Table of Contents

[Prerequisite 2](#_Toc123822557)

[Overview 2](#_Toc123822558)

[Automatic search configuration 3](#_Toc123822559)

[Automatic search attributes 3](#_Toc123822560)

[TEXT TYPES 8](#_Toc123822561)

[Visibility and sort order 9](#_Toc123822562)

[Search text types 9](#_Toc123822563)

[Search query configuration 10](#_Toc123822564)

[Search text type 10](#_Toc123822565)

[Search query 11](#_Toc123822566)

[Using @ComponentOwner in the search query 11](#_Toc123822567)

[Reindex 12](#_Toc123822568)

[ADDING NEW SEARCH TEXT TYPES 12](#_Toc123822569)

[CHANGING THE SORTING CONFIGURATION 12](#_Toc123822570)

[CHANGING THE SORTING OR FILTERING CONFIGURATION IN THE SEARCH RESULT TABLE 12](#_Toc123822571)

[Configuration examples 12](#_Toc123822572)

[Example 1 - automatic search displayed as a list 12](#_Toc123822573)

[Example 2 - automatic search displayed as a table with selectable columns 13](#_Toc123822574)

# Prerequisite

**Important**: This information is applicable only for existing customers.

Dependencies have been removed from the following control attributes in the automatic search:

* @NumberOfSearchResultsPerPage
* @SearchForModelTextIds
* @SearchForModelTypes
* @SearchQuery
* @SearchTextType

In case existing definitions were using dependencies in these attributes, they will need to be recreated as new automatic searches. To facilitate that, the migration message contains the full dependency rule.

# Overview

Automatic search component consists of one or more predefined searches which are represented as separate tabs in the user interface. Each tab will display the implementations matching the predefined query configured for that tab.

Automatic search can be displayed in a process or in a zone. To do that, a component attribute pointing to the automatic search must be created in the current process object or in the zone. The position of the automatic search within the process or a zone can be defined using the visibility and sort order properties of the component attribute.

The search query of the automatic search can use the attributes of the current process object or the zone in which the automatic search is displayed.

# Automatic search configuration

To configure an automatic search, a few attributes can be used on the root and the bricks. Each brick will represent a separate search and will be displayed as a separate tab in the UI.

Graphical user interface, text, application

Description automatically generated

## Automatic search attributes

The following attributes on the root can be used to configure an automatic search:

|  |  |
| --- | --- |
| **Control attribute text-id** | **Description** |
| @ComponentOwner | To display an automatic search within a process or a zone, a component attribute referring to the automatic search must be created on the current process object or the zone. This current process object (or the zone) is then the so called "Component owner" and the value of the @ComponentOwner reference attribute will then contain the UUID of that current process object (or zone) implementation.  The @ComponentOwner reference attribute may be used in the search query to navigate to the component owner to use the values of its attributes in the configuration of the search logic.  Example:   * The search query in an automatic search which is embedded in the Home product can depend on the value of attributes in the Home implementation |

Every brick in an automatic search component represents a separate search displayed as a tab. The following attributes can be used on the brick to configure a search tab:

|  |  |  |
| --- | --- | --- |
| **Control attribute text-id** | **Description** | **Mandatory** |
| @SearchQuery | The value of this attribute defines the search query with which the search is executed. The following format must be followed:  QUERY:({searchQuery})  Where?  {searchQuery} is the real query to be executed.  Recommendations on how to configure a search query and the search text type on the definitions are described [below](https://confluence.innoveo.com/is902/automatic-search-131140926.html#Automaticsearch-searchQuery). | Yes |
| @SearchTextType | The value of this attribute specifies the name of the search text type in which the search query will be executed. This value must be equal to the search text defined on the root of the definitions that the automatic search should look for.  What is specified in this search text on the root of the definitions, defines what will be stored for an implementation in the index in which the search will be executed. In other words, this text defines **where to search**for what was specified in the search query.  In case the search results are displayed as a list and not as a table, this text also defines **what will be be displayed** for an implementation which matches the search query.  See [below](https://confluence.innoveo.com/is902/automatic-search-131140926.html#Automaticsearch-searchTextTypes) for more information. | Yes |
| @NumberOfSearchResultsPerPage | The value of this attribute defines the number of search results displayed per page. | Yes |
| @SearchForModelTypes | This attribute defines a list of model types to search for. It is possible to add the following model types: campaign, generic, product, user, usergroup.  If the list is empty, then the model type is not considered as a filter for the search.  This attribute is used only when the search results are displayed as a list, and not as a table. | No |
| @SearchForModelTextIds | This attribute defines a list of model text-ids to search for, that belong to the specified model types in @SearchForModelTypes attribute. It is possible to add model text-ids that belong to the following model types: campaign, generic, product, user, usergroup.  If the list is empty, then the model text-id is not considered as a filter for the search.  This attribute is used only when the search results are displayed as a list, and not as a table. | No |
| @SearchSorting | This attribute is used only when the search results are displayed as a list, and not as a table.  It defines the default sorting parameter (LastModified or CreationDate) and the default sorting order (Ascending or Descending) for the matching search results.  It is also possible to add a list of additional sorting parameters by which the results can be sorted in case the search is done in **one** single root. In that case it is also possible to specify an attribute as the default sorting parameter.  The attribute @SearchSorting needs to be visible to display a label on the search component and to have the sorting angles visible where the user can change the sort order manually.  It is possible to sort after attributes of type generic, counter, flexdata single, service and status. In case several sorting parameters are configured, a drop-down menu will be displayed on top of the search component, where the user can select the field the search results should be sorted after. The labels of these lists are displayed if at least 2 sorting parameters are in the list. | Yes, only for list display |
| @SearchResultTable | If this attribute is present in the brick of the automatic search, then the search results for the respective tab will be displayed as a table.  Several properties can be configured for this attribute:   * **Default sorting parameter** (LastModified or CreationDate) and the **default sorting order** (Ascending or Descending) for the matching search results. * **Definitions to search for**: a list of model text-ids whose implementations should be found in the automatic search, * Table display settings, which include: * A list of **table headers** and the corresponding **data type**, * Defining if a table header should be **filterable**and/or **sortable**, * Defining **what should be displayed** under each table header for each definition whose implementation matches the search query. Variable resolvers and text values are supported. * Text to be displayed when no matching records are found.   See [below](https://confluence.innoveo.com/is902/automatic-search-131140926.html#Automaticsearch-Example2) for an example of an automatic search configuration using a tabular display of the search results.  When the @SearchResultTable attribute is present, the @SearchSorting, @SearchForModelTypes and @SearchForModelTextIds attributes cannot be used. | No |
| @SelectableAutosearch | The configuration of this attribute will add extra columns (selectable columns) in the automatic search result. Several properties can be configured for this attribute:   * **Where to display the selection:**define if the additional columns will be displayed at the end or at the beginning, * **autosearchButtonLabel:**define the label of the button that will execute the "set value" defined in the automatic search, * **Multiple selection per row**: if this option is selected, the row selection will be done with checkboxes. Otherwise, the selection will be done with radio buttons. * **Definition to set values:**defines the root in which the "set value" will happen * **Label of the column:**The label of the additional column in the automatic search * **Attribute:**Attribute where the value will be set * **Value:**Value to be set for the attribute   See below for an example of an automatic search configuration using selectable columns. | No |

### Text types

The following text types can be configured for an automatic search:

|  |  |  |
| --- | --- | --- |
| **Text Type** | **Location** | **Description** |
| header | Root | Text displayed as the header of the automatic search component. |
| searchInfo | Root | Text displayed below the search results.  For example: Showing records \_START\_ to \_END\_ of a total of \_TOTAL\_ entries |
| searchNoRecords | Root | Text displayed if no search results are found. |
| header | Brick | Text displayed as the header of the tab represented by this brick. |

Graphical user interface, text, application, email

Description automatically generated

## Visibility and sort order

Visibility and sort order of all bricks can be defined by default and per context. The sort order defines the order in which the search tabs are displayed. The visibility defines whether the search tab is displayed at all or not.

In case of defining the brick visibility and sort order per context, a generic context page (for example, searchPage) can be defined and used for that purpose.

## Search text types

At least one search text type must be added to every definition of which implementations should be found in a search. Search text defines **where to search** for what was specified in the search query. In case the search results are displayed as a list and not as a table, then the search text also defines **what is displayed** as a search result.

Every search text type that is added to any definition defines a value that will be stored for an implementation in the index in which the search is performed. If in one search it should be possible to search for implementations of different model text-ids, it is mandatory to create the same search text type on all the corresponding definitions. It is possible to add more than one search text type to a definition. In this case, an implementation will be put multiple times into the search index. Like all other texts, the search text types are also language specific. The search will always be done only for a search text type of the current language. The attribute @SearchTextType defines per search component where to search for in an index. Search text types can be added to definitions of the following model types:

* campaign
* generic
* product
* user
* usergroup.

## Search query configuration

Below are recommendations for configuring the search query and search text type of automatic searches.

