Oracle® Endeca Information Discovery

Studio Developer's Guide

Version 2.4.0 • November 2012

ORACLE

Copyright and disclaimer

Copyright © 2003, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners. UNIX is a registered trademark of The Open Group.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Table of Contents

Preface

Oracle® Endeca Information Discovery Studio is an enterprise data discovery platform for advanced, yet intuitive, exploration and analysis of complex and varied data.

Information is loaded from disparate source systems and stored in a faceted data model that dynamically supports changing data. This integrated and enriched data is made available for search, discovery, and analysis via interactive and configurable applications. Oracle Information Discovery Studio includes a Provisioning Service that allows you to upload data directly from spreadsheet files.

Oracle Endeca Information Discovery Studio enables an iterative “model-as-you-go” approach that simultaneously frees IT from the burdens of traditional data modeling and supports the broad exploration and analysis needs of business users.

About this guide

This guide provides information on extending Studio to use customized versions of the Studio SecurityManager and StateManager. It also provides information on using Studio's Component SDK.

Who should use this guide

This guide is intended for developers who want to extend Studio.

Contacting Oracle Customer Support

Oracle Customer Support provides registered users with important information regarding Oracle software, implementation questions, product and solution help, as well as overall news and updates from Oracle.

You can contact Oracle Customer Support through Oracle's Support portal, My Oracle Support at *https://support.oracle.com*.

Chapter 1

Security Extensions to Studio

You may require more than the default data source role-based security discussed in the *Oracle Endeca Information Discovery Studio User's Guide*. If so, you can customize the automated filtering of data from the Oracle Endeca Server (based on user profile details such as the user's role or group association) by creating a custom Security Manager.

Security Manager class summary

A Security Manager is a concrete class that implements `com.endeca.portal.data.security.MDEXSecurityManager`.

Abstract base class	`com.endeca.portal.data.security.MDEXSecurityManager`
Default implementation class	`com.endeca.portal.data.DefaultMDEXSecurityManager`
Description	Handles pre-execution query modification based on the user, role, or group-based security configuration of filters.

Default implementation behavior	The default Security Manager implementation uses the following properties: • `securityEnabled`. If the value is not present, then `securityEnabled` defaults to `false`. • `securityFilters`. Record filters are the only supported type of `securityFilter`. • `rolePermissions` • `inheritSecurity`. If the data source has a parent, then `inheritSecurity` defaults to `true`. Otherwise, the value defaults to `false`. • `parentDataSource` These properties are defined in data source configurations in order to apply role-based security filters to queries issued to the Endeca Server backing a given data source. Users are assigned to Studio roles in the **Control Panel**. The related associations are made available to every component throughout the user's session. Users who have not yet logged in are automatically assigned the Guest user role. Any role-based restrictions for the Guest role are also applied to these users. For each data source, the Security Manager maintains an internal map of security filters to always apply to queries issued during that user's session.

Creating a new Security Manager

The Studio Component SDK includes Windows and Linux batch scripts for creating a new Security Manager.

To create a new Security Manager project:

1. In a terminal, change your directory to `endeca-extensions` within the Component SDK's root directory (normally called `components`).
2. Run one of the following commands:
   - On Windows: `.\create-mdexsecuritymanager.bat <your-security-manager-name>`
   - On Linux: `./create-mdexsecuritymanager.sh <your-security-manager-name>`

   This command creates a `your-security-manager-name` directory under `endeca-extensions`. This directory is an Eclipse project that you can import directly into Eclipse, if you use Eclipse as your IDE.

   This directory also contains a sample implementation that you can use to help understand how the Security Manager can be used. The sample implementation is essentially identical to the default implementation of the Security Manager used by Studio.

Implementing a new Security Manager

Your Security Manager must implement the `applySecurity` method.

There are two versions of the `applySecurity` method, one of which your Security Manager must implement:

```
public void applySecurity(PortletRequest request, MDEXState mdexState, Query query) throws
MDEXSecurityException;
```

The Query class in this signature is `com.endeca.portal.data.Query`. This class provides a simple wrapper around an `ENEQuery`.

Using the Security Manager

In order to use your Security Manager, you must specify a new class for Studio to use in place of the default Security Manager implementation.

The `your-security-manager-name` directory you created contains an ant build file. The `ant deploy` task places a .jar file containing your Security Manager into the `portal/tomcat-<version>/lib/ext` directory.

To configure Studio to use your new class:

1. From the Studio menu, select **Control Panel**.
2. In the **Information Discovery** section of the **Control Panel** menu, select **Framework Settings**.
3. Change the value of the `df.mdexSecurityManager` property to the full name of your class, similar to following example:
   ```
   df.mdexSecurityManager = com.endeca.portal.extensions.YourSecurityManagerClass
   ```
4. Click **Update Settings**.
5. Restart Studio so the change can take effect. You may also need to clear any cached user sessions.

