EVIDEN

Identity and Access Management

DirX Audit

Customization Guide

Version 9.0, Edition July 2025

Table of Contents

Preface

This manual provides information how to customize DirX Audit. It consists of the following chapters:

- Chapter 1 describes how to generally customize the DirX Audit installation and configuration files.
- Chapter 2 describes how to customize the DirX Audit Manager Classic user interface.
- Chapter 3 describes how to set up and customize reports.
- Chapter 4 describes the view type APIs used with report template definitions.
- Chapter 5 describes how to collect audit messages from third-party applications.
- Chapter 6 describes how to configure digest (event) and tag (dimension) producers.
- Chapter 7 describes how to configure and add fact tables, facts and dimensions.
- Chapter 8 describes how to customize history synchronization.
- Chapter 9 describes how to customize color schemes for Dashboard components in DirX Audit Manager Classic.

DirX Audit Documentation Set

DirX Audit provides a powerful set of documentation that helps you configure your audit server and its applications.

The DirX Audit document set consists of the following manuals:

- *DirX Audit Introduction*. Use this book to obtain a description of DirX Audit architecture and components.
- *DirX Audit Tutorial*. Use this book to get familiar quickly with your DirX Audit installation.
- *DirX Audit Administration Guide*. Use this book to understand the basic tasks of DirX Audit connectivity administration.
- *DirX Audit Manager Classic Guide*. Use this book to obtain a description of the DirX Audit Manager Classic user interface provided with DirX Audit.
- *DirX Audit Manager Guide*. Use this book to obtain a description of the DirX Audit Manager user interface provided with DirX Audit.
- *DirX Audit Command Line Interface Guide*. Use this book to obtain a description of the command line interface provided with DirX Audit.
- *DirX Audit Customization Guide*. Use this book to customize your DirX Audit environment.
- *DirX Audit History Synchronization Guide*. Use this book to obtain information about the DirX Audit History synchronization jobs.
- *DirX Audit Best Practices*. Use this book to obtain guidelines and tips for avoiding common mistakes and improving the experience of a DirX Audit installation.
- *DirX Audit Installation Guide*. Use this book to install DirX Audit.
- *DirX Audit Migration Guide*. Use this book to migrate DirX Audit from previous versions.
- *DirX Audit Release Notes*. Use this book to understand the features and limitations of the current release.
- *DirX Audit History of Changes*. Use this book to understand the features of previous releases.

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.

install_path
The exact name of the root of the directory where DirX Audit programs and files are installed. The default installation directory is *userID_home_directory*/**DirX Audit** on UNIX systems and **C:\Program Files\DirX\Audit** 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*.

install_media
The exact path where the DirX Audit installation media is located.

1. General Customization

This chapter describes how to generally customize the DirX Audit installation and configuration files. The goal is to maintain separate default and customized files.

The next sections describe:

- General concepts
- Customization levels and file resolving priority
- File customization examples

1.1. General Concepts

Custom files allow to override default files partially or fully at different customization levels. These levels are described in the next chapter.

Default file:

- Delivered with the installation
- Overwritten or deleted during installation/uninstallation/reinstallation
- Must be present

Custom file:

- Defined during customization process
- Kept during installation/uninstallation/reinstallation
- Optional

When creating customizations, the same folder structure must be preserved.

1.2. Customization Levels

The customization can be done at the core level and the tenant level. The core level customization applies to properties set using the Configuration Wizard for the Core Configuration. The tenant level customization applies to all customizable properties. The order is starting from the least specific up to the most specific folder structure.

1.2.1. Tenant Levels

There are four customization levels for the tenant level:

- **Core default** – default files for core components with the path starting with:

  *install_path***/conf/defaults/**

- **Tenant default** – default files for tenant components with the path starting with:

  *install_path***/conf/defaults/tenant/**

- **Core custom** – custom files for core components with the path starting with:

  *install_path***/conf/**

- **Tenant custom** – custom files for tenant components with the path starting with: *install_path***/conf/tenants/***tenantID***/**

1.2.2. Core Levels

They consist of **Core default** and **Core custom** levels which contain customizable properties declared only for core components. Such properties should not be customized on the tenant level and should be kept in the core level files.

1.2.3. File Resolving Priority

The priority of files to be used depends on the situation and is detailed in the customization process of specific feature. Generally, there are four methods:

- Finding the first file – it looks for the least specific file (e.g., when searching for report definition file)
- Finding the last file – it looks for the most specific file (e.g., when searching for SQL script)
- Merging – it iterates through all available default and custom files in the specific order and merges their content (e.g., JSON and XML files)
- Custom processing

1.3. Examples

Customization paths according to customization levels:

1.3.1. Customizing Configuration Files

When customizing fact configuration files, the merging resolving priority is applied.

Core default path:

- *install_path***/conf/defaults/fact-configuration/***confFactTables.xml*

Core custom path:

- *install_path***/conf/fact-configuration/***confFactTables.xml*

Tenant custom path:

- *install_path***/conf/tenants/***tenantID***/fact-configuration/***confFactTables.xml*

1.3.2. Customizing SQL Scripts

When customizing SQL scripts, the find last file resolving priority is applied.

If you want to customize SQL scripts, it is possible to do that for all tenants or for a specific tenant. Copy the core default SQL script with the exact folder structure from the folder:

*Install_path***/conf/defaults/sql/common/***folder_structure*

into the core custom configuration folder for all tenants in:

- *install_path***/conf/sql/common/***folder_structure*

and/or into the tenant custom configuration folder in:

- *install_path***/conf/tenants/***tenantID***/sql/common/***folder_structure*
  where *tenantID* is a unique tenant identifier.

For example, core default SQL script for filling fact table FCT_ACCOUNTS is located at:

*install_path***/conf/defaults/sql/common/factpopulation/data/fct/FCT_ACCOUNTS/***fill_01_FCT_ACCOUNTS_merge_MSSQL.sql*

To customize this SQL script, create and modify the script in location:

- *install_path***/conf/sql/common/factpopulation/data/fct/FCT_ACCOUNTS/***fill_01_FCT_ACCOUNTS_merge_MSSQL.sql*

By default, the most specific SQL script that is found, will be used. To customize this SQL script on tenant custom level, create and modify the script in location:

- *install_path***/conf/tenants/***tenantID***/sql/common/factpopulation/data/fct/FCT_ACCOUNTS/***fill_01_FCT_ACCOUNTS_merge_MSSQL.sql*

1.3.3. Customizing Report Definitions for Dirx Audit Manager Classic

When customizing report definitions, the find first file resolving priority is applied.

Core default path:

- *install_path***/conf/defaults/report-definitions/***EvnLogins/EvnLogins.xml*

Core custom path:

- *install_path***/conf/report-definitions/***EvnLogins/EvnLogins.xml*

Tenant custom path:

- *install_path***/conf/tenants/***tenantID***/report-definitions/***EvnLogins/EvnLogins.xml*

1.3.4. Customizing Report Definitions for DirX Audit Manager

When customizing report definitions, the find first file resolving priority is applied.

Core default path:

- *install_path*/conf/defaults/report-definitions/*EvnLogins/EvnLogins.json*

Core custom path:

- *install_path*/conf/report-definitions/*EvnLogins/EvnLogins.json*

Tenant custom path:

- *install_path*/conf/tenants/*tenantID*/report-definitions/*EvnLogins/EvnLogins.json*

1.3.5. Customizing Report Components for DirX Audit Manager

When customizing report components, the merging resolving priority is applied.

Core default path:

- *install_path*/conf/defaults/report-components/dxtComponents.json

Core custom path:

- *install_path*/conf/report-components/dxtComponents.json

Tenant custom path:

- *install_path*/conf/tenants/*tenantID*/report-components/dxtComponents.json

1.3.6. Customizing Report Indicators and Indicator Queries for DirX Audit Manager

When customizing report indicators, the merging resolving priority is applied.

Core default path:

- *install_path*/conf/defaults/report-indicators/dxtIndicators.json

Core custom path:

- *install_path*/conf/report-indicators/dxtIndicators.json

Tenant custom path:

- *install_path*/conf/tenants/*tenantID*/report-indicators/dxtIndicators.json

When customizing report indicator queries, the merging resolving priority is applied.

Core default path:

- *install_path*/conf/defaults/report-indicators/dxtIndicatorQueries.json

Core custom path:

- *install_path*/conf/report-indicators/dxtIndicatorQueries.json

Tenant custom path:

- *install_path*/conf/tenants/*tenantID*/report-indicators/dxtIndicatorQueries.json

For customizing report indicator queries's SQL scripts, see the chapter "Customizing SQL Scripts".

1.3.7. Customizing Report Resources

When customizing report resources, the find last file resolving priority is applied.

Core default path:

- *install_path*/conf/defaults/reports/*HstUsers/HstUsers.jrxml*

Core custom path:

- *install_path*/conf/reports/*HstUsers/HstUsers.jrxml*

Tenant custom path:

- *install_path*/conf/tenants/*tenantID*/reports/*HstUsers/HstUsers.jrxml*

2. Customizing the DirX Audit Manager Classic User Interface

This chapter describes how to customize DirX Audit Manager Classic's user interface. It describes how to:

- Set up single sign on
- Customize the table page navigator
- Configure privileged groups
- Customize Audit analysis
- Customize the History view
- Customize the History timeline
- Customize reports
- Customize the user interface layout
- Localize Dashboard component titles
- Configure database connection caching
- Enable (and disable) HTTP compression
- Configure the default view displayed

2.1. Setting Up Single Sign On

You can use HTTP header injection to integrate DirX Audit Manager Classic with an external authentication system such as DirX Access. The external system can pass the username and tenant identification to the DirX Audit Manager Classic application in the request headers, by default DXT_USER (user header) and DXT_TENANT (tenant header). If the DirX Audit Manager Classic detects a valid username and tenant identification combination in the HTTP request by comparing it with an appropriate LDAP source, it bypasses the HTML form authentication.

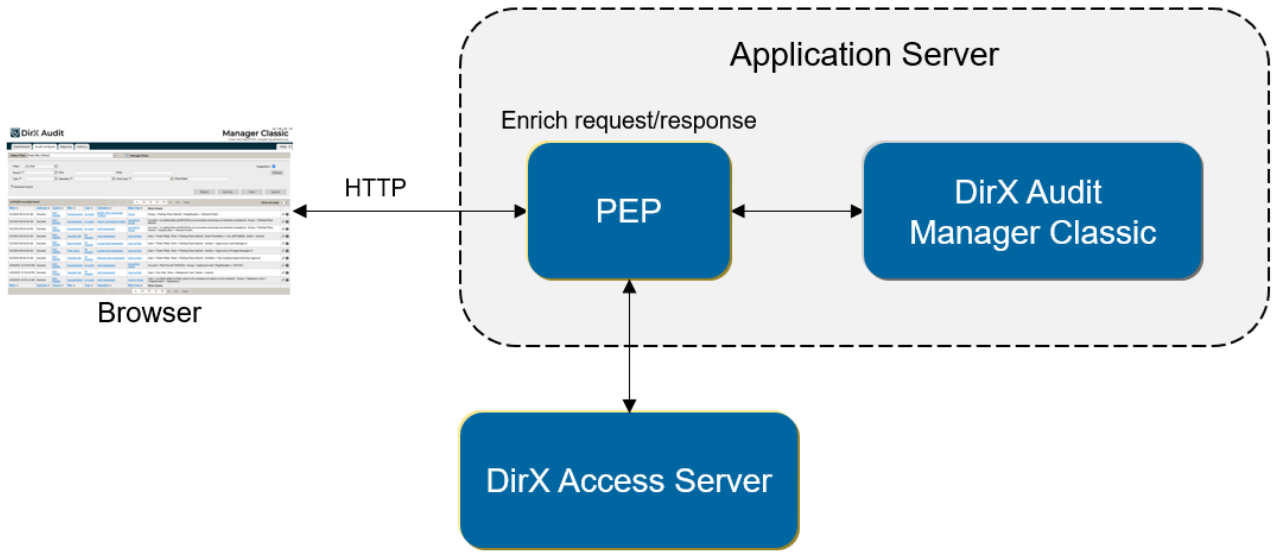

Figure 1. DirX Audit Manager Classic Integration with DirX Access as SSO Provider

This functionality is disabled by default because it introduces a significant security risk. Before enabling SSO, you must ensure that the DirX Audit Manager Classic container is accessible only from a portal that forwards the HTTP request containing the username and tenant identification injected into the request headers.

To enable SSO, run the Core Configuration Wizard and go to the "Audit Manager Classic Authentication" step. You can also modify the request header names for username and tenant identification there. Continue with the SSO settings in the Tenant Configuration Wizard in the "Authentication Configuration" step.

It is also necessary to open DirX Audit Manager Classic without specifying the tenant parameter in the URL because the user and tenant information will be provided in the header; for example: **https://localhost:8443/AuditManager/**.

If you decide to use SSO, you must ensure that only one value for each HTTP header will be sent, otherwise there may be issues with HTTP headers concatenation and SSO may not work correctly.

2.2. Customizing the Table Page Navigator

To customize the table page navigator settings, edit the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties**

You can change the values of the following keys in this file:

- **Table.maxPages** – the maximum number of next and previous pages to which a user can jump by clicking on the link. The default value is 5.
- **Table.fastStep** – the number of pages to switch to when fast scrolling is used. The default value is 5.

2.3. Configuring Privileged Groups

DirX Audit Manager Classic supports the following internal roles: **Auditor**, **Audit administrator** and **Restricted auditor**. The LDAP groups to be mapped to each internal role are set up during DirX Audit configuration in the "Authentication Configuration" step.

Users with the **Auditor** role can only modify their private area of the predefined Dashboard components and Audit analysis filters.

Users with the **Audit administrator** role can modify all public areas of the dashboard components and Audit analysis filters in addition to their private area.

Users with the **Restricted auditor** role have no access to Dashboards, Audit analysis or History View; they can access only a selected subset of reports via the Reports tab.

The internal roles are mapped to configured LDAP groups during the authentication procedure. For more information, see the section "Managing Application Roles" in the "Authorization" chapter of the *DirX Audit Administration Guide*.

2.4. Customizing Audit Analysis

To customize Audit analysis settings, edit the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties**

You can change the values of the following keys in this file:

- **Events.suggestionList.max** – the limit setting for the maximum number of displayed suggestions, which applies to suggestions fields except the **Source**, **Type**, and advanced search fields. The default value is 50.
- **Events.populateSuggestionLists.maxCount.TYPE** – if the total number of event types exceeds the configured value, the autocomplete component is used in the user interface instead of the selection box component. The default value is 30.
- **Events.populateSuggestionLists.maxCount.SOURCE** – if the total number of event sources exceeds the configured value, the autocomplete component is used in the user interface instead of the selection box component. The default value is 30.
- **Events.populateSuggestionLists.maxCount.OPERATION** – if the total number of event operations exceeds the configured value, the autocomplete component is used in the user interface instead of the selection box component. The default value is 100.
- **Events.populateSuggestionLists.maxCount.WHAT_TYPE** – if the total number of event what types exceeds the configured value, the autocomplete component is used in the user interface instead of the selection box component. The default value is 100.
- **Events.detail.attributeChanges.expanded** – whether (**true**) or not (**false**) to enable the expanded overview of attribute changes included in the Event details popup. The default value is true.

- **Events.detail.attributeChanges.expanded.maxRecords** – the maximum number of records that control whether or not the overview of attribute changes included in the Event details popup is displayed and expanded by default. When the number of attribute changes in the event detail is higher than the **maxRecords** variable, the overview is hidden by default. The default value is 20.
- **Events.detail.show.legacy.attributes** – whether (**true**) or not (**false**) to show legacy fields in the event detail view. Affected fields are “Sensitivity” and “Lifecycle” in “Identification” and “What” sections. The default value is false.
- **Configuration.cache.suggestions.initialDelay** – Suggestions for fields **Source**, **Type**, **Operation** and **What Type** are cached and synchronized with the database state at intervals. This key sets the initial suggestions load delay in seconds after the application starts. The default value is 10.
- **Configuration.cache.suggestions.delay** – Suggestions for fields **Source**, **Type**, **Operation** and **What Type** are cached and synchronized with the database state at intervals. This key sets the interval in seconds at which the suggestions are periodically refreshed. The default value is 1800.

2.5. Customizing the History View

To customize History view settings, edit the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties**

You can change the values of the following keys in this file:

- **History.entry.tab.PREVIEW** – the number of rows to display on the page for the Overview tab. The default value is 20.
- **History.entry.tab.ATTRIBUTES** – the number of rows to display on the page for the Attributes tab. The default value is 20.
- **History.entry.tab.ROLES** – the number of rows to display on the page for the Roles tab. The default value is 20.
- **History.entry.tab.PERMISSIONS** – the number of rows to display on the page for the Permissions tab. The default value is 20.
- **History.entry.tab.GROUPS** – the number of rows to display on the page for the Groups tab. The default value is 20.
- **History.entry.tab.ACCOUNTS** – the number of rows to display on the page for the Accounts tab. The default value is 20.
- **History.entry.tab.USERS** – the number of rows to display on the page for the Users tab. The default value is 20.
- **History.entry.tab.RISKS** – the number of rows to display on the page for the Risks tab. The default value is 20.
- **History.entry.tab.ASSIGNMENT_CAUSE** – the number of rows to display on the page for the Assignment cause tab. The default value is 10.

- **History.entry.tab.CC_ENTRIES** – the number of rows to display on the page for the Certification campaign entries tab. The default value is 20.
- **History.entry.attributes.maxAttrNameCount** – the maximum number of attribute values shown for multivalue attributes in the history Attributes view.
  The default value is 100.

