NETVIBES | Exalead CloudView

CloudView CV23

# Mashup

# Table of Contents

# Mashup Builder

The Mashup Builder application allows you to design the search application, also called Mashup UI, which is the front end of the Exalead CloudView solution.

It lets you customize the home page, the search results page and additional pages, through a user-friendly drag and drop interface. This guide gives procedures and examples illustrating the most common use cases of the Mashup Builder, but does not cover all the possibilities of the application.

## Audience

This guide is mainly destined to consultants, developers, or system administrators who are new to Exalead CloudView and its Mashup Builder application.

## Access the Mashup Builder

The Mashup Builder application is accessible at: http://<HOSTNAME>:<BASEPORT+1>/mashup-builder

The default Mashup UI front end is accessible at: http://<HOSTNAME>:<BASEPORT>/mashup-ui

## Further Reading

You might need to refer to the following guides:

| Guide | for more details on |
|---|---|
| Mashup Programmer | Mashup UI customization. |
| Configuration | indexing and search concepts, as well as advanced functionality. |

# What's New?

There are no enhancements in this release.

# About Mashup Builder

This section describes the basic background information to understand the Mashup Builder environment.

## Mashup Builder or Mashup Builder Premium

There are two editions of the Mashup Builder application:

- By default, Exalead CloudView includes a standard Mashup Builder that lets you work with two feeds and several widgets.
- With an additional license, you can use the Mashup Builder Premium edition, which extends Mashup Builder with more feed and widget types.

The following table presents the differences between the Mashup Builder and the Mashup Builder Premium editions.

*Mashup Builder VS Mashup Builder Premium*

| Element | Mashup Builder | Mashup Builder Premium |
|---|---|---|
| Widget access | Limited set of widgets. | Unlimited<br><br>Premium widgets are flagged by 🅿 in the *Widget Reference*. |
| Feed access | Exalead CloudView **Search** & Exalead CloudView **Data Model** feeds only.<br><br>One feed per page only. | Unlimited |
| Trigger access | No | Yes |
| Create Mobile Applications | No | Yes |

## Mashup Builder Terminology

This section describes the most important terms and concepts in Mashup Builder.

**Note:** All throughout the application, you can hover over the ‘i’ information icons to display tooltip descriptions.

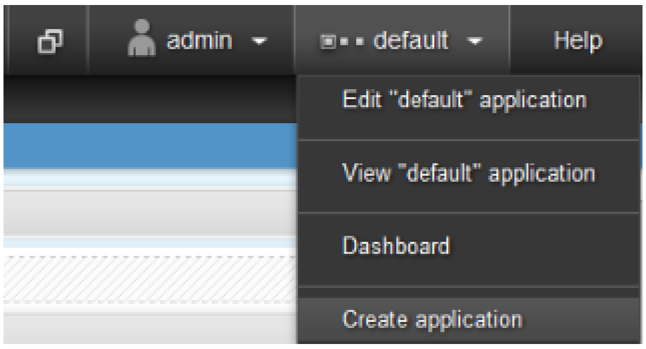
- CloudView Administration Console is the main interface for configuring data sources, indexing options and search processes in the Exalead CloudView platform. To access, go to the product’s `BASEPORT+ 1` (for example, if the installation is at `http://host:10000`, admin will be at `http://host:10001/admin`). Log in with the user `app-admin` and password `exalead`.
- Composite widgets can be created with the standard widget library to handle very specific needs. Your new composite widgets are added to the **Widgets > Composite Widgets** group. They can then be reused as any other widget on any page of your application. Updating a composite widget updates all its instances at once, which is useful when the same widgets are repeated on every page (for example, menus, headers, etc.).
- Facets are used to narrow search results. Use them to drill-down into an area, such as language, author, or file type. They are typically used in dashboard analytics widgets, or in the Refinements panel for enterprise search.
- Feeds are generic content fetchers for heterogeneous sources. The Mashup API retrieves and searches hits from feeds and returns the results in standard Atom format. Feeds can also be enriched using nested and/or parallel requests.
- Mashup API is the API which retrieves the contents of data sources to make them accessible to the data feeds. Note that Mashup Builder Premium also uses other APIs for non-Exalead CloudView feeds (for example, FlickR search).
- Mashup UI is the search front-end of the Exalead CloudView solution.
- MEL Functions (Mashup Expression Language) are dynamic variable expressions which allow you to construct text that contains dynamic content from your feeds, for example, `${feeds["persons"].metas["name"]}`. These expressions provide much more than just dynamic variables and support common operations that would usually require editing JSP files.
- Triggers can be added to the applications created with a Mashup Builder Premium instance. They allow you to launch specific actions and alter the default behavior of feeds, widgets and pages.
- Widgets are the graphic components which build up the Mashup Builder. There are widgets for:
    - Creating search forms

- ◦ Displaying search results in different forms (lists, tables, charts, maps...)
- ◦ Controlling search refinements (facets)

For complete widget descriptions, see the Widget Reference.

## Overall Description of the Menus

Mashup Builder contains the following menus.



| Menu | Description |
|---|---|
| **Edit <application name> application** | This is the main menu where you design your Mashup UI applications, using feeds and widgets.<br>See Building Mashup Applications.<br>This is also here that you can administer you mashup applications, deploy and delete them, etc. using the Application tools.<br>See Understanding the Edit Application Menu. |
| **Create application** | This is the main menu where you can create new web or mobile applications.<br>See Creating New Applications. |
| **Dashboard** | This menu displays the following panes:<br>• **Users** – listing all the users connected to the Mashup Builder instance. You can log out users if necessary.<br>• **Storage** – listing all the components using the data storage.<br>• **Statuses** - listing the statuses of the various mashup services. |

# Understanding the Edit Application Menu

This section shall give you the basics to understand the key components of the Edit application menu.

## Pages

In Mashup Builder, applications are made of different pages, for example **/index**, **/search**, etc.

Each page has the following views:

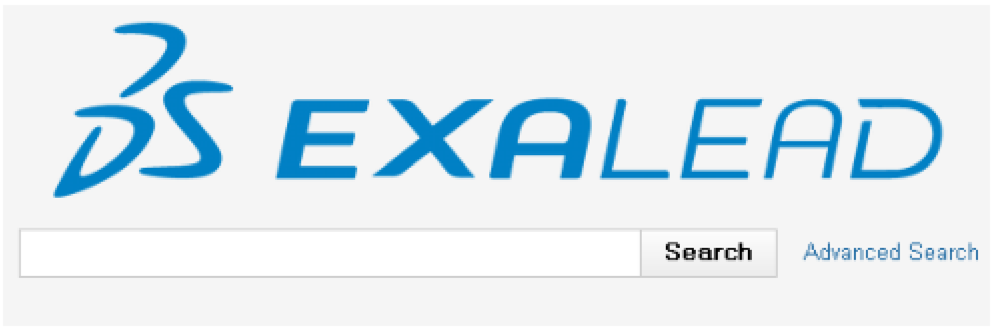
- **Feeds**: which lets you add and configure the data sources.
- **Design**: a canvas where you can set up the look and feel of the Mashup Builder and display feeds using widgets.
- **Preview**: which gives you a preview of your page as it will be displayed in the Mashup Builder.

### Index & Search page

By default, the mashup application contains:

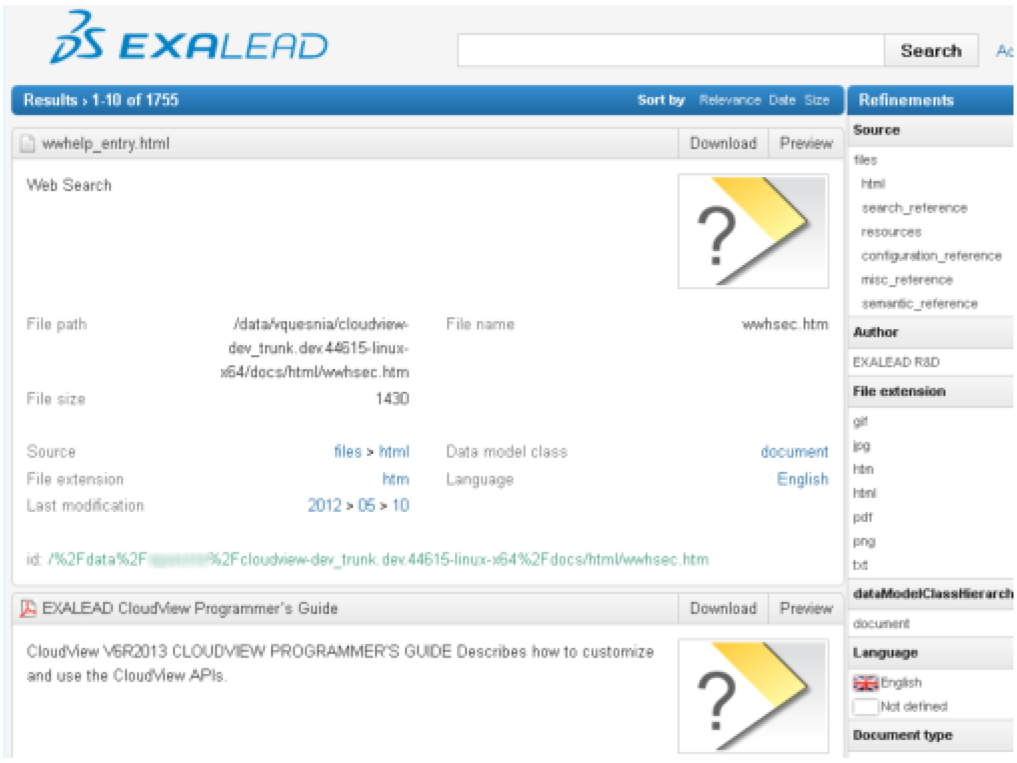
- An **/index** page which stands as the home page of your Search Based Application. It is the only required page.

  *Index page on Mashup Builder*

- A **/search** page that displays the search results/hits.

*Search page on Mashup Builder*



wwhelp_entry.html

Web Search

| | | | |
|---|---|---|---|
| File path | /data/vquesnia/cloudview-dev_trunk.dev.44615-linux-x64/docs/html/wwhsec.htm | File name | wwhsec.htm |
| File size | 1430 | | |
| Source | files > html | Data model class | document |
| File extension | htm | Language | English |
| Last modification | 2012 > 05 > 10 | | |

id: /%2Fdata%2F[redacted]%2Fcloudview-dev_trunk.dev.44615-linux-x64%2Fdocs/html/wwhsec.htm

EXALEAD CloudView Programmer's Guide

CloudView V6R2013 CLOUDVIEW PROGRAMMER'S GUIDE Describes how to customize and use the CloudView APIs.

Source
files
html
search_reference
resources
configuration_reference
misc_reference
semantic_reference

Author
EXALEAD R&D

File extension
gif
jpg
htm
html
pdf
png
txt

dataModelClassHierarch

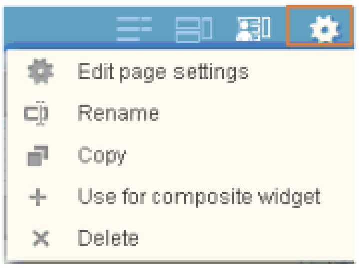
document

Language
English
Not defined

Document type

## Page tools

To create an interface you can then add, copy, modify and delete pages. To do so, use the **Add page** button and the page editing tools at the right of the screen.



Use the page settings to control the behavior of your current page, and customize the:

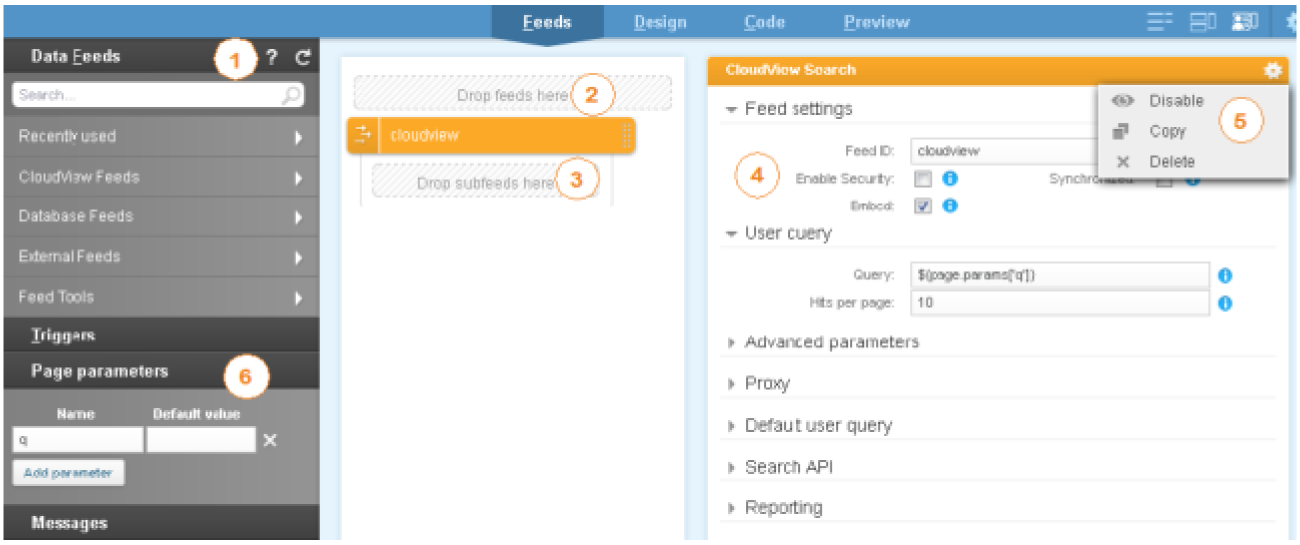
- Page Title
- Page description
- Additional meta headers
- CSS attributes (CSS id, CSS class, CSS rules)

## Feeds view

The **Feeds** view allows you to configure the feeds that can be integrated into your page.

These are the data sources that can be used by your application.

To the left of the screen, the **Data Feeds** panel lists the various feeds that you can configure and make available in the design of your application.



Feed ID: cloudview
Enable Security: ☐
Embed: ☑
Query: ${page.params["q"]}
Hits per page: 10
Name: q

| No. | Element | Description |
|---|---|---|
| 1 | Data Feeds | This panel lists the various feeds that you can configure and make available in the design of your application.<br><br>**Note:** The search field at the top of the panel lets you search in all feed categories. When a query is launched the **Data Feeds** panel turns into a **Search Results** panel displaying all corresponding entries. |
| 2 | Feed drop zone | To add a feed, drag and drop it from the panel to the drop zone. |
| 3 | Subfeed drop zone | To add a subfeed, that is to say a feed depending on a parent feed, drag and drop it from the panel to the drop zone. |
| 4 | Feed properties | Each feed has a set of properties which define its behavior. For most properties, you can click in the field to display a contextual menu listing all available Values and MEL Functions on the left. |
| 5 | Feed tools | To disable, copy or delete the feed. |
| 6 | Page Parameters | To declare reusable parameters for your page. Once declared, parameters are available in the Values tab of the contextual menu which opens when you click in the Feed properties fields.<br>You can notice that by default:<br>The **index** page does not contain any parameters. As it is the required home page of the search application, this page is static.<br>Unlike the **index** page, the **search** page is dynamic and takes a parameter `q` (our query). |

| No. | Element | Description |
|---|---|---|
| | | **Note:** In your Mashup Builder, you can notice that the page parameter is shown in the URL field, between a question mark and an equal sign. For example, `search?q=` means that on the `search` page, the `q` parameter is passed with the value defined after the = sign. You will also frequently see several page parameters in URLs, separated by ampersands (`&`). |

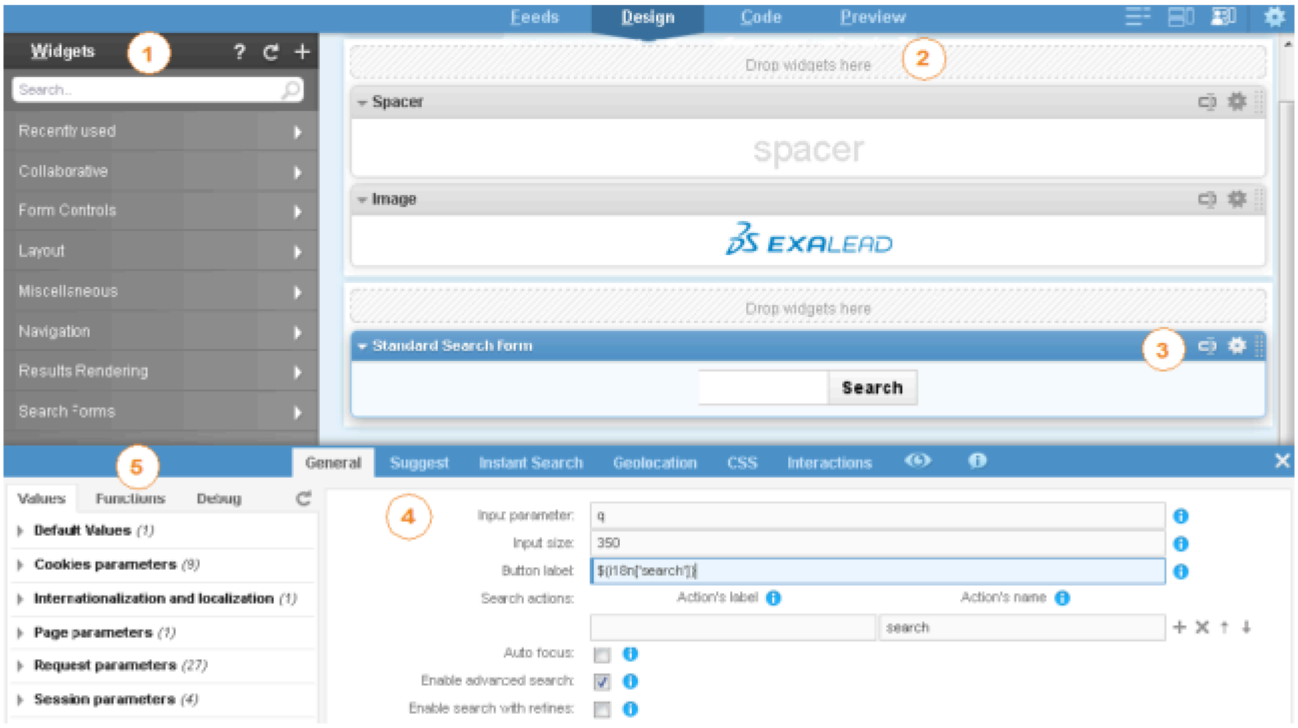
## Design view

The **Design** view allows you to design your page and integrate the feeds defined in the Feeds view. Adding widgets to the **Design** view can be done with simple drag-and-drops from the **Widgets** panel into cells. You can then select the feeds that will be called by each widget.