Chapter 2

Managing Data Source State in Studio

Studio allows you to define your own interaction model for data sources by creating a custom State Manager. For information on the default interaction model between related data sources, see the *Oracle Endeca Information Discovery Studio User's Guide*.

About the State Manager interface

The State Manager controls how data sources interact during updates and query construction.

Interface (required)	`com.endeca.portal.data.MDEXStateManager`
Abstract base class (optional)	`com.endeca.portal.data.AbstractMDEXStateManager`
Default implementation class	`com.endeca.portal.data.DefaultMDEXStateManager`
Description	Handles: • Updating a data source with a new query state (called from `DataSource.setQueryState(QueryState newState)`) • Retrieving the current query state from a data source (called from `DataSource.getQueryState()`) • Resetting a data source's query state to its initial state (called from `DataSource.resetQueryState()`) • Retrieving a copy of the data source's initial state without resetting the data source (called from `DataSource.getInitialQueryState()`)

Default implementation behavior	The default State Manager implementation uses the `ParentDataSource` property from the data source configuration to propagate state changes throughout the hierarchy of data source relationships. When a component changes the query state of its data source, that modification is applied to: • The parent data source • All of the children of the parent data source This is recursive, applying all the way up and back down an ancestor tree. Configuring a hierarchy of data source relationships allows application developers to create more advanced interfaces, such as a tabbed result set where a single **Guided Navigation** component controls the query state for **Results Table** components on different tabs.

Creating a new State Manager

The `endeca-extensions` directory of the Component SDK includes scripts for creating a State Manager project on either Windows or Linux.

To create a new State Manager project:

1. In a terminal, change to the `endeca-extensions` directory within the Component SDK's root directory (normally called `components`).
2. Run one of the following commands:
   - On Windows: `.\create-mdexstatemanager.bat` <*`your-state-manager-name`*>
   - On Linux: `./create-mdexstatemanager.sh` <*`your-state-manager-name`*>

   This command creates a <*`your-state-manager-name`*> directory under `endeca-extensions`. This directory is an Eclipse project. If you use Eclipse as your IDE, you can import the project directly into Eclipse.

   The directory also contains a sample implementation, which is essentially identical to the default implementation of the State Manager used by Studio. You can use this sample implementation to help understand how to use the State Manager.

Implementing a State Manager

Custom State Managers implement the `MDEXStateManager` interface. There are methods for updating, retrieving, and resetting the data source query state.

Recommendations for implementing

To create a custom State Manager, you must at minimum implement the `com.endeca.portal.data.MDEXStateManager` interface. The recommended approach is to extend `com.endeca.portal.data.AbstractMDEXStateManager`, which in turn implements `MDEXStateManager`.

You also should extend `com.endeca.portal.data.AbstractMDEXStateManager`, which in turn implements `MDEXStateManager`. The `AbstractMDEXStateManager` abstract class contains the useful utility method `addEventTrigger(PortletRequest, MDEXState)`.

The default state manager implementation is `com.endeca.portal.data.DefaultMDEXStateManager`. The Studio Component SDK creates state managers that extend `DefaultMDEXStateManager`, because they will work without any modification. If you want your custom state manager to inherit some of the default functionality, you can extend `DefaultMDEXStateManager` instead of `AbstractMDEXStateManager`.

Required methods

Your State Manager must implement the following methods:

```
public void handleStateUpdate(PortletRequest request, MDEXState mdexState, QueryState newQueryState)
throws QueryStateException;

public QueryState handleStateMerge(PortletRequest request, MDEXState mdexState) throws
QueryStateException;

public void handleStateReset(PortletRequest request, MDEXState mdexState) throws QueryStateException;

public QueryState handleStateInitial(PortletRequest request, MDEXState mdexState) throws
QueryStateException;
```

`handleStateUpdate()`	Called when a component calls `DataSource.setQueryState(qs)`. This method should eventually call `mdexState.setQueryState()`. However, it is not required to make this call if it determines that the `MDEXState's QueryState` should not change. If the data source state is changed by `handleStateUpdate()`, you must mark the affected data sources. To mark the data sources, you call the `addEventTrigger(PortletRequest request, MDEXState ds)` method, passing in the request object and any `MDEXState` objects that are changed.
`handleStateMerge()`	Called when a component calls `DataSource.getQueryState()`. You are expected to return the `QueryState` that the component should get access to for the data source represented by the `mdexState`, taking into account any data source relationships or other aspects of your `State Manager` that might affect the query state.