To customize History view tenant-specific settings, edit the file:

*install_path***/conf/tenants/***tenantID***/configuration.cfg**

Edit the section **`[manager.history]`** or create it if it does not exist. You can add or change the values of the following configuration keys in this file. (If these keys do not exist, you can create them.):

- **search.attribute_name.provider_type** – the provider to be used to provide attribute names according to the selected entry type. Possible values are:
  - CONFIGURATION – attribute lists are loaded from the configuration file
  - DATABASE – attribute lists are loaded from the database (default setting)
- **search.attribute_name.list.default** – the value to be used as the list of attribute names when there is no defined value for a defined history entry type. The default value is cn.
- **search.attribute_name.list.***EntryType* – the attribute names for the specified history entry type. Replace the *EntryType* with the desired entry type name. The entry type name must be equal to the value you can see in the history view search form in the **Type** selection box and is case sensitive; for example, search.attribute_name.list.**User**.
- **search.attribute_name.preset.default** – the preset attribute name that is used when there is no definition for the history entry type. The default value is cn.
- **search.attribute_name.preset.***EntryType* – the preset attribute name for the specified history entry type. Replace *EntryType* with the desired entry type name. The entry type name must be equal to the value you can see in the history view search form in the **Type** selection box and is case sensitive; for example, search.attribute_name.preset.**User**.

For example, when you want to provide **cn**, **uid**, **dxruid**, **sn**, and **customAttribute** as a list of attribute names for the **User** entry type and preset the **customAttribute** attribute name as the value in the **Attribute** selection box, the snippet of configuration in *install_path***/conf/tenants/***tenantID***/configuration.cfg** file should look like this:

```
[manager.history]
search.attribute_name.provider_type = CONFIGURATION
search.attribute_name.list.User = cn, uid, dxruid, sn,
customAttribute
search.attribute_name.preset.User = customAttribute
```

2.6. Customizing the History Timeline

To customize History timeline settings, edit the file:

install_path/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties

You can change the values of the following keys in this file:

- **History.entry.timeline.from** – the beginning of the timeline scope. The default value is 90 days before now, **$before($now();90d)**.
- **History.entry.timeline.to** – the end of the timeline scope. The default value is now, **$now()**.

You can also use the time functions **$now()**, **$before()** and **$after()**.

To customize History time point settings when accessing the History page via links from Events detail, edit the file:

install_path/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties

You can change the values of the following keys in this file:

- **Events.history.entry.timepoints** – the time points definition in the history timeline. Use the hash tag character (#) for the time point separation. The default value contains a list of three dates: seven days before the event was triggered, the day when the event was triggered and now:
  [$before($date(@__WHEN);7d)#$date(@__WHEN)#$now()].
- **Events.history.entry.timeline.from** – the beginning of the timeline scope when jumping to the History view from Audit analysis. The default value is seven days before the event was triggered: **$before($date(@__WHEN);7d)**.
- **Events.history.entry.timeline.to** – the end of the timeline scope when jumping to the History view from Audit analysis. The default value is the day when the event was triggered: **$date(@__WHEN)**.

In the definition, you can use time functions **$now()**, **$before()**, **$after()**. The **@__WHEN** function allows you to adopt the when date from the audit event.

2.7. Customizing Reports

To customize report configuration settings, edit the file:

install_path/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties

You can change the values of the following keys in this file:

- **Reports.preview.rowlimit** – the number of results to which a report preview is limited. The default value is 20.

- **Reports.default.rowlimit** – the number of results set as the default report result limit for all new reports (this value can be manually edited for each report).
  The default value is 100.
- **Reports.tooltip.default.picklist.rowlimit** – the number of report parameters to be displayed in the report tooltip that is visible for each report in the report set. The default value is 10.
- **Reports.attribute.name.cache.delay** – the refresh period for the reports attribute name cache. Identifying attribute names in report definitions can be cached. A value of **-1** turns off the cache. A value of **0** loads cache data only once, when the DirX Audit Manager Classic application starts. A positive value defines the refresh period in minutes. The default value is **720** minutes (12 hours).

2.8. Customizing the User Interface Layout

You can change the interface layout by using images or cascading style sheets (CSS).

New or modified images and CSS files must be stored in predefined directories that are analogous to the paths in the default installation.

The following table shows the default installation and customization paths:

	Default Folder	**Customization Folder**
Images	*install_path***/web/audit-manager-classic.war/img**	*install_path***/conf/custom/***tenantID***/manager/theme/img**
CSS	*install_path***/web/audit-manager-classic.war/css**	*install_path***/conf/custom/***tenantID***/manager/theme/css**

For example, when you want to change the company logo that is displayed in the left upper corner, follow these steps:

- Find the image in the default installation path. In this case, it is the **company-logo-125.png** image file which represents the company logo and is stored in the default installation directory *install_path***/web/audit-manager-classic.war/img/dxt-design/**.
- Create or check the existence of the customization folder where the modified image must be stored.
  In this example, it is *install_path*/**conf/custom/***tenantID***/manager/theme/img/dxt-design/**.
- Replace the placeholder *tenantID* with the appropriate tenant identifier string.

The file name must be preserved.

2.9. Localizing Dashboard Component Titles

All existing Dashboard component titles are localized using keys starting with the **dashboard.component.def** prefix from the *install_path***/conf/i18n/messages.properties** localization bundle file. When you want to localize a new or custom component title, you need to add a new localization key and an appropriate value (localized title) to the localization bundle file and use the key as the component title. Note that when you add a new key to the localization bundle file, you must restart the Apache Tomcat service (DirX Audit Manager Classic container) for the modification to take effect.

2.10. Configuring Database Connection Caching

Audit database connections are cached when the Apache Tomcat service (DirX Audit Manager Classic container) starts in order to speed up user logins and the initialization of user data and the interface. You can configure an initial delay for loading the database connection configuration and the time interval after which the connection settings are periodically rechecked.

Edit the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties**

You can change the values of the following keys in this file:

- **Configuration.cache.initialDelay** – the time (in seconds) of the delay for loading the database connection settings. The default value is 10.
- **Configuration.cache.delay** – the time interval (in seconds) after which the database connection settings are rechecked. The default value is 1800.

2.11. Enabling HTTP Compression

DirX Audit Manager Classic supports built-in HTTP compression. When compression is enabled, each HTTP response is compressed before it is sent to the client and is automatically decompressed on the client side. HTTP compression can be useful when you want to improve transfer speed and bandwidth. To enable and disable HTTP compression, edit the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/configuration.properties**

and use the following key:

- **Application.use_response_compression** – whether (**true**) or not (**false**) HTTP compression is enabled. The default value is false.

2.12. Configuring the Default View Displayed

The Dashboard view is the default view displayed after a user logs in to the DirX Audit Manager Classic. When a user is not authorized to view a defined tab, the first permitted tab is selected for display. This feature can be configured using the Core Configuration Wizard in the "Audit Manager Classic Application" step. For details, see the section "Audit Manager Classic Application" in the "Using the Configuration Wizard for the Core Configuration" in the *DirX Audit Installation Guide*.

3. Customizing Reports

DirX Audit reporting is based on TIBCO JasperReports Library, which is an open source reporting engine. JasperReports defines a report with an XML file (suffix: **.jrxml**). Its format is XML-based and can be edited with any XML editor. We recommend that you use TIBCO Jaspersoft Studio to create and maintain your JasperReports definitions. It is a free, open source, Eclipse-based report designer and allows you to design complex JasperReports definitions easily and quickly.

The report design usually contains a SQL query to set the scope of database records to be reported. However, DirX Audit can optionally provide the input records via its API. See the chapter "Working with View Type APIs" for more details.

A report configuration comprises the following items:

- Report definition – an XML-formatted file that contains meta data for the report. DirX Audit evaluates it when you set up a schedule for the report and configure its scope and when the report is produced either in DirX Audit Manager, DirX Audit Manager Classic or scheduled by DirX Audit Server. The report definition refers to a SQL query and Report output format.
- SQL query – a file that contains a SQL query. DirX Audit executes this query and passes the resulting database records to JasperReports when the report is generated. The SQL query can contain placeholders, which are variables that must be set in DirX Audit Manager or DirX Audit Manager Classic when the report is configured and scheduled. See the chapter "Using Reports Page" in the *DirX Audit Manager Guide* for DirX Audit Manager or see the chapter "Using the Reports View" in the *DirX Audit Manager Classic Guide* for details.
- Report output format – a ***.jrxml** file typically created using TIBCO Jaspersoft Studio.

These files must be stored in the following locations in the DirX Audit installation folder:

- The report definition and SQL query files must be stored in the folder *install_path***/conf/defaults/report-definitions/***reportName*. The file names are *reportName***.xml** for the report definition used in DirX Audit Manager Classic, *reportName***.json** for the report definition used in DirX Audit Manager and *reportName***.sql** for the SQL query.
- The report output format must be stored in *install_path* **/conf/defaults/reports/***reportName***/***reportName***.jrxml**.

Make sure these files are stored in these locations on every host where either DirX Audit Manager, DirX Audit Manager Classic or DirX Audit Server is installed.

This chapter describes how to:

- Configure a report definition file
- Configure a SQL query with placeholders
- Set up TIBCO Jaspersoft Studio
- Create a new report and change an existing report

- Add TIBCO JasperReports templates to DirX Audit Manager's Audit Analysis and History
- Solve problems with reports

For more information, see:

- The TIBCO Jaspersoft Studio homepage:
  https://community.jaspersoft.com/project/jaspersoft-studio
- The TIBCO JasperReports homepage:
  http://sourceforge.net/projects/jasperreports

3.1. Configuring Custom Files

If you want to customize a report and its resources, it is possible to do that for all or for a specific tenant. Configuration of custom files apply concepts described in the "General Customization" chapter.

3.1.1. Configuring Custom Files in Report-definitions Folders

Default report definition files are located in the common core default folder:

*install_path***/conf/defaults/report-definitions/***reportName*

It is not recommended to modify Report definitions and SQL queries located at this location, as with new installation or upgrade the entire folder will be overwritten or deleted. To create a customized version, copy the folder with its content to core customization folder for all tenants:

*install_path***/conf/report-definitions/***reportName*

For specific tenant, you can copy the folder to:

*install_path***/conf/tenants/***tenantID***/report-definitions/***reportName*

where *tenantID* is a unique tenant identifier. During installation or upgrade, these folders will be preserved.

It is necessary to rename the folder and its content, as report definitions must have unique names in order to load properly. As it is not allowed to overwrite default report-definitions the find first file resolving priority is applied. Modify the SQL query and Report definition according to the following chapters.

3.1.2. Configuring Custom Files in Reports Folders

Default report files are located in the common core default folder:

*install_path***/conf/defaults/reports/**

Customized version for all tenants is located at:

*install_path***/conf/reports/**

For specific tenant, the folder is:

*install_path***/conf/tenants/***tenantID***/reports/**

where *tenantID* is a unique tenant identifier.

As the find last file resolving priority is applied, all files can be customized. These files are:
.jrxml – report output format
.svg – images used in report output format located at **reports/img**
.jrtx – report output format style used in report output format located at **reports/_StyleTemplate**

For example, the report output format .jxrml file for report HstApprovers located at:

*install_path***/conf/defaults/reports/***HstApprovers/HstApprovers.jrxml*

Can be customized for all tenants by copying the file to:

*install_path***/conf/reports/***HstApprovers/HstApprovers.jrxml*

And/or for specific tenant to:

*install_path***/conf/tenants/***tenantID***/reports/***HstApprovers/HstApprovers.jrxml*

During the report creation, the most specific .jrxml file will be used. The same applies for .svg and .jrtx files. This includes common shared subreports located at **reports/_common_shared** folder.

Note, that subreports defined in specific report must always be present inside the report's folder and only those will be compiled.

3.1.3. Customizing Company Logo in Report Templates

A default company logo is located at:

*install_path***/conf/defaults/reports/img/***company_logo.svg*

The logo can be customized in a similar way to the report content as explained in the previous chapter. The customized logo file must be named *company_logo.svg*.

To disable the logo in generated reports by default, copy the provided transparent image at:

*install_path***/conf/defaults/reports/img/***company_logo_empty.svg*

For all tenants to:

*install_path***/conf/reports/img/***company_logo_empty.svg*

For a specific tenant to:

*install_path***/conf/tenants/***tenantID***/reports/img/***company_logo_empty.svg*

And rename the file *company_logo_empty.svg* in the new location to *company_logo.svg*. During report generation, the transparent image will be used, and the logo will not be visible in generated files.

3.2. Configuring a Report Definition File for Dirx Audit Manager Classic

A report definition is stored in an XML file according to the XML namespace http://definition.report.persistence.xml.audit.dirx.atos.net. To configure a report definition, you need to:

- Set up report definition XML elements
- Extend DirX Audit Manager Classic's application context, if you have created customized versions of Java Beans for **PickListControl** controls defined in the report definition.

3.2.1. Setting Report Definition Elements for DirX Audit Manager Classic

You need to set the following XML elements:

id
– the unique identifier for the report.

name and **description**
– the name and description of the report to be displayed by DirX Audit Manager Classic when selecting and configuring a report. DirX Audit Manager Classic expects a variable in both fields and replaces them with the property found in the corresponding properties files **messages**_*language*.**properties** in the folder *install_path*/**conf/i18/**. If you want to override default properties or add your own ones, we recommend creating your own properties files and putting them into the same folder with the report definitions: *install_path*/**conf/defaults/report-definition/***reportName*/*reportName*.**properties** for English (default), *reportName*_**de.properties** for German and *reportName*_**fr.properties** for French.

query
– the name of the file that contains the SQL query. The file must be in the same folder as the report definition file.

dataType
– the type of event on which to report. Set the value **DXT_DATA** if the report is on audit events and **DXT_HISTORY** if the report is on history entries.

outputType
– the type of report output. Currently, only **LIST** is supported to indicate that the report outputs a list of records.

reportTemplates, **reportTemplate**
– an identifier (*reportName*) of a report output format; that is, a name reference to the ***.jrxml** file.

tags
– the list of <tag> sub-elements that characterize the report and help an auditor when selecting an appropriate report in DirX Audit Manager Classic. Note that these tags can be variables for message properties and be localized into supported languages.

controls
– the list of <control> sub-elements that you need for each placeholder in your SQL scope definition. A control represents a placeholder ${*placeholder*} in the SQL query string that you refer to in the element query. Each placeholder is a variable that must be replaced when the report is produced. When the report is configured in DirX Audit Manager Classic, the auditor can select value(s) for these variables. If a placeholder is flagged as optional, no value needs to be provided. DirX Audit Manager Classic uses the control elements to generate the HTML view for configuration of the report scope. The variables and sub-elements of <control> are:

- **label** – the string or key to a message properties file. DirX Audit Manager Classic displays it as the head of the control. It should help the auditor to understand what to select.
- **controlType** – the name of a user interface component. These components include:

  BooleanControl – a checkbox.

  DateControl – a calendar.

  HiddenControl – a control that is not displayed. This component can be used to pass (nationalized) values into the report output.

  IntegerControl – an input box.

  PickListControl – a powerful control that helps the auditor to search and select from values found in the database; for example, to search and select a list of users by entering their last names.

  PickListControlWithoutSearch – control that helps the auditor to select from pre-selected values set in report definition.

  PickListCsvColumnsControl – control that allows the auditor to select which columns are included in the output when generating a report in spreadsheet (CSV) format.

  SelectControl – a combo box.

  StringControl – an input box.

  TimePointControl – a combo box for defining a time point with a pre-defined option.

  TimeRangeControl – a combo box for defining a time range (from – to) with a pre-defined option.

- **controlName** – the name of a Java Bean that the control uses to obtain attribute names, attribute values and search results. The control must be configured with this name in the Spring report context file *install_path*/**web/audit-manager-classic.war/WEB-INF/classes/reportContext.xml**. See the section "Extending DirX Audit Manager Classic Report Context" for more information on the bean interface.
- **srchSelLabel** – applicable only for **PickListControl**. Represents string or key to a message properties file. DirX Audit Manager Classic displays it as the label of **PickListControl** select list.
- **srchValLabel** – applicable only for **PickListControl**. Represents string or key to a message properties file. DirX Audit Manager Classic displays it as the label of **PickListControl** value list.
- **params** with sub-elements **param** or **paramList** – parameters specific for the control:
  - The **param** element represents a single-value parameter. It is either the name of a placeholder used in the SQL query or passed to JasperReport as a variable. The **params** element has the following XML parameters:
    - **name** – the name of the parameter; exactly as used in the SQL query or in the JasperReports output file.
    - **passToReport** – a Boolean value. If true, it is passed to the JasperReports output file.
    - **passAsValue** – a Boolean value.
    - **passToReportAsValue** – a Boolean value.
    - **type** – the Java full class name that represents the type of the attribute.
    - **use** – whether or not the parameter must be populated. The value **required** (default) tells DirX Audit Manager Classic that a value for the parameter is required. If the value is optional, no value needs to be provided. If it refers to a placeholder in the SQL query, the corresponding section is dropped which requires specific formatting, and no constraints are applied regarding this placeholder.
    - **initialValue** – the initial value if displayed first time in DirX Audit Manager Classic. Note: this is not a default value.
    - **initialDisplayValue** – the initial display value, if displayed for the first time in DirX Audit Manager Classic and **initialValue** is not user-friendly enough.
  - The **paramList** element represents a multi-value parameter. It contains the same XML attributes and sub-elements as **param** with the exception of **initialValue** and **initialDisplayValue**: here they are sub-elements that can occur several times rather than XML attributes.