Each page is structured as a table, with columns, rows and cells. To build your page, select the **Page Layout** panel and add as many columns, rows and cells as you need.

### Widgets

The following screenshot and its accompanying legends describe the main elements of the **Design > Widgets** view.



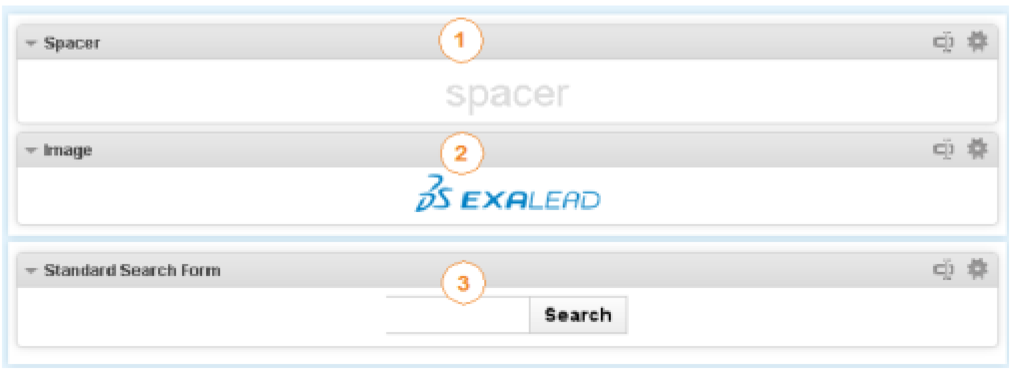
Input parameter: q
Input size: 350
Button label: ${i18n['search']}
Search actions: Action's label / Action's name: search
Auto focus: ☐
Enable advanced search: ✓
Enable search with refines: ☐

| No. | Element | Description |
|---|---|---|
| 1 | Widgets | The **Widgets** panel contains many standard widgets grouped by categories. You can easily drag and drop widgets on the **Design** view to create your own page(s). |

| No. | Element | Description |
|---|---|---|
| | | To configure a widget, you must define its properties (see No. 4).<br>The search field at the top of the panel lets you search in all widget categories. When a query is launched the **Widgets** panel turns into a **Search Results** panel displaying all corresponding entries.<br>For widget descriptions, see the *Widget Reference*. |
| 2 | Widget drop zone | To add a widget, drag and drop it from the panel to the drop zone. |
| 3 | Widget tools | To edit the widget properties, annotate the widget title, use the widget to create a composite widget, and disable, delete, copy or delete the widget. |
| 4 | Widget properties | To configure a widget, just click its header. A panel opens at the bottom of the screen to let you define the widget properties.<br>For example, in a search form, the main properties are:<br>**Input parameter**: Parameter name of the input search form that will be sent to the action page (just like in a standard HTML form).<br>**Action**: Name of the page where the form should send parameters. If you click on the field, you will see a list of possible values (**pages**) on the left. |
| 5 | Contextual menu | Provides the available widget configuration options depending on the content of the option (for example, when a facet is required, it will list the available facets for the selected feed(s)). |

## Index page widgets

The following screenshot and its accompanying table describe the default widgets of the **index** page **Design** view.

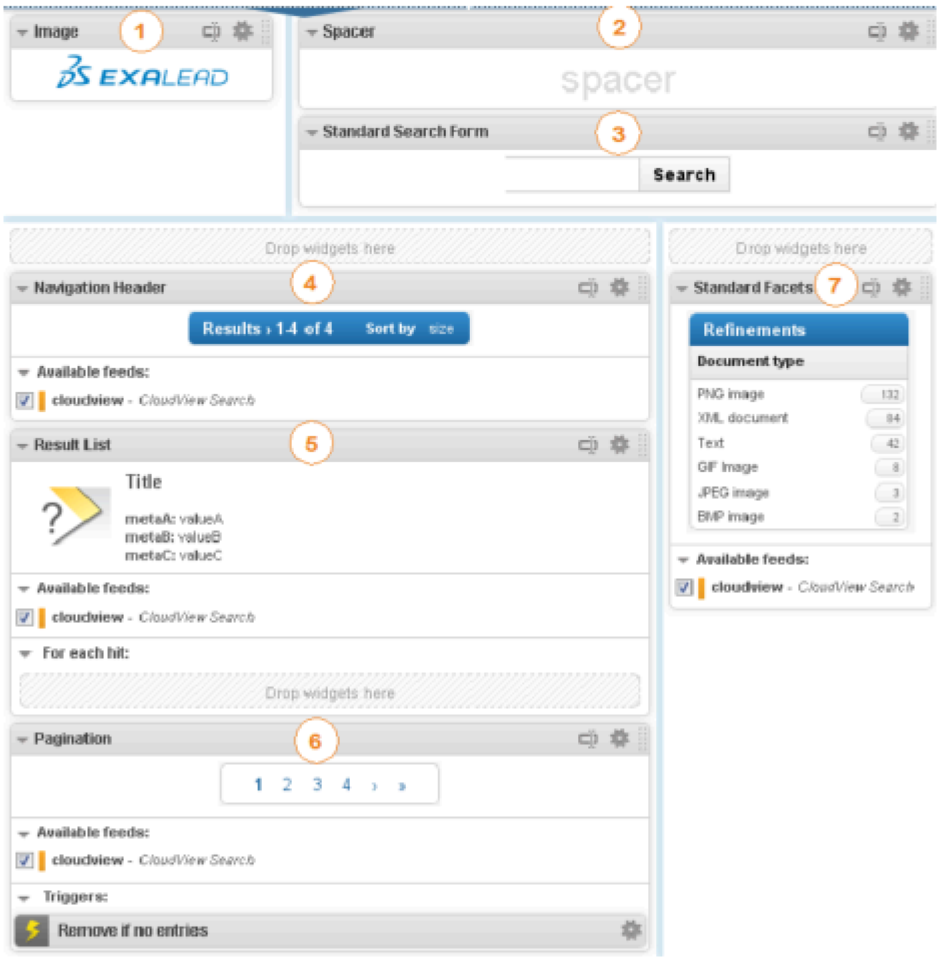


| No. | Widget | Description |
|---|---|---|
| 1 | **Spacer** | An empty block to center the search form with the logo. |

| No. | Widget | Description |
|---|---|---|
| 2 | **Image** | Exalead logo. |
| 3 | **Standard Search form** | Our standard search form made of a free-form field and a Search button. |

## Search page widgets

The following screenshot describes and its accompanying table describe the default widgets of the **search** page **Design** view.

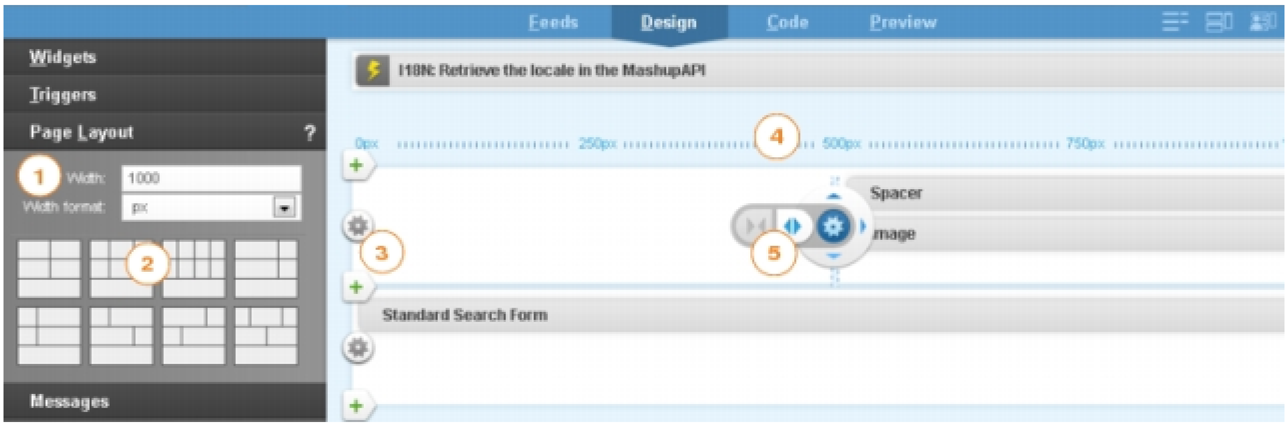


| No. | Widget | Description |
|---|---|---|
| 1 | **Image** | A small Exalead logo. |
| 2 | **Spacer** | An empty block to center the search form. |
| 3 | **Standard Search form** | Exactly the same search form as on the index page. |
| 4 | **Navigation Header** | Our first dynamic widget, displaying information on the number of hits and on the number of pages. |

| No. | Widget | Description |
|---|---|---|
| 5 | **Result List** | A dynamic widget displaying the hits (results) of the feed, in our case the **CloudView Search** feed. |
| 6 | **Pagination** | Displays the pagination for the given feeds, in our case the **CloudView Search** feed. |
| 7 | **Standard Facets** | The refinements widget that is configured to display facets from the feed, in our case the **CloudView Search** feed. |

## Page Layout

The following screenshot and its accompanying legends describe the main elements of the **Design > Page Layout** view.



| No. | Element | Description |
|---|---|---|
| 1 | Width and width format | Specify the width and the width format (pixels or percentage) of the page. |
| 2 | Predefined Layouts | You can select between a variety of different row layouts depending on your needs, or use the row tools (3 & 5) to add cells above or below. |
| 3 | Add row (+) | To add a row below. |
| 4 | Horizontal ruler | This ruler allows you to measure the widgets' width. The units of measurement are expressed in pixels or percentage, depending on the selected **Width format**. |
| 5 | Cell designer | The cell designer allows you to split or merge the current cell. |

## Code view

Use the **Code** view to customize an application page by adding custom CSS or JavaScript code.

In:

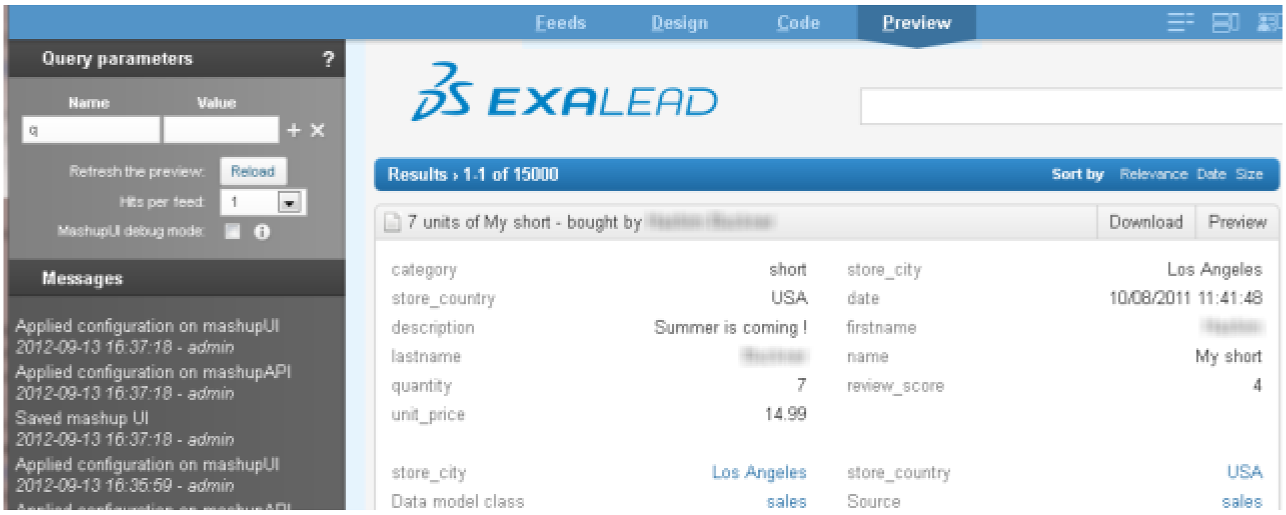
- **Styles**: Enter the inlined CSS code that you want to use on the page.
- **JavaScript**: Enter the JavaScript code that you want to use on the page.

## The Preview

The **Preview** allows you to verify the behavior and the look and feel of your page.

It is useful to tune your page and avoid saving and applying the whole configuration to see its results on the Mashup Builder.

**Note:** When you select the **Preview**, a Save action is performed in the background.



Applied configuration on mashupUI
2012-09-13 16:37:18 - admin
Applied configuration on mashupAPI
2012-09-13 16:37:18 - admin
Saved mashup UI
2012-09-13 16:37:18 - admin
Applied configuration on mashupUI
2012-09-13 16:35:59 - admin

7 units of My short - bought by [redacted]

| | | | |
|---|---|---|---|
| category | short | store_city | Los Angeles |
| store_country | USA | date | 10/08/2011 11:41:48 |
| description | Summer is coming ! | firstname | [redacted] |
| lastname | [redacted] | name | My short |
| quantity | 7 | review_score | 4 |
| unit_price | 14.99 | | |
| store_city | Los Angeles | store_country | USA |
| Data model class | sales | Source | sales |

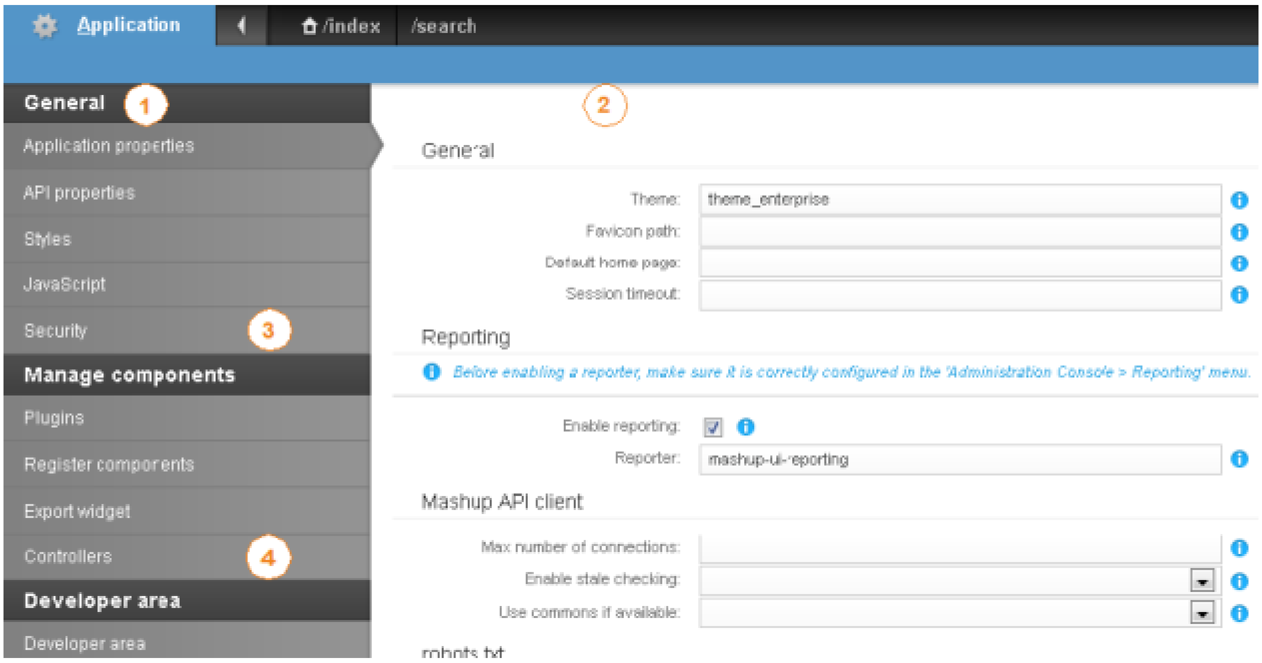
The **Query parameters** panel lets you enter other parameters than the default `q` parameter. You can test the behavior of your feed queries as defined in the **Query** properties of your feeds.

## Application tools

Mashup Builder contains a set of application tools allowing to configure settings globally on all your application pages.

You can access application tools by clicking the **Application** button at the top left of the screen.

The following screenshot and its accompanying legends give an overview of the application tools.



