the RequestWorkflow TimoutCheck flag in LDAP for the target Java-based server (active/not active)
- Switches Tcl scripts from one C++-based Server to the other
- Sets the Java-based / C++-based Server as inactive (active/not active)
- Moves the configuration handler from one Java-based Server to the other (active/not active)
- Adjusts associated servers at connected directories
- Sets the Status Tracker flag on the C++-based Server (active/not active)
- Sets the Notes connector on the C++-based Server (active/not active)
- Start the services C++-based Server, Java-based Server, Message Broker and Tomcat server (optional)

5.1. Prerequisites

The tool requires a setup with one active and one passive DirX Identity configuration. The tool must be installed on the host where the passive part of DirX Identity is running, which implies that it must be installed on both systems, since the active configuration will change

to a passive configuration when the tool is used.

5.2. Command-line Interface

To use the tool, specify the command:

activatePassiveConfiguration.bat *parameters*

Where *parameters* are all of the following:

-host *hostname* – the LDAP server that holds the Connectivity/Provisioning store

-port *port* – the port number on which the LDAP server is listening

-user *ldap_user_dn* – the user DN (domain admin)

-pass *password* – the password for the user DN

-ssl true | **false** – whether (**true**) or not (**false**) an SSL connection is in use

-domain *domain* - the domain name; for example, **My-Company**

-c *configuration_file* – the path and file name of the switching configuration file to use

Always call the tool on the host where you want to activate the configuration.

Here is an example command line:

```
activatePassiveConfiguration.bat -host jupiter.my-company.com
-port 389 -user cn=domainAdmin,cn=My-Company -pass ****
-domain My-Company -ssl false -c switchHAConfigFromTo.xml
```

5.3. Switching Configuration File

The **activatePassiveConfiguration.bat** tool uses a switching configuration file to control its operation. Switching configuration file samples are provided in the tool's installation directory; these files can be changed and/or copied, renamed and relocated according to on-site requirements.

The switching configuration file contains **configuration**, **logging**, and **process** elements. The next sections describe these elements and their attributes.

5.3.1. configuration Element

The **configuration** element has the following attributes

- **startServices** – if set to **true**, the tool starts the services C++-based Server, Java-based Server, Message Broker and Tomcat server on this host.
- **moveNotesConnector** – if set to **true**, the tool activates the Notes Connector(s) on the new active configuration. If false, Notes connectors are ignored.

5.3.2. logging Element

The attributes of the **logging** element are similar to the corresponding parameters of Identity Manager in the DirX Identity configuration file **dxi.cfg**. They are:

- **fileName** – the name of the trace file. The absolute or relative path is allowed.
- **level** – possible values are:

0 – no trace, no error

1 – error

2-**4** – warnings

5-**8** – flow trace

9 – debug

Higher levels include the content of lower levels. For example, if you specify **5**, errors and warnings are also written.

- **timestampformat** – a format string to enable time stamp information to be included before each log entry in the trace file. For example:

timestampformat="EEE MMM d HH:mm:ss.SSS yyyy:"

If the **timestampformat** attribute is not specified, timestamps are not written into the trace file.

5.3.3. process Element and its mode Attribute

Note: in the descriptions in this section, the server to be activated is called the “to server”. The other server is called the “from server”.

The **mode** attribute of the **process** element in the switching configuration file specifies the method the tool is to use to determine the active configuration and which configuration should be activated. The following values are available:

- **auto** – the “to server” name is determined by the local hostname.
- **fromto** – the “from server” name and the “to server” name are specified explicitly.
- **state** – the configuration to be switched to is evaluated by the state/registered attribute of the Java-based and C++-based Servers.

These values are described in more detail in the next sections. The sample configuration files delivered with the tool include examples of all three modes.

5.3.3.1. auto

When the **process mode** attribute value is **auto**, the tool assumes the host where it’s running must become the active configuration. The hostname is determined via the Java method **InetAddress.getLocalHost().getHostName()** and the fully qualified hostname read

from the registry.

The tool looks for Java-based and C++-based servers that run on these hosts (one of the hostnames must match). Java-based/C++-based Server found is treated as the “to server”. The other Java-based / C++-based Server is treated as the “from server”. If exactly two Java-based /C++-based Servers are not found, an error is generated.

The file **switchHAConfig.xml** is a sample configuration file for **auto** mode.

5.3.3.2. fromto

When the **mode** attribute value is **fromto**, the hostnames for “from server” and “to server” servers are explicitly specified as follows:

```
<process mode="fromto" >
     <from>fromHostname</from>
     <to>toHostname</to>
</process>
```

The specified hostnames must match the names stored in LDAP for the Java-based and C++-based Servers. The file **switchHAConfigFromTo.xml** is a sample configuration file for the **fromto** mode.