    Parameters supported by the controls include:
  - **BooleanControl** – supports only one parameter of type java.lang.Boolean.
  - **DateControl** – supports only one parameter of type java.util.Date.
  - **HiddenControl** – supports 1..*n* parameters, either **param** or **paramList**. They need to have an initial value; otherwise the parameter makes no sense.

- **IntegerControl** – supports only one parameter of type java.lang.Integer.
- **PickListControl** – supports only one parameter of type java.lang.String. It requires the **controlName** attribute for the Java Bean that obtains the attributes.
- **SelectControl** – supports only one parameter of type java.lang.String. It requires the **controlName** attribute for the Java Bean that obtains the attributes or one **paramList** element to provide the initial values from which to select.
- **StringControl** – supports only one parameter of type java.lang.String.
- **TimePointControl** – requires the following parameters:
  - Type java.lang.String that holds a constant for **WhenType**. Possible values are: END_OF_PREVIOUS_DAY, END_OF_PREVIOUS_WEEK, END_OF_PREVIOUS_MONTH, END_OF_PREVIOUS_YEAR, CUSTOM_TIME_POINT.
  - Type java.util.Date with the Date value for the specification of the time point.
- **TimeRangeControl** – requires the following parameters:
  - Type java.lang.String that holds a constant for **WhenType**. Possible values are: LAST_HOUR, LAST_24HOURS, LAST_WEEK, LAST_7DAYS, LAST_30DAYS, LAST_MONTH, LAST_3MONTHS, LAST_YEAR, TODAY, YESTERDAY, PREVIOUS_DAY, PREVIOUS_WEEK, WEEK_TO_DATE, PREVIOUS_MONTH, MONTH_TO_DATE, PREVIOUS_YEAR, YEAR_TO_DATE, CUSTOM_TIME and ANYTIME. They correspond to the values in the DirX Audit Manager Classic Audit Analysis tab: last hour / 24 hours, last week, last 7 days, last 30 days, last month, last 3 months, last year, today, yesterday, previous day, previous week, week to date, previous month, month to date, previous year, year to date, custom time (you set fixed start and end date and time) and all (any time). See the section "Filtering Audit Events" in the chapter "Using the Audit analysis" of the *DirX Audit Manager Classic Guide* for details.
  - Type java.util.Date with the **from** value for the start of the time interval.
  - Type java.util.Date with the **to** value for the end of the time interval.

3.2.2. Extending DirX Audit Manager Classic Report Context

When you configure your own Java Bean for a **PickListControl**, you need to extend DirX Audit Manager Classic's report context.

The DirX Audit Manager Classic's report context is in the file:

*install_path***/web/audit-manager-classic.war/WEB-INF/classes/reportContext.xml**

You need to define a bean that reflects your **PickListControl** and you need to register it at the reportConfigurationHolder bean. The registration name must be identical to the name used in the report definition in the controlName XML attribute. See the description of this attribute in the section "Setting Report Definition Elements for DirX Audit Manager Classic" for details.

For example, suppose the controlName is `"SelectUserUidsComponentService"`. We need to register it at the `reportConfigurationHolder` in the following way:

```
<bean
  id="reportConfigurationHolder" …>
  …
  <constructor-arg
    name="componentMap">
    <map>
      …
      <entry
        key="SelectUserUidsComponentService"
        value-ref="selectUserUidsComponentService" />
    </map>
  </constructor-arg>
</bean>
```

In the XML attribute **value-ref**, enter the (case-sensitive) name of your new PickListControlBean; for example, **selectUserUidsComponentService** in the snippet above.

The best way to define the PickListControl bean is to copy one of the existing bean definitions and then change the values. You need to change the bean name in the XML attribute **id** and the following constructor argument values:

- **dbType** – the database type. Use **DXT_DATA** for a report on audit events and **DXT_HISTORY** for reports on history entries.
- **attributeNamesSqlFile** – the name of the file that contains the SQL query for finding attribute names. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.
- **attributeValuesSqlFile** – the name of the file that contains the SQL query for finding attribute values for the above found attribute names. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.
- **searchResultSqlFile** – the name of the file that contains the SQL query for finding records that match an attribute name and value selected from the result set of the above queries. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.

Note, that you can add or customize these SQL files for all and/or specific tenant as described in the “General Customization” chapter. Examples are available in the chapter “Customizing SQL Scripts”.

Here is an example for the bean registered in the previous example:

```
<bean
  id="selectUserWhoUidsComponentService"
  factory-bean="reportComponentServiceFactory"
  factory-method="createComponentService">
<constructor-arg
  name="dbType"
  value="DXT_DATA" />
<constructor-arg
  name="attributeNamesSqlFile"
  value="data_user_who_uid_attribute_names.sql" />
<constructor-arg
  name="attributeValuesSqlFile"
  value="data_user_who_uid_attribute_values.sql" />
<constructor-arg
  name="searchResultSqlFile"
  value="data_user_who_uid_search_result.sql" />
</bean>
```

The control searches in the database on audit events.

The query for attribute names is intended to find all attribute names that were used as an identifying attribute for the "who" of an audit message. Typical examples of these names are: AltName, First Name, Last name, employeeNumber, Organization.

The query for attribute values is intended to find all values for a given attribute name obtained from the query above. It contains two placeholders: the first is replaced with the attribute name. For example, the auditor could select **Last Name** from the list above. The second placeholder is replaced with the sub-string the auditor entered. For example, the auditor wants to find all users whose last name starts with **a**, then the placeholders would become **a%**.

The query in **searchResultSqlFile** is intended to find – in this example – all users that match the attribute name (for example, **Last Name**) and the attribute value sub-string (for example, **a**). It returns the UID of these entries – here, users – along with other attributes that help the auditor to identify them. For users, these identifying attributes are often **AltName** and **Organizational Unit**. The placeholders for the attribute name and value sub-string are **${KEY}** and **${VALUE_LIKE}**. Note the ${...} notation here: these are not JDBC placeholders as in the query above, but placeholders as used in the Apache FreeMarker template engine. This gives you the option to use a placeholder several times in the query.

3.3. Configuring a Report Definition File for Dirx Audit Manager

A report definition is stored in an JSON file according to the JSON schema https://audit.dirx.solutions/schemas/report/report-definition-schema.json. To configure a report definition, you need to:

- Set up report definition JSON properties
- Extend DirX Audit Manager's report components, if you have created customized versions of Components for **array-picklist-modal** or **dxtselect** UI parameters defined in the report definition.

3.3.1. Setting Report Definition Elements for DirX Audit Manager

The JSON schema for a report definition is structured into three main sections: `itemUuid`, `metadata`, and `data`.

itemUuid – the unique identifier for the report definition, represented as a version-4 UUID.

metadata – contains meta-information about the report definition, including:

- **version** – the version of the report definition.
- **revision** – the revision number of the report definition.
- **accessLevel** – the access level required for the report definition. For customization purposes, set this property to **PUBLIC**.
- **name** – the name of the report definition.
- **description** – the description of the report definition.
- **creationTimestamp** – the timestamp indicating when the report definition was created.

data – contains the actual configuration and structure of the report definition, including query references, UI parameters, templates, and other report definition-specific settings:

- **id** – the unique identifier for the report.
- **name** and **description** – the name and description of the report to be displayed by DirX Audit Manager when selecting and configuring a report. DirX Audit Manager expects a variable in both fields and replaces them with the property found in the corresponding properties files **messages**_*language*.**properties** in the folder *install_path*/**conf/i18/**. If you want to override default properties or add your own ones, we recommend creating your own properties files and putting them into the same folder with the report definitions: *install_path*/**conf/defaults/report-definition/***reportName*/*reportName*.**properties** for English (default), *reportName*_**de.properties** for German and *reportName*_**fr.properties** for French.
- **query** – the name of the file that contains the SQL query. The file must be in the same folder as the report definition file.
- **dataType** – the type of event on which to report. Set the value **DXT_DATA** if the report is on audit events and **DXT_HISTORY** if the report is on history entries.

- **outputType** – the type of report output. Currently, only **LIST** is supported to indicate that the report outputs a list of records.
- **reportTemplates** – an array of objects, each representing a report output format. Each object contains:
  - **id** – a unique identifier for the report template.
  - **label** – the display name or key for the template, which can be localized into supported languages.
  - **path** – the reference to the **.jrxml** file.
- **tags** – the array of tags that characterize the report and help an auditor when selecting an appropriate report in DirX Audit Manager. Note that these tags can be variables for message properties and be localized into supported languages.
- **parameters** – the array of UI parameters that you need for each placeholder in your SQL scope definition. A UI parameter represents a placeholder ${*placeholder*} in the SQL query string that you refer to in the query property. Each placeholder is a variable that must be replaced when the report is produced. When the report is configured in DirX Audit Manager, the auditor can select value(s) for these variables. If a placeholder is flagged as optional, no value needs to be provided. DirX Audit Manager uses the UI parameters for configuration of the report scope. The variables and properties of UI parameter are:
  - **key** – the unique key of a UI parameter
  - **reportTemplateIds** – references **id** values in **reportTemplates** for which the UI parameter is shown; the UI parameter is only shown for the specific selected template id(s) listed here, and hidden for others. If omitted, the UI parameter is visible for all templates.
  - **params** or **paramsList** – parameters specific for the UI parameter:
    - The **params** property represents a single-value parameter. It is either the name of a placeholder used in the SQL query or passed to JasperReport as a variable. The **params** property has the following properties:
      - **name** – the name of the parameter; exactly as used in the SQL query or in the JasperReports output file.
      - **passToReport** – a Boolean value. If true, it is passed to the JasperReports output file.
      - **passAsValue** – a Boolean value.
      - **passToReportAsValue** – a Boolean value.
      - **type** – the Java full class name that represents the type of the attribute.
      - **use** – whether or not the parameter must be populated. The value **required** (default) tells DirX Audit Manager that a value for the parameter is required. If the value is optional, no value needs to be provided. If it refers to a placeholder in the SQL query, the corresponding section is dropped which requires specific formatting, and no constraints are applied regarding this placeholder.
      - **initialValue** – the initial value if displayed first time in DirX Audit Manager.

Note: this is not a default value.

- **initialDisplayValue** – the initial display value, if displayed for the first time in DirX Audit Manager and **initialValue** is not user-friendly enough.

- The **paramsList** property represents a multi-value parameter. It contains the same properties and sub-elements as **params** with the exception of **initialValue** and **initialDisplayValue**: here they are sub-elements that can occur several times.

- **formlySchema** – definition of the UI, contains:
  - **type** – simple or complex type of UI parameter:

    boolean – a checkbox, supports only one parameter of type java.lang.Boolean.

    integer – an input box, supports only one parameter of type java.lang.Integer.

    string – an input box, supports only one parameter of type java.lang.String.

    object – a complex input described further.

  - **widget** – contains definition of **formlyConfig**:
    - **type** – complex types:

      period – a combo box for defining a time range (from – to) with a pre-defined option, requires the following parameters:

      Type java.lang.String that holds a constant for **WhenType**. Possible values are: LAST_HOUR, LAST_24HOURS, LAST_WEEK, LAST_7DAYS, LAST_30DAYS, LAST_MONTH, LAST_3MONTHS, LAST_YEAR, TODAY, YESTERDAY, PREVIOUS_DAY, PREVIOUS_WEEK, WEEK_TO_DATE, PREVIOUS_MONTH, MONTH_TO_DATE, PREVIOUS_YEAR, YEAR_TO_DATE, CUSTOM_TIME and ANYTIME. They correspond to the values in the DirX Audit Manager Audit analysis tab: last hour / 24 hours, last week, last 7 days, last 30 days, last month, last 3 months, last year, today, yesterday, previous day, previous week, week to date, previous month, month to date, previous year, year to date, custom time (you set fixed start and end date and time) and all (any time). See the section "Filtering Audit Events" in the chapter "Using Audit Analysis" of the *DirX Audit Manager Guide* for details.

      Type java.util.Date with the **from** value for the start of the time interval.

      Type java.util.Date with the **to** value for the end of the time interval.

      timepoint – a combo box for defining a time point with a pre-defined option, requires the following parameters:

      Type java.lang.String that holds a constant for **WhenType**. Possible values are: END_OF_PREVIOUS_DAY, END_OF_PREVIOUS_WEEK, END_OF_PREVIOUS_MONTH, END_OF_PREVIOUS_YEAR, CUSTOM_TIME_POINT.

      Type java.util.Date with the Date value for the specification of the time point.

dxtselect – a combo box, supports only one parameter of type java.lang.String.

array-picklist-modal – a control that helps the auditor to search and select from values found in the database, pre-selected values set in report definition or select which columns are included in the output when generating a report in spreadsheet (CSV) format. Supports only one parameter of type java.lang.String.

- **templateOptions** – contains miscellaneous options
  - **label** – label of a UI parameter
  - **componentId** – id of a component providing values from database, relevant only for **array-picklist-modal** or **dxtselect**
  - **options** – property providing values, relevant only for **array-picklist-modal** or **dxtselect**
- **default** – preselected value of UI parameter
- **required** – boolean value deciding if values must be provided

3.3.2. Extending DirX Audit Manager Report Components

When you configure your own Component for a **array-picklist-modal** or **dxtselect**, you need to extend DirX Audit Manager's report components.

The DirX Audit Manager's report components are in the file:
install_path/**conf/defaults/report-components/dxtComponents.json**

For adding or customizing a report component(s), see the chapter "Customizing Report Components for DirX Audit Manager" in the "General Customization" chapter.

To define a component that reflects your **array-picklist-modal** or **dxtselect**. The **id** must be identical to the name used in the report definition in the componentId property.

```
{
  "components": [
    …
    {
      "id": "selectUserWhoUidsComponentService",
      "dbType": "DXT_DATA",
      "attributeNamesSqlFile":
"data_user_who_uid_attribute_names.sql",
      "attributeValuesSqlFile":
"data_user_who_uid_attribute_values.sql",
      "searchResultSqlFile": "data_user_who_uid_search_result.sql"
    },
    …
```

```
    {
      "id": "selectUsrAttrComponentService",
      "dbType": "DXT_HISTORY",
      "attributeNamesSqlFile": "hst_usr_attrs.sql",
      "attributeValuesSqlFile": null,
      "searchResultSqlFile": null
    },
    ...
  ]
}
```

The best way to define the Component is to copy one of the existing component definitions and then change the values. You need to change the following constructor argument values:

- **id** – componentId defined in **formlySchema**.
- **dbType** – the database type. Use **DXT_DATA** for a report on audit events and **DXT_HISTORY** for reports on history entries.
- **attributeNamesSqlFile** – the name of the file that contains the SQL query for finding attribute names. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.
- **attributeValuesSqlFile** – the name of the file that contains the SQL query for finding attribute values for the above found attribute names. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.
- **searchResultSqlFile** – the name of the file that contains the SQL query for finding records that match an attribute name and value selected from the result set of the above queries. A file with this name must be placed in the folder *install_path***/conf/defaults/sql/common/report_search_queries/**.

Note, that you can add or customize these SQL files for all and/or specific tenant as described in the “General Customization” chapter. Examples are available in the chapter “Customizing SQL Scripts”.

The component searches in the database on audit events.

The query for attribute names is intended to find all attribute names that were used as an identifying attribute for the "who" of an audit message. Typical examples of these names are: AltName, First Name, Last name, employeeNumber, Organization.

The query for attribute values is intended to find all values for a given attribute name obtained from the query above. It contains two placeholders: the first is replaced with the attribute name. For example, the auditor could select **Last Name** from the list above. The second placeholder is replaced with the sub-string the auditor entered. For example, the auditor wants to find all users whose last name starts with **a**, then the placeholders would become **a%**.

The query in **searchResultSqlFile** is intended to find – in this example – all users that match the attribute name (for example, **Last Name**) and the attribute value sub-string (for example, **a**). It returns the UID of these entries – here, users – along with other attributes that help the auditor to identify them. For users, these identifying attributes are often **AltName** and **Organizational Unit**. The placeholders for the attribute name and value sub-string are **${KEY}** and **${VALUE_LIKE}**. Note the ${...} notation here: these are not JDBC placeholders as in the query above, but placeholders as used in the Apache FreeMarker template engine. This gives you the option to use a placeholder several times in the query.

3.4. Configuring a Report Indicator for Dirx Audit Manager

A report indicator is used for creating conditions, see the chapter "Using Reports Page" in the *DirX Audit Manager Guide* for details. Report indicator is stored in an JSON file according to the JSON schema https://audit.dirx.solutions/schemas/report/indicator-schema.json. To configure a report indicator, you need to:

- Set up report indicator JSON properties
- Extend DirX Audit Manager's report indicators

3.4.1. Setting Report Indicator Properties

The JSON schema for a report indicator is structured into three main sections: `itemUuid`, `metadata`, and `data`.

itemUuid – the unique identifier for the indicator, represented as a version-4 UUID.

metadata – contains meta-information about the report indicator, including:

- **version** – the version of the report indicator.
- **accessLevel** – the access level required for the report indicator. For customization purposes, set this property to **PUBLIC**.

data – contains the actual configuration and structure of the report indicator:

- **id** – the unique identifier for the indicator.
- **name** and **description** – the name and description of the report indicator to be displayed by DirX Audit Manager when selecting and configuring an indicator.
- **indicatorQueryId** – **id** of a referenced report indicator query
- **queryParameterValues** – an array of query parameters, see the chapter "Configuring a SQL Query with Placeholders" for more information:
  - **queryParameterId** – a unique identifier for the parameter.
  - **value** – value of the given parameter.

3.4.2. Extending DirX Audit Manager Report Indicators

When you configure your own Indicator, you need to extend DirX Audit Manager's report indicators.