Theme: theme_enterprise
Favicon path:
Default home page:
Session timeout:

Before enabling a reporter, make sure it is correctly configured in the 'Administration Console > Reporting' menu.

Enable reporting: ✓
Reporter: mashup-ui-reporting

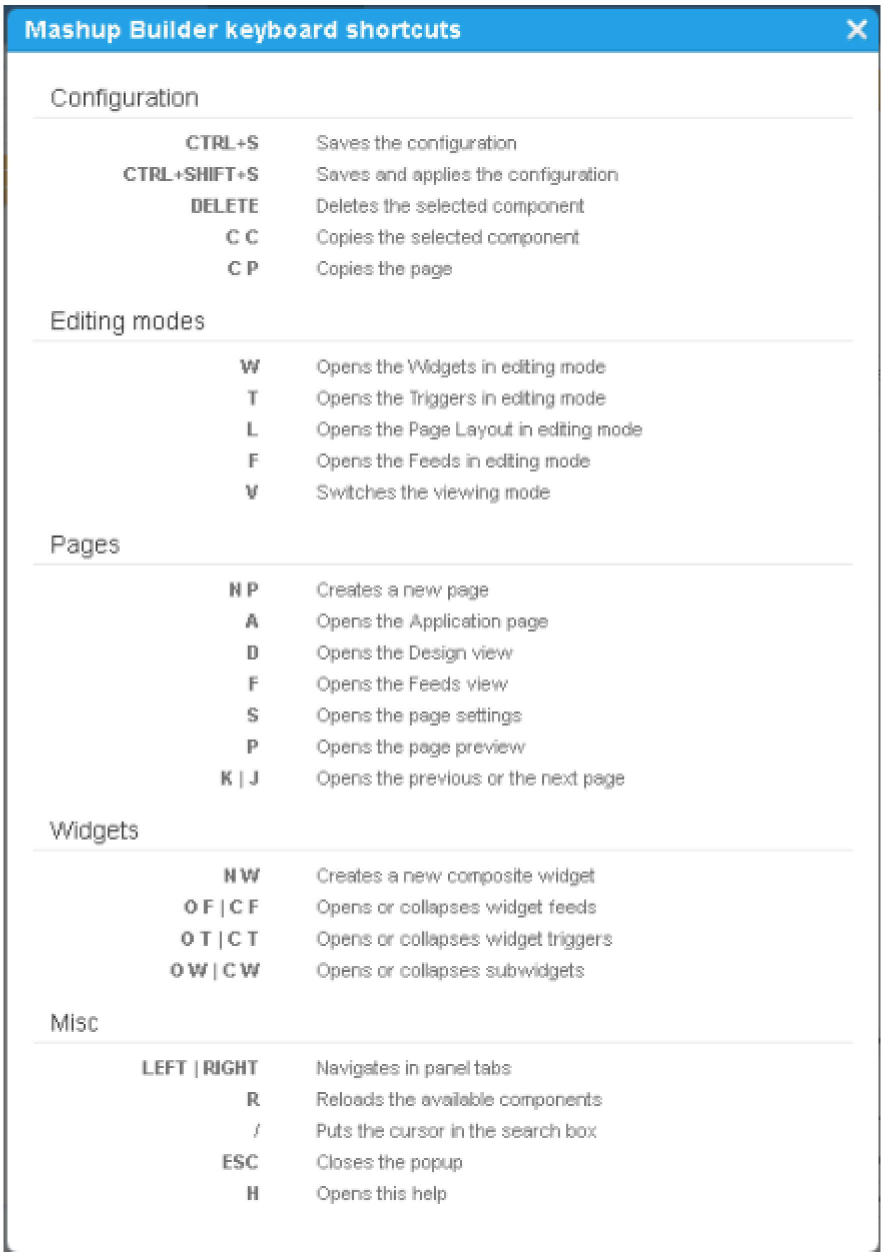
Max number of connections:
Enable stale checking:
Use commons if available:

| No. | Element | Description |
|---|---|---|
| 1 | General tools | These tools allow you to parameter general settings for your mashup application.<br>You can:<br>• Tune the overall look and feel, reference CSS and javascript.<br>• Set up security. |
| 2 | Application Properties view | You can configure applications properties to control the behavior of your current application. For example, you can specify the default home page of the application, the icon to display in the URL bar (favicon), the theme to use, etc. |
| 3 | Manage components | Mashup Builder Premium gives you the possibility of adding plugins for feeds, widgets and triggers. You can therefore install and deploy custom components on your applications. |
| 4 | Developer area | The Developer area includes several tools that can be useful to develop and debug your own search applications. For more information, see "Using Developer Tools" in the Exalead CloudView Mashup Programmer's Guide. |

## Keyboard Shortcuts

Mashup Builder includes many keyboard shortcuts allowing you to get a quick access to features.

Press **H** to display the Keyboard Shortcuts dialog box that lists all shortcuts.



Mashup Builder keyboard shortcuts

Configuration

| | |
|---|---|
| CTRL+S | Saves the configuration |
| CTRL+SHIFT+S | Saves and applies the configuration |
| DELETE | Deletes the selected component |
| C C | Copies the selected component |
| C P | Copies the page |

Editing modes

| | |
|---|---|
| W | Opens the Widgets in editing mode |
| T | Opens the Triggers in editing mode |
| L | Opens the Page Layout in editing mode |
| F | Opens the Feeds in editing mode |
| V | Switches the viewing mode |

Pages

| | |
|---|---|
| N P | Creates a new page |
| A | Opens the Application page |
| D | Opens the Design view |
| F | Opens the Feeds view |
| S | Opens the page settings |
| P | Opens the page preview |
| K \| J | Opens the previous or the next page |

Widgets

| | |
|---|---|
| N W | Creates a new composite widget |
| O F \| C F | Opens or collapses widget feeds |
| O T \| C T | Opens or collapses widget triggers |
| O W \| C W | Opens or collapses subwidgets |

Misc

| | |
|---|---|
| LEFT \| RIGHT | Navigates in panel tabs |
| R | Reloads the available components |
| / | Puts the cursor in the search box |
| ESC | Closes the popup |
| H | Opens this help |

# Building Mashup Applications

This section describes how to build your own mashup applications within Mashup Builder.

The behavior and the layout of your applications can be fully designed using the various functionalities of the **Edit <application name> application** menu. You can add and configure feeds and widgets, create pages, add triggers, customize the look and feel of your application, etc.

## Adding Feeds

The **Feeds** view lets you add and configure the feeds (data sources) that will then be available in the **Design** view, to be called by widgets.

## About Mashup Builder feeds

The following table describes the various Feeds available in Mashup Builder Premium edition. The Mashup Builder edition only contains the Exalead CloudView Search and Exalead CloudView Data Model feeds.

*Mashup Builder Premium Edition - Feeds*

| Category - Feeds | Description |
|---|---|
| **CloudView Feeds** | |
| **CloudView Data Model** | This feed is linked to the data model defined in the Administration Console. Its search capabilities are therefore enhanced by the classes, semantic types, advanced processors, etc. used in the data model. |
| **CloudView Search** | This is the standard Exalead CloudView search feed.<br>It is independent from the data model defined in the Administration Console. It works for search on versions V6R2012x and higher. |
| **Database Feeds** | |
| **Advanced JDBC Query** | This feed allows you to query a JDBC database directly.<br>You need to specify the following SQL queries:<br>• One to retrieve results with both a `LIMIT` and an `OFFSET` clauses (for pagination).<br>• One to retrieve the total number of results.<br>• One to retrieve only one result (when a hit is opened directly via its URL). |
| **Drupal Query** | This feed allows you to query a drupal database (Opens Source CMS tool) directly. |
| **JDBC Query** | This feed allows you to query a JDBC database directly.<br>You need to specify a single SQL query to retrieve results. |
| **Feed Tools** | |
| **Concat Feeds** | This feed must gather one or more subfeeds (nested feeds).<br>It merges the hits of feeds based on their IDs. A comparison is performed and if:<br>• IDs are equal, only one hit is displayed for the two feeds.<br>• IDs are different, all hits are displayed. |

| Category - Feeds | Description |
|---|---|
| | **Note:** Concat feeds do not support synthesis. You cannot use this feed on widgets rendering synthesis, for example, the Pie Chart widget. |
| **Joined Feed** | This feed must gather one or more subfeeds (nested feeds). It combines the hits of nested feeds based on a join key meta. A comparison is performed and if: • Keys are equal, only one hit is displayed for the two feeds. • Keys are different, all hits are displayed. |

## Add a feed

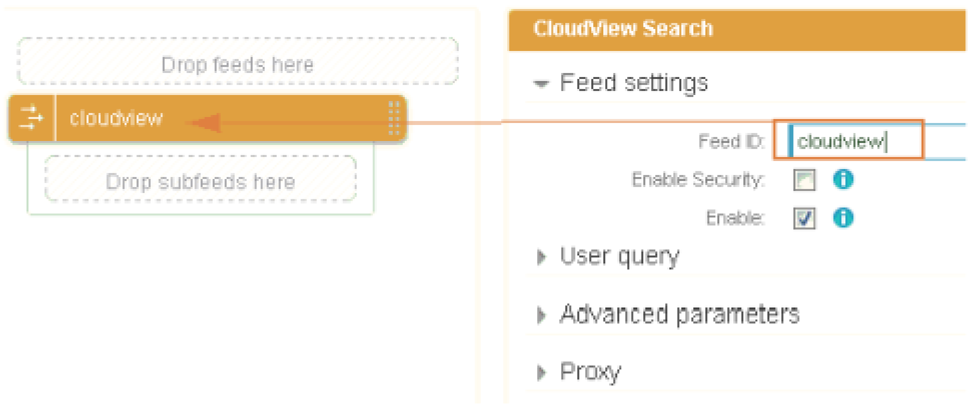
The **CloudView Search** feed is usually at the very heart of your application. As already stated, it is independent from the data model defined in the Administration Console.

Everything that can be done using the Search Logic, can also be done at query time by the **CloudView Search** and the **CloudView Data model** feeds. Using these feeds to perform the query is a way to capitalize the default Search Logic when you have specific needs. For example, if you want to call Exalead CloudView with a few parameters that are different from those defined in the default Search Logic, you don't need to create another Search Logic and you can just edit these parameters at the feed level instead.

This section describes how to add a **CloudView Search** feed and configure its main properties.

1. In Mashup Builder, select an application page, for example, the **index** page.
2. Select the **Feeds** view.
3. Drag a **CloudView Search** feed in the drop zone.
4. In the Feed properties panel, expand the **Feed settings** section and using the Feed ID field, call it `cloudview`.

   The feed now displays `cloudview` in the drop zone.



Feed ID: cloudview
Enable Security: ☐
Enable: ☑

5. Set its properties as needed. The following table describes the main properties of the **CloudView Search** feed.

| Property | Description |
|---|---|
| **Query** | The dynamic query to send to the index.<br>For example the `${page.params["q"]}` variable stands for parameter q applied to the current page.<br>**Note:** Using `#all` queries, to retrieve all documents, is not recommended since it can impact the performances of the search server. |
| **Hits per page** | Number of hits (results) to display per page. |
| **Default User Query** | The default query executed when the **Query** parameter is empty. |
| **Properties/ Mappings** | The output format of the Mashup API is a standard XML output format, Atom .<br>This format has specific tags for defining the title or the thumbnail of an entry, this is why these parameters can be configured at the feed level, but they can also be configured on a hit displaying widget.<br>PRE: Maps the property before subfeeds execution.<br>POST: Maps the property after subfeeds execution. |

6. Click **Save**.

## Make parallel requests with feeds

Creating parallel requests is useful if you have different indexes or if you want to search for specific content in the same index using different queries.

The following use case describes how to proceed if you want your application to fetch hits from two sources using the same query basis:

- One of these source is a filesystem crawl retrieving data through a **Files** connector. In our example, the set of documents located under the `<INSTALLDIR>/docs/` directory.
- The other one is a web crawl retrieving data through an **HTTP** connector. In our example, the Exalead website (`http://www.exalead.com/software/`).

Our application is configured in the Mashup Builder to use two **CloudView Search** feeds that will select the 2 most relevant hits for the request on the search results page `${page.params["q"]}` to focus on the two specific data sources.

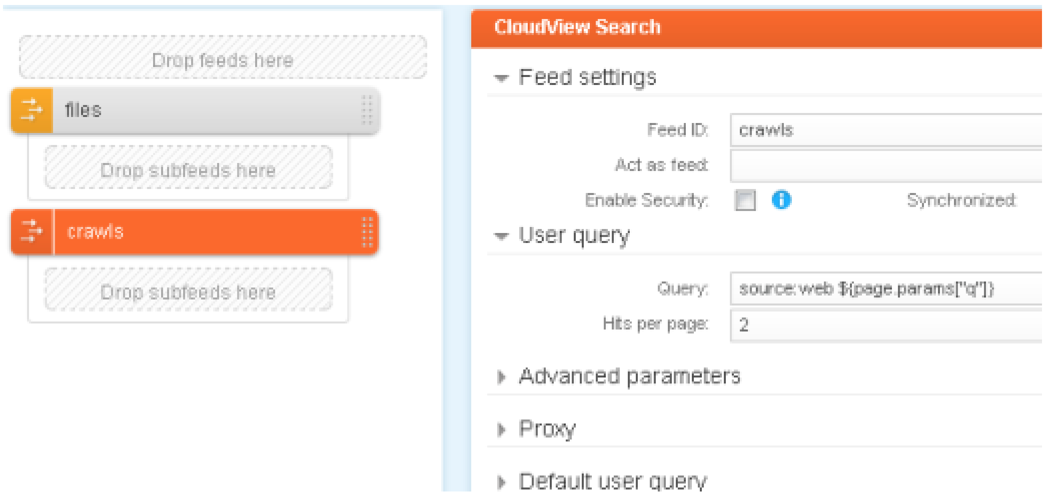
By making a query on the Mashup Builder, we see below that the search page displays results coming from the two data sources; one from an Exalead website URL and another from the filesystem.

Using more than one feed per page is only possible with Mashup Builder Premium.

This procedure describes the steps to create the example presented above.

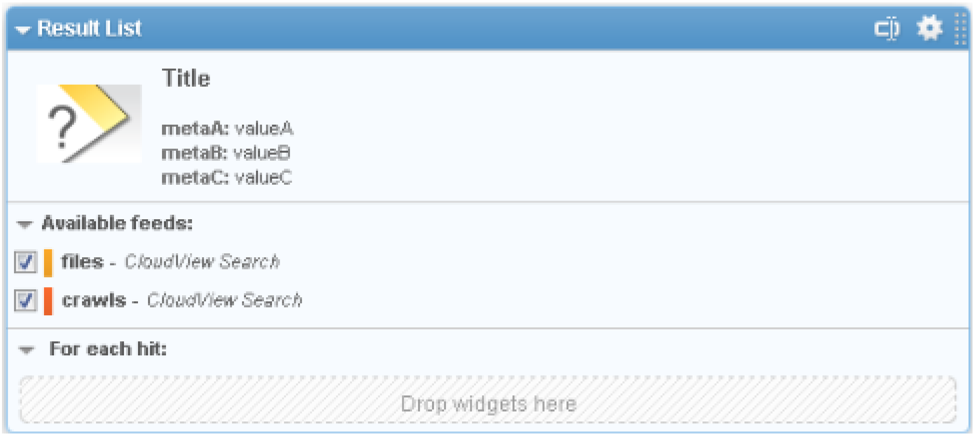
1. In **Mashup Builder**, open one of your application pages, for example **search**, and select the **Feeds** view.
2. Drag and drop a **CloudView Search** feed, and from the **Feed ID** field, call it `files`.
3. Set the feed properties as follows:
   a. For **User query > Query**, enter for example `source:fs ${page.params["q"]}` (where `fs` stands for a file system source managed by a Files connector).
   b. For **User query > Hits per page**, enter `2`.
   c. Under **Properties / Mapping**, for **TITLE**, enter `${entry.metas["file_name"]}`
4. Drag and drop another **CloudView Search** feed below the first one, and call it `crawls`.
5. Set the feed properties as follows:
   a. For **User query > Query**, enter for example `source:web ${page.params["q"]}` (where `web` stands for a web source managed by an HTTP connector).
   b. For **User query > Hits per page**, enter `2`.

   The configuration should look as follows.