`handleStateReset()`	Called when a component calls `DataSource.resetQueryState()`. This method returns the data source to the "initial state" defined by your state manager. The default implementation (`DefaultMDEXStateManager`) clears all query functions from the data source except those defined in the `baseFunctions` key of the data source's .json file, and similarly updates all parent and child data sources. If the data source state changes while it is being reset, you must mark the affected data sources. To mark the data sources, you call the `addEventTrigger(PortletRequest request, MDEXState ds)` method, passing in the request object and any `MDEXState` objects that are changed.
`handleStateInitial()`	Called when a component calls `DataSource.getInitialQueryState()`. This method returns a copy of the data source's initial state as defined by your state manager. The default implementation (`DefaultMDEXStateManager`) returns a `QueryState` with query functions made up of the union of the `baseFunctions` from: • The current data source • All of the current data source's parents

Using the State Manager

In order to use your State Manager, you must specify a new class for Studio to use in place of the default State Manager implementation.

The *<your-state-manager-name>* directory you created contains an ant build file. The `ant deploy` task places a `.jar` file containing your State Manager into the `portal/tomcat-<version>/lib/ext` directory.

To configure Studio to use your State Manager:

1. From the Studio menu, select **Control Panel**.
2. In the **Information Discovery** section of the **Control Panel** menu, select **Framework Settings**.
3. Change the value of `df.mdexStateManager` property to the full name of your class, similar to following example:
   ```
   df.mdexStateManager = com.endeca.portal.extensions.YourStateManagerClass
   ```
4. Click **Update Settings**.
5. Restart Studio so the change can take effect. You may also need to clear any cached user sessions.

Chapter 3

Installing and Using the Component SDK

The Studio Component SDK is a packaged development environment that you can use to add or modify components, themes, and layout templates.

Note: The Studio Component SDK is designed to work with the Studio Tomcat bundle. It will not work out-of-the-box on other platforms.

Software and licensing requirements for component development

To develop custom components, you need the following software and licenses.

Software requirements

In addition to the Studio Component SDK, component development requires the following software:

- Eclipse
- JDK 1.5 or above
- Apache Ant 1.7.1 or higher

Ext JS license requirement

Studio uses *Ext JS* in its components and in the default components created by its SDK.

The Oracle Endeca Information Discovery license does not bundle licensing for Ext JS.

Therefore, customers developing components with Ext JS must either purchase their own development licenses from Ext JS, or remove Ext JS and develop components without using that Javascript framework.

About obtaining junit.jar for component unit tests

If you are planning to create unit tests for your custom components, you will need to first obtain `junit.jar`.

The Component SDK can use JUnit for unit tests, but does not come with the `junit.jar` file.

Downloading and configuring the Component SDK

The Studio Component SDK is available with the Studio installer.

Before installing the Component SDK, download and unzip `EID_<version>_Studio_portal.zip`, as described in the Studio portion of the *Oracle Endeca Information Discovery Installation Guide*. This is the base Studio code, upon which the Component SDK depends. You do not have to start Studio.

Note: Do not install the Component SDK in a directory path that contains spaces.

Note: On Windows, for steps b and d below, backslashes in paths must be escaped. That is, use a path similar to the following:

```
portal.base.dir=C:\\my_folder\\EID-portal
```

instead of:

```
portal.base.dir=C:\my_folder\EID-portal
```

To install the Component SDK:

1. Download and unzip `EID_<version>_Studio_components_sdk.zip` to a separate directory.

   This is the Component SDK itself.

2. Perform the following steps within the Component SDK:
   (a) Create a file `components/build.<user>.properties`

   where `<user>` is the user name with which you logged on to this machine.

   (b) Within that `properties` file, add a single property `portal.base.dir=<absolute_path_to_portal>`

   where `<absolute_path_to_portal>` is the path to the unzipped `EID_<version>_Studio_portal.zip`.

   (c) Create a `shared.properties` file in the `shared/` directory.

   (d) Edit `shared/shared.properties` and set the single property `portal.base.dir=<absolute_path_to_portal>`

   where `<absolute_path_to_portal>` is the path to the unzipped `EID_<version>_Studio_portal.zip`.

Configuring Eclipse for component development

Before using the Component SDK to develop Studio components in Eclipse, you need to create two Eclipse classpath variables.

Note: Depending on your version of Eclipse, the steps below may vary slightly.

To configure the Eclipse classpath variables for Studio component development:

1. In Eclipse, go to **Window>Preferences>Java>Build Path>Classpath Variables**.

2. Create two new variables:

Name	Path
`DF_GLOBAL_LIB`	Path to the application server global library. Example: `C:/endeca-portal/tomcat-<version>/lib`
`DF_PORTAL_LIB`	Path to the Studio ROOT Web application library. Example: `C:/endeca-portal/tomcat-<version>/webapps/ROOT/WEB-INF/lib`

   Once these variables have been created, the components generated by the Component SDK can be imported into Eclipse.