The DirX Audit Manager's report indicators are in the file:
install_path/**conf/defaults/report-indicators/dxtIndicators.json**

For adding or customizing a report indicator(s), see the chapter "Customizing Report Indicators and Indicator Queries for DirX Audit Manager" in the "General Customization" chapter.

```
{
  "indicators": [
    …
    {
      "itemUuid": "fe3a9947-9e74-4c81-8387-ead535f79b16",
      "metadata": {
      "version": "1.0.0",
      "accessLevel": "PRODUCT"
      },
      "data": {
        "id": "evn_added_users_in_5_days",
        "name": "indicator.def.evn_added_users_in_5_days.name",
        "description":
"indicator.def.evn_added_users_in_5_days.description",
        "indicatorQueryId": "evn_added_objects_in_number_of_days",
        "queryParameterValues": [
          {
            "queryParameterId": "NUMBER_OF_DAYS",
            "value": "5"
          },
          {
            "queryParameterId": "OBJECT_TYPE",
            "value": "User"
          }
        ]
      }
    },
    …
  ]
}
```

The best way to define the Indicator is to copy one of the existing indicators and then change the values. It is necessary to change **itemUuid** to a **unique** valid value.

3.5. Configuring a Report Indicator Query for Dirx Audit Manager

A report indicator query is stored in an JSON file according to the JSON schema https://audit.dirx.solutions/schemas/report/indicator-query-schema.json. To configure a report indicator, you need to:

- Set up report indicator query JSON properties
- Extend DirX Audit Manager's report indicator queries

3.5.1. Setting Report Indicator Queries Properties

The JSON schema for a report indicator query is structured into two main sections: `metadata`, and `data`.

metadata – contains meta-information about the report indicator query, including:

- **version** – the version of the report indicator query.
- **accessLevel** – the access level required for the report indicator query. For customization purposes, set this property to **PUBLIC**.

data – contains the actual configuration and structure of the report indicator query:

- **id** – the unique identifier for the indicator query.
- **name** and **description** – the name and description of the indicator query.
- **dxtDbType** – the database type. Use **DXT_DATA** for a indicator query on audit events and **DXT_HISTORY** for indicator query on history entries.
- **queryPath** – the reference to the **.sql** file.
- **resultDataType** – type of a result retruned by the query.
- **queryParameters** – defined parameters, which the query expects.
  - **id** – a unique identifier for the parameter.
  - **name** and **description** – the name and description of the report indicator query.
  - **parameterDataType** – type of parameter passed as a placeholder to SQL query
  - **required** – boolean value deciding if values must be provided

3.5.2. Extending DirX Audit Manager Report Indicator Queries

When you configure your own Indicator Query, you need to extend DirX Audit Manager's report indicator queries.

The DirX Audit Manager's report indicator queries are in the file:
install_path/**conf/defaults/report-indicators/dxtIndicatorQueries.json**

For adding or customizing a report indicator queries, see the chapter “Customizing Report Indicators and Indicator Queries for DirX Audit Manager” in the “General Customization” chapter.

```
{
  "indicatorQueries": [
    …
    {
      "metadata": {
        "version": "1.0.0",
        "accessLevel": "PRODUCT"
      },
      "data": {
        "id": "evn_added_objects_in_number_of_days",
        "name":
"indicatorQuery.def.evn_added_objects_in_number_of_days.name",
        "description":
"indicatorQuery.def.evn_added_objects_in_number_of_days.description",
        "dxtDbType": "DXT_DATA",
        "queryPath": "data/evn_added_objects_in_number_of_days.sql",
        "resultDataType": "number",
        "queryParameters": [
          {
            "id": "NUMBER_OF_DAYS",
            "name": "queryParameter.def.NUMBER_OF_DAYS.name",
            "description":
"queryParameter.def.NUMBER_OF_DAYS.description",
            "parameterDataType": "number",
            "required": true
          },
          {
            "id": "OBJECT_TYPE",
            "name": "queryParameter.def.OBJECT_TYPE.name",
            "description":
"queryParameter.def.OBJECT_TYPE.description",
            "parameterDataType": "string",
            "required": false
          }
        ]
      }
    },
```

```
    ...
  ]
}
```

The best way to define the Indicator Query is to copy one of the existing indicator queries and then change the values.

3.6. Configuring a SQL Query with Placeholders

In the query element of a report definition, you refer to a file that holds a SQL query. The file must be located in the same folder as the report definition. DirX Audit executes the SQL query when it generates the report. It passes the query result to JasperReports.

The file contains a normal SQL query that must be recognized by the database you have deployed. Typically, you'll use some variables or placeholders that are replaced at runtime. The most common are the start and end of the time range in which you are interested. They should not be fixed, but cover, for example, the previous month. Other use cases can include:

- An option for whether or not to include only orphaned or disabled accounts.
- A list of users whose data should be reported.
- A list of privileges whose users should be reported.
- A list of target systems whose accounts or groups should be reported.

These placeholders can be mandatory or optional. An auditor might provide exactly one value, multiple values or none.

These placeholders must be configured in the report definition as **param** or **paramList** sub-elements of control elements.

Use these placeholders in the SQL query to set a value of the form **${***placeholder***}** instead of a real value. For example, use the placeholders **${WHEN_FROM}** and **${WHEN_TO}** for the beginning and end of a time range or **${UIDS}** for a list of unique identifiers.

Pay attention to optional parameters. In most cases, it is not sufficient just to enter the placeholder name. If no values are given, they can render the query incorrectly or produce an unexpected result. The typical solution in these cases is to drop certain sections of the query that contain the optional parameter.

DirX Audit supports these use cases by leveraging the Apache FreeMarker open source template engine. See http://freemarker.org/docs/index.html for the Apache FreeMarker user guide. The conditional if statement is particularly useful for excluding sections depending on a parameter value. Here is a sample snippet for a parameter that represents lists of users occurring as a "what" in an audit event:

```
<#if UIDS?? >
  and what_user.WHAT_UID in ${UIDS}
</#if>
```

The condition for the WHAT_UID column is only included into the SQL query when the UIDS placeholder is not empty.

Check the queries delivered with the product for additional samples.

3.7. Setting up TIBCO Jaspersoft Studio

This section provides information on how to install and set up TIBCO Jaspersoft Studio in the DirX Audit environment.

3.7.1. Installing TIBCO Jaspersoft Studio

To install TIBCO Jaspersoft Studio:

- Download Jaspersoft Studio from https://sourceforge.net/projects/jasperstudio/files/
- Run the downloaded file to install TIBCO Jaspersoft Studio

You can now run TIBCO Jaspersoft Studio from the installation location.

3.7.2. Setting Up the TIBCO Jaspersoft Studio Classpath

TIBCO Jaspersoft Studio no longer requires a large, unique classpath for the entire application like iReport does. Each report is intended to be a part of a project. Each project has a classpath where **jar** libraries are added. To set up the TIBCO Jaspersoft Studio classpath for the selected project:

- Start TIBCO Jaspersoft Studio.
- Right-click on the project you selected.
- Select **Properties** → **Java Build Path** → **Libraries**.
- Click **Add external JARs...** . A file dialog opens.
- Add the files:

  install_path/**Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/DirXjdiscoverAPI-***version*.**jar**

  install_path/**Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-common-config-***version*.**jar**

  install_path/**Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-common-reports-support-***version*.**jar**

  install_path/**Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-db-**

api-config-*version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-db-apiquery-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-db-api-raw-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-db-schema-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-dbv3-event-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-dxi-digest-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-history-api-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-pep-api-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/dxt-utils-lib-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/log4j-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/slf4j-api-***version***.jar**

*install_path***/Deployment/Manager/audit-manager-classic.war/WEB-INF/lib/slf4j-log4j12-***version***.jar**

- Click **Apply**.

3.8. Creating a New Report

Creating a new report consists of the following tasks:

- Creating the top-level page of the report
- Creating a sub report

The following sections describe these tasks.

3.8.1. Creating the Top-Level Page

Creating the top-level page consists of the following tasks:

- Creating the report file
- Setting the classpath
- Selecting the field list

3.8.1.1. Creating the Report File

To create the report file:

- Start TIBCO Jaspersoft Studio.
- Select **File** → **New** → **Jasper Report**.

The Report Wizard starts. It will guide you through the following steps:

1. Report Templates
2. Report File
3. Data Source
4. Finish

1. Select a template; for example, **BlankA4**. Click **Next**.
2. Enter **MyTableReport** as the report name. You can select a parent folder where the report will be created. Click **Next**. (Alternatively, once you finish with your changes and save the report, you can just copy your report to the folder of your choice.)
3. In the **Data Adapter** drop-down list, select **One Empty Record - Empty Rows**. You must specify the data source later. Click **Next**.
4. Click **Finish**. If no errors occur, the new report is created.

3.8.1.2. Setting the Classpath

To set up the classpath, follow the instructions in “Setting Up the TIBCO Jaspersoft Studio Classpath” if you have not already done so.

3.8.1.3. Setting the Data Adapter

To set the data adapter:

- Switch to the **Repository Explorer**.
- Right-click **Data Adapters** and choose **Create Data Adapter**.
- The Data Adapter Wizard opens.
- Choose **Database JDBC Connection** and follow the wizard to specify your database connectivity. Pay special attention to defining the database connectivity when your audit events and history entries are stored in different databases.

3.8.1.4. Defining Report Parameters

To define report parameters:

- In **Outline**, right-click **Parameters** and then select **Create Parameter**.
- Provide **Name**, **Class** and additional properties of the new parameter.
- Repeat the procedure for additional report parameters.

3.8.1.5. Selecting the Field List

To select the field list:

- In **Outline**, right-click the name of your report and then select **Dataset** and **Query ...**
- Click the Query tab.
- Select the data adapter you have defined in "Setting the Data Adapter".
- Select **sql** in **Language**.
- Place your SQL query in the **Texts**. The query should correspond to your SQL query file as defined in the "Setting Report Definition Elements for DirX Audit Manager" or in the "Setting Report Definition Elements for DirX Audit Manager Classic" and in the "Configuring a SQL Query with Placeholders".

  The query cannot contain any Apache FreeMarker elements and placeholders must be transferred to JasperReports parameters set in the "Defining Report Parameters". This means that **${***placeholder***}** must be replaced with **$P{***placeholder***}** in the query. The query can be simplified. In the TIBCO Jaspersoft Studio, it serves only to support designing the report output format. The query will not be applied during the report execution in DirX Audit Manager or DirX Audit Manager Classic and DirX Audit Server.
- Click **Read Fields** to automatically read a list of report fields from the SQL query. Review the automatically generated data type for each field.
- Now the list of fields is available in the **Outline** view under the **Fields** node. You can drag these fields to the report layout area and then set up your report as required. See the TIBCO Jaspersoft Studio and TIBCO JasperReports documentation for more information.

Don't forget to save your report definition from time to time. You can also see what your current report looks like by clicking the **Preview** tab.

3.8.2. Creating a Subreport

Top-level pages can contain subreports, which should be used for parts that may occur multiple times per audit message, especially the **Extension**, **What** and **What Details** sections in audit events. To create subreports:

- Create the subreport element in the current page.
- Create the subreport definition as a separate TIBCO JasperReports file (JRXML).

3.8.2.1. Creating the Subreport Element

You need to create a subreport element in your table layout for each subreport. For each of these subreport elements, you need to assign the corresponding TIBCO Jaspersoft Studio file that contains the subreport definition. This section shows how to create a What section in a top-level report and then display the field What – Name.

First, we will create the subreport element in our top-level page:

- Start TIBCO Jaspersoft Studio. Click **File** → **Open File ...** and then select the JasperReport to which you want to add subreport.
- Select the subreport item in the Palette tab and then drag it to the report pane. A wizard opens.
- Select **Just create the subreport element** and then click **Finish**. The new subreport element is visible.
- Adjust the size and position of the subreport element.
- Select the element to open its properties and then select the Subreport tab.
- In **Expression**, enter the string **$P{SUBREPORT_DIR} + "WhatReport.jasper"**.
  The "WhatReport.jasper" expression specifies the external TIBCO JasperReports file with the definition of the subreport.
- Edit **Data Source Expression**.
- In **Connection Expression**, enter the string **$P{DXT_SUBREPORT_CONNECTION}**.
- Add a parameter reference to the subreport:
  - Click **Edit Parameters**.
  - In the **Subreport Parameters** dialog, click **Add**. The **Parameter Configuration Dialog** is opened.
  - Fill in the **Parameter Name** to the new parameter name **WHEN_FROM**.
  - Click the edit button of the **Parameter Expression**. A new **Expression Editor** dialog opens.
  - Enter **$P(WHEN_FROM)** to create an expression for the parameter, and then click **Finish**.
  - Click **OK** to close the **Parameter Configuration Dialog**.

If you want to create additional subreport elements, copy an existing subreport and then simply change the **Expression** field in the Subreport tab of the element's Properties view. This action changes the file name for the TIBCO JasperReports file.

3.8.2.2. Creating the Subreport TIBCO JasperReports File

Now we will create a TIBCO JasperReports definition file (a JRXML file) for the subreport. Name it **WhatReport.jrxml**.

Follow the steps in the "Creating the Top-Level Page". Now you can place the fields from the sub-level within this page (in this case, the What level). To create, for example, the What – Name field, simply create a field with the expression $F{Name}.

3.9. Modifying a Report

Use TIBCO Jaspersoft Studio to modify a report definition:

- Select **File** → **Open** from the menu bar.
- Navigate to the file you want to change.
- Select the file and then click **Open**.

Now let us make some typical changes.

3.9.1. Changing the Report Title

To change the report title:

- Select the text field and change it, for example, to **$R{report.title.myreportcopy}**.
- Create a new file *install_path***/conf/defaults/reports/***reportName***/***reportName***.properties** (or *reportName***_de.properties** for reports in German, *reportName***_fr.properties** for reports in French) to store new keys and labels. Create a new line containing: **report.title.myreportcopy = My Report Copy**, where **My Report Copy** can be easily replaced by any string.

3.9.2. Changing the Color of the Column Headers

All data fields have style properties. You simply change the property value to change the field.

- In the Design view, click on a field that you want to change.
- In the Properties tab, select **Appearance**.
- In the **Color** section, you can set **Forecolor** and **Backcolor** by clicking on the square that shows the color.
- Select your color and then click **OK**. The color of the field changes.

You can now change the top-level page or sub-reports using TIBCO Jaspersoft Studio features. See the TIBCO Jaspersoft Studio documentation for more information.

Note: if you create a new field and/or your color does not change, it's because the **Transparent** flag is set. To clear the **Transparent** flag:

- In the Properties tab, select **Appearance**.
- In the **Color** section, uncheck **Transparent**.

3.9.3. Changing Common Parts of Reports

Reports have defined common parts, which are reused as subreports. These are page header, page footer and no data message and can be customized as is described in the section "Configuring Custom Files" in reports folders:

- Common shared subreport files:
  install_path/**conf/defaults/reports/*_common_shared*/*_common_pageHeader.jrxml*
  install_path/**conf/defaults/reports/*_common_shared*/*_common_pageFooter.jrxml*
  install_path/**conf/defaults/reports/*_common_shared*/*_common_noData.jrxml*

3.10. Adding JasperReports Templates to Audit Analysis for Dirx Audit Manager Classic

You can add your own TIBCO JasperReports templates to DirX Audit Manager Classic's Audit analysis. To add a JasperReport template to the Audit analysis:

- Navigate to *install_path*/**conf/defaults/reports/**. Create a new folder that starts with **EventMonitor**; for example, **EventMonitorTest**.
- In TIBCO Jaspersoft Studio, create a new report with the same name as the folder you created in the previous step. For example, **EventMonitorTest.jrxml**.
- Customize your report. The Audit analysis reports are based on Java Bean, not a SQL query. See EventMonitorAll, EventMonitorPlain and EventMonitorStd reports and their definitions. For more details on the Java Bean API, see the chapter "Working with View Type APIs".
- Make sure that the name in the Outline view is same as the name of the report.
- Save the report to your TIBCO Jaspersoft Studio workspace.
- Navigate to your workspace. Right-click on your report (in this case, **EventMonitorTest.jrxml**) and then click **Copy**.
- Navigate to your new folder – for example, *install_path*/**conf/defaults/reports/EventMonitorTest** - and then paste your report file here.
- Make sure that the name of your report is the same as the name of the folder in which it is located.

If you cannot see your new template in DirX Audit Manager Classic / Audit analysis / Report / Template list, restart the Apache Tomcat service used for DirX Audit Manager Classic.

Note, that you can add or customize these reports for all and/or specific tenant as described in the chapter "Configuring Custom Files".

3.11. Adding JasperReports Templates to Audit Analysis for Dirx Audit Manager

You can add your own TIBCO JasperReports templates to DirX Audit Manager's Audit Analysis. To add a new report template to the Audit analysis:

- Navigate to *install_path*/**conf/defaults/reports/**. Create a new folder, for example, **EvnAuditAnalysisExportTest**.
- In TIBCO Jaspersoft Studio, create a new report with the same name as the folder you created in the previous step. For example, **EvnAuditAnalysisExportTest.jrxml**.

- Customize your report.
- Make sure that the name in the Outline view is same as the name of the report.
- Save the report to your TIBCO Jaspersoft Studio workspace.
- Navigate to your workspace. Right-click on your report (in this case, **EvnAuditAnalysisExportTest.jrxml**) and then click **Copy**.
- Navigate to your new folder – for example, *install_path***/conf/defaults/reports/EvnAuditAnalysisExportTest** - and then paste your report file here.
- Make sure that the name of your report is the same as the name of the folder in which it is located.
- Navigate to *install_path***/conf/defaults/reports-definitions/EvnAuditAnalysisExport/EvnAuditAnalysisExport.json** and extend the **reportTemplates** array with new template.
- Here is an example of a new template in **reportTemplates**:

```
...
{
  "id": "EvnAuditAnalysisExportTest",
  "label": "EvnAuditAnalysisExportTest_Label",
  "path": "EvnAuditAnalysisExportTest"
},
...
```

If you cannot see your new template in DirX Audit Manager / Audit analysis / Report / Report templates, restart the Apache Tomcat service used for DirX Audit Manager.