Feed ID: crawls
Act as feed:
Enable Security: ☐ Synchronized
Query: source:web ${page.params["q"]}
Hits per page: 2

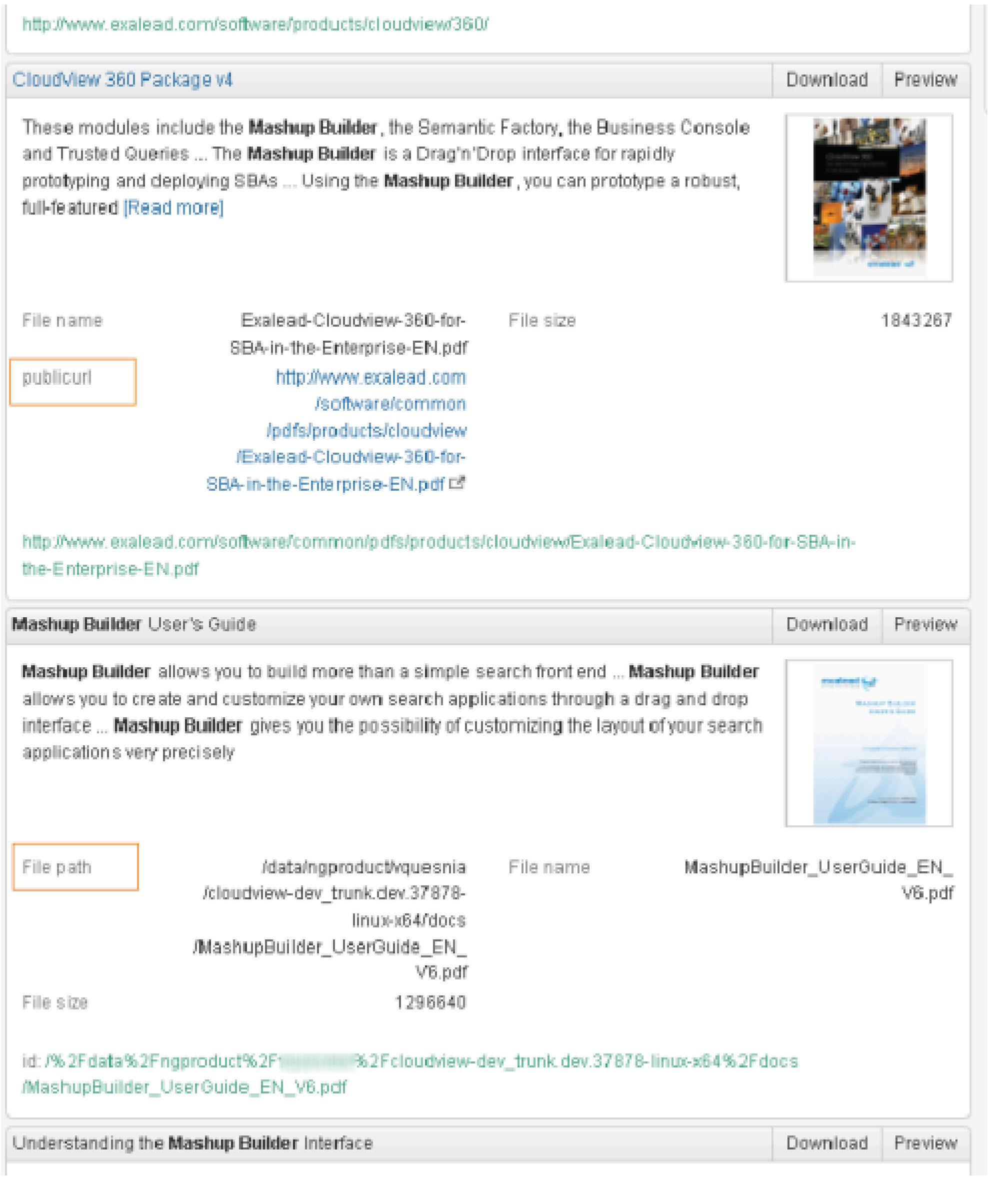
6. Select the **Design** view, and add the two feeds (`files` and `crawls`) to a widget, for example **Result List**.



7. Select the **Preview** to check your configuration changes.
8. Click **Apply**.

The Mashup Builder displays results coming from the 2 data sources.

*Example of parallel requests on the Mashup Builder*



http://www.exalead.com/software/products/cloudview360/

CloudView 360 Package v4 | Download | Preview

These modules include the **Mashup Builder**, the Semantic Factory, the Business Console and Trusted Queries ... The **Mashup Builder** is a Drag'n'Drop interface for rapidly prototyping and deploying SBAs ... Using the **Mashup Builder**, you can prototype a robust, full-featured [Read more]

| | | | |
|---|---|---|---|
| File name | Exalead-Cloudview-360-for-SBA-in-the-Enterprise-EN.pdf | File size | 1843267 |
| publicurl | http://www.exalead.com/software/common/pdfs/products/cloudview/Exalead-Cloudview-360-for-SBA-in-the-Enterprise-EN.pdf | | |

http://www.exalead.com/software/common/pdfs/products/cloudview/Exalead-Cloudview-360-for-SBA-in-the-Enterprise-EN.pdf

**Mashup Builder** User's Guide | Download | Preview

**Mashup Builder** allows you to build more than a simple search front end ... **Mashup Builder** allows you to create and customize your own search applications through a drag and drop interface ... **Mashup Builder** gives you the possibility of customizing the layout of your search applications very precisely

| | | | |
|---|---|---|---|
| File path | /data/ngproduct/vquesnia/cloudview-dev_trunk.dev.37878-linux-x64/docs/MashupBuilder_UserGuide_EN_V6.pdf | File name | MashupBuilder_UserGuide_EN_V6.pdf |
| File size | 1296640 | | |

id: /%2Fdata%2Fngproduct%2F[redacted]%2Fcloudview-dev_trunk.dev.37878-linux-x64%2Fdocs/MashupBuilder_UserGuide_EN_V6.pdf

Understanding the **Mashup Builder** Interface | Download | Preview

## Enrich hits with nested feeds

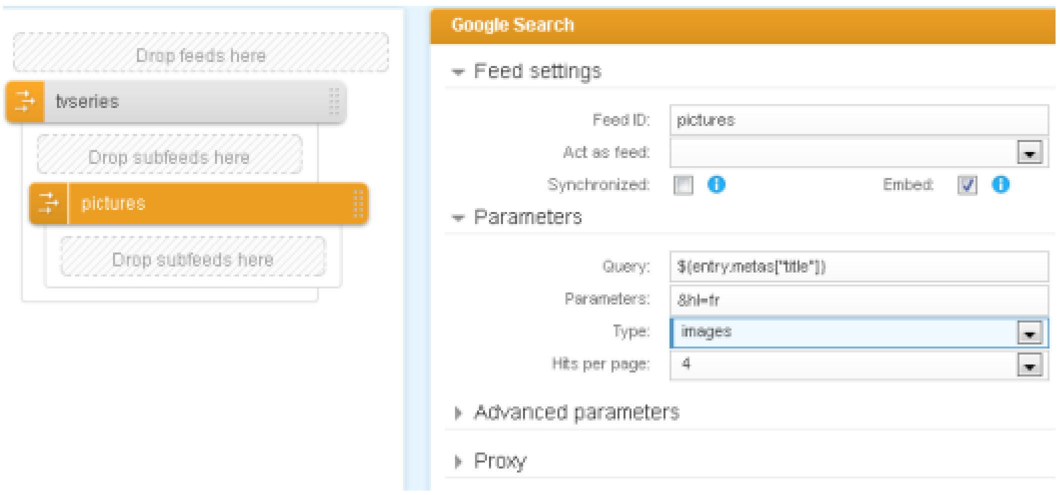
Nesting feeds in one another gives you the possibility of enriching the hits of a query with other sources to bring related content to the user. This can be achieved because nested feeds can use parent metas.

In the following use case, our application starts by retrieving information about TV series using a **CloudView Search** feed. It then tries to enrich each hit with images retrieved by a **Google Search** feed using the TV series title. We assume that the data source (`http://www.allocine.fr/series/`) is crawled by an HTTP connector called `series`, properly configured in the Administration Console.

Using more than one feed per page is only possible with Mashup Builder Premium.

1. In Mashup Builder, open one of your application pages, for example **search**, and select the **Feeds** view.

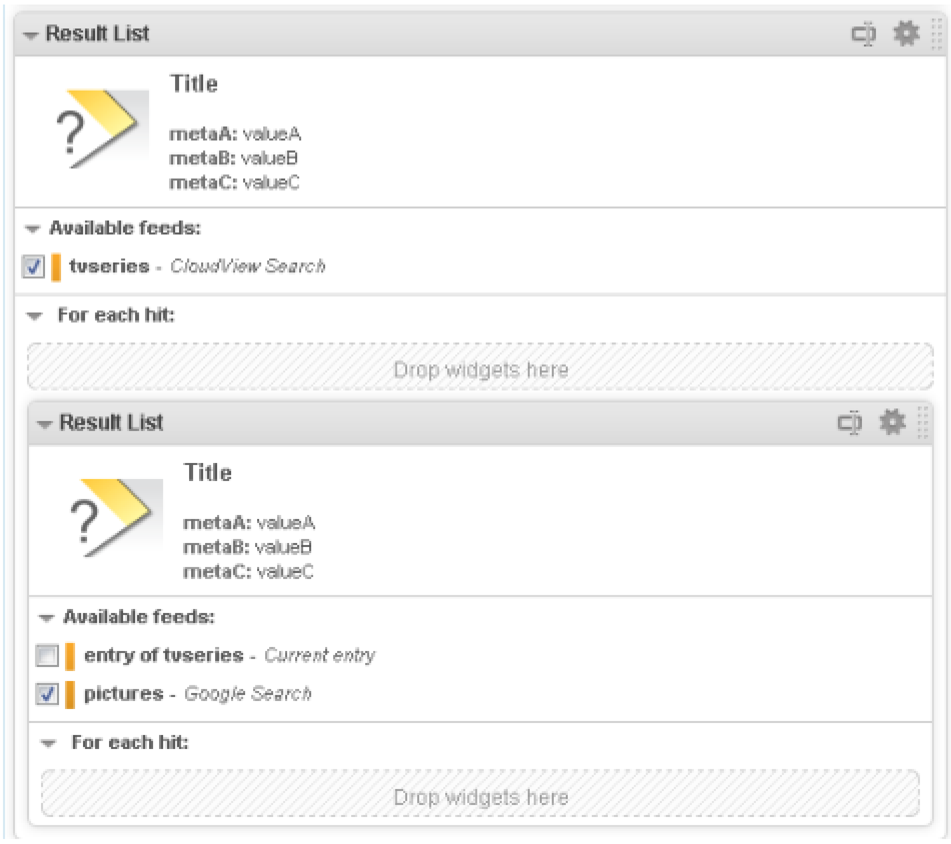
2. Drag and drop a **CloudView Search** feed and from the **Feed ID** field, call it `tvseries`.
3. Set the feed properties as follows:
   a. For **Query**, enter `source:series ${page.params["q"]}` where `source:series` specifies the name of the HTTP connector that crawls TV series in the Administration Console.
   b. For **Hits per page**, enter the number of hits to display on the search results page, for example `4`
4. Drag and drop an **Google Search** feed within the **CloudView Search** feed, and call it `pictures`.
5. Set the feed properties as follows:
   a. For **Query**, enter `${entry.metas["title"]}` (the title meta for each result of the `tvseries` **CloudView Search** feed).
   b. For **Type**, select **images**.



Feed ID: pictures
Act as feed:
Synchronized: ☐
Embed: ✓
Query: ${entry.metas["title"]}
Parameters: &hl=fr
Type: images
Hits per page: 4

6. Select the **Design** view, and add the parent feed (`tvseries`) to a widget, for example **Result List**.
7. Expand for **For each hit**, and drag and drop another widget within the parent widget.

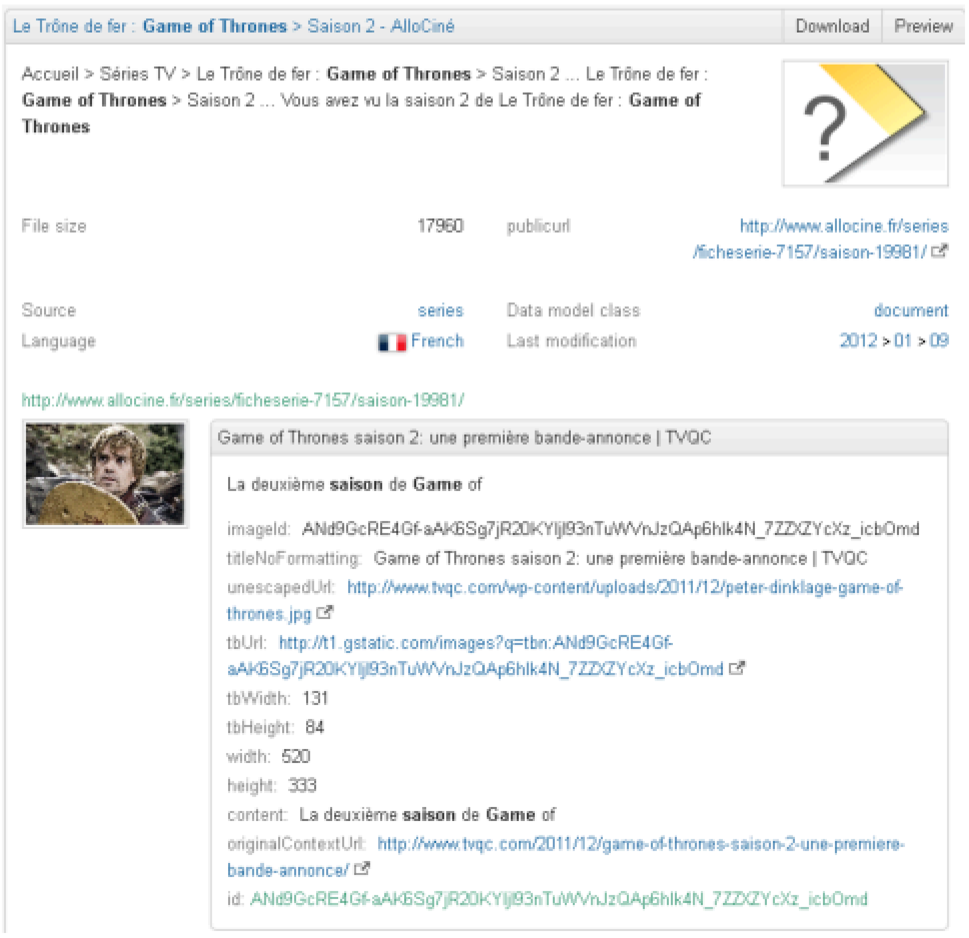
   By default the nested feed (`pictures`) is selected in the second widget.



8. Select the **Preview** to check your configuration changes.
9. Click **Apply**.

The Mashup UI displays results coming from the 2 data sources. The data retrieved by the HTTP connector is enriched by pictures retrieved by Google.

*Example of hit enrichment in Mashup UI*



Le Trône de fer : **Game of Thrones** > Saison 2 - AlloCiné | Download | Preview

Accueil > Séries TV > Le Trône de fer : **Game of Thrones** > Saison 2 ... Le Trône de fer : **Game of Thrones** > Saison 2 ... Vous avez vu la saison 2 de Le Trône de fer : **Game of Thrones**

| | | | |
|---|---|---|---|
| File size | 17960 | publicurl | http://www.allocine.fr/series/ficheserie-7157/saison-19981/ |
| Source | series | Data model class | document |
| Language | French | Last modification | 2012 > 01 > 09 |

http://www.allocine.fr/series/ficheserie-7157/saison-19981/

Game of Thrones saison 2: une première bande-annonce | TVQC

La deuxième **saison** de **Game** of

imageId: ANd9GcRE4Gf-aAK6Sg7jR20KYIjI93nTuWVnJzQAp6hIk4N_7ZZXZYcXz_icbOmd
titleNoFormatting: Game of Thrones saison 2: une première bande-annonce | TVQC
unescapedUrl: http://www.tvqc.com/wp-content/uploads/2011/12/peter-dinklage-game-of-thrones.jpg
tbUrl: http://t1.gstatic.com/images?q=tbn:ANd9GcRE4Gf-aAK6Sg7jR20KYIjI93nTuWVnJzQAp6hIk4N_7ZZXZYcXz_icbOmd
tbWidth: 131
tbHeight: 84
width: 520
height: 333
content: La deuxième **saison** de **Game** of
originalContextUrl: http://www.tvqc.com/2011/12/game-of-thrones-saison-2-une-premiere-bande-annonce/
id: ANd9GcRE4Gf-aAK6Sg7jR20KYIjI93nTuWVnJzQAp6hIk4N_7ZZXZYcXz_icbOmd

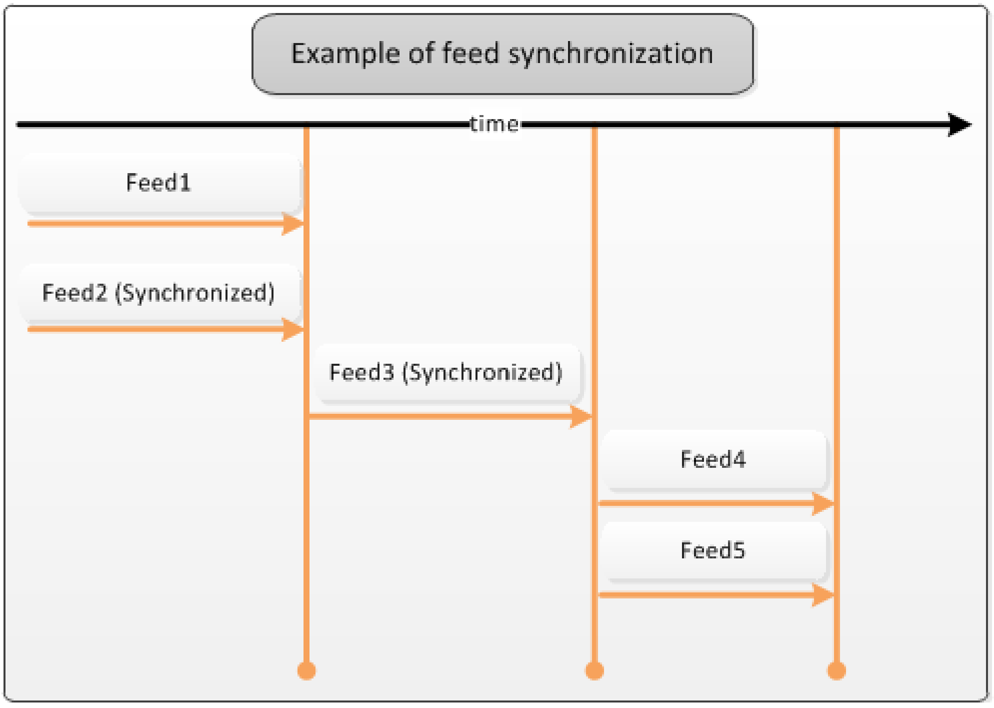
## Synchronizing feeds on a page

With Mashup Builder Premium, you can use more than one feed per page.