Developing a new component

Here is a high-level overview of the component development process.

To develop a new Studio component:

1. Create the component.
2. Import the project in Eclipse.
3. Build and test the new component.

Creating a new component

New Studio components are extensions of the `EndecaPortlet` class.

To create a new component:

1. At a command prompt, navigate to the Component SDK directory, and from there to `components/portlets`.
2. Run the command:

   ```
   create.bat <component-name-no-spaces> "<ComponentDisplayName>"
   ```

   For example:

   ```
   create.bat johns-test "John's Test Component"
   ```

   In the command, the first argument is the component name. The component name:

   - Cannot have spaces.
   - Cannot include the string `-ext`, because it causes confusion with the ext plugin extension. For example, `my-component-extension` would not be a valid name.

- Has the `-portlet` automatically appended to the name. For example, if you set the name to `johns-test`, the name will actually be `johns-test-portlet`.

The second argument is intended to be a more human-friendly display name. The display name can have spaces, but if it does, it must be enclosed in quotation marks.

Importing the project in Eclipse

Before beginning component development, you have to import the component project you just created into Eclipse.

To import the Studio Component SDK project you just created into Eclipse:

1. Within Eclipse, choose **File>Import>General>Existing Projects into Workspace**.
2. As the root directory from which to import, select the directory where you installed the Component SDK.

   You should see multiple projects to import.
3. Import the components you need to work with.

   If your components depend on shared library projects located within the `/shared` directory, import those as well.

   

   Note: It takes some time for projects to build after they are imported.

Building and testing your new component

Next, you can build your new component in Eclipse and ensure that it is available in Studio.

To build your new component in Eclipse:

1. In your new project, open the `build.xml` file at the top level.
2. In the outline view, right-click the deploy task and select **Run as...>Ant Build**.

   

   Note: This step is only necessary if you do not have **Build Automatically** checked in the Eclipse **Project** menu.

3. If Studio is not already running, start Studio and log in.
4. Look at the Studio logs to confirm that the component was picked up successfully.
5. To test your new component within Studio:
   (a) From the Studio menu, select **Add Component**.
   (b) In the **Add Component** dialog, expand the **Sample** category.

   Your component should be listed in that category.

   (c) To add the new component to the Studio page, drag and drop it from the **Add Component** dialog.

Adding and removing components from the WebLogic .ear file

If you have installed Studio on Oracle WebLogic Server, then you can also add the component to the deployed .ear file, so that it will be deployed automatically the next time you deploy the file, for example when installing a production instance after you have completed testing on a development instance.

To add components to and remove components from the WebLogic .ear file:

1. To add a custom component to the .ear file:
   (a) Copy your component to the `<LIFERAY_HOME>/deploy` directory.
   (b) After the component has been processed and moved to the `<LIFERAY_HOME>/weblogic-deploy` directory, undeploy the .ear file.
   (c) Add the processed component .war file to the root of the zipped .ear file.
   (d) In the .ear file, add an entry for the new component to `META-INF/application.xml`.
2. To remove a component from the .ear file:
   (a) Remove the component .war file from the root of the .ear file.
   (b) In the .ear file, remove the component entry from `META-INF/application.xml`.

Modifying the build enhancements to the Component SDK

The `build.xml` file in the root directory of each component created by the Component SDK contains properties that control whether to include the build enhancements.

By default, these properties are:

```
<property name="shared.libs" value="endeca-common-resources,endeca-discovery-taglib" />
<property name="endeca-common-resources.includes" value="**/*" />
<property name="endeca-common-resources.excludes" value="" />
```

The properties control the following behavior:

`shared.libs`	Controls which projects in the `shared/` directory to include in your component. These shared projects are compiled and included as `.jar` files where appropriate.
`endeca-common-resources.includes`	Controls which files in the `shared/endeca-common-resources` project are copied into your component. The default value is `"**/*"`, indicating that all of the files are included, These files provide: • AJAX enhancements (`preRender.jspf` and `postRender.jspf`) • The ability to select a different data source for the component (`dataSourceSelector.jspf`)

`endeca-common-resources.excludes`	Controls which files from the `shared/endeca-common-resources` project are excluded from your component. By default, the value is `""`, indicating that no files are excluded. If your component needs to override any of these files, you must use this build property to exclude them. If you do not exclude them, your code will be overwritten.