5.3.3.3. state

When the **mode** attribute value is **state**, the tool treats the Java-based Server with the state STOPPED as the “to server” Java-based Server and treats the C++-based Server with an unchecked registered flag as the “to server” C++-based Server.

5.3.3.4. How the Tool Determines Hostnames from LDAP

The tool determines the hostnames from LDAP as follows:

- Java-based Server – the last part of the name is used as the hostname (LDAP attribute **dxmDisplayname**)
- C++-based Server – the name is used as the hostname (LDAP attribute **dxmDisplayname**)

5.4. Sample Activation

This section shows a sample trace of an activation. The activation occurs on **dxi-sample01** (the “to server”). The other host (the “from server”) is **dxi-sample03**. The domain suffix is **iam.sampledomain.net**. Before calling the tool, the active configuration is on **sample03**. The following figure illustrates this configuration:

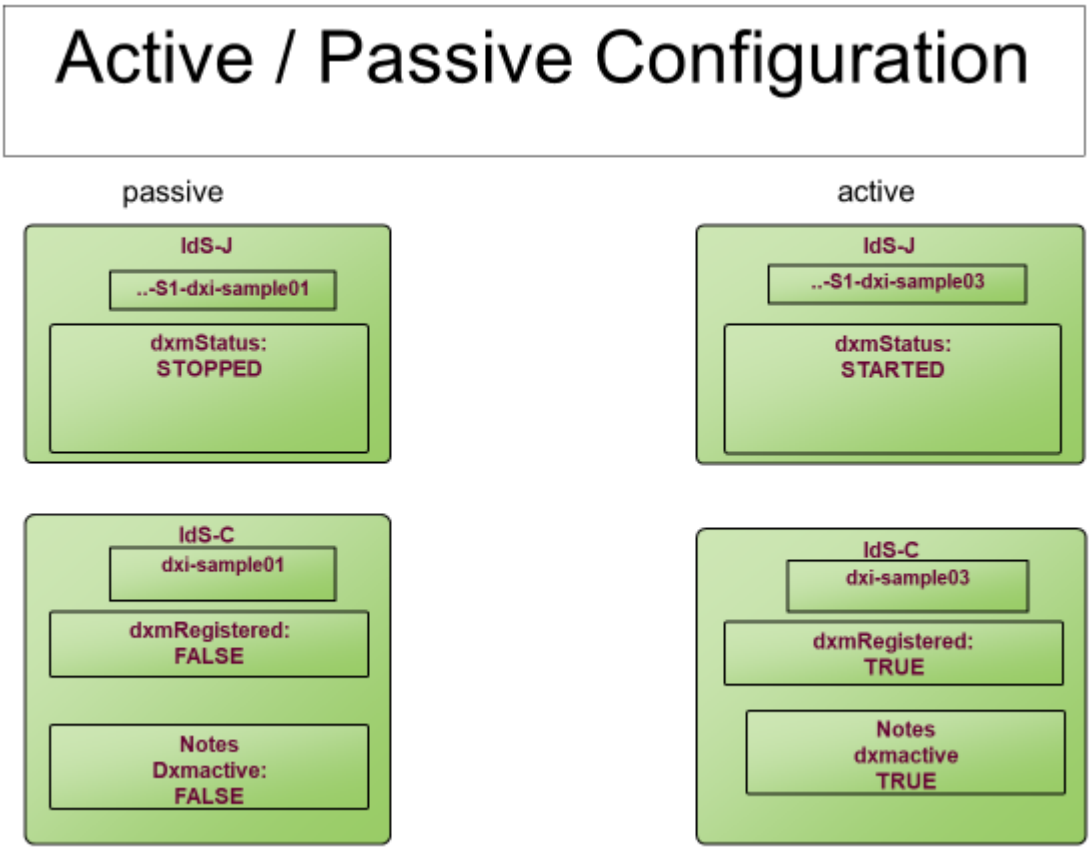

Figure 5. Active / Passive Configuration

Here is the sample trace:

```
LOG(STG200): SwitchConfiguration called at 18.12.19, 13:48:42 MEZ
with the following Parameters:
LOG(STG200):     tracefile: ./testMove.txt
LOG(STG200):     tracelevel: 5
LOG(STG200):     mode: automatic by local hostname
LOG(STG200):     move Notes Connector: true
LOG(STG200):     start Services: true
LOG(STG200):
LOG(STG200):     host: localhost
LOG(STG200):     port: 636
LOG(STG200):     user: cn=admin,dxmc=dirxmetahub
LOG(STG200):     ssl: true
LOG(STG200):     domain: My-Company
LOG(STG200): ------------------------------------------------------
LOG(STG200):
LOG(STG200):  hostname from getHostname: dxi-sample01
LOG(STG200):  fqn hostname from registry: dxi-
```

```
sample01.iam.sampledomain.net
LOG(STG200): Moving from My-Company-S2-dxi-
sample03.iam.sampledomain.net to My-Company-S1-dxi-
sample01.iam.sampledomain.net
LOG(STG200): Moving CServer from dxi-sample03.iam.sampledomain.net to
dxi-sample01.iam.sampledomain.net
LOG(STG200):
LOG(STG200): ------------------------------------------------------
LOG(STG200):
INF(ADC215): Moving dxmRunsScheduler flag from Server 'My-Company-S2-
dxi-sample.iam.sampledomain.net' to Server 'My-Company-S1-dxi-
sample01.iam.sampledomain.net'.
INF(ADC216): Moving 'Monitor C++-based Servers' flag from Server 'My-
Company-S2-dxi-sample03.iam.sampledomain.net' to Server 'My-Company-
S1-dxi-sample01.iam.sampledomain.net'.
INF(ADC200): Moving Primary from Server 'dxi-
sample03.iam.sampledoamin.net' to Server 'dxi-
sample01.iam.sampledomain.net'
LOG(STG200): Moving ConfigurationHandler.
LOG(STG200): Adjust associated servers at connected directories .
INF(ADC214): Moving Tcl workflows from Server 'dxi-
sample03.iam.sampledomain.net' to Server 'dxi-
sample01.iam.sampledomain.net'.
INF(ADC200): Moving StatusTracker from Server 'dxi-
sample03.iam.sampledomain.net' to Server 'dxi-
sample01.iam.sampledoamin.net'
INF(ADC200): Moving Notes connector active flag from Server 'dxi-
sample03.iam.sampledoamin.net' to Server 'dxi-
sample01.iam.sampledomain.net'
LOG(STG200): ------------------------------------------------------
LOG(STG200):  starting the services
LOG(STG200):
LOG(STG200): starting MessageBroker service : DirX Identity Message
Broker 1 returned: 0
LOG(STG200): starting ids-j service : DirX Identity IdS-J-My-Company-
S1 returned: 0
LOG(STG200): starting ids-c service returned: 0
LOG(STG200): starting TOMCAT service : Tomcat9 returned: 0
LOG(STG200): Ended with rc: 0
```

DirX Product Suite