By default, feeds are executed in parallel but sometimes, you need to wait for the result from a previous feed to execute another feed. This can be done using the **Synchronized** option.

When you enable the synchronized option on one of your feeds, remember that feed order is important.

Feeds that are located after a synchronized feed will wait for it to answer before triggering themselves.



## Enable security on a Exalead CloudView Search feed

The following procedure describes how to enable security on a **CloudView Search** feed. When the user authenticates on a page, the security provider retrieves his tokens from the security source(s) which in turn, sends the user's query and the security tokens to the index.

The index only fetches documents that match both the user's query and the security tokens.

For example, if a user has the `Everybody` and `group1` tokens, and if security is enabled on the feed, Exalead CloudView will filter the documents and return only the documents matching `Everybody` and `group1`.

- A security source has been set in the Administration Console. For more information, see "Configuring security sources" in the Exalead CloudView Administration Guide.
- A security provider has been set, see Adding Security to Your Application.

1. In **Mashup Builder**, select an application page, for example, the **search** page.
2. Select the **Feeds** view and add two **CloudView Search** feeds.

   Call the first feed `secured` and the second feed `unsecured`.
3. For the first feed, select **Enable Security**.
4. Select the **Design** view, drag the **Table** widget, and drop two **Result List** widgets inside it.
   a. In the first **Result List** widget, select the `secured` feed.
   b. In the second **Result List** widget, select the `unsecured` feed.
5. Click **Save** and **Apply** the configuration.
6. Open the Mashup Builder search page:

- The secured result list shows only the documents matching the users' security tokens.
- The unsecured result list shows all documents, as security tokens are not taken into account.

## Adding Widgets

Once you have defined a feed in the **Feeds** view, you can add the widget(s) that will use this feed in the **Design** view.

Add widgets

Specify widget interactions

### Add widgets

Mashup Builder and particularly its premium version comes with many widgets grouped by type. This section focuses on three widgets, the Result Table, Pie Chart and Stacked Column Chart widgets. You will find examples using other widgets throughout this guide.

**Note:** Most chart widgets come from the highcharts and highstock libraries, we recommend reading their documentation on `http://www.highcharts.com/` for advanced configuration (in the **Javascript** tab of the chart widget properties).

#### Add a Result Table widget

This example shows how to display results in a **Result Table** widget, and set it to refine document search.

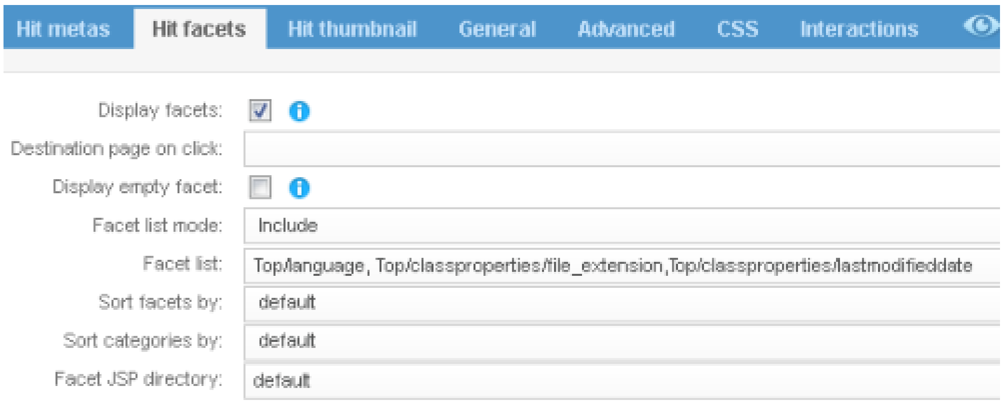
1. In **Mashup Builder**, go to the **search** page and select the **Design** view.
2. In the **Widgets** panel, expand the **Results Rendering** group, and drag the **Result Table** widget just above the **Result List** widget.
3. Click the **Result table** widget header.

   The widget properties panel opens at the bottom of the screen.
4. On the **Hit Config** tab:
   a. For **Hit title**, select **Hit Metas > title** from the **Values** contextual menu on the left.

      You will get `${entry.metas['title']}`
   b. For **Hit URL**, select **Hit Metas > url** from the **Values** contextual menu on the left.

      You will get `${entry.metas['url']!uh}`
5. On the **Hits metas** tab, deselect **Display metas**.

6. On the **Hit facets** tab, set the list of facets to include as columns in the table. For example, we want to get three columns displaying the File extension, the language, and the last modification of each document.
   a. For **Facet list mode**, select **include**.
   b. In the **Facet List** field that displays, select the facets to include in the Result Table, using the **Values** contextual menu on the left. These must be separated by commas. For example: `Top/language, Top/classproperties/file_extension,Top/classproperties/lastmodifieddate`



| | |
|---|---|
| Display facets: | ✓ |
| Destination page on click: | |
| Display empty facet: | ☐ |
| Facet list mode: | Include |
| Facet list: | Top/language, Top/classproperties/file_extension,Top/classproperties/lastmodifieddate |
| Sort facets by: | default |
| Sort categories by: | default |
| Facet JSP directory: | default |

7. Select the **Preview** to check your configuration changes.
8. Click **Apply**.

   To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

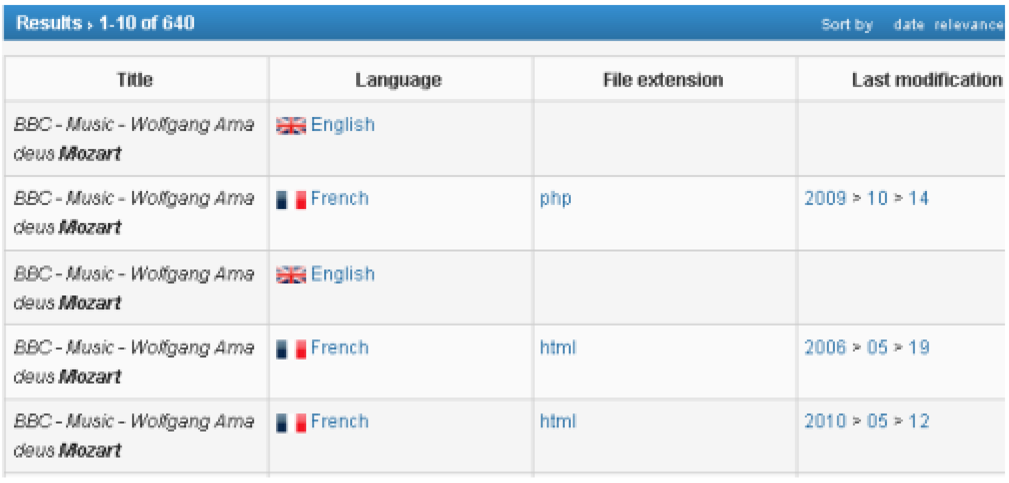
Your Mashup UI application should now have a search result page with a Result Table above the standard result list. You can use this table to refine your document search.

## Add a Pie Chart widget

In this example, we will add a **Pie Chart** widget to refine on result segments.

This example is based on the set of documents located under the `<INSTALLDIR>/docs/` directory.

*Example of Mashup UI application with Result Table*



Results › 1-10 of 640 Sort by date relevance

| Title | Language | File extension | Last modification |
|---|---|---|---|
| BBC - Music - Wolfgang Amadeus **Mozart** | English | | |
| BBC - Music - Wolfgang Amadeus **Mozart** | French | php | 2009 > 10 > 14 |
| BBC - Music - Wolfgang Amadeus **Mozart** | English | | |
| BBC - Music - Wolfgang Amadeus **Mozart** | French | html | 2006 > 05 > 19 |
| BBC - Music - Wolfgang Amadeus **Mozart** | French | html | 2010 > 05 > 12 |

This widget is available in Mashup Builder Premium only.

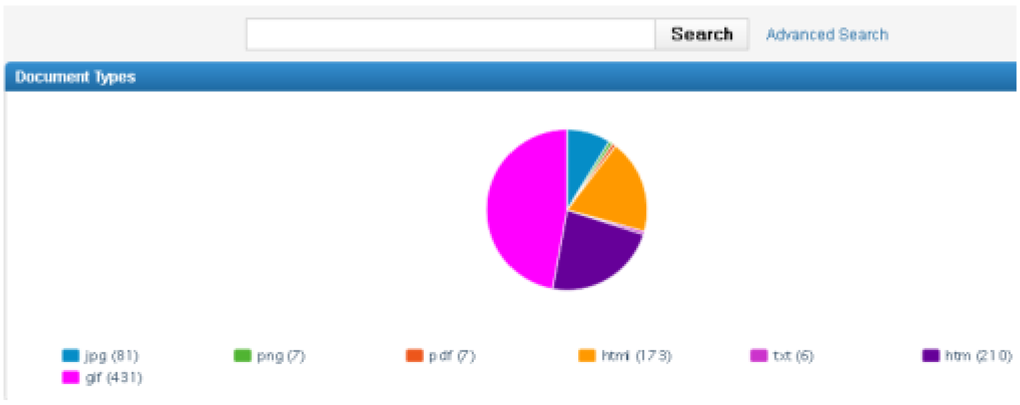
1. In **Mashup Builder**, go to the **index** page and select the **Design** view.
2. In the **Widgets** panel, expand the **Visualizations > Charts** group, and drag the **Pie Chart** widget to the last row of the canvas.
3. Click the **Pie Chart** widget header.

   The widget properties panel opens at the bottom of the screen.
4. From the **General** tab, set up the pie chart to launch a search with the corresponding refinement when you click a slice of the pie.
   a. Enter a **Widget title**, for example `Document Types`.
   b. In **Facet**, select the facet to refine on from the **Values** contextual menu on the left, for example **Top/classproperties/file_extension** to refine on document file extensions.
5. From the **Advanced** tab, set the search page as the page that will display the result corresponding to the slice selected on the pie chart.
   a. In **Destination page on click** select **search**.
   b. In **Max number of categories**, increase the default number if there are more than 5 categories (for example, `7`). This is to avoid having an **Other** category gathering all the categories that will not be displayed with the default value.
6. If you want to customize the colors used by the pie chart, go to the **JavaScript** tab and specify the HTML hexadecimal codes of your favorite colors.
   a. Uncomment the `colors` line by removing `/*` and `*/`
   b. Specify your favorite colors. For example, for the 7 categories we can define the following HTML colors: `colors: ['#058DC7', '#50B432', '#ED561B', '#FF9900', '#CC33CC', '#660099', '#FF00FF']`
7. Select the **Preview** to check your configuration changes.
8. Click **Apply**.

To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

Your Mashup UI application should now have a Home (**index**) page that resembles the following screenshot.

*Example of Mashup UI application with integrated pie chart.*



You can refine your search by selecting a pie chart section, and view its related results only.

## Add a Stacked Column Chart

This section gives several examples of **Stacked Column Chart** widget configuration:

- display multiple dimension in stacked bars
- display both bars and lines
- remove legends
- add labels to the chart axes

This widget is available in Mashup Builder Premium only.

### Display multiple dimensions in stacked bars

This example is based on the sales sample database located under the `<INSTALLDIR>/docs/sample_database` directory.

We assume you have followed the “Connect to the database” and “Create a new class in the data model” sections of the *CloudView Getting Started Guide*.

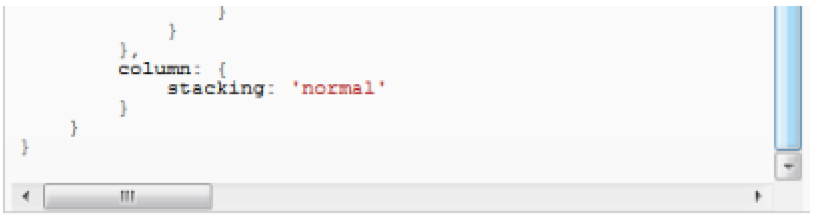
1. In the Administration Console, go to **Search Logics > Facets**.
2. Click **Add facet** and:
   a. For **Name**, enter `multidim`
   b. For **Type**, select **Multi-dimension**.
   c. Click **Accept**.
3. Add two dimensions to the `multidim` facet.

| Facet | Sort by |
|---|---|
| store_city | Count |
| category | Count |

4. Click **Apply**.
5. In **Mashup Builder**, drag the **Stacked Column Chart** widget on a page.
6. Click the widget header.

   The widget properties panel opens at the bottom of the screen.

7. On the **General** tab:
   a. For **Widget title**, enter `Multidim`.
   b. For **Facet type**, select **multi_dimension**.
   c. For **X**, select **multidim** from the **Values** contextual menu on the left.
8. Go to the **JavaScript** tab and uncomment the `/*stacking: 'normal'*/` line, to get `stacking: 'normal'`

```
                }
            }
        },
        column: {
            stacking: 'normal'
        }
    }
}
```

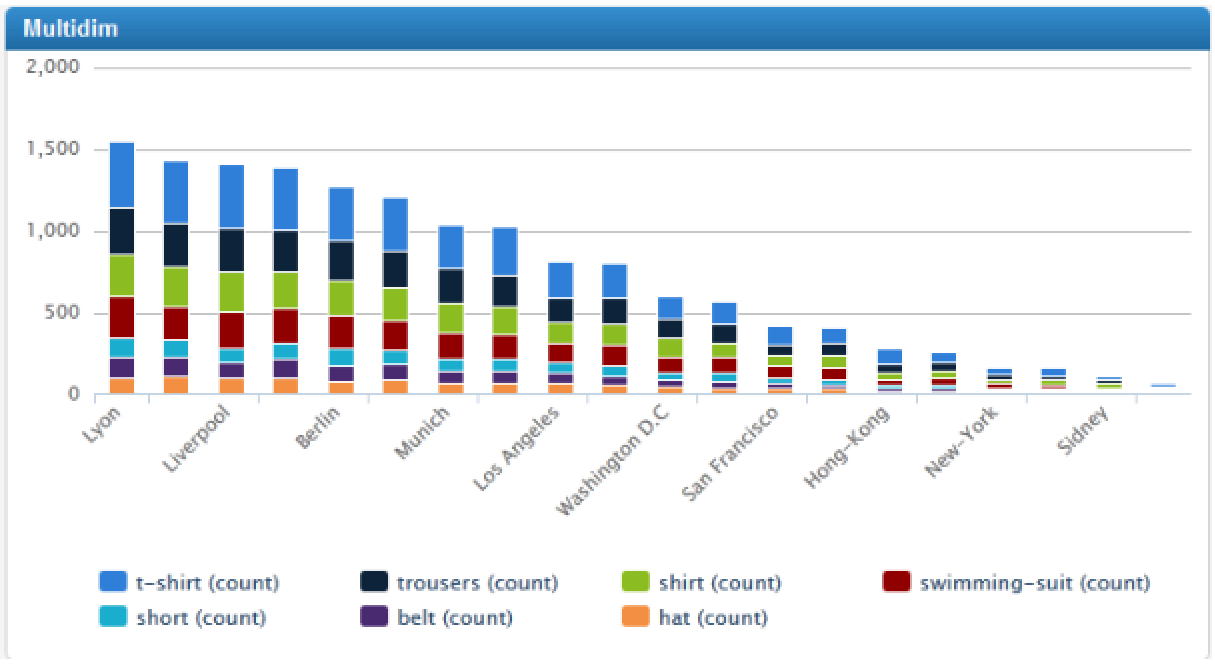
   **Note:** Press F11 to switch the code editor to fullscreen mode

9. Click **Apply**.

   To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

Your Mashup Builder application should now have a Chart that resembles the following screenshot.

*Display of multiple dimensions in stacked bar*



## Display both bars and lines

This example is based on the set of documents located under the `<INSTALLDIR>/docs/` directory

1. In the Administration Console, go to **Search Logics > Facets**.
2. Expand the **file_extension** facet and define the following aggregations:

| Id | Type | Expression |
|---|---|---|
| **file_size_avg** | **AVG** | document_file_size |
| **file_size_min** | **Min** | document_file_size |
| **file_size_max** | Max | document_file_size |

3. Click **Apply**.
4. In **Mashup Builder**, select a page and then select the **Design** view.
5. In the **Widgets** panel, select **Visualizations > Charts**, and drag the **Stacked Column Chart** widget on your page.
6. Click the widget header.

   The widget properties panel opens at the bottom of the screen.