The `includes` and `excludes` properties can be specified for any shared library, for example:

```
<property name="endeca-discovery-taglib.includes" value="**/*" />
<property name="endeca-discovery-taglib.excludes" value="" />
```

Chapter 4

Working with QueryFunction Classes

Studio provides a set of `QueryFunction` classes to allow you to filter and query data. You can also create and implement your own `QueryFunction` classes.

Provided QueryFunction filter classes

Studio provides the following filter classes. Filters are used to change the current query state. They can be used in the definition of a Studio data source, or called by a custom component.

The available filter classes are:

- `DataSourceFilter`
- `RecordFilter`
- `RefinementFilter`
- `NegativeRefinementFilter`
- `RangeFilter`
- `SearchFilter`

Note that the examples below use the syntax for calling the filters from a component. For details on configuring filters in a data source definition, see the *Oracle Endeca Information Discovery Studio User's Guide*.

DataSourceFilter

Uses an EQL snippet to provide the filtering.

When used in a data source definition, a `DataSourceFilter` is a permanent filter designed to be used for security purposes.

The available properties are:

`filterString`	The EQL snippet containing the filter information. For a `DataSourceFilter`, this would be the content of a `WHERE` clause for an EQL statement. For details on the EQL syntax, see the *Oracle Endeca Server Query Language Reference*.

For example, to filter data to only show records from the Napa Valley region with a price lower than 40 dollars:

```
ExpressionBase expression = dataSource.parseLQLExpression("Region='Napa Valley' and P_Price<40");
DataSourceFilter dataSourceFilter = new DataSourceFilter(expression);
```

RecordFilter

A `RecordFilter` can be configured to include multiple filters with Boolean logic.

When used in a data source definition, a `RecordFilter` provides permanent filtering of the data.

The properties for a `RecordFilter` are:

`recordFilter`	String The filter content. For details on the `RecordFilter` syntax, see the *Oracle Endeca Server Developer's Guide*.

In the following example, the data is filtered to only include records that have a value of Midwest for the Region attribute.

```
RecordFilter recordFilter = new RecordFilter("Region:Midwest");
```

RefinementFilter

Used to filter data to include only those records that have the provided attribute values. End users can remove `RefinementFilter` refinements.

The properties for a `RefinementFilter` are:

`attributeValue`	String The attribute value to use for the refinement. For a managed attribute, this is the value ID.
`attributeKey`	String The attribute key. Identifies the attribute to use for the refinement.

`multiSelect`	`AND` |`OR` | `NONE` For multi-select attributes, how to do the refinement if the filters include multiple values for the same attribute. If set to `AND`, then matching records must contain all of the provided values. If set to `OR`, then matching records must contain at least one of the provided values. If set to `NONE`, then multi-select is not supported. Only the first value is used for the refinement.

In the following example, the data is refined to only include records that have a value of 1999 for the Year attribute.

```
RefinementFilter refinementFilter = new RefinementFilter("1999", "Year");
```

NegativeRefinementFilter

Used to filter data to exclude records that have the provided attribute value. End users can remove `NegativeRefinementFilter` refinements.

The properties for a `NegativeRefinementFilter` are:

`attributeValue`	String The attribute value to use for the refinement.
`attributeKey`	String The attribute key. Identifies the attribute to use for the refinement.

For example, to refine the data to only include records that do NOT have a value of 2003 for the Year attribute:

```
NegativeRefinementFilter negativeRefinementFilter = new NegativeRefinementFilter("Year", "2003");
```

RangeFilter

Used to filter data to include only those records that have attribute values within the specified range. End users can remove `RangeFilter` refinements.

The properties for a `RangeFilter` are:

`attributeKey`	String The attribute key. Identifies the attribute to use for the filter.

rangeOperator	LT | LTEQ |GT |GTEQ| BTWN |GCLT |GCGT | GCBTWN The type of comparison to use. • LT - Less than • LTEQ - Less than or equal to • GT - Greater than • GTEQ - Greater than or equal to • BTWN - Between. Inclusive of the specified range values. • GCLT - Geocode less than • GCGT - Geocode greater than • GCBTWN - Geocode between
rangeType	NUMERIC | CURRENCY | DATE | GEOCODE The type of value that is being compared.
value1	Numeric The value to use for the comparison. For BTWN, this is the low value for the range. For the geocode range operators, the origin point for the comparison.
value2	Numeric For a BTWN, this is the high value for the range. For GCLT and GCGT, this is the value to use for the comparison. For GCBTWN, this is the low value for the range.
value3	Numeric Only used for the GCBTWN operator. The high value for the range.