### **Search text type**

On the search text type, it is recommended to define the search parameters by concatenating for every parameter the parameter key (Example: UserCreated) and the parameter value (Example: %{root->userCreated}) in one string. In most cases those search parameters should not be displayed, which can be achieved by using "display:none". The value of the search parameter can be dynamic or fixed. For dynamic parameters the parameter value depends on the implementation (Example: on the value of the attribute @Status of a referral). For fixed parameters the value is fixed (Example: the editable role for a referral is always "underwriter").

**Example**

Referral created by: %{root->userCreated}. Creation date: %{root->creationDate}

<**div** style="display:none">UserCreated%{root->userCreated} CurrentStatus%{Referral.@Status->rawValue} EditableRoleunderwriter</**div**>

What would be displayed in the above example is something like: "Referral created by: userCC. Creation date: 21.02.2015 12:22:45".

In the second section there are three search parameters defined:

|  |  |  |  |
| --- | --- | --- | --- |
| **Parameter key** | **Parameter value** | **Concatenated string** | **Parameter type** |
| UserCreated | %{root->userCreated} | UserCreated%{root->userCreated} | dynamic |
| CurrentStatus | %{Referral.@Status->rawValue} | CurrentStatus  %{Referral.@Status->rawValue} | dynamic |
| EditableRole | underwriter | EditableRoleunderwriter | fixed |

### **Search query**

The search query on the attribute @SearchQuery should now use those search parameters combined with one or multiple search operators:

* AND
* OR
* NOT
* AND NOT
* OR NOT

**Example**:

QUERY:(UserCreated%{UserContext['userId']} AND EditableRole%{UserContext['skyeRole']} AND (CurrentStatusInReview OR CurrentStatusInReferral)

This search query would give back all referral implementations that were created by the logged in user which are either in status "InReview" or "InReferral” if the user has role "underwriter".

### **Using @ComponentOwner in the search query**

If the attributes from the current process object, in which the automatic search is displayed, should be used as dynamic parameters in the search query, this can be accomplished by using the @ComponentOwner reference attribute.

For example, if an automatic search "QuoteSearch" should be displayed on the customer process object to return all Home quotes created for that customer, the following configuration could be used. The Component owner for the automatic search would in this case be the Customer definition and the value of the @ComponentOwner attribute would contain the UUID of the Customer implementation.

The following search text could be defined on the Home definition:

Home quote %{Home.quoteNumber->shortOrFormattedValue}, created by: %{root->userCreated}, created on: %{root->creationDate}

<**div** style="display:none">ComponentOwner%{Home.CustomerRef->shortOrFormattedValue}</**div**>

What would be displayed in the above example is something like: "**Home quote Q-0343, created by: userCC, created on: 16.03.2021 13:01:45**".

The "display:none" block defines a search parameter ComponentOwner which looks at the value of the reference to customer on a Home implementation.

## Reindex

If certain changes are made to the automatic search configuration, it is recommended to restart the application and perform a reindex on the relevant definitions using the admin console, so that the changes to the search configuration are properly applied.

Most common examples where a reindex is required are described below.

### Adding new search text types

If there are already implementations in the application and a search text type is added to a certain definition afterwards, it is important to know that all existing implementations are not automatically added to the index with the new search text type value. To add them to the index as well so that they can be found afterwards, a re-indexing must be done. As re-indexing takes quite some time if there is a lot of data to be re-indexed, please check first with the support whether this can be done in a reasonable time or not.

### Changing the sorting configuration

If any sorting field configuration is **created or modified** in an automatic search in the @SearchSorting attribute, or in case an attribute contained in the sorting fields has its **type modified from / to decimal or integer**, the following actions are required:

* Restart the application
* Reindex affected products

Reindex may either be executed with Skye admin console or through JMX. For productive deployments, please request a re-index of freetext search indexes.

### Changing the sorting or filtering configuration in the search result table

If any sorting or filtering configuration is **created or modified** in the @SearchResultTable attribute (Example: adding another sortable or filterable column, or changing the default sorting parameter), it is necessary to:

* Restart the application
* Reindex affected products

Reindex may either be executed with Skye admin console or through JMX. For productive deployments, please request a re-index of freetext search indexes.

## Configuration examples

### Example 1 - automatic search displayed as a list

In this example we will configure an automatic search which will be displayed in a zone and return a list of internal users.