7. On the **General** tab:
   a. For **Widget title**, enter `Bars & Lines`.
   b. For **X**, select **file_extension** from the **Values** contextual menu on the left .
   c. For **Y** , create the following series:

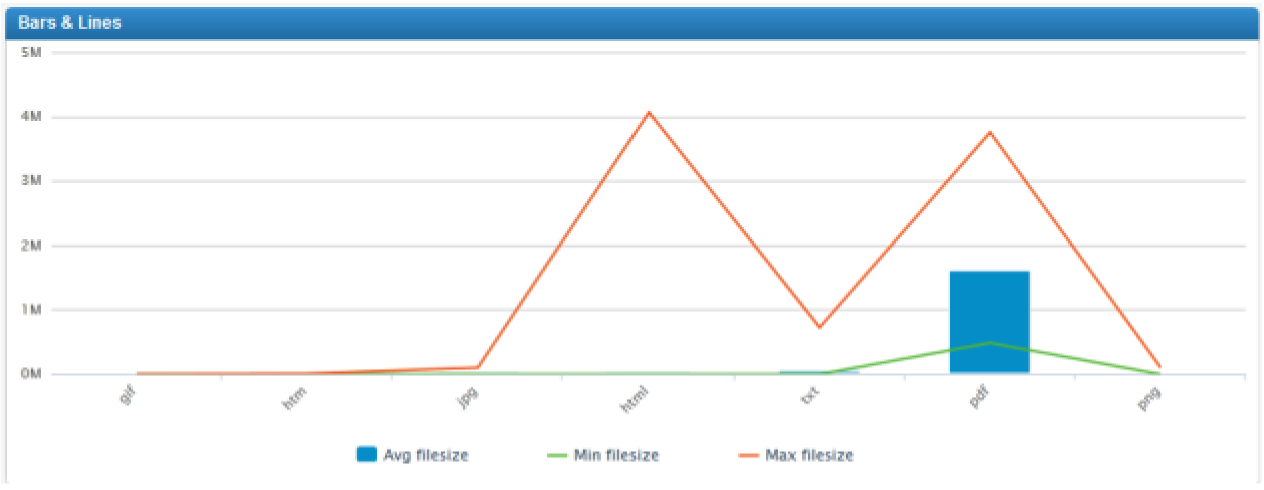
| Aggregation | Series type | Legend |
|---|---|---|
| file_size_avg | column | Avg file size |
| file_size_min | line | Min file size |
| file_size_max | line | Max file size |

8. Select the **Preview** to check your configuration changes.
9. Click **Apply**.

   To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

Your Mashup Builder application should now have a Chart that resembles the following screenshot.

*Display of bars and lines in Stacked Column Chart widget*



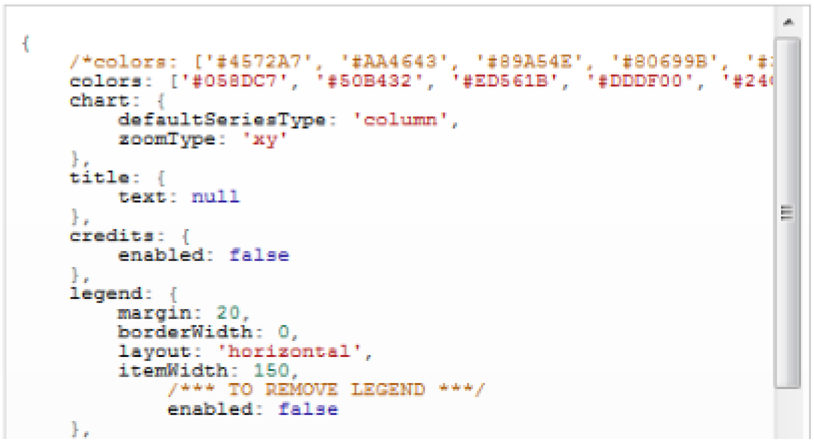
## Remove the chart legend

This example is based on the set of documents located under the `<INSTALLDIR>/docs/` directory.

1. In **Mashup Builder**, drag the **Stacked Column Chart** widget on a page.
2. Click the widget header.

   The widget properties panel opens at the bottom of the screen.

3. On the **General** tab:
   a. For **Widget title**, enter `No Legend`.
   b. For **Facet type**, select **normal**.
   c. For **X**, select **file_extension** from the **Values** contextual menu on the left.
4. Go to the **JavaScript** tab and in the `legend` node, set the enabled attribute to `false`.



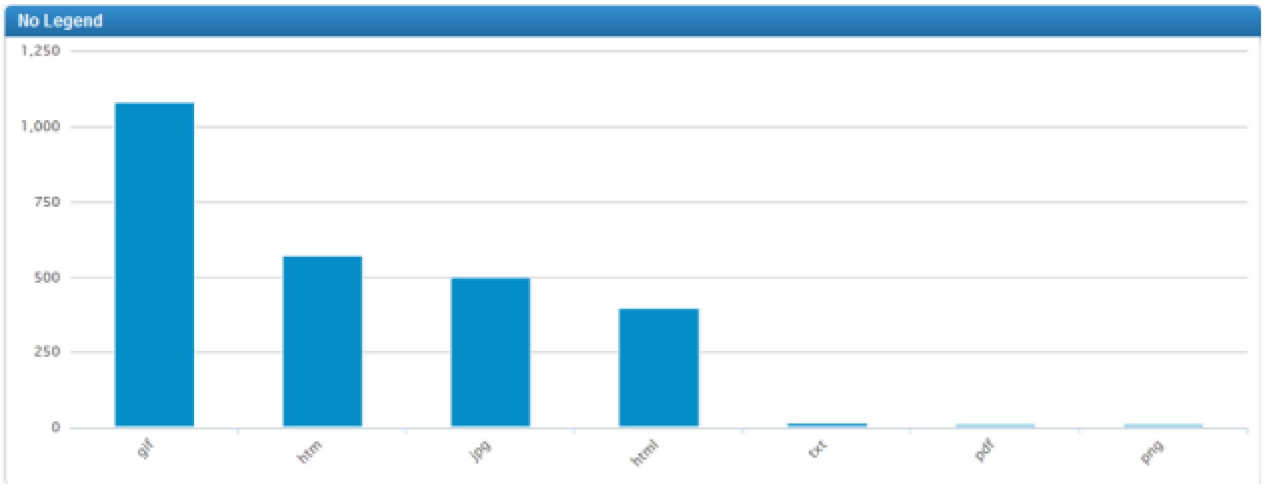
```
{
    /*colors: ['#4572A7', '#AA4643', '#89A54E', '#80699B', '#:
    colors: ['#058DC7', '#50B432', '#ED561B', '#DDDF00', '#24
    chart: {
        defaultSeriesType: 'column',
        zoomType: 'xy'
    },
    title: {
        text: null
    },
    credits: {
        enabled: false
    },
    legend: {
        margin: 20,
        borderWidth: 0,
        layout: 'horizontal',
        itemWidth: 150,
            /*** TO REMOVE LEGEND ***/
            enabled: false
    },
```

5. Select the **Preview** to check your configuration changes.
6. Click **Apply**.

   To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

Your Mashup Builder application should now have a Chart without legend, as shown in the following screenshot.

*Chart without legend*



## Add labels to the chart axes

This example is based on the set of documents located under the `<INSTALLDIR>/docs/` directory.

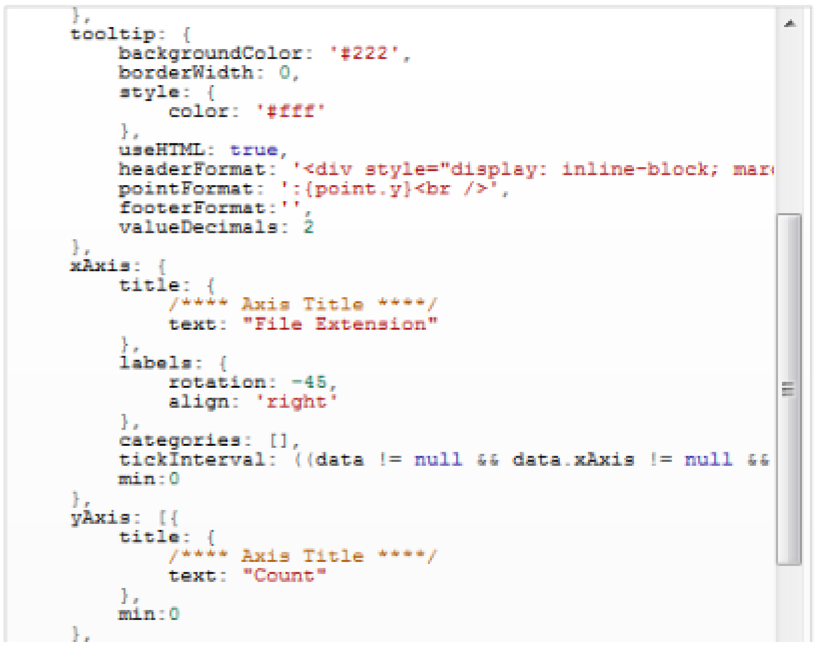
1. In **Mashup Builder**, drag the **Stacked Column Chart** widget on a page.
2. Click the widget header.

   The widget properties panel opens at the bottom of the screen.

3. On the **General** tab:
   a. For **Widget title**, enter `Labels on axes`.
   b. For **Facet type**, select **normal**.

c. For **X**, select **file_extension** from the **Values** contextual menu on the left.

4. Go to the **JavaScript** tab and enter the labels of your choice for the `text` attributes of the `xAxis` and `yAxis` nodes.



```
    },
    tooltip: {
        backgroundColor: '#222',
        borderWidth: 0,
        style: {
            color: '#fff'
        },
        useHTML: true,
        headerFormat: '<div style="display: inline-block; mar
        pointFormat: ':{point.y}<br />',
        footerFormat:'',
        valueDecimals: 2
    },
    xAxis: {
        title: {
            /**** Axis Title ****/
            text: "File Extension"
        },
        labels: {
            rotation: -45,
            align: 'right'
        },
        categories: [],
        tickInterval: ((data != null && data.xAxis != null &&
        min:0
    },
    yAxis: [{
        title: {
            /**** Axis Title ****/
            text: "Count"
        },
        min:0
    },
```

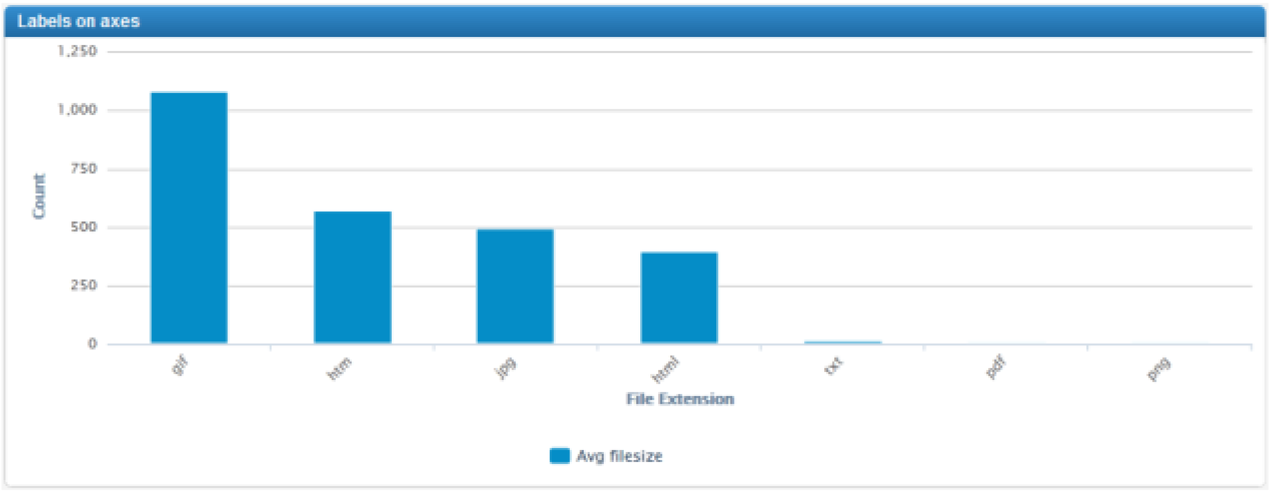
5. Select the **Preview** to check your configuration changes.

6. Click **Apply**.

   To see your changes, go to http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.

Your Mashup Builder application should now have a Chart with labels on axes, as shown in the following screenshot example.

*Chart with labels on axes*



## Specify widget interactions

Widget interactions allows certain widgets to interact with one another, for example, Result List, Google Map, Bookmarks etc. All these widgets contain an **Interactions** tab in their properties panel.

This feature provides:

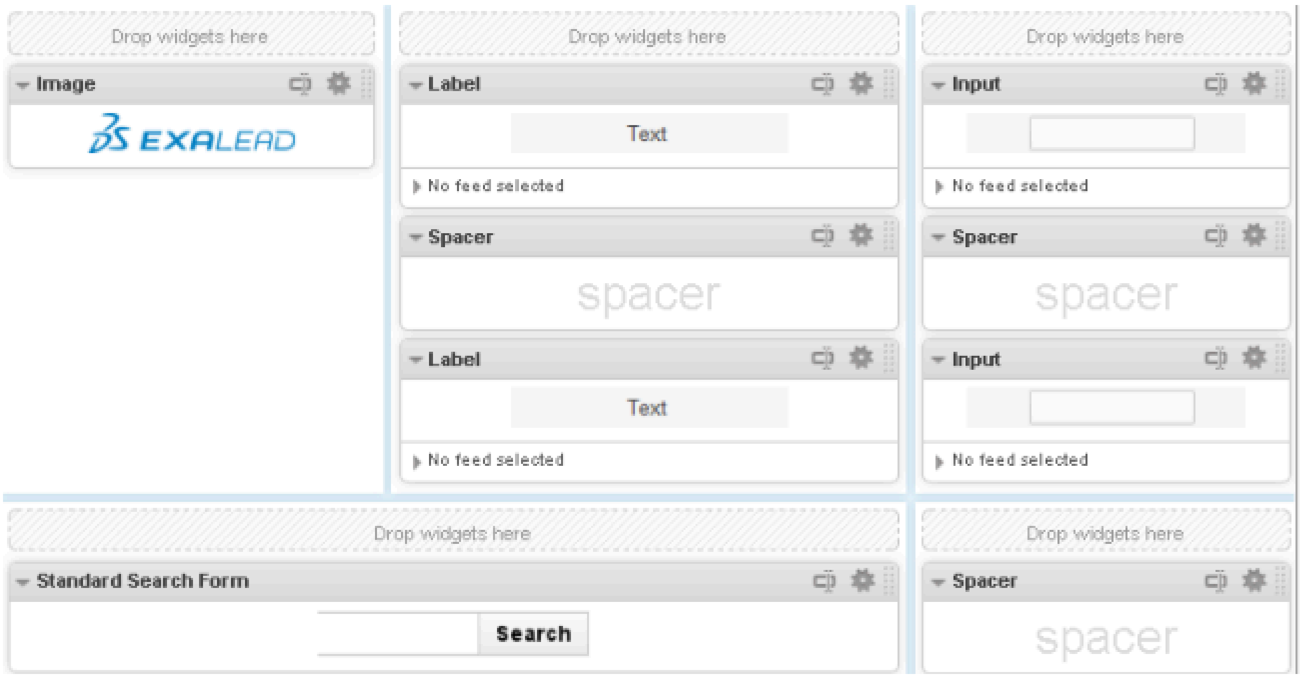
- a list of widgets on which you can create interactions.
- a field to specify the name of the widget that must be linked to the current widget, in **exa.io.**<property name> fields.

The following use case shows how to create a form with widget interactions. In this example we want to restrict the search results with two input widgets: one to select an author and one to select a start date using a datepicker. The aim is to link the input widgets to the Standard Search Form widget so as to submit the form with its Search button. The results will be filtered according to the selected author and start date.

1. In **Mashup Builder**, select a page, for example **/search**, and select the **Design** view.
2. From the **Widgets** panel, drag and drop widgets to assemble your form.

   In our example, we add a set of **Label** and **Input** widgets to create a form, as represented in the following table and screenshots.

| Row # | Label Widget | Input Widget |
|---|---|---|
| 1 | **Text**: `Author` | **Name**: `author`, **Type**: `text`<br>Interactions: `Standard Search Form` |
| 2 | **Text**: `Start Date` | **Name**: `startdate`, **Type**: `datepicker`<br>Interactions: `Standard Search Form` |

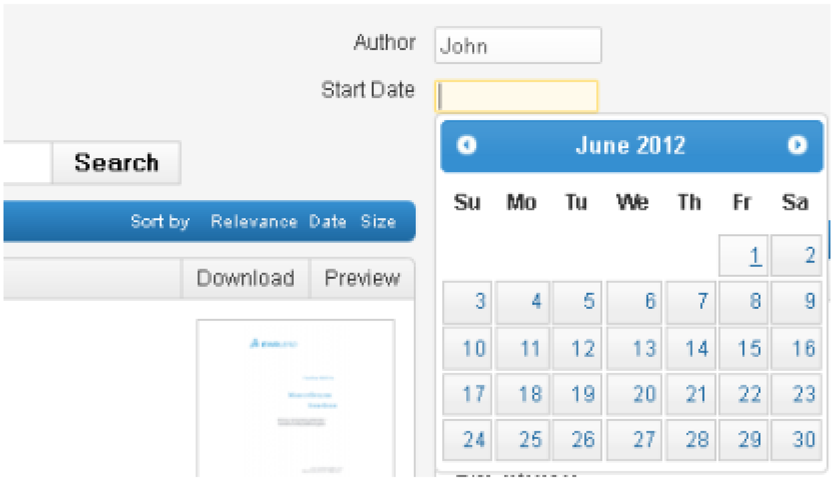


3. Select the **Preview** to check your configuration changes.

   You should see the form on your Mashup Builder page, and be able to complete it.

4. Click **Apply**.

   *Form inputs linked to the Search button of the Standard Search Form*



## Adding Triggers

For each application created within your Mashup Builder Premium instance, you can configure Feed and Design Triggers.