In the following example, the data is refined to only include records where the value of P_Score is a number between 80 and 100:

```
RangeFilter rangeFilter
= new RangeFilter("P_Score", RangeType.NUMERIC, RangeOperator.BTWN, "80", "100");
```

SearchFilter

Used to filter the data to include records that have the provided search terms. End users can remove SearchFilter refinements.

The properties for a `SearchFilter` are:

`searchInterface`	String Either the name of the search interface to use, or the name of an attribute that is enabled for text search.
`terms`	String The search terms.
`matchMode`	`ALL` | `PARTIAL` | `ANY` | `ALLANY` | `ALLPARTIAL` | `PARTIALMAX` | `BOOLEAN` The match mode to use for the search.
`enableSnippeting`	boolean Whether to enable snippeting. Optional. If not provided, the default is `false`.
`snippetLength`	int The number of characters to include in the snippet. Required if `enableSnippeting` is `true`. To enable snippeting, set `enableSnippeting` to `true`, and provide a value for `snippetLength`.

In the following example, the filter uses the "default" search interface to search for the terms "California" and "red". The matching records must include all of the search terms. Snippeting is supported, with a 100-character snippet being displayed.

```
SearchFilter.Builder builder = new SearchFilter.Builder("default", "California red");
builder.matchMode(MatchMode.ALL);
builder.enableSnippeting(true);
builder.snippetLength(100);
SearchFilter searchFilter = builder.build();
```

Provided QueryConfig functions

Studio provides the following `QueryConfig` functions, used to manage the results returned by a query. These are more advanced functions for component development.

Each `QueryConfig` function generally has a corresponding function in `DiscoveryServiceUtils` to get the results.

`QueryConfig` functions are specific to a component. Because of this, `QueryConfig` functions should never be persisted to a data source using `setQueryState()`, as this would affect all of the components bound to that data source. Instead, `QueryConfig` functions should only be added to a component's local copy of the `QueryState` object.

The available `QueryConfig` functions are:

- `AttributeValueSearchConfig`

- `BreadcrumbsConfig`
- `ExposeRefinement`
- `LQLQueryConfig`
- `NavConfig`
- `RecordDetailsConfig`
- `ResultsConfig`
- `ResultsSummaryConfig`
- `SearchAdjustmentsConfig`
- `SearchKeysConfig`
- `SortConfig`

AttributeValueSearchConfig

Used for typeahead in search boxes. For example, used in Guided Navigation to narrow down the list of available values for an attribute.

`AttributeValueSearchConfig` has the following properties:

`searchTerm`	String The term to search for in the attribute values.
`maxValuesToReturn`	int (optional) The maximum number of matching values to return. If you do not provide a value, then the default is 10.
`attribute`	String (optional) The attribute key for the attribute in which to search. Use the `attribute` property to search against a single attribute. To search against multiple attributes, use `searchWithin`.
`searchWithin`	List<String> (optional) A list of attributes in which to search for matching values.
`matchMode`	`ALL`|`PARTIAL`|`ANY`|`ALLANY`|`ALLPARTIAL`|`PARTIALMAX`|`BOOLEAN` (optional) The match mode to use for the search.
`relevanceRankingStrategy`	String (optional) The name of the relevance ranking strategy to use during the search.

The following example searches for the term "red" in the WineType attribute values:

```
AttributeValueSearchConfig attributeValueSearchConfig
= new AttributeValueSearchConfig("red", "WineType");
```

BreadcrumbsConfig

Used to return the breadcrumbs associated with the query. Allows you to specify whether to display the full path for hierarchical attribute values.

`BreadcrumbsConfig` has the following property:

`returnFullPath`	boolean (optional) For a hierarchical managed attribute, whether to return the full path to the selected value. The default is `true`, indicating to return the full path. To not return the full path, set this to `false`.

This example returns the breadcrumbs, but does not return the full path for hierarchical managed attributes:

```
BreadcrumbsConfig breadcrumbsConfig = new BreadcrumbsConfig(false);
```

ExposeRefinement

Affects results from a `NavConfig` function. Used to implement Guided Navigation. Controls whether to display available attributes within groups, and whether to display available refinements for attributes.

`ExposeRefinement` has the following properties:

`dimValId`	String The ID of the selected attribute value. You would provide an attribute value ID if you were displaying the next level of available values in a managed attribute hierarchy.
`dimensionId`	String The name of the attribute. You must provide at least one `dimValId` or `dimensionId`.
`ownerId`	String (optional) The ID of the associated `NavConfig` instance. If not provided, then uses the first `NavConfig` instance.
`dimExposed`	boolean (optional) Whether to display the available values for the attribute, to the number specified in `maxRefinements`. The default is `true`.