The DirX product suite provides the basis for fully integrated identity and access management; it includes the following products, which can be ordered separately.

DirX Identity

DirX Identity provides a comprehensive, process-driven, customizable, cloud-enabled, scalable, and highly available identity management solution for businesses and organizations. It provides overarching, risk-based identity and access governance functionality seamlessly integrated with automated provisioning. Functionality includes lifecycle management for users and roles, cross-platform and rule-based real-time provisioning, web-based self-service functions for users, delegated administration, request workflows, access certification, password management, metadirectory as well as auditing and reporting functionality.

DirX Directory

DirX Directory provides a standards-compliant, high-performance, highly available, highly reliable, highly scalable, and secure LDAP and X.500 Directory Server and LDAP Proxy with very high linear scalability. DirX Directory can serve as an identity store for employees, customers, partners, subscribers, and other IoT entities. It can also serve as a provisioning, access management and metadirectory repository, to provide a single point of access to the information within disparate and heterogeneous directories available in an enterprise network or cloud environment for user management and provisioning.

DirX Access

DirX Access is a comprehensive, cloud-ready, scalable, and highly available access management solution providing policy- and risk-based authentication, authorization based on XACML and federation for Web applications and services. DirX Access delivers single sign-on, versatile authentication including FIDO, identity federation based on SAML, OAuth and OpenID Connect, just-in-time provisioning, entitlement management and policy enforcement for applications and services in the cloud or on-premises.

DirX Audit

DirX Audit provides auditors, security compliance officers and audit administrators with analytical insight and transparency for identity and access. Based on historical identity data and recorded events from the identity and access management processes, DirX Audit allows answering the “what, when, where, who and why” questions of user access and entitlements. DirX Audit features historical views and reports on identity data, a graphical dashboard with drill-down into individual events, an analysis view for filtering, evaluating, correlating, and reviewing of identity-related events and job management for report generation.

For more information: support.dirx.solutions/about

Legal remarks

On the account of certain regional limitations of sales rights and service availability, we cannot guarantee that all products included in this document are available through the Eviden sales organization worldwide. Availability and packaging may vary by country and is subject to change without prior notice. Some/All of the features and products described herein may not be available locally. The information in this document contains general technical descriptions of specifications and options as well as standard and optional features which do not always have to be present in individual cases. Eviden reserves the right to modify the design, packaging, specifications and options described herein without prior notice. Please contact your local Eviden sales representative for the most current information. Note: Any technical data contained in this document may vary within defined tolerances. Original images always lose a certain amount of detail when reproduced.