About Feed and Design Triggers

Add triggers to an application or a page

Add triggers to a widget

Add triggers to a feed

## About Feed and Design Triggers

### Feed Triggers

A Feed Trigger is an entry point to alter the behavior of a Feed.

It is called by the Mashup API before and after the feed execution, allowing for query manipulation, context modification and results manipulation.

Therefore, Feed Triggers can do the following:

- Decide to override the query to be issued to the actual ‘execute’ method based on query expansion (`beforeQuery` method)
- Decide to replay the feed execution because the result obtained is not satisfying, for example, if there are no results (`afterQuery` method that returns `Result.EVAL_AGAIN`)

Example:

You can use a Feed Trigger on any Feed in your configuration to customize query processing or feeds behavior for different purposes such as:

- Query rewriting
- Query computing from previously retrieved results
- Enabling / Disabling feeds

### Design Triggers

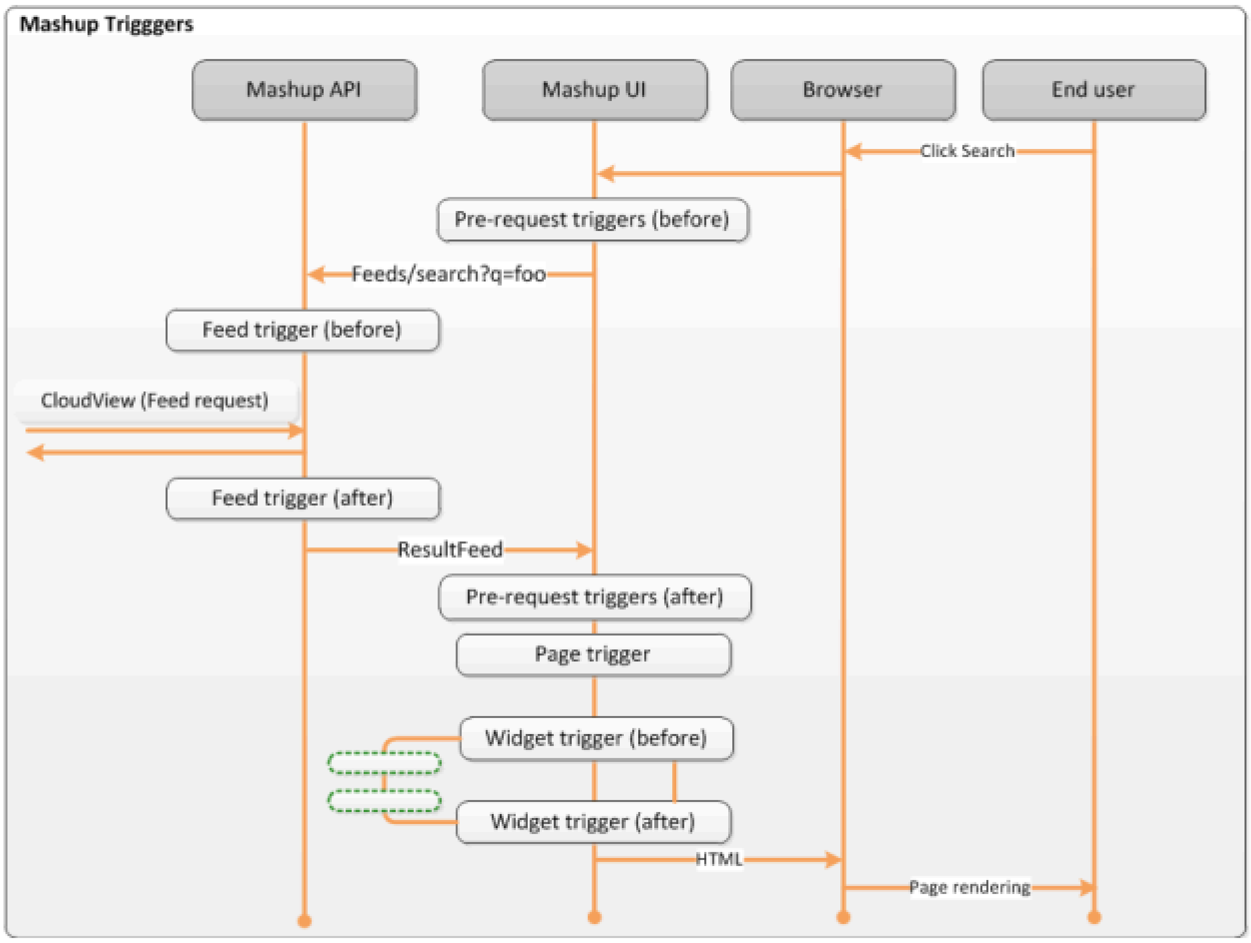
Design triggers are called by the Mashup Builder

They include:

- **Pre-request triggers** which can be used to decide whether the user should be redirected to another page or not. The back end is not yet called, so no special load is triggered on the system. For example, redirect the user to the page called `/imagesearch` if the query starts with `‘image(s)’`.
- **Page** and **widget triggers** which have the possibility to alter the behavior of the display by making decisions to draw things or even change the configuration (each user request gets a fresh copy of the original configuration, so any changes at query time are safe). For example: decide whether a widget should be displayed or not.
- **Application triggers** which are executed on all application pages.

## Execution Flow

The Mashup trigger sequence is as follows:



## Add triggers to an application or a page

This section describes how to add Feed and Design Triggers to a given application or a given page.

Note that:

- Application triggers will be applied globally throughout all the pages of the application.
- Page triggers will be applied on the selected page only.

**Note:** The Feed and Design triggers that can be applied at the ‘page level’ are the same as the ones that can be applied at the ‘application level’.

### Add Feed triggers to a given application or page

1. In **Mashup Builder**, select a page and then select its **Feeds** view.

2. From the **Triggers > Feed Triggers** pane, drag the Feed (or Mashup API) triggers that you want to apply to the application level or to the page level.

*Feed triggers description*

| Feed Trigger | Description |
|---|---|
| **Category name regexp processing** | Uses regular expressions to search and replace category names for a given feed.<br>It contains the following properties:<br>• **Facets**– Enter the facet name.<br>• **Search for** – Enter a Java regular expression string to search for a specific category name.<br>• **Replace with** – Enter a Java regular expression string to replace the category name found by the **Search for** string by another category name.<br>Look up the reference for `String.replaceAll()` for further information. |
| **Create facet** | Creates pseudo-facets on the fly. This can be useful to create virtual facets for feeds returning a lot of unsorted hits.<br>For example, an XML feed returns several hits, one hit corresponds to a color meta and we want to use this meta as a facet in a pie chart to represent a pie section.<br>It contains the following properties:<br>• **meta name** – Enter the meta name.<br>• **facet name** – Enter the facet name.<br>• **facet description** – Enter a description |
| **Query Builder** | This is the most important trigger.<br>It computes simple queries out of a feed result to override another feed's query. In other words, it allows you to programmatically build a query using previously fetched results, assembling metas, facets, etc.<br>**Important:** The current feed must be synchronized, otherwise unexpected behavior may occur! To synchronize the feed, select the **Feeds** view, and in the **Feed Options** section, set the **Synchronized** property to **true**. Sorting is important if you select the **Synchronized** option in the feed properties. It determines the execution order of your feeds. The first feed |

| Feed Trigger | Description |
|---|---|
| | in the drop zone is executed first, when results are retrieved, the second feed is executed etc.<br><br>It contains the following properties:<br><br>• **xml** – Displays the XML code that will be used by the trigger. This code contains the following nodes (the values in green are sample values only):<br>◦ `<string value="str" />`: Appends the string "str" to the query. It is useful for static query chunks.<br>◦ `<join glue=" OR ">`: Joins the values generated by the embedded nodes using the specified glue.<br>◦ `<metaValues metaName="title" max="3" quotes="true"/>`: Retrieves up to 3 values of the "title" meta and surrounds them with quotes for an exact match query.<br>◦ `<facetLeaves facetId="coffee" max="10" quotes="true" />`: Retrieves up to 10 values of the "coffee" facet and surrounds them with quotes for an exact match query.<br>◦ `<expr expr="${feeds["feedname"].metas["metaname"] + 10}" />`: Executes the specified MEL expression.<br>◦ `<replace pattern="fo[uo]" replacement="bar" caseInsensitive="false">`: Applies the specified replacements on the values generated by embedded nodes.<br>◦ `<if test="${expr}">`: Executes the specified MEL expression test, and if true, calls the embedded generator nodes. You must format the node as follows: `<if test="> <then> ... </then> <else> ... </else> </if>`<br>Important:<br>Using only `<if test="> ... </if>` does not work.<br><br>• **Feed to override** – Enter the name of the feed to override or simply select it from the list of available feeds from the **Values** tab (on the left).<br><br>• **Parameter to override** – Enter the feed parameter to override.<br><br>• **Execution mode** – Select the trigger mode:<br>◦ **beforeQuery**: builds the query for the current feed. |

| Feed Trigger | Description |
|---|---|
| | ◦ **afterQuery**: allows to take the feed's results to build a query for another feed.<br>◦ **Query Builder**<br>◦ **Enable feed if disabled** – Enables the feed if it is disabled, that is to say, if the **Feed options > Enable** property is set to **false**. You can indeed choose to disable a feed by default to launch it only if the query builder is executed to build a specific query. Note: This property is used only in **afterQuery** mode, that is to say after the execution of a first query.<br>◦ **Launch if empty** – Launches the QueryBuilder trigger if the feed is empty or has no results. Note: This property is used only in **afterQuery** mode, that is to say after the execution of a first query. |
| **JSON to Meta Trigger** | Transforms JSON meta values into metas. This is useful for the collaborative widgets returning unreadable and unusable JSON strings that bring valuable information.<br>It contains the following property:<br>**jsonMetaNames** – Enter the JSON meta name that must be transformed into a standard Exalead CloudView meta. |
| **Recommendation Trigger** | Defines the feed that will be used for result-condition-based analysis in the Content Recommender. It checks whether the triggers have matched on the page.<br>**Important:** The current feed must be synchronized and placed on top of the recommendation feeds, otherwise unexpected behavior may occur. To synchronize the feed, select the **Feeds** view, and in the **Feed Options** section, set the **Synchronized** property to **true**. |
| **Transform Categories** | Applies transformations to a single category using a JavaScript expression. It contains the following property:<br>**JavaScript expression** – Enter a JavaScript expression to transform a specific category (available object: `category`) For example: `category.count = category.count / 2` |
| **Transform Facets** | Applies transformations to a single facet using a JavaScript expression.<br>It contains the following property: |

| Feed Trigger | Description |
|---|---|
| | **JavaScript expression** – Enter a JavaScript expression to transform a specific facet (available object: `facet`) For example: `facet.name = \"New facet name\"` |

*Page Feed Triggers description*

| Page Feed Trigger | Description |
|---|---|
| **Aggregation Builder** | This trigger can be added to the Page or to the Application level. It adds on-the-fly aggregations to a given facet. For example, you can add an aggregation to get the percentage corresponding to each category.<br>The calculation can also be performed using the values of different feeds. For example, FeedA relates to a group of items, FeedB relates to one of these items, and you want to make a comparison between these two.<br>Limitation: This trigger does NOT work with MultiDimension facets and Hierarchical2D facets.<br>It contains the following properties:<br>• **Target Feed** – Specifies the target feed on which you want to create on-the-fly aggregations.<br>• **Target Facet** – Specifies the target facet on which you want to create on-the-fly aggregations.<br>• **Facet Iteration mode** – Specifies the iteration mode used to iterate each facet category. For example, the "Year" facet is set to YYYY/MM/dd:<br>◦ ALL – recursive iteration,<br>◦ FLAT – iteration on the first facet level, in our example: 2010, 2011, 2012, etc.<br>◦ LEAVES – iteration on the last facet level, in our example: Monday, Tuesday, Wednesday, etc.<br>• **Aggregation name** – Specifies the aggregation name.<br>• **MEL expression (Facet level)** – Enter the MEL expression that will be evaluated at facet level to create the target aggregation value.<br>• **MEL expression (Category level)** – Enter the MEL expression that will be evaluated at category level to create the target aggregation value.<br>Example: |

| Page Feed Trigger | Description |
|---|---|
| | Target feed: `cloudview`, Target Facet: `area`, Aggregation name: `percent`, MEL expression (Facet level): `100`, MEL expression (Category level): `${(category.count*100)/feed.facets["myfacet"].count}` |
| **Join feed trigger** | This trigger allows you to Join several feeds using a meta as key. It should be used with a query builder<br>It contains the following property:<br>**Feeds to join**: A list of feeds that you want to join using a given joinKey. The first feed of the list will contain the result of the join. |

3. Click **Save** and then **Apply** your configuration changes.

## Add triggers to a given application or page

1. In **Mashup Builder**, select a page and then select its **Design** view.
2. From the **Triggers** pane, select the type of trigger that you want to apply. For example, **Pre Request Trigger**.
3. Drag the Design triggers that you want to apply to the application level or to the page level. In the following screenshot the Pre Request Trigger **I18N...** is applied at the application level.

*Mashup Page Triggers description*

| Mashup Page Trigger | Description |
|---|---|
| **Business Console Debug Trigger** | This trigger is required for the Business Console Test UI.<br>It communicates a list of matched rules to the Business Console (Content Recommender tool) and will override subfeeds according to matched rules output configurations. |
| **Google Analytics Trigger** | Launches a Google Analytics event trigger. Google Analytics allows you to track how often viewers click particular links on your website.<br>It contains the following property:<br>**Google Analytics profile**: Enter the name of your Google Analytics account ID. For example a code like ‘UA-10876-1’. |
| **Override Page Title** | Overrides the page title using result feeds. For example, if you want to set a title from the first hit name.<br>It contains the following property: |

| Mashup Page Trigger | Description |
|---|---|
| | **title**: Enter the page title that will override the title configured in the **Page properties**. If blank, it will use the title defined in the `WEB-INF/i18n/commons.properties` file. |

*Pre Request Triggers description*

| Pre Request Trigger | Description |
|---|---|
| **Cookie to parameter** | Reads a cookie and sends its value to a new parameter that can be used by feeds as a page parameter.<br>By default, a `timezone.js` file (specified in **Application > General > JavaScript**) retrieves the local timezone of the user session and stores it in the`mashup-timezone` cookie. The Cookie to parameter trigger retrieves this cookie value and pushes it to the `timezone` page parameter. The user timezone information is provided for each feed in the **Advanced Parameters > Timezone Parameter**<br>This trigger contains the following properties:<br>• **Parameter name**: Specifies the name of the parameter that will contain the cookie value.<br>• **Cookie name**: Specifies the name of the cookie which contains the value. |
| **I18N: Retrieve the locale in the MashupAPI** | Adds a new parameter containing the locale used in the Mashup Builder to the MashupAPI.<br>The MashupAPI has no 'user session' information whatsoever, and does not know which locale is used by the Mashup Builder. It can however be required when a trigger is launched so as to use it in your feed.<br>It contains the following property:<br>**parameterName**: Enter the name of the parameter which contains the locale. |
| **Page Redirection** | Redirects to another page used on a javascript routine.<br>It contains the following property:<br>**expr**: Enter a Javascript expression to redirect the user to another page. You must return the name of the page to be called, or null to stay on this page. |

| Pre Request Trigger | Description |
|---|---|
| | **Important:** If you are looking for an optimized way to do this, a Java PreRequestTrigger may be preferred. |
| **Redirect by group** | Redirects authorized/unauthorized group of users to specific page.<br>It contains the following properties:<br>• **Regular expression**: Enter a regular expression to extract the group value from security tokens.<br>• **Rules**: Define the group of users impacted by the redirection. |
| **Redirect if mobile device** | Redirects the user to a given URL if the User Agent comes from a mobile device.<br>It contains the following property:<br>**url**: Enter the URL where the user must be redirected to if the User Agent comes from a mobile device |

4. Select the **Preview** to check your configuration changes.
5. Click **Apply**.

## Add triggers to a widget

Triggers can be configured for a given widget. These triggers will be applied on the selected widget only.

1. In **Mashup Builder**, select a page and then select its **Design** view.
2. From the **Triggers** pane, select **Mashup Widget Trigger**.
3. Choose a widget and drag the triggers that you want to apply to its drop zone:

*Mashup Widget Triggers description*

| Mashup Widget Trigger | Description |
|---|---|
| **Asynchronous ajax loading** | Loads the widget asynchronously. |
| **Conditional display** | Adds a conditional display to the widget, using a MEL expression parameter. |
| **Remove if facet is empty** | Hides a widget when the given facet is empty. |
| **Remove if logged in** | Hides a widget when the user is logged in. |

| Mashup Widget Trigger | Description |
|---|---|
| **Remove if no entries** | Hides a widget when its sources contain no entries. |
| **Remove if not logged in** | Hides a widget when the user is not logged in. It is also hidden, if security is disabled or ill-configured. |
| **Transform to Ajax links** | Transforms all widget links to Ajax links. This is useful when you want to reload some widget components only, and NOT reload the whole page. |

4. Click **Save** and then **Apply** your configuration changes.

### Add triggers to a feed

Triggers can be configured for a given feed. These triggers will be applied on the selected feed only.

**Note:** The Feed triggers that can be applied at the 'feed level' are the same as the ones that can be applied at the 'application level'.

1. In **Mashup Builder**, select a page and then select its **Feeds** view.
2. Choose a feed and drag the triggers that you want to apply to its drop zone.
3. Click **Save** and then **Apply** your configuration changes.