`exposeAll`	boolean (optional) Whether to display the complete list of available values. For example, on the **Guided Navigation** component, would indicate whether the "More..." link is selected. The default is `false`.
`maxRefinements`	int (optional) The maximum number of available values to display. The default is `1000`.
`groupKey`	String (required) The name of a group.
`groupExposed`	boolean (optional) Whether to display all of the attributes in the specified group. The default is `true`.

The following example shows the available attributes for the Flavors attribute within the Characteristics group.

```
ExposeRefinement exposeRefinement = new ExposeRefinement("/", "Flavors", "Characteristics");
```

LQLQueryConfig

Executes an EQL query on top of the current filter state.

`LQLQuery` has the following property:

`LQLQuery`	AST The EQL query to add. To retrieve the AST from the query string, call `DataSource.parseLQLQuery`.

The following example retrieves the average of the P_Price attribute grouped by Region:

```
Query query
= dataSource.parseLQLQuery("return mystatement as select avg(P_Price) as avgPrice group by Region",
true);
LQLQueryConfig lqlQueryConfig = new LQLQueryConfig(query);
```

NavConfig

Used to retrieve a navigation menu, such as in the Guided Navigation component.

`NavConfig` has the following properties:

`exposeAllRefinements`	boolean Whether to display all of the available values for the attributes. Determines the initial state of the menu. The associated `ExposeRefinement` function is then applied. The default is false.
`List<RefinementGroupConfigs>`	List of groups for which to return the available attributes. If no `RefinementGroupConfigs` are specified, no attribute groups or attributes are returned.

The following examples returns attributes in the Source and Characteristics groups:

```
List<RefinementGroupConfig> refinementGroups = new ArrayList<RefinementGroupConfig>();
RefinementGroupConfig source = new RefinementGroupConfig();
source.setName("Source");
source.setExpose(true);
refinementGroups.add(source);
RefinementGroupConfig characteristics = new RefinementGroupConfig();
characteristics.setName("Characteristics");
characteristics.setExpose(true);
refinementGroups.add(characteristics);
NavConfig navConfig = new NavConfig();
navConfig.setRefinementGroupConfig(refinementGroups);
```

RecordDetailsConfig

Sends an attribute key-value pair to assemble the details for a selected record. The complete set of attribute-value pairs must uniquely identify the record.

`RecordDetailsConfig` has the following property:

`recordSpecs`	List<RecordSpec> Each new `RecordDetailsConfig` is appended to the previous `RecordDetailsConfig`.

The following example sends the value of the P_WineID attribute:

```
List<RecordSpec> recordSpecs = new ArrayList<RecordSpec>();
recordSpecs.add(new RecordSpec("P_WineID", "37509"));
RecordDetailsConfig recordDetailsConfig = new RecordDetailsConfig(recordSpecs);
```

ResultsConfig

Used to manage the returned records. Allows for paging of the records.

`ResultsConfig` has the following properties:

`recordsPerPage`	long The number of records to return at a time.
`offset`	long (optional) The position in the list at which to start. The very first record is at position 0. For example, if `recordsPerPage` is 10, then to get the second page of results, the offset would be 10.
`columns`	String[] (optional) The columns to include in the results. If not specified, then the results include all of the columns.
`numBulkRecords`	int (optional) The number of records to return. Overrides the value of `recordsPerPage`.

The following example returns a selected set of columns for the third page of records, where each page contains 50 records:

```
ResultsConfig resultsConfig = new ResultsConfig();
resultsConfig.setOffset(100);
resultsConfig.setRecordsPerPage(50);
String[] columns = {"Wine_ID", "Name", "Description", "WineType", "Winery", "Vintage"};
resultsConfig.setColumns(columns);
```

ResultsSummaryConfig

Gets the number of records returned from a query.

```
ResultsSummaryConfig resultsSummaryConfig = new ResultsSummaryConfig();
```

SearchAdjustmentsConfig

Returns "Did you mean" and auto-correction items for a search.

```
SearchAdjustmentsConfig searchAdjustmentsConfig = new SearchAdjustmentsConfig();
```

SearchKeysConfig

Returns the list of available search interfaces.

```
SearchKeysConfig searchKeysConfig = new SearchKeysConfig();
```

SortConfig

Used to sort the results of a query. Used in conjunction with `ResultsConfig`.

`SortConfig` has the following properties:

`ownerId`	String (optional) The ID of the `ResultsConfig` that this `SortConfig` applies to. If not provided, uses the default `ResultsConfig` ID. If you configure a different ID, then you must provide a value for `ownerId`.
`property`	String The attribute to use for the sort.
`ascending`	boolean Whether to sort in ascending order. If set to `false`, then the results are sorted in descending order.