Note, that you can add or customize these reports for all and/or specific tenant as described in the chapter “Configuring Custom Files”.

3.12. Adding JasperReports Templates to History tab for Dirx Audit Manager Classic

You can add your own TIBCO JasperReports templates to DirX Audit Manager Classic’s History tab.To add a JasperReport template to the History tab:

- Navigate to *install_path***/conf/defaults/reports/**. Create a new folder that starts with **HistoryEntries**; for example, **HistoryEntriesTest**.
- In TIBCO Jaspersoft Studio, create a new report with the same name as the folder you created in the previous step. For example, **HistoryEntriesTest.jrxml**.
- Customize your report. The History reports are based on Java Bean, not a SQL query. See **HistoryEntries** and **HistoryEntriesPlain** reports and their definitions.
- Make sure that the name in the Outline view is same as the name of the report.

- Save the report to your TIBCO Jaspersoft Studio workspace.
- Navigate to your workspace. Right-click on your report (in this case, **HistoryEntriesTest.jrxml**) and then click **Copy**.
- Navigate to your new folder – for example, *install_path***/conf/defaults/reports/HistoryEntriesTest** - and then paste your report file here.
- Make sure that the name of your report is the same as the name of the folder in which it is located.

If you cannot see your new template in DirX Audit Manager Classic / History / Report / Template list, restart the Apache Tomcat service used for DirX Audit Manager Classic.

Note, that you can add or customize these reports for all and/or specific tenant as described in the chapter “Configuring Custom Files”.

3.13. Adding JasperReports Templates to History tab for Dirx Audit Manager

You can add your own TIBCO JasperReports templates to DirX Audit Manager’s History. To add a new report template to the History:

- Navigate to *install_path***/conf/defaults/reports/**. Create a new folder, for example, **HstHistoryExportTest**.
- In TIBCO Jaspersoft Studio, create a new report with the same name as the folder you created in the previous step. For example, **HstHistoryExportTest.jrxml**.
- Customize your report.
- Make sure that the name in the Outline view is same as the name of the report.
- Save the report to your TIBCO Jaspersoft Studio workspace.
- Navigate to your workspace. Right-click on your report (in this case, **HstHistoryExportTest.jrxml**) and then click **Copy**.
- Navigate to your new folder – for example, *install_path***/conf/defaults/reports/HstHistoryExportTest** - and then paste your report file here.
- Make sure that the name of your report is the same as the name of the folder in which it is located.
- Navigate to *install_path***/conf/defaults/reports-definitions/HstHistoryExport/HstHistoryExport.json** and extend the **reportTemplates** array with new template.

- Here is an example of a new template in **reportTemplates**:

```
...
{
  "id": "HstHistoryExportTest",
  "label": "HstHistoryExportTest_Label",
  "path": "HstHistoryExportTest"
},
...
```

If you cannot see your new template in DirX Audit Manager / Audit analysis / Report / Report templates, restart the Apache Tomcat service used for DirX Audit Manager.

Note, that you can add or customize these reports for all and/or specific tenant as described in the chapter “Configuring Custom Files”.

3.14. Designing Reports for CSV Format

3.14.1. Modify an Existing Report Template

To modify an existing report template for CSV format:

- Copy the report template file and extend its name with the **_csv** suffix.
- Remove **Title**, **Page Header**, **Page Footer**, **Last Page Footer**, **Summary**, **No Data** and **Background** bands.
- Modify the **Report Page** properties (**Page Height**, **Page Width**, **Page Orientation**, **Bottom margin**, **Left Margin**, **Page Width**, **Right Margin**). Height, width and margins can be set to 0. Page orientation is recommended to be **Landscape**.
- Set the report property **net.sf.jasperreports.print.keep.full.text** to **true** in the jrxml file: `<property name="net.sf.jasperreports.print.keep.full.text" value="true"/>`
- Edit the **Detail** band and optionally also the **Group Header** and/or the **Group Footer** bands.
- Check the **Print Repeated Values** (**isPrintRepeatedValues**) property for all text fields.

See *Designing Reports for CSV Export*, https://community.jaspersoft.com/wiki/designing-reports-csv-export, for more details, including how to set a different delimiter; for example, a semicolon.

3.14.2. Use a Common CSV Format Report Template for Dirx Audit Manager Classic

To use common CSV format template:

- Set **reportTemplate** element described in chapter "Setting Report Definition Elements for DirX Audit Manager Classic" to **reports/_common_template_csv/_common_template_csv** to reference reusable common CSV format template. This will present all columns from SQL query used for specific report.
- Optionally, to generate only desired columns, add **PickListCsvColumnsControl** control named **DXT_SELECTED_COLUMNS** to report definition with initial values set to column aliases in report SQL query.
- Here is an example, in which SQL query returns three columns named **USER_NAME**, **ORGANIZATIONAL_UNIT** and **MANAGER**:

```
<control
  controlType="PickListCsvColumnsControl"
  label="report.param.select_columns" >
  <params>
    <paramList
      name="DXT_SELECTED_COLUMNS"
      type="java.lang.String"
      passToReport="true" >
      <initialValue>USER_NAME</initialValue>
      <initialValue>ORGANIZATIONAL_UNIT</initialValue>
      <initialValue>MANAGER</initialValue>
    </paramList>
  </params>
</control>
```

3.14.3. Use a Common CSV Format Report Template for Dirx Audit Manager

To use common CSV format template:

- Set **path** in **reportTemplates** element described in the chapter Setting Report Definition Elements to **reports/_common_template_csv/_common_template_csv** to reference reusable common CSV format template. This will present all columns from SQL query used for specific report.
- Optionally, to generate only desired columns, add **array-picklist-modal** control named **DXT_SELECTED_COLUMNS** to report definition with initial values set to column aliases in report SQL query.

- Here is an example, in which SQL query returns three columns named **USER_NAME**, **ORGANIZATIONAL_UNIT** and **MANAGER**:

```
{
  "key": "DXT_SELECTED_COLUMNS",
  "paramsList": [
    {
      "name": "DXT_SELECTED_COLUMNS",
      "passToReport": true,
      "type": "java.lang.String"
    }
  ],
  "formlySchema": {
    "type": "object",
    "widget": {
      "formlyConfig": {
        "type": "array-picklist-modal",
        "templateOptions": {
          "label": "report.param.select_columns",
          "options": [
            {
              "label": "USER_NAME",
              "value": "USER_NAME"
            },
            {
              "label": "ORGANIZATIONAL_UNIT",
              "value": "ORGANIZATIONAL_UNIT"
            },
            {
              "label": "MANAGER",
              "value": "MANAGER"
            }
          ],
          "preselectAll": true
        }
      }
    },
    "required": true
  }
}
```

3.15. Solving Problems

This section provides tips and tricks to ease report creation, test and execution, including information on how to handle:

- Report design errors
- Problems with report definition changes
- Report execution errors

3.15.1. Handling Report Design Errors

After designing the report, run the compiler and check the output for errors. Double-click an error message in the **Report State** window to open the definition with the error condition. Correct the error and run the compiler again.

If compilation is successful without errors, you can save the record and test it.

3.15.2. Handling Problems with Report Definition Changes

If the DirX Audit Manager does not recognize changes you have made to the report definition, try this procedure:

- Navigate to the folder where your report definitions are stored.
- Delete all ***.jasper** files.
- Run your report in DirX Audit Manager again. This action forces recompilation of the JRXML files.

3.15.3. Handling Report Execution Errors

When you test your reports, you may encounter the error message "Transaction has failed, please try again." If you receive this error, check the Apache Tomcat Application Server log file(s) for error messages:

- Navigate to the folder *tomcat_install_path***/logs**.
- Check the file **dirxaudit-manager.log** for error messages and try to solve the problem. A typical error is:
  `java.lang.NoClassDefFoundError: org/jfree/util/classname`

  JasperReports has many features and this message indicates that a Java library is not available.
- Check the website http://www.findjar.com/ and search for the library *classname*. If found, download it and store it under the following path:
  *tomcat_install_path***/lib**

  Warning: If the library already exists, rename it with an extension that is not **jar**. This action allows you to reset the library if something goes wrong.
- Restart the Apache Tomcat Application Server and run your report again.

3.16. About the Reports Overview

The Reports Overview file is designed to help you identify the most suitable report for your data needs. It provides a categorized list of available reports, including:

- Report groups
- Tags and descriptions
- Source data and output format
- The DirX Audit version in which each report was introduced

Additionally, the overview links each report to:

- Its configuration file in the *install_path***/conf/defaults/report-definitions/** folder
- Sample report files located in the *install_media***/Additions/Data/SampleReports** folder, where the overview file itself is also stored

4. Working with View Type APIs

This chapter describes the Application Programming Interfaces (APIs) that are used for report template definitions. It describes:

- Raw API
- Audit Event API

4.1. Raw API

ReportConfig.xml / API:

- Event

Record data type:

- **solutions.dirx.audit.persistence.api.UniversalRecord**

Field:

- **rawData** : solutions.dirx.audit.persistence.api.Event

Subreports:

- Identification - Extension
- Where From - Extension
- Who - Extension
- What

Column **Column Group**	**Text Field Expression** **Data Source Expression**
Identification - When	$F{rawData}.getWhen()
Identification - Operation	$F{rawData}.getOperation()
Identification - UID	$F{rawData}.getUid()
Identification - Cause	$F{rawData}.getCause()
Identification - Outcome	$F{rawData}.getOutcome().toString()
Identification - Sensitivity	$F{rawData}.getSensitivity()
Identification - Type	$F{rawData}.getType()
Identification - Source	$F{rawData}.getSource()
Identification - Category	$F{rawData}.getCategory()
Identification - Extension	new JRBeanCollectionDataSource($F{rawData}.getExtension())
Where From - Application	$F{rawData}.getWhereFrom().getApplication()

Column Column Group	Text Field Expression Data Source Expression
Where From - Address	$F{rawData}.getWhereFrom().getAddress()
Where From - Type	$F{rawData}.getWhereFrom().getType()
Where From - Extension	new JRBeanCollectionDataSource($F{rawData}.getWhereFrom().getExtensions())
Who - Name	$F{rawData}.getWho().getName()
Who - UID	$F{rawData}.getWho().getUid()
Who - DN	$F{rawData}.getWho().getDn()
Who - From Address	$F{rawData}.getWho().getFromAddress()
Who - From Type	$F{rawData}.getWho().getFromType()
Who - Role	$F{rawData}.getWho().getRole()
Who - Path	$F{rawData}.getWho().getPath()
Who - Extension	new JRBeanCollectionDataSource($F{rawData}.getWho().getExtensions())
What	new JRBeanCollectionDataSource($F{rawData}.getWhat())

4.1.1. Identification - Extension Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.EventExtension**

Fields:

- **type** : java.lang.String
- **value** : java.lang.String

Subreports:

- -

Column	Text Field Expression
Identification - Extension - Type	$F{type}
Identification - Extension - Value	$F{value}

4.1.2. Where From - Extension Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.WhereFromExtension**

Fields:

- **type** : java.lang.String
- **value** : java.lang.String

Subreports:

- -

Column	Text Field Expression
Where From - Extension - Type	$F{type}
Where From - Extension - Value	$F{value}

4.1.3. Who - Extension Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.WhoExtension**

Fields:

- **type** : java.lang.String
- **value** : java.lang.String

Subreports:

- -

Column	Text Field Expression
Who - Extension - Type	$F{type}
Who - Extension - Value	$F{value}

4.1.4. What Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.What**

Fields:

- **name** : java.lang.String
- **dn** : java.lang.String
- **uid** : java.lang.String
- **sensitivity** : java.lang.String
- **lifecycle** : java.lang.String
- **query** : java.lang.String

- **type** : java.lang.String
- **path** : java.lang.String

Subreports:

- What - Extension
- What - Detail

Column Column Group	Text Field Expression Data Source Expression
What - Name	$F{name}
What - DN	$F{dn}
What - UID	$F{uid}
What - Sensitivity	$F{sensitivity}
What - Lifecycle	$F{lifecycle}
What - Query	$F{query}
What - Type	$F{type}
What - Path	$F{path}
What - Extension	new JRBeanCollectionDataSource($F{extension})
What - Detail	new JRBeanCollectionDataSource($F{detail})

4.1.4.1. What - Extension Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.WhatExtension**

Fields:

- **type** : java.lang.String
- **value** : java.lang.String

Subreports:

- -

Column	Text Field Expression
What - Extension - Type	$F{type}
What - Extension - Value	$F{value}

4.1.4.2. What - Detail Subreport

Record data type:

- **solutions.dirx.audit.persistence.api.WhatDetail**

Fields:

- **operation** : java.lang.String
- **type** : java.lang.String
- **value** : java.lang.String

Subreports:

- -

Column	Text Field Expression
What - Operation - Type	$F{operation}
What - Extension - Type	$F{type}
What - Extension - Value	$F{value}

4.2. Audit Event API

The Audit Event API is used for reports in Audit analysis. The API references the related audit message and so it can contain all fields listed in the Raw API and reports can also contain subreports. The list below is not complete, but only illustrative.

ReportConfig.xml / API:

- -

Record data type:

- **solutions.dirx.audit.persistencev3.event.api.record.Identification**

Fields:

- **operation** : java.lang.String
- **type** : java.lang.String
- **detail** : java.lang.String
- **auditMessage** : solutions.dirx.audit.persistence.AuditMessage

Subreports:

- -

Column	Text Field Expression
When	$F{auditMessage}.getIdentificationWhen()
Source	$F{auditMessage}.getIdentificationSource()
Application	$F{auditMessage}.getWhereFromApplication()
Address	$F{auditMessage}.getWhereFromAddress()
Who	$F{auditMessage}.getWhoName()
Type	$F{auditMessage}.getIdentificationType()
Operation	$F{operation}
What Type	$F{type}
What Details	$F{detail}

5. Collecting Audit Messages from Third-party Applications

This chapter describes how to collect audit messages from a third-party application. To achieve this task:

- The third-party application needs to provide its audit messages in the DirX Audit format and send them to a message queue (recommended) or store them in files.
- A generic collector must be configured to read the messages from the queue or from the files.
- You need to implement and configure a digest producer (also called a digest generator) that extracts the digest of the message.
- You need to provide and configure a tag producer (also called a tag generator or a dimension generator) that calculates the desired tags for the audit message and associated events.
- If you provide additional tags, you might want to configure and populate dimension and fact tables to show your aggregated data in charts.

5.1. About the Audit Message Data Flow

The following figure illustrates the audit message data flow and its important components:

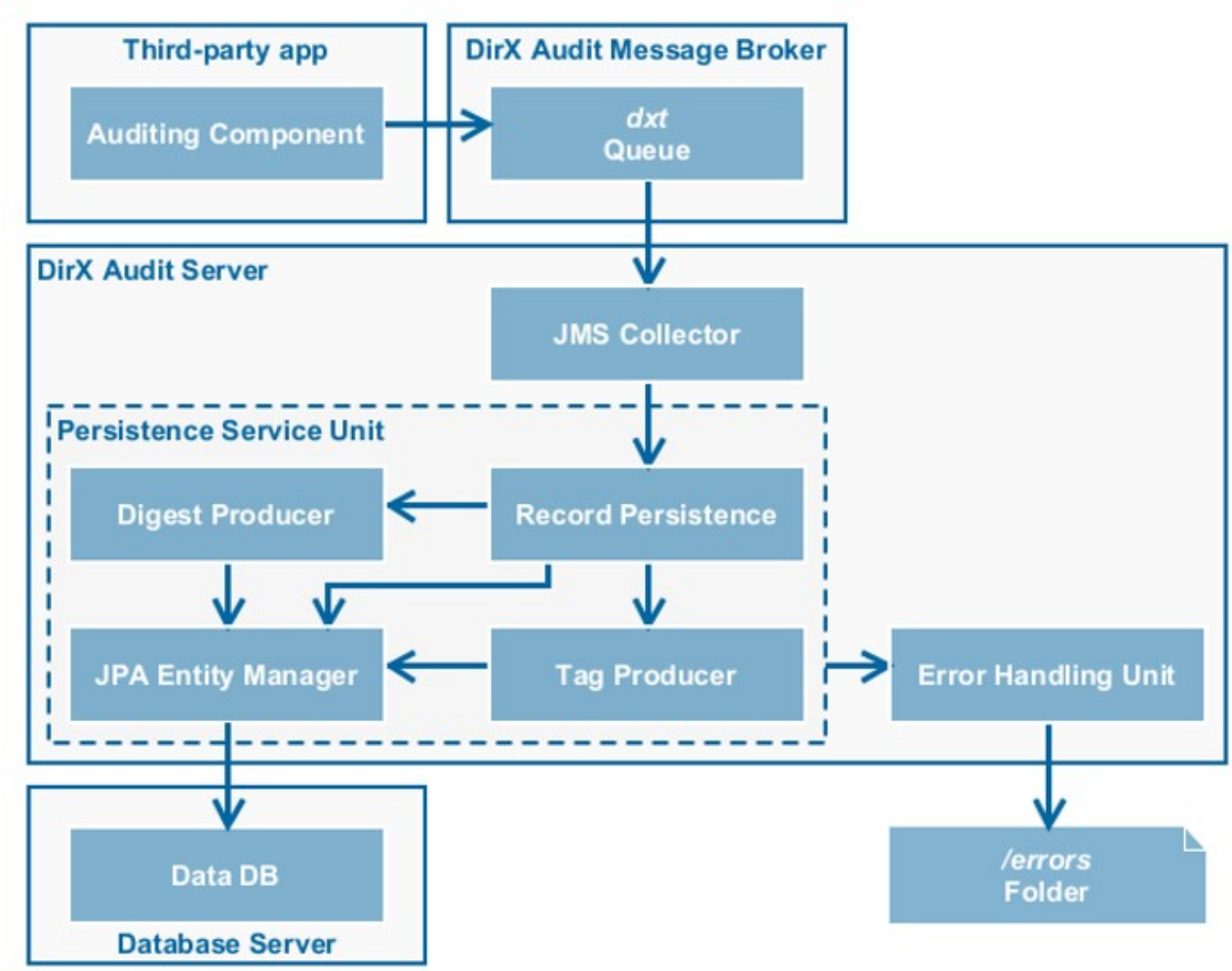

Figure 2. DirX Audit Message Data Flow

As shown in the figure, the application sends the audit messages to a queue (***dxt* Queue** in the figure). The generic JMS collector picks up the messages from the queue and then passes them to the Persistence Service Unit. The Persistence Service Unit first transforms the message from XML structure into the corresponding in-memory object. Next, it forwards the object to the digest producer and to the tag producer. Finally, it stores the whole data structure containing the object representing the audit message, digests and tags into the corresponding database tables.