[Graphical user interface, text, application

Description automatically generated with medium confidence](https://confluence.innoveo.com/is902/files/131140926/131140934/1/1670408031000/image2022-12-5_13-26-35.png)

The configuration is as follows:

1. Add a new automatic search definition "UsersAutomaticSearch".
2. Define the texts on the root.
3. Add the @SearchForModelTextIds control attribute and add "InternalUser" to the list of model text-ids.
4. In the @SearchQuery control attribute, enter value:

QUERY:(InternalUser)

1. In @SearchTextType control attribute, add a new search text type "UsersAutoSearch".
2. In the InternalUser definition, on the Search texts tab, add a search text type of the same name as in the previous step "UsersAutoSearch" and define how the search results should be displayed and what should be stored in the index for the InternalUser implementations.

Example:

<**div** style="display:block"><**b**>Username: </**b**>[%{InternalUser.@Username->shortOrFormattedValue}](mailto:%25%7bInternalUser.@Username-%3eshortOrFormattedValue%7d)

<**p**>Full name: %{InternalUser.firstName->shortOrFormattedValue} %{InternalUser.lastName->shortOrFormattedValue},email address: %{InternalUser.@Email->shortOrFormattedValue}, created on %{root->creationDate}</**p**></**div**>

<**div** style="display:none">InternalUser</**div**>

1. On the Zone definition, add a component attribute to the corresponding brick where the search should be displayed. Add "UsersAutoSearch" to the "Component model text-id" field in the Component tab of the attribute. Make sure the visibility of the component attribute is set to "read write" in the default or a specific context.
2. Activate the definitions. All newly created internal users will be displayed in the search.

To display the already existing internal users, a reindex would have to be done.

### Example 2 - automatic search displayed as a table with selectable columns

In this example we will configure an automatic search that will be displayed in a process of editing an internal organization. The search will return a list of internal users that belong to that organization.

Graphical user interface, text, application, chat or text message

Description automatically generated

The configuration is as follows:

1. Add a new automatic search definition "OrgUsersAutomaticSearch".
2. Define the texts on the root.
3. Add the @SearchResultsTable attribute to the brick and make the following changes in its properties:

* Add "InternalUser" to the list of definitions to search for
* Add the following headers to the list of table headers:
  + Username
  + Full name
  + Email
* Check the "Sortable" and "Filterable" checkboxes next to all the columns to enable sorting and filtering by each table column.
* Add the following values in the fields to display under each table header for the found implementations:
  + %{InternalUser.@Username->shortOrFormattedValue}
  + %{InternalUser.firstName->shortOrFormattedValue} %{InternalUser.lastName->shortOrFormattedValue}
  + %{InternalUser.@Email->shortOrFormattedValue}
* Add value "No records found" in the text type 'No matching records found'.

1. In @SearchTextType control attribute, add a new search text type "OrgUsersAutoSearch".
2. In the @SearchQuery control attribute, enter value:

**QUERY**:(ComponentOwner%{{OrgUsersAutomaticSearch.@ComponentOwner->shortOrFormattedValue}})

We are referring to the component owner of the search, which in this case will be the InternalOrganization definition (because the search will be displayed in a process where InternalOrganization is the current process object). The value of the @ComponentOwner attribute will contain the UUID of the organization in which the search is displayed.

1. In the InternalUser definition, on the Search texts tab, add a search text type of the same name as in the previous step "OrgUsersAutoSearch".

There we will define the search parameter by which the InternalUser implementations should be found.

ComponentOwner%{InternalUser.organizationRef->shortOrFormattedValue}

We are using the reference to the organization on the InternalUser implementation as a dynamic search parameter. The search query will return all InternalUser implementations for which the organizationRef attribute has the UUID of the organization in which the search is displayed.

1. Add the @SelectableAutosearch control attribute to the brick and make the following changes in its properties:

* Select "Last column" in the "Where to display the selection" dropdown
* Click on the "Configure the extra columns in the autosearch" option:
  + Add "InternalUser" to the Definition to set values list
  + Check the "Multiple selection per row" option
  + Click on the "Add column to the autosearch" option
  + Enter "Select" as the column name
  + Add "InternalUser@Status" to the attribute list and enter value "Blocked" in the set value field
  + Click OK
* Enter value "Block user" in the autosearchButtonLabel text field.

1. Define a context in which the automatic search will be visible. Example: admin-admin-searchPage.
2. Add a component attribute to the InternalOrganization definition. Add "OrgUsersAutoSearch" to the "Component model text-id" field in the Component tab of the attribute. Add searchPage as the Context page.

Make sure the visibility of the component attribute is set to "read write" in the context in which the search should be visible (Example: admin-admin-EditOrganizationProcess:OrganizationData).

1. Activate the definitions.

In the process for editing an internal organization, a search displaying all users of that organization is displayed.

To display the already existing internal users, a reindex would have to be done.