For example, with the following `SortConfig`, the results are sorted by the P_Score attribute in descending order:

```
SortConfig sortConfig = new SortConfig("P_Score", false);
```

Creating a custom QueryFunction class

The Component SDK directory includes scripts for creating new `QueryFunction` classes.

Note: Before you can create `QueryFunction` classes, you must install the Component SDK, which is a separate download. See *Downloading and configuring the Component SDK on page 9*.

To create a new `QueryFilter` or `QueryConfig` class:

1. In a terminal window, change to the `endeca-extensions` subdirectory of the Component SDK's root directory (normally called `components`).
2. Run the appropriate command to create the `QueryFilter` or `QueryConfig` class.

   To create a `QueryFilter` class:

On Windows:	`.\create-queryfilter.bat` <*`your-query-filter-name`*>
On Linux:	`./create-queryfilter.sh` <*`your-query-filter-name`*>

   To create a `QueryConfig` class:

On Windows:	`.\create-queryconfig.bat` <*`your-query-config-name`*>
On Linux:	`./create-queryconfig.sh` <*`your-query-config-name`*>

The command creates in the endeca-extensions directory a new directory for the QueryFilter or QueryConfig class:

- For a QueryFilter, the directory is <*your-query-filter-name*>-filter.
- For a QueryConfig, the directory is <*your-query-config-name*>-config.

This directory is an Eclipse project that you can import directly into Eclipse, if you use Eclipse as your IDE.

It contains an empty sample implementation of a QueryFilter or QueryConfig. This has no effect on QueryState in its original form.

The skeleton implementation creates source files that:

- Extend either QueryFilter or QueryConfig.
- Create stubs for the applyToDiscoveryServiceQuery, toString, and beforeQueryStateAdd methods.

  applyToDiscoveryServiceQuery and toString are required methods that you must implement.

  beforeQueryStateAdd is an optional method to verify the query state before the function is added. This method is used to prevent invalid query states such as duplicate refinements.
- Create a no-argument, protected, empty constructor. The protected access modifier is optional, but recommended.
- Create a private member variable for logging.

Implementing a custom QueryFunction class

After you create your new QueryFunction class, you then implement it.

To implement your new QueryFunction, you must:

- Add private filter or configuration properties.
- Create getters and setters for any filter properties you add.
- Define a no-argument constructor (protected access modifier optional, but recommended).
- Optionally, implement the beforeQueryStateAdd(QueryState state) method to check the current query state before the function is added.

Deploying a custom QueryFunction class

Before you can use your new QueryFunction, you must deploy it to Studio.

The directory that you created for the new QueryFilter or QueryConfig contains an ant build file.

The ant deploy task places a .jar file containing the custom QueryFunction into the endeca-portal/tomcat-<version>/lib/ext directory.

Note: If you are not using the default portal bundle, put the new QueryFunction.jar into the container's global classpath.

To deploy the new QueryFunction:

1. Run the ant build.

2. Restart Studio.

   The portal picks up the new class.

After you deploy your custom `QueryFunction`, you can use it in any component.

Adding the custom QueryFunction .jar file to your Eclipse build path

If you are using Eclipse as your IDE, you need to add the new `.jar` file to the build path of your custom component.

To add the new `.jar` file to your Eclipse build path:

1. Right-click the project, then select **Build Path>Configure Build Path**.
2. Click the **Libraries** tab.
3. Click **Add Variable**.
4. Select **DF_GLOBAL_LIB**.

   You should have added this variable when you set up the SDK.
5. Click **Extend**.
6. Open the `ext/` directory.
7. Select the `.jar` file containing your custom `QueryFunction`.
8. Click **OK**.

After adding the `.jar` file to the build path, you can import the class, and use your custom `QueryFilter` or `QueryConfig` to modify your `QueryState`.

Obtaining query results

The `Results` class is used to represent results of queries.

You must add the relevant `QueryConfigs` to a component in order to specify the types of results it needs.

```
QueryState query = getDataSource(request).getQueryState();
query.addFunction(new NavConfig());
QueryResults results = getDataSource(request).execute(query);
```

You can then get the underlying API results and do whatever manipulation is required by your component.

```
Results discoveryResults = results.getDiscoveryServiceResults();
```

Before executing, you can also make other local modifications to your query state by adding filters or configurations to your query:

```
QueryState query = getDataSource(request).getQueryState();
query.addFunction(new ResultsConfig());
query.addFunction(new RecordFilter("Region:Midwest"));
QueryResults results = getDataSource(request).execute(query);
```

When you need to update a data source's state to update all of the associated components, you must use `QueryState` instances.

```
DataSource ds = getDataSource(request);
QueryState query = ds.getQueryState();
query.addOperation(new RecordFilter("Region:Midwest"));
ds.setQueryState(query);
```

Index