If an error occurs, the message is passed to the Error Handling Unit, which stores it along with error information in the configured **/errors** folder.

5.2. Setting up the Third-party Application

The application must produce audit messages according to the XML namespace "urn:com:siemens:dxt:persistence:schemadb:2.0". You can find it in the schema file **AuditMessages.xsd** in the folder *install_path***/conf/schemas**.

The application can put multiple messages into one XML document; the root element to use is always <auditMessages>. Individual messages are then contained sequentially in the sub-elements named <auditMessage>. The application can send this XML document to the configured JMS queue or write it to files in the configured input folder for the file collector.

For sample messages, see the files in the folder **Additions/Data/SampleData/Access** on the installation media.

For examples on how to write a JMS producer for Apache ActiveMQ, see the JMS specification (for example, https://jcp.org/aboutJava/communityprocess/final/jsr914/index.html) and the Apache ActiveMQ documentation; for example, http://activemq.apache.org/hello-world.html.

In order to easily distinguish its messages from those of DirX Identity and DirX Access, the application should fill the attribute source of the <identification> element with an appropriate name.

5.3. Configuring the Generic Collector for Third-party Messages

DirX Audit provides two generic collectors: one for reading messages from a JMS message queue and one for reading messages from files. The message queue should be selected for a production environment; the file collector is mainly intended for testing purposes or initial load.

Both collectors expect the messages in the DirX Audit XML format. See the "Setting up the Third-party Application" for details.

For information on how to configure these collectors, see the section "Configuring Generic Collectors" in the chapter "Configuring DirX Audit Collectors" of the *DirX Audit Administration Guide*.

5.4. Implementing a Digest Producer

A digest producer – also called a digest generator – is responsible for generating one or more digests for an audit message. A digest producer is specific to the audit message provider.

A digest producer is a Java class and must implement the interface **solutions.dirx.audit.persistence.digest.api** with its only method **getDigests(AuditMessage auditMessage)**. The method accepts one audit message in its Java object (not XML) representation and returns a set of audit events, the digests.

A digest consists of the properties **operation**, **type** and **detail**. It is intended to give a summary of the audit message that can be understood by auditors that are not experts in the audited product. In addition to **Add Object**, **Update Object** or **Delete Object**, the **operation** can also be more high-level; for example, **Login**, **Set Password** or **Accept** and **Reject**. The **type** usually denotes the type of the affected object; this can be something like **User**, **Account** and **Role**, but can also denote relationships like **User to Role** or **Account to Group**. The **detail** of a digest depends on the operation and type; it typically contains the human-readable names of the affected objects; for example, the user's pseudonym, the login name of an account along with the target system display name (if available) or the name of a workflow and an approval activity.

Thread safety is not an issue for the digest producer because it is a Spring bean (see https://docs.spring.io/spring-framework/reference/core.html) and is instantiated by the Spring dependency injection framework for each message. This design permits configuring the Java class name along with any custom class properties in external XML configuration files.

To configure the digest producer, see the section "Configuring a Digest Producer" in the chapter "Customizing Digest and Tag Producers" in this guide.

To deploy the digest producer, see the section "Deploying Digest and Tag Producers" in this guide.

For a sample of a digest producer Java class, see the digester for Apache Tomcat in the folder **Additions/Data/SampleJava** on the installation media.

5.5. Implementing a Tag Producer

A tag producer – also called a tag generator or dimension generator – generates tags for an audit message and audit event. A tag consists of a key (type) – value pair.

The tags are evaluated for filling fact tables and are used as dimension values in charts. They can also be used to filter events in DirX Audit Manager's and DirX Audit Manager Classic's Audit analysis. Tag keys and values can be any string that seems appropriate for the event. For example, typical tag keys for DirX Identity are:

- APPLICATION (Target system) – to associate an audit event with a target system. The values are the display names of the target systems.
- WHO_OU (Who – Organizational unit) – to associate an audit message with the organizational unit of the active participant.

A tag key APPLICATION can be used, for example, in charts on password changes to slice the number of password changes per target system and drill through to the password change events for a selected target system. For the configuration of fact and dimension tables, see the chapter "Customizing Fact and Dimension Tables" in this guide.

A tag producer is a Java class that implements the interfaces **solutions.dirx.audit.persistence.dim.api.DigestDimensionProducer** for audit event dimensions and **solutions.dirx.audit.persistence.dim.api.MessageDimensionProducer** for audit message dimensions. For each of these interfaces, it must implement one method that accepts either the audit message or the audit event with the digest and returns a set of key – value pairs, the tags. Like the digest producer, a tag producer is a Spring bean instantiated for each message and event.

To configure a dimension generator, see the section "Adding a Tag Producer" in the chapter "Customizing Digest and Tag Producers" in this guide.

For a sample of a tag producer Java class, see the one for Apache Tomcat in the folder **Additions/Data/SampleJava** on the installation media.

To deploy a tag producer, see the section "Deploying Digest and Tag Producers" in this

guide.

5.6. Deploying Digest and Tag Producers

All of the digest and tag producers – along with a lot of other libraries – are deployed as separate jar files in DirX Audit Server deployment folders. Custom producers, especially those for a third-party collector, must be added to the correct server deploy folder.

Place the valid Java jar file with Java producer classes into one of the following folders depending on whether it is tenant-specific or shared for all tenants:

- The common server deployment folder, if they are shared by all configured tenants: *install_path***/server_container/core/BOOT-INF/lib/**
- The tenant-specific deployment folder: *install_path***/server_container/tenants/***tenantid***/deploy/**

Next:

- Restart DirX Audit Server.

6. Customizing Digest and Tag Producers

This chapter describes how to customize the DirX Audit digest producers (also called digest generators) and tag producers (also called dimension generators or tag generators).

DirX Audit supports the generation of digests and tags for the imported audit messages. A digest producer calculates a list of audit events for each audit message. At least one audit event should be produced for an audit message. A tag producer calculates a list of tags both for an audit message and its associated audit events. These lists can be empty.

Digest and tag producers are product-specific: They work on the same format – the audit message – but depend on the content. An additional digest producer and tag producer must be configured for each audit producer (DirX Identity, DirX Access, third party).

By default, the configuration is stored in the following file:

*install_path***/conf/event-dim-configuration/dgtDimConfig.xml**

The next sections describe:

- The data flow in the DirX Audit Persistence Service Unit
- How to configure the digest dimension generator
- How to configure a digest producer
- How to add a digest producer
- How to configure a tag producer
- How to add a tag producer

6.1. About Persistence Service Unit Data Flow

Digests and tags are normally generated in the DirX Audit Server during import of audit messages. Digest and tag producers are part of the DirX Audit Persistence Service Unit and are called after the transformation of the original format to the DirX Audit internal standard audit message. The following figure shows the important components in the Persistence Service Unit:

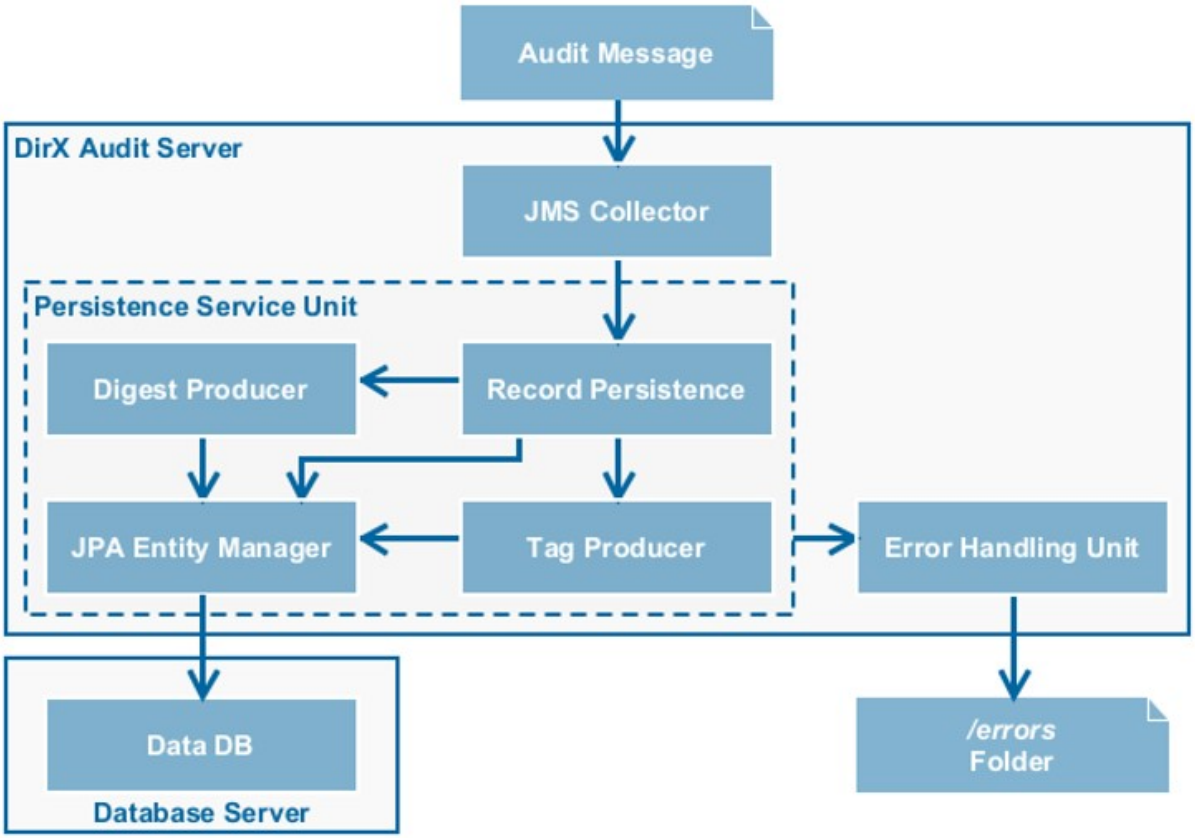

Figure 3. DirX Audit Persistence Service Unit Components

The Persistence Service Unit passes the transformed audit message to the Record Persistence component. First, it stores the audit message in the database (table: DAT_AUDITMESSAGES). Next, it calls the digest producer and next calls the tag producer for each message and stores each digest as an audit event (table: DAT_AUDITEVENTS) and the tags into the tag tables (TAG_*) of the database.

The Persistence Service Unit instantiates the digest and tag producers and leverages the Dependency Injection framework Spring (see https://spring.io/projects/spring-framework) for their flexible configuration as Spring beans.

6.2. Configuring Producers

Digest producers generate an audit event which is the digest or summary for an audit message. More than one event can be produced for an audit message. Tag producers generate a list of tags either for an audit event or for an audit message.

All of these producers are configured using Spring Bean configuration. The file is named **dgtDimConfig.xml** and located in the folder *install_path*/**conf/event-dim-configuration/**.

The structure of the file is as shown in the following snippet:

```
<beans>
  <bean id="DigestDimensionGenerator" … />
  <bean id="DxiDigestProducer" … />
  <bean id="DxiDigestDimensionProducer" … />
  <bean id="DxiEventDimensionProducer" … />
  <bean id="DxiStandardDimensions" … />
  <bean id="OuDimensionProducer" … />
  <bean id="AssignmentModeDimProducer" … />
  <bean id="AccessDigestProducer" … />
  <bean id="DxaEventDimensionProducer" … />
  <bean id="DxaStandardDimensions" … />
  <bean id="TomcatDigestProducer" … />
  <bean id="TomcatEventDimensionProducer" … />
  <bean id="TomcatStandardDimensions" … />
</beans>
```

The next sections describe these beans and their configurators.

6.2.1. Configuring the Digest Dimension Generator

The Persistence Service unit within the DirX Audit Server uses the Spring framework to instantiate the **DigestDimensionGenerator**, which is the bean that produces the digests and tags for all imported audit messages and events. It contains three maps with the bean references to digest, event dimension and digest dimension – tag – producers. These maps are indexed by the name of the product in lowercase that produces the audit message. The referenced generators are configured as separate <bean> elements. Here is the list of bean properties that hold these maps:

- digestProducers – a map with all digest (event) producers indexed by the audit producer name as contained in the audit message.
- digestDimensionProducers – a map with all tag producers for an audit event indexed by the audit producer name as contained in the audit message.
- eventDimensionProducers – a map with all tag producers for an audit message indexed by the audit producer name as contained in the audit message.

Here is the snippet with the default configuration for the bean **DigestDimensionGenerator**:

```
<bean id="DigestDimensionGenerator"
      class="….DigestDimensionGeneratorImpl"
      scope="prototype">
  <property name="digestProducers">
    <map>
      <entry key="dirx identity">
        <ref bean="DxiDigestProducer" />
      </entry>
      <entry key="dirx access">
        <ref bean="AccessDigestProducer" />
      </entry>
      <entry key="tomcat">
        <ref bean="TomcatDigestProducer" />
      </entry>
    </map>
  </property>
  <property name="digestDimensionProducers">
    <map>
      <entry key="dirx identity">
        <ref bean="DxiDigestDimensionProducer" />
      </entry>
    </map>
  </property>
  <property name="eventDimensionProducers">
    <map>
      <entry key="dirx identity">
        <ref bean="DxiEventDimensionProducer" />
      </entry>
      <entry key="dirx access">
        <ref bean="DxaEventDimensionProducer" />
      </entry>
      <entry key="tomcat">
        <ref bean="TomcatEventDimensionProducer" />
      </entry>
    </map>
  </property>
</bean>
```

6.2.2. Configuring a Digest Producer

You need to configure a digest (event) producer bean for each supported source product. For DirX Identity, this is the bean with the name **DxiDigestProducer**. For DirX Access, it is the bean **AccessDigestProducer**. You can simply stay with the default configuration that is shown in the next snippet for the DirX Identity digest producer:

```
<bean id="DxiDigestProducer"
  class="….DxiDigestProducerImpl"
  scope="prototype">
</bean>
```

The configuration of the digest producer for DirX Access is similar.

6.2.3. Adding a Digest Producer

To support a third-party audit producer, you need to add a digest producer into the configuration.

First, you implement the producer as a Java bean and then deploy it; see the section "Implementing a Digest Producer" in the chapter "Collecting Audit Messages from Third-party Applications" in this guide.

Next, add this bean to the <digestProducers> map of the digest dimension generator, for example:

```
<bean id="DigestDimensionGenerator">
  …
  <property name="digestProducers">
    <map>
      …
      <entry key="yourThirdPartyApp">
        <ref bean="{YourDigestProducer}" />
      </entry>
    </map>
  </property>
</bean>
```

The map contains a reference to the bean that configures your digest producer. The key value must match (case insensitive) the name that your application puts into the attribute source of the <identification> element of the audit message.

Finally, add the new bean into the configuration file with the ID you used in the map reference:

```
<bean id="YourDigestProducer"
  class="{fully.qualified.name.of.YourDigestProducerImpl}"
  scope="prototype">
</bean>
```

Note that you can add any property definitions into the bean; see the Spring framework definition for more details.

6.2.4. Configuring a Tag Producer

The message and event tag (dimension) producers for DirX Identity are composed of a list of producer beans that is easily extensible. The following snippet shows the configuration of the tag producer for DirX Identity audit messages:

```
<bean id="DxiEventDimensionProducer"
    class="….DxiEventDimensionProducer"
    scope="prototype">
  <property name="producers">
    <list>
      <ref bean="DxiStandardDimensions" />
      <ref bean="OuDimensionProducer" />
    </list>
  </property>
</bean>
```

By default, it supports a standard tag producer (named **DxiStandardDimensions**) and another producer that generates tags for organizational units associated with the active participant (who) or the object (what) of the message. The tags are named WHO_OU and WHAT_OU respectively. This producer looks for the identifying attribute of the active participant or the object that describes the organizational unit. As this attribute name depends on the configuration of the corresponding audit policy in DirX Identity, the property for the attribute name **identifierType** is configurable and must be adapted to the project settings:

```
<bean id="OuDimensionProducer"
      class="….OuDimensionProducer"
      scope="prototype">
  <property name="identifierType"
            value="Organizational Unit"/>
</bean>
```

For example, suppose you change the label in the DirX Identity audit policy to **OU**. As a result, you need to adapt the value in the bean configuration as follows:

```
    <property name="identifierType"
              value="OU"/>
```

6.2.5. Adding a Tag Producer

If you want to add a tag producer for generating additional tags or tags for a third-party application, you need to:

- Implement the producer as a Java class and deploy it; see the section "Implementing a Tag Producer" in the chapter "Collecting Audit Messages from Third-party Applications" in this guide.
- Configure a bean for your producer.
- Enter a reference to your bean into the appropriate tag producer.

First, configure the bean for your tag producer:

```
<bean id="YourTagProducer"
      class="{fully.qualified.name.of.YourTagProducerImpl}"
      scope="prototype">
</bean>
```

Use <property> child elements to configure any properties of your class; see the previous sections in this chapter for examples.

Next, add your tag producer bean to the appropriate product tag producer. For DirX Identity, this is either the DxiEventDimensionProducer for tags applicable to audit messages or DxiDigestDimensionProducer for tags applicable to digests ~ audit events.

The following example shows the insertion into the digest tag producer:

```
<bean id="DxiDigestDimensionProducer"
      class="….DxiDigestDimensionProducer"
      scope="prototype">
  <property name="producers">
    <list>
      <ref bean="DxiStandardDimensions"/>
      <ref bean="AssignmentModeDimProducer"/>
      <ref bean="{YourTagProducer}"/>
    </list>
  </property>
</bean>
```

If you want to add the tag producer for a third-party application, you need to add a reference to your bean into the map **digestDimensionProducers** or **eventDimensionProducers** of the **DigestDimensionGenerator** bean (see the section "Configuring the Digest Dimension Generator"). Here is an example for the **eventDimensionProducers** map:

```
<bean id="DigestDimensionGenerator">
  …
  <property name="digestDimensionProducers">
    <map>
      …
      <entry key="yourThirdPartyApp">
        <ref bean="{YourTagProducer}" />
      </entry>
    </map>
  </property>
</bean>
```

7. Customizing Fact and Dimension Tables

This chapter describes how to configure the OLAP tables with facts and dimensions that are the source for the Dashboard components in DirX Audit Manager Classic.

The default configuration of these tables is located in the following folder:

install_path/**conf/defaults/fact-configuration**

It is evaluated by all components that create or update facts and dimensions, including:

- The fact population job in the DirX Audit Server. It is responsible for regularly filling the tables. See the section "Managing Fact and Dimension Tables" in the *DirX Audit Administration Guide* for a description of its schedule.
- The command line tool **db_fact_population**. It is intended to calculate the facts and dimensions for previously collected or for re-imported audit messages. See the section "Populate Fact Tables" in the *DirX Audit Command Line Interface Guide* for a more detailed description.

Note that both components create the fact and dimension tables according to the configuration when necessary. But they don't delete existing columns from tables when they are removed from the configuration.

7.1. About Fact and Dimension Tables

The OLAP cubes are essentially fact tables with columns for facts and columns for dimensions.

The table names for facts and dimensions should follow these rules:

- Fact table names start with **FCT_**
- Dimension table names start with **DIM_**

The next sections describe fact and dimension tables for audit messages and history entries.

7.1.1. Fact and Dimension Tables for Audit Messages

A typical fact table for audit messages has the following items:

- Three fact columns. Most often these are FCT_TOTAL, FCT_SUCCEEDED and FCT_FAILED for the count of all, succeeded and failed events associated with the table. A calculated FCT_FAILED_RELATIVE can provide the relative number of failed events. For audit messages dealing with approvals, the facts FCT_APPROVED and FCT_REJECTED for the number of accepted and rejected approvals are used.
- The mandatory dimension DIM_DATETIME. It contains the day as timestamp with time 12am.

- A list of other dimension columns which are specific to the fact table. These columns frequently include the dimensions for the operation and the Where_From_Application. These dimension columns must follow the naming schema: *table_name*_**ID**.

The following diagram shows the columns of the fact table for audit messages related to accounts with the dimension tables for operation and application:

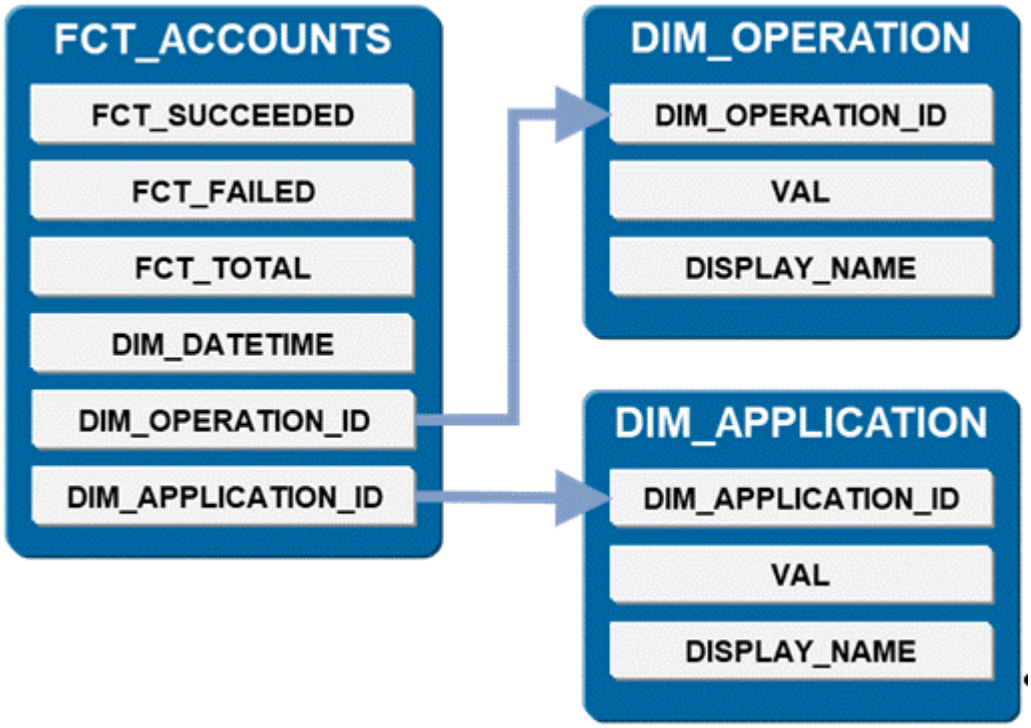

Figure 4. Fact and Dimension Tables Example

For more information, see the section “Configuring Fact Tables”.

A fact column contains numbers: the count of all (or all succeeded or failed) audit messages associated with that table and with the dimensions of that row. The dimension columns contain a foreign key reference into the corresponding dimension table.

All of the dimension tables have the same columns:

- VAL – contains a value for that dimension. It is restricted to a maximum of 255 characters.
- DISPLAY_NAME – by default, this column contains the same value as the VAL column. It is used when the dimension is displayed in DirX Audit Manager Classic and can be changed to a customer-specific value.
- *table_name*_**ID** – contains the primary key for the dimension value. It is used as the foreign key reference from the dimension columns in the fact tables. Note that this is the same column name as the one in the fact tables that contains the foreign key to this dimension table.

Note that due to restrictions for Oracle Database in previous versions, the maximum number of characters for a dimension table name is 23. The reason is that DirX Audit must use a trigger for generating the primary key in the dimension tables. The trigger name has the structure **GEN_***table name***_ID** and Oracle Database in previous versions supports trigger names with a maximum length of 30 characters.

A sample row for the table FCT_ACCOUNTS might have the following values:

FCT_SUCCEEDED: 9
FCT_FAILED: 0
FCT_TOTAL: 9
DIM_DATETIME: 2011-02-21 12:00:00.000
DIM_OPERATION_ID: 6
DIM_APPLICATION_ID: 3

The fact table contains only audit events related to accounts. The facts of this row correspond to audit events produced on February 21, 2011, which are associated to the operation with primary key 6 and application with primary key 3. There are 9 audit events matching these conditions. All of them were successful, none failed. Looking for the primary keys in the dimension tables, we might find the application (here: target system) “Intranet Portal” in the table DIM_APPLICATION and “Add Object” in the table DIM_OPERATION.

7.1.2. Fact and Dimension Tables for History Entries

Tables for fact and dimensions associated with history entries are much the same as those for audit messages. The tables for facts and dimensions for history entries are saved in the DirX Audit History Database.

The table names start with **FCT_HST_** for fact tables and **DIM_HST_** for dimension tables.

Fact tables on history entries have the following mandatory columns:

- The dimension columns DIM_DATETIME and DIM_MONTH. They contain the day as a timestamp with the time 12am. The column DIM_MONTH is empty for all days except the last day of the month or the current day.
- The only fact column FCT_HST_TOTAL. It contains the number of entries existing at the end of the day and matching the other dimensions. For history entries dealing with certifications, there are additional facts FCT_HST_CERTIFIED and FCT_HST_UNCERTIFIED for the number of certified and uncertified entries. For history entries dealing with users, there are additional facts FCT_HST_WOUT_ROLE, FCT_HST_WOUT_PERMISSION, FCT_HST_WOUT_GROUP and FCT_HST_WOUT_PRIVILEGE for the number of users without a role, a permission, a group and any privilege. For history entries dealing with approvals, there is an additional fact FCT_HST_DURATION_MI for the length of approval duration in minutes. The record with a non-empty DIM_MONTH shows the number of entries at the end of the month.

The fact table might contain additional dimension columns such as DIM_HST_ENTRY_TYPES for distinguishing the entry types User, Role, and so on, or dimensions DIM_HST_OU for distinguishing users according their organizational unit. The columns for such a dimension table are exactly the same as those for audit events.
The dimension DIM_HST_ENTRY_TYPES doesn't need an extra dimension table: the fact population generator just takes the dimension values from the existing table HST_ENTRY_TYPES.

7.1.3. Fact Table for Imported Memberships

The imported membership database table is called FCT_HST_IMPORTED_MEMBERSHIPS.

- The dimension columns DIM_HST_DATETIME and DIM_HST_MONTH contain the day as a timestamp with the time 12am. The column DIM_HST_MONTH is empty for all days except the last day of the month or the current day.
- DIM_HST_APPLICATION_ID is distinguishing applications where imported group memberships belong.
- The only fact column FCT_HST_TOTAL contains the number of entries existing at the end of the day and matching the other dimensions.

7.2. Configuring Fact Tables

The configuration for the fact tables is defined in the file **confFactTables.xml**. This file is an XML document that conforms to the XML schema with the namespace **https://factpopulation.persistence.audit.dirx.solutions/tables**. The root element <confFactTables> contains a number of child elements <confFactTable>. Each <confFactTable> describes one fact table and has the following child elements:

- <name> – the name of the fact table. The fact population component creates a table in the database with exactly that name. The `customDrilldown="true"` attribute and value indicates that customized result table of drill-through data will be shown. The `enabled="false"` attribute and value indicates that fact table will not be created and populated, by default it is set to `"true"`.
- <onRecordType> – the element is optional. For tables on history entries only, it must contain the value **HISTORY_ENTRY**.
- <description> – a string describing the content of the fact table.
- <common> – contains shared elements for dimension and tact tables. The `<ObjectType>` enumeration provides information, whether the fact or dimension is object in database and if it is table or view.
- <facts> – the list of <fact> child elements for the facts. The element <fact> contains the name of a fact and must correspond to a fact definition in the file **confFacts.xml**.
- <dimensions> – the list of <dimension> child elements for the dimensions. The element <dimension> contains the name of a dimension and must correspond to a dimension definition in the file **confDimensions.xml**.

- <query> – the filter conditions to identify the audit events covered in this fact table. The component transforms this element into a SQL select statement and collects only facts and dimensions for audit events matching these conditions. Alternatively for history entries, there are the <statements> element instead of the <query> element.
- <statements> – references to files containing SQL statements for creating the fact table, populating the fact table and drilling the data.

The sub-elements <query> and <onRecordType> are defined in the XML schema namespace **https://factpopulation.persistence.audit.dirx.solutions/query**.

The sub-element <statements> is defined in the XML schema namespace **https://factpopulation.persistence.audit.dirx.solutions.net/statements**.

The sub-element <common> is defined in the XML schema namespace **https://factpopulation.persistence.audit.dirx.solutions.net/common**.

The next sections describe fact tables on audit events and on history entries.

7.2.1. Fact Tables on Audit Events

Here is a sample snippet showing the structure of a fact table definition on audit events:

```
<t:confFactTable>
  <t:name>FCT_ACCOUNTS</t:name>
  <t:description>Account related operations</t:description>
  <t:facts>
    <t:fact>FCT_SUCCEEDED</t:fact>
      …
  </t:facts>
  <t:dimensions>
    <t:dimension>DIM_DATE_DAY</t:dimension>
    …
  </t:dimensions>
  <q:query>
    …
  </q:query>
  <s:statements>
    <s:create>
      <s:script path='data/fct/FCT_ACCOUNTS/create.sql'/>
    </s:create>
    <s:fill>
      <s:script
path='data/fct/FCT_ACCOUNTS/fill_01_FCT_ACCOUNTS_merge.sql'/>
    </s:fill>
  </s:statements>
</t:confFactTable>
```

The <query> element conforms to the XML schema **https://factpopulation.persistence.audit.dirx.solutions/query**. The important child elements of <query> are:

- <table> – the name of a table, which is related to audit events and must be one of the following tables:
  - DAT_AUDITEVENTS – this table contains the audit events. The columns that may be used for querying here are: OP (Operation), TYPE, DETAIL.
  - DAT_AUDITMESSAGES – this table contains the audit messages as transformed from the native audit trails. The columns most useful for fact identification are: IDENTIFICATION_SOURCE, IDENTIFICATION _OUTCOME, WHEREFROM_APPLICATION.
- <column> – the column name within the table defined in <table>.

- <operator> – the SQL comparison operator. Allowed values are: **=** (the default if not supplied), **!=**, **like**.
- <value> – the comparison value. Use the wildcard **%** (zero or more characters) for searching with the **like** operator.

<query> elements can be combined by the logical operators <and>, <or> and <not>. They can even be nested as in the following sample:

```
<q:query>
  <q:and>
    <q:or>
      <q:query>
        <table>DAT_AUDITEVENTS</table>
        <column>OP</column>
        <operator>like</operator>
        <value>Accept%</value>
      </q:query>
      <q:query>
        <table>DAT_AUDITEVENTS</table>
        <column>OP</column>
        <operator>like</operator>
        <value>Reject%</value>
      </q:query>
    </q:or>
    <q:query>
      <table>DAT_AUDITEVENTS</table>
      <column>TYPE</column>
      <operator>like</operator>
      <value>User to%</value>
    </q:query>
  </q:and>
</q:query>
```

This query searches for audit events that correspond to an approval operation: either Accept or Reject. It is not interested in all approvals, only those for User to Privilege assignments. See the delivered default configurations for more examples.

7.2.2. Fact Tables on History Entries

The following sample shows a fact table on history entries:

```
<t:confFactTable>
  <t:name>FCT_HST_USERS</t:name>
  <q:onRecordType>HISTORY_ENTRY</q:onRecordType>
  <t:description>History Users with states and organizational
units</t:description>
  <t:facts>
    <t:fact>FCT_HST_TOTAL</t:fact>
    <t:fact>FCT_HST_WOUT_ROLE</t:fact>
    <t:fact>FCT_HST_WOUT_PERMISSION</t:fact>
    <t:fact>FCT_HST_WOUT_GROUP</t:fact>
    <t:fact>FCT_HST_WOUT_PRIVILEGE</t:fact>
  </t:facts>
  <t:dimensions>
    <t:dimension>DIM_HST_DATETIME</t:dimension>
    <t:dimension>DIM_HST_MONTH</t:dimension>
    <t:dimension>DIM_HST_DXRSTATE</t:dimension>
    <t:dimension>DIM_HST_OU</t:dimension>
    <t:dimension>DIM_HST_O</t:dimension>
    <t:dimension>DIM_HST_L</t:dimension>
    <t:dimension>DIM_HST_DXRRSKLEVEL</t:dimension>
  </t:dimensions>
  <s:statements>
    <s:create>
      <s:script path='history/fct/FCT_HST_USERS/create.sql'/>
    </s:create>
    <s:fill>
      <s:script
path='history/fct/FCT_HST_USERS/fill_01_FCT_HST_USERS_TMP_delete.sql'
/>
       …
    </s:fill>
    <s:drilldown>
      <s:script path='history/fct/FCT_HST_USERS/drilldown.sql'/>
    </s:drilldown>
  </s:statements>
</t:confFactTable>
```

The table FCT_HST_USERS is on users and distinguishes according the dimensions state, organizational unit, organization and locality.

The fact tables are created and populated with SQL scripts executed by schedule and stored in the *install_path*\conf\defaults\sql\common\factpopulation folder. It also contains drilldown statements to supporting drill-through from aggregated data to individual records.

7.3. Configuring Facts

The configuration for the facts is defined in the file **confFacts.xml**. This file is an XML document that conforms to the XML schema with the namespace https://factpopulation.persistence.audit.dirx.solutions/facts. The root element <confFacts> contains a number of child elements <confFact>. Each <confFact> describes one fact and has the following child elements:

- <name> – the name of the fact. The fact population tool creates a column with that name in every fact table that references this fact. This name must match the <fact> names used in fact table configurations.
- <description> – a string describing the fact.

  The fact population tool transforms the following elements into a SQL select statement and counts only audit events matching these conditions. The elements describe a query similar to the one used in the fact table configurations.

Note that in most cases the only supported fact on history entries is FCT_HST_TOTAL. It counts the records in the table HST_ENTRIES separated by their primary key in HST_ENTRIES_ID.

Here is a sample definition for the fact for all successful operations:

```
<f:confFact>
  <f:name>FCT_SUCCEEDED</f:name>
  <f:description>Successful Operations</f:description>
</f:confFact>
```

There is also a virtual relative fact called FCT_FAILED_RELATIVE, which is a representation of the failed entries as a relative part of total entries. This fact is computed based on the FCT_FAILED and FCT_TOTAL facts.

```
<f:confFact
  virtual="true"
  relative="true" >
  <f:name>FCT_FAILED_RELATIVE</f:name>
  <f:description>Failed Operations</f:description>
  <f:display-sql>100.0*sum(FCT_FAILED)/sum(FCT_TOTAL)</f:display-sql>
  <f:additional-sql>sum(FCT_FAILED)</f:additional-sql>
</f:confFact>
```

7.4. Configuring Dimensions

The configuration for the dimensions is defined in the file **confDimensions.xml**. This file is an XML document that conforms to the XML schema with the namespace https://factpopulation.persistence.audit.dirx.solutions/dimensions. The root element <confDimensions> contains a number of child elements <confDimension> or <confDimensionVirtual>. Each <confDimension> describes one dimension based on the database view or table and is created with SQL DDL scripts and has the following child elements:

- <name> – the name of the dimension. The fact population tool creates a table with this name. Exceptions are the DATE dimensions, which do not need a table. This name must match the <dimension> names used in fact table configurations. The `enabled="false"` attribute and value indicates that dimension will not be created and populated, by default it is set to `"true"`. Note, that setting `enabled="false"` to dimension could also disable fact table(s), which are dependend on this dimension.
- <onRecordType> – the element is optional. For tables on history entries only, it must contain the value **HISTORY_ENTRY**.
- <description> – a string describing the dimension.
- <table>: the name of a table, which is related to audit messages. You can use the tables TAG_EVENT_DIMENSIONS and TAG_MESSAGE_DIMENSIONS for dimension filtering in addition to the tables allowed for dimension tables (DAT_AUDITEVENTS, DAT_AUDITMESSAGES).
- <column> – the column name within the table defined in <table>.
- <common> – contains shared elements for dimension and tact tables. The <ObjectType> enumeration provides information, whether the fact or dimension is object in database and if it is table or view.
- <statements> – references to files containing SQL statements for creating and populating the dimension table or database view.

- <query> – the filter conditions to identify the dimension. The fact population tool transforms this element into a SQL select statement and uses the resulting list of distinct values for grouping the facts.

The following child elements describe a query similar to the one used in the fact and fact table configurations:

- <table> – the name of a table. In addition to the tables allowed for facts (DAT_AUDITEVENTS, DAT_AUDITMESSAGES, TAG_EVENT_DIMENSIONS, TAG_MESSAGE_DIMENSIONS) you can use the following tables for dimension filtering:
  - TAG_EVENT_DIMENSION_TYPES – this table contains the distinct tag names for audit events, not their values. The only column that may be used for querying is "NAME".
  - TAG_ MESSAGE _DIMENSION_TYPES – this table contains the distinct tag names for audit messages, not their values. The only column that may be used for querying is "NAME".
    Note that for dimensions on history entries only, the tables **HST*** can be used. They are listed in the section “Fact Tables on History Entries”.
- <column> – the column name within the table defined in <table>.
- <operator> – the SQL comparison operator. Allowed values are: = (the default if not supplied), **!=**, **like**.
- <value> – the comparison value. Use the wildcard **%** (zero or more characters) for searching with the **like** operator. If the value is not supplied, all distinct values of the table column are considered.

Note that in the same way as fact tables, dimensions are created and populated with SQL scripts executed by scheduled jobs and stored in the *install_path***conf\defaults\sql\common\factpopulation** folder.

Note that the file also contains a couple of virtual dimensions <confDimensionVirtual>. Please do not change them! DirX Audit Manager Classic evaluates them for grouping the charts by days, weeks, months and years.

7.5. Adding and Disabling a Fact Table

To create a new fact table, add an element <confFactTable> into the file **confFactTables.xml**. See the section “Configuring Fact Tables” for information about the format.

When the fact table population is executed next time, the table is created. The fact population job in the DirX Audit Server automatically updates the table according to the configured schedule.

To disable a fact table, set the attribute `enabled="false"` of an existing element <confFactTable>. See the section “Configuring Fact Tables” for information about the format.

When the fact table population is executed next time, the table will not be created and

updated.

7.6. Adding a Fact

To create a new fact, add an element <confFact> into the file **confFacts.xml**. See the section "Configuring Facts" for information about the format.

Enter the new fact as element <fact> into an existing or a new fact table. See the previous sections for details.

Adapt your database schema accordingly.

When the fact table population is executed next time, the new column is populated too. The fact population job in the DirX Audit Server automatically updates the table according to the configured schedule.

7.7. Adding and Disabling a Dimension

To create a new dimension, add an element <confDimension> into the file **confDimensions.xml**. See the section "Configuring Dimensions" for a description of the format.

Enter the new dimension as the element <dimension> into a fact table. See the previous sections for details.

Adapt your database schema accordingly.

When the fact table population is executed next time, the new dimension table and the new dimension column are populated too. The fact population job in the DirX Audit Server automatically updates the table according to the configured schedule.

To disable a dimension, set the attribute `enabled="false"` of an existing element <confDimension>. See the section "Configuring Dimensions" for information about the format.

When the fact table population is executed next time, the dimension table will not be created and updated.

Note, that setting `enabled="false"` to dimension could also disable fact table(s), which are dependend on this dimension.

7.8. Configuring Custom Fact Tables

If you want to customize fact table configuration, it is possible to do that for all tenants or for a specific tenant. Copy the common core default configuration from the folder:

*install_path***/conf/defaults/fact-configuration**

into the core custom configuration folder for all tenants in:

*install_path***/conf/fact-configuration**

and/or into the tenant custom configuration folder in:

*install_path***/conf/tenants/***tenantID***/fact-configuration**

where *tenantID* is a unique tenant identifier. Now you can customize configuration files for all and/or a specific tenant by following the instructions given in the previous sections of this chapter.

Remove all <confFactTable> elements from the customized **confFactTables.xml** file and keep only those you want to modify, disable, or add a new one as described in previous chapters.

Remove all <confFact> elements from the customized **confFacts.xml** file and keep only those you want to modify or add a new one as described in previous chapters.

Remove all <confDimension> and <confDimensionVirtual> elements from the customized **confDimensions.xml** file and keep only those you want to modify, disable, or add a new one as described in previous chapters.

Components that create or update fact tables, facts and dimensions will implicitly use the core default configuration. If configuration folder for all tenants is available, its configuration will be merged, same applies for tenant specific customization.

For example, table FCT_HST_USERS is defined on core default level in **confFactTables.xml** file with specific elements and values. If the table is defined/customized on core custom level, its elements and values will be overwritten. And if the table is also defined on tenant custom level, its elements and values will be overwritten again.

To easily identify, which fact tables, dimensions or facts are customized, it is recommended to only include in customized configuration files those elements, which you want to add, modify, or disable.

7.9. Configuring SQL scripts

If you want to customize SQL scripts for creating, filling, or drilling data, it is possible to do that for all tenants or for a specific tenant. Follow the instructions described in the General Customization chapter. Examples are available in chapter Customizing SQL scripts.

8. Customizing History Synchronization

There are four basic methods for customizing DirX Audit History Synchronization:

- Modifying, adding, and removing entry types from History Synchronization
- Customizing attribute mapping
- Excluding attributes from synchronization
- Explicitly specifying attributes for synchronization

For more information, see the section "Configuring and Customizing DirX Audit History Synchronization Jobs" in the *DirX Audit History Synchronization Guide*.

9. Customizing Dashboard View Chart Colors

Dashboard components in DirX Audit Manager Classic present fact and dimension values according to selected Color scheme. This chapter describes how to configure specific colors for specific fact and dimension values. It is required to have basic knowledge of JSON files.

This chapter describes customizable JSON files and how to:

- Configure a new color scheme
- Setup tenant specific JSON files
- Configure custom colors
- Configure colors for fact and dimension values
- Reload JSON files and clear chart coloring cache

It also contains a reference to the *DirX Audit Manager Classic Guide* and sample JSON files.

9.1. Default Chart Coloring JSON Files

The default chart coloring JSON files are common for all tenants and are located in the following *install_path*/**conf/colors** folder.

- **color_scheme.json** – contains a predefined color schemes for the component display; for example **Ocean** or **Forest**.
- **chart_colors.json** – contains predefined configuration of colors to be used in Dashboard view charts. Colors depend on specified dashboard, chart, dimension or fact table and their value.
- **colors.json** – contains customizable colors and their name to be reused in **chart_colors.json**

9.1.1. Configuring a New Color Scheme

Before modifying *install_path*/**conf/colors/color_scheme.json** make sure to back up the original file.

To add a new color scheme, open *install_path*/**conf/colors/color_scheme.json** in any text editor. Easiest way to add a new color scheme is to copy an existing JSON node and modify in accordingly.
In this tutorial, we will copy **Traffic lights** color scheme.

Node to be copied and modified:

```
…
{
  "schemeName": "TRAFFIC_LIGHTS",
  "label": "Dashboard.widget.colorScheme.TRAFFIC_LIGHTS",
  "description": "",
  "disabled": false,
  "defaultLabel": "Traffic lights",
  "colors": [
    "#00FF00",
    "#FFF200",
    "#FF0000"
  ]
},
…
```

Each color scheme JSON node contains these key-pair values:

- **schemeName** – name of a color scheme
- **label** – localization label
- **description** – optional description
- **disabled** – boolean value
- **defaultLabel** – label to be used, when **label** is missing or cannot be found in properties file
- **colors** – ordered array of colors in hexadecimal format, for more possible color input, check “Configuring Custom Colors” chapter

Modify the copied node as:

```
...
{
  "schemeName": "NEW_COLOR_SCHEME",
  "label": "Dashboard.widget.colorScheme.NEW_COLOR_SCHEME",
  "description": "",
  "disabled": false,
  "defaultLabel": "New color scheme",
  "colors": [
    "#FF00FF",
    "#00FFFF",
    "#FFFF00"
  ]
},
...
```

Save the modified file. Optionally, you can use any JSON validator to make sure the modified file is valid. Modifications on this file have immediate effect in DirX Audit Manager Classic application.

All existing color schemes are localized using keys starting with the `Dashboard.widget.colorScheme` prefix from the *install_path*/**conf/i18n/messages.properties** localization bundle file. When you want to localize a new or custom color scheme label, you need to add a new localization key and an appropriate value (localized color scheme) to the localization bundle file. Note that when you add a new key to the localization bundle file, you must restart the Apache Tomcat service (DirX Audit Manager Classic container) for the modification to take effect.

9.2. Tenant Specific Chart Coloring JSON Files

Modifications and customizations in the following chapters could be also applied to default JSON files located at *install_path*/**conf/colors**. Modifications in default JSON files will be applied to all tenants. It is **strongly** recommended to keep backup of original JSON files, if you want to modify them.

Customizations in tenants specific JSON files have higher priority than defaults stored at *install_path*/**conf/colors**. Overall, always the most specific color will be used to represent value in a chart.

9.2.1. Setting Up Tenant Specific JSON Files

If you want to assign specific colors for specific facts and dimension in a specific tenant, it is necessary to create a folder that will contain customizable JSON files.

Copy folder **chart_coloring_template** containing two JSON files located at *install_path*/**conf/colors** to *install_path*/**conf/tenants/***tenantID*/, where *tenantID* represents identification of the tenant you want to customize. Then rename the **chart_coloring_template** folder to **chart_coloring**. Modifications done to **chart_colors.json** and **colors.json** files in this location will be relevant only for the specific tenant.

It is also possible to copy and accordingly modify sample JSON files located at: *install_media*/**Additions/Data/SampleChartColors/chart_coloring**.

After this step, there should be a new folder in your installation path: *install_path*/**conf/tenants/***tenantID*/**chart_coloring** containing **chart_colors.json** and **colors.json**.

9.2.2. Configuring Custom Colors

It is possible to configure new custom colors that may be reused multiple times without specifying the color using hexadecimal format `#00FF00`, RGB format `(255, 0, 0)` or java.awt.Color `Color.BLUE` constructor. Therefore, in case of any required changes to the customized color, it will not be necessary to modify it at multiple places. File to modify: *install_path*/**conf/tenants/***tenantID*/**chart_coloring/colors.json**.

Example JSON node holding three custom colors:

```
…
"customColors": {
  "Custom.My-Company_RED": "(255, 0, 0)",
  "Custom.My-Company_GREEN": "#00FF00",
  "Custom.My-Company_BLUE": "Color.BLUE"
},
…
```

It is necessary to use **`Custom.`** prefix for naming customized colors. These customized colors can be used when setting specific fact and dimension colors in the next chapter.

For testing purposes, modified JSON file might have this content:

```
{
  "version": 1,
  "data": {
    "customColors": {
      "Custom.TEST_RED": "(255, 0, 0)"
    }
  }
}
```

The configured color `Custom.TEST_RED` will be used in the following chapter, it represents RGB color value of `(255, 0, 0)`, which is a red color.

9.2.3. Configuring Colors for Fact and Dimension Values

To configure colors for fact and dimension values, it is necessary to modify file located at: *install_path*/**conf/tenants/***tenantID*/**chart_coloring/chart_colors.json**.

The JSON file contains two arrays, `dimensions` and `facts`.
Common key-pair values in their nodes are:

- **dashboardSpecific** – array of dashboard names which these predefined colors will be used in, currently not supported and should be left as an empty array
- **chartSpecific** – array of chart names which these predefined colors will be used in
- **tableSpecific** – specific fact table, for example FCT_HST_USERS or dimension table, for example DIM_HST_OU in which these predefined colors will be used. If the value is set to “default”, colors will be used for all fact or dimension tables
- **valueColor** – node containing values and their assigned colors

Example JSON node holding dimension colors for specific chart, dimension table and values:

```
...
"dimensions": [
  {
    "dashboardSpecific": [],
    "chartSpecific": [
      "hst__dxi__users__total__datemonth_dxrstate"
    ],
    "tableSpecific": "DIM_HST_OU",
    "valueColor": {
      "Product Testing": "Custom.My-Company_RED",
      "Finances": "Custom.My-Company_GREEN",
      "Information Technology": "Custom.My-Company_BLUE"
    }
  },
]
...
```

Example JSON node holding fact colors that will be applied to all fact tables:

```
…
"facts": [
  {
    "dashboardSpecific": [],
    "chartSpecific": [],
    "tableSpecific": "default",
    "valueColor": {
      "FCT_TOTAL": "Color.BLUE",
      "FCT_SUCCEEDED": "#00FF00",
      "FCT_FAILED": "(255, 0, 0)"
    }
  },
]
…
```

For testing purposes, modified JSON file might have this content:

```
{
  "version": 1,
  "data": {
    "dimensions": [
      {
        "dashboardSpecific": [],
        "chartSpecific": [],
        "tableSpecific": "DIM_HST_OU",
        "valueColor": {
          "Information Technology": "Custom.TEST_RED"
        }
      },
      {
        "dashboardSpecific": [],
        "chartSpecific": [],
        "tableSpecific": "default",
        "valueColor": {
          "No value": "(0, 0, 0)"
        }
      }
    ],
```

```
    "facts": [
      {
        "dashboardSpecific": [],
        "chartSpecific": [],
        "tableSpecific": "FCT_HST_USERS",
        "valueColor": {
          "FCT_HST_TOTAL": "Color.YELLOW"
        }
      },
      {
        "dashboardSpecific": [],
        "chartSpecific": [],
        "tableSpecific": "default",
        "valueColor": {
          "FCT_HST_TOTAL": "Color.GREEN"
        }
      }
    ]
  }
}
```

In dimensions node, there are colors set for dimension table “DIM_HST_OU”. The value `Information Technology` will be represented by red color, as in previous chapter `Custom.TEST_RED` was configured as such. Next node sets color of value `No value` to black color, it will be applied to all dimension tables, as `tableSpecific` attribute is set to `default`.

In facts node, fact `FCT_HST_TOTAL` in fact table “FCT_HST_USERS” should be represented by yellow color. Fact `FCT_HST_TOTAL` should be represented by green color in all other fact tables, as `tableSpecific` attribute is set to `default`.

See the chapter “Dashboard Data in DirX Audit Manager Classic” in the *DirX Audit Administration Guide* for more information about facts, dimensions, fact tables and components.

To test changes after modifying JSON files, you can either logout and login back to the DirX Audit Manager Classic application or use the refresh button in top right corner in the Dashboard view tab. The refresh button will reload JSON files and clear cached colors. For details on how to use the Dashboard view, see the section “Using the Dashboard View” in the *DirX Audit Manager Classic Guide*.

In the Dashboard view, add component **Risk users based on overall risk by month and risk level** and open component settings. The **Fact table** should be set as `Users (HST)`, **Chart class** should be `1 fact 2 dimensions`. Set second dimensions from `Risk level` to `Organizational unit` and press **OK**. Value **Information technology** should be represented by red color and **No value** by black color as is set in the **chart_colors.json** file.

To test colors of facts, open component settings and change the **Chart class** to `1 fact 1 dimension` and make sure that `Total` is selected as a **Facts**, proceed with **OK**. Total number of users should be represented by yellow color, as is set specifically for “FCT_HST_USERS” fact table.

To check default color of “FCT_HST_TOTAL” fact, add component **DirX Identity total history account entries by month and state**, open component settings, **Fact table** should be set as `Accounts (HST)`, change **Chart class** to `1 fact 1 dimension` and **Facts** to `Total`. After pressing **OK** total number of accounts should be represented by green color.

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

EVIDEN

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.