## Configuring Data Storage for Collaborative Widgets

With collaborative widgets (**Comments**, **Star Rating**, etc.) you can enrich your document with added data saved in the data storage.

**Note:** Collaborative widgets are available in Mashup Builder Premium only.

Storage in Mashup allows you to:

- Get arbitrary chunks of data (JSON strings, images, Java objects, etc.) attached to your Mashup Builder application.
- Delete, replace and append metas and categories to Exalead CloudView documents.
- Store private data per user.

By default collaborative widgets only save and retrieve data from the data storage. They manage their own data display and safeguard. They do not communicate with the index. This default behavior guarantees that each action performed within the collaborative widgets will be instantaneously propagated to all search pages.

Optionally, you can choose to index storage data to improve your data model. You will then be able to apply all index features on collaborative data (analysis, processing, search, refinement, etc.). Every change made to the document will be notified to Exalead CloudView. Changes will therefore be available after the next indexing action. The elapsed time between each indexing action can be configured in the Administration Console.

## Configure storage to index collaborative data

The following procedures describe how to configure your index to use data coming from the storage.

It includes:

- the Exalead CloudView document cache has to be enabled on the document build group.
- the `RepushFromCache` setting must be set to `true` in `StorageService.xml`.
- the **StorageServiceDocumentProcessor** must be configured in the Analysis pipeline.
- the data of the collaborative widget should be mapped to metas.

### Requirements

When you use a collaborative widget with the “`DOCUMENT`” scope, it uses the build group, the source and the doc URI, to identify each document. These values are required and must be set to be used for hit content.

1. In the Administration Console, go to **Search Logics > Facets**, expand the **Source** category.
2. Make sure that the **Root** property is set to `Top/source`. This source category is required to retrieve the hit information used to create primary keys in the storage mechanism, so as to link the collaborative widget to a document.
3. Make sure that the **Use for hit content** option is selected. Otherwise, your collaborative will have no source on the Mashup Builder.

### Enable the build group cache

1. In the Administration Console, go to **Deployment > Build groups**.
2. Select **Document cache**.

## Enable the RepushDocuments property

1. Go to your Exalead CloudView `<DATADIR>/config/360` directory.
2. Open the `StorageService.xml` file.
3. Set the `RepushDocuments` property to `true`.

## Configure the StorageServiceDocumentProcessor in the Analysis pipeline

1. In the Administration Console, go to **Index > Data Processing**.
2. Select your analysis pipeline, for example `ap0`.

   The **Document Processors** tab displays.
3. From the **Processor Types** panel:
   a. expand the **Other** node,
   b. drag a **Storage Service Document Processor** to the list of **Current Processors** (preferably near the top of the list).
4. Click the **StorageServiceDocumentProcessor** link and define the **Instance** property.

   Usually the instance is `sts0`. You can check the instance name from the **Roles** menu, by expanding the **Storage Service** role.
5. Click **Save** and then **Apply** your configuration changes.

## Make data searchable

1. In Mashup Builder, check the **meta name** property of your collaborative widget.

   For example, the **Tags meta name** for a **Tags** widget is set by default to `tags[]`.
2. In the Administration Console, go to **Data Model**.
3. Click **Add Property** to add a property matching the meta name specified in the Mashup Builder (see step 1). For example:
   - **Name**: tags
   - **Data Type**: Alphanum
   - **Semantic Type**: text
   - **Field Type**: Dedicated field only
4. Click **Accept**.

   By default the property is searchable.
5. Select the **Preview** to check your configuration changes.
6. Click **Apply**.

You should now be able to search on your meta in the Mashup Builder. For example, you can search on your tag values by entering `tags:MyTag`

### Index aggregated results

1. Make sure you followed the standard indexing workflow described in the previous procedures.
2. Go to your Exalead CloudView `<DATADIR>/config/360/applications/<APPLICATION NAME>` directory.
3. Open the `Storage.xml` file.
4. Add as many `StorageKey` nodes containing `Aggregate` nodes as necessary. In the following example, we want to index the `COUNT` and `SUM` aggregation values for the `bagkey` StorageKey.

```
<Storage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSche
   <Indexing>
     <StorageKey name="bagKey[]">
     <Aggregate type="COUNT"/>
     <Aggregate type="SUM"/>
     </StorageKey>
    </Indexing>
</Storage>
```

5. Build your configuration:
   a. Go to your Exalead CloudView `<DATADIR>/bin/`directory.
   b. Launch the following command: `buildgct.<sh|bat>`

The **StorageServiceDocumentProcessor** will create the corresponding metas with aggregate operations as suffixes. For our example: `bagKey_count` and `bagKey_sum`

## Storage Administration

### Back up the Storage SQLite database

1. Go to `<DATADIR>/storageService/storage.db.sqlite`
2. Back up the database.

### Enhance the Storage performance

By default the storage works with an SQLite engine and is therefore limited in terms of concurrent accesses, as each write action blocks the whole database; and scalability, when the number of entries is really big.

However, you can configure its JDBC connection to work with compatible databases (SQL Server, MySQL or Oracle) if you need the storage to be more scalable.

1. For more information, see "Creating Collaborative Widgets Using Storage Service" in the *Mashup Programmer Guide* .

### Set up a storage linked to a replicated MySQL server

The storage service uses a standard JDBC configuration. A JDBC URL can therefore be configured to manage failover.

For example, with MySQL, we could have:

```
jdbc:mysql://[host][,failoverhost...][:port]/[database] » ?propertyName1[=propertyVal
[=propertyValue2]...
```

However, this MySQL failover configuration does not guarantee real High Availability, as database queries will be read-only queries, preventing users to enrich documents with storage information (for example, saving the preferences of the advanced table widget, comments, star rating, saved queries, etc.).

See the MySQL documentation: `http://dev.mysql.com/doc/refman/5.5/en/connector-j-reference-configuration-properties.html`

## Troubleshooting

### Repush is not launched after document modifications

1. Enable the cache before indexing. Note that dumping the storage database will not trigger an automatic repush from cache.

### I can’t search on my collaborative data

A meta corresponding to the collaborative data must be mapped to a "searchable" field or facet in your data model.

1. Check that the collaborative widget data has been properly indexed.

   To do so, you can use a debug processor as explained in Troubleshoot all issues related to the storage service behavior.

### Apply actions fail if the search server and the related storage service are not on the same host

Storage can't be used from a remote search server due to SQLite limitations.

It is yet fully compatible with MySQL and Oracle.

### Troubleshoot all issues related to the storage service behavior

1. In the Administration Console, go to **Logs**.

2. Set the **Default logging level** to **Debug** and check the storage service's logs.
3. Make sure that the `repushFromCache` request is issued without problems.

   **Important:** If you get an error saying `invalid source:"`, it means that the source is empty, that is to say that there is no source category to display. Go to **Search Logics > Facets**, expand the **Source** category and set the **Root** property to `Top/source`

### Troubleshoot indexing issues

1. In the Administration Console>, go to **Index > Data Processing >** *pipeline name* **> Document Processors**.
2. Add a **Debug Processor** to the bottom of your analysis pipeline.
3. Make sure that values are indexed.
   a. Go to **Logs**.
   b. From the **Processes** select box, select **analyzer-<BUILD GROUP NAME>**.

The Debug Processor will write explicit logs.

## Creating Composite Widgets

When you have very specific needs, you can create your own Composite Widgets with the standard widget library. Your composite widgets will be added to the **Widgets > Composite Widgets** group.

Inside your composite widgets, you can also create your own widget properties and therefore configure your custom widget from A to Z.

It is also possible to create a composite widget out of:

- a page and reuse that page as a widget in other pages,
- an existing widget.

  **Note:** A composite widget is actually a plugin. Once created, you can export and import it like any other plugin.

Create composite widgets from scratch

Create custom widget properties

Create a composite from a page or widget

Delete a composite widget

## Create composite widgets from scratch

This section describes two composite widget use cases to:

- Create a custom form using a set of **Label** and **Input** widgets.
- Link addresses to Google Maps.

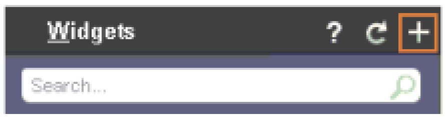
These use cases show two of the many possibilities offered by composite widgets.

### Create a custom form

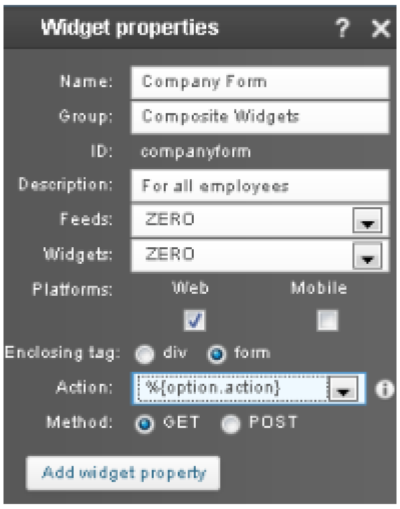
The following use case describes how to proceed if you want to create an HTML form that has an impact on the page results.

**Note:** For another example of form creation with widget interactions, see Specify widget interactions.

1. In **Mashup Builder**, select a page, for example **/search**, and select the **Design** view.
2. In the **Widgets** panel header, click the **+** sign.



3. In the **Widgets properties** panel:
   a. Fill in the **Name** field, for example `Company Form`.
   b. Enter a **description** (optional).
   c. For **Feeds**, define how the widget supports feeds by selecting a cardinality.
   d. For **Widgets**, define how this widget supports sub-widgets by selecting a cardinality.
   e. Select the platforms for which the composite widget will be available, **WEB** and/or **MOBILE**, for example **WEB**.
   f. For **Enclosing tag**, select the type of enclosing tag used by the widget. In our example, we must select **form** to create a new form.
   g. For **Action**, select the action to launch when the form is submitted.
   h. For **Method**, select how to send the data once the form is submitted. For example, select **GET** to attach the form data onto the page URL.


Widget properties

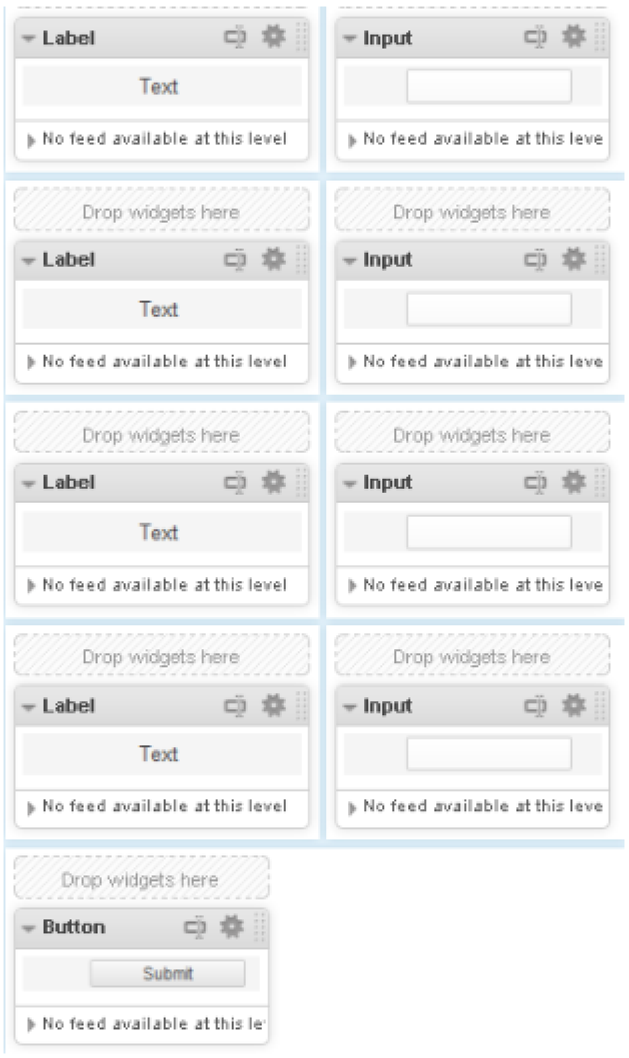
Name: Company Form
Group: Composite Widgets
ID: companyform
Description: For all employees
Feeds: ZERO
Widgets: ZERO
Platforms: Web [✓] Mobile [ ]
Enclosing tag: div [✓ form]
Action: %{option.action}
Method: [✓ GET] POST

4. From the **Page Layout** panel, organize the layout. For example, two columns and five rows.
5. From the **Widgets** panel, drag and drop widgets to assemble your composite widget.
   a. For the first four rows, add a set of **Label** and **Input** widgets in the cells and configure them as described in the following table.

| Row # | Label Widget | Input Widget |
|---|---|---|
| 1 | **Text**: `Company` | **Name**: `company`, **Type**: `text` |
| 2 | **Text**: `Login` | **Name**: `username`, **Type**: `text` |
| 3 | **Text**: `Start Date` | **Name**: `startdate`, **Type**: `datepicker` |
| 4 | **Text**: `End Date` | **Name**: `enddate`, **Type**: `datepicker` |

   b. Drag a **Button** widget in the last row of the form and set its properties as follows: **Button label**: `Send`, **Type**: `submit`.

   *Composite widget form example*


Label
Text
No feed available at this level
Input
No feed available at this leve
Drop widgets here
Button
Submit
No feed available at this le

6. Click **Save**.

You then need to add your composite widget on the Design view of your application page, as described in Add and configure your form composite widget on a page .

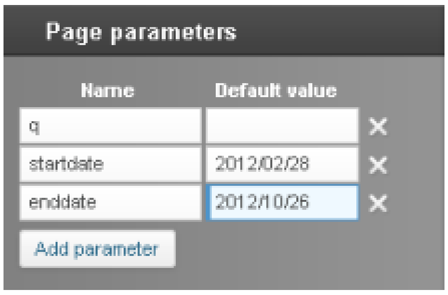
## Add and configure your form composite widget on a page

1. Once your composite widget is saved, close the **Widget properties** panel to return to the main **Design** view.
2. Select the page on which you want to add your composite widget.
3. From the **Widgets > Composite Widgets** group, select your `Company Form` composite widget, and drag and drop it on the canvas.
4. From its properties panel, select the page that will display the search results of your form from the **Action** property.
5. Select the **Feeds** view and edit the feed called by the composite widget to define the behavior of the query so as to search after the start date and before the end date delimited by the two datepickers.

a. Select the feed called by the widget and expand the **Advanced parameters** section.

b. In the **Query restrictions** field, restrict the query to use the span of time between your `startdate` and `enddate` datepickers: `after:${page.params["startdate"]} before:${page.params["enddate"]}`

6. From the **Page parameters** panel create two page parameters for `startdate` and `enddate` and choose their default values. The time format of these dates is `YYYY/MM/dd`.



Page parameters

| Name | Default value |
|---|---|
| q | |
| startdate | 2012/02/28 |
| enddate | 2012/10/26 |

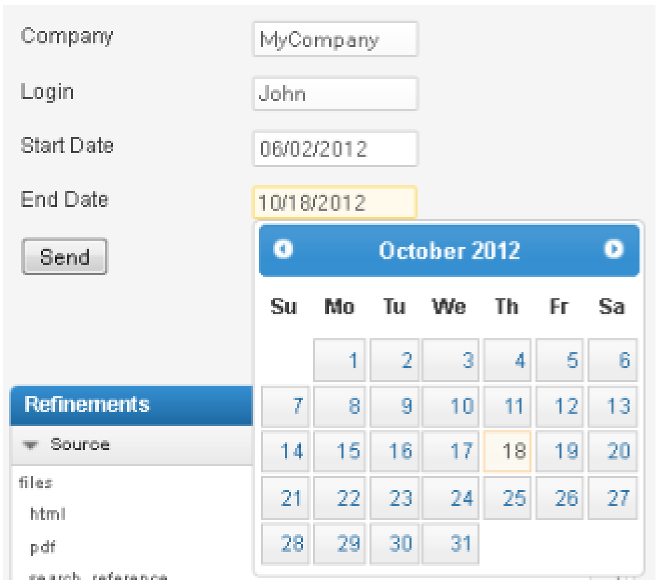
7. Select the **Preview** to check your configuration changes.

   You should now see the composite widget on your Mashup Builder page, and be able to complete the form. See Figure 12.

   You can use the datepickers to filter the search results.

8. Click **Apply**.

   *Form composite widget to refine the page results*



| | |
|---|---|
| Company | MyCompany |
| Login | John |
| Start Date | 06/02/2012 |
| End Date | 10/18/2012 |

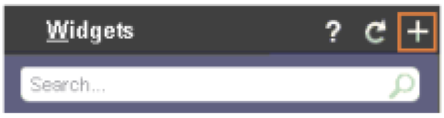
## Link addresses to static Google Maps

The following use case describes how to proceed if you want to link postal addresses to static Google Maps in a result list. Each address will be displayed as both text and image, as shown in  .

## Create a composite widget linking addresses to static images

This example requires that an **address** meta is defined in your data model.

1. In **Mashup Builder**, select a page, for example **/search**, and select the **Design** view.
2. In the **Widgets** panel header, click the **+** sign.



3. In the **Widgets properties** panel:
   a. Fill in the **Name** field, for example **Static Google Map**.
   b. Enter a **description** (optional).
   c. For **Feeds**, define how the widget supports feeds by selecting a cardinality, for example **ONE**.
   d. Select the platforms for which the composite widget will be available, for example **WEB**.
   e. >For **Enclosing tag**, select the type of enclosing tag used by the widget, for example **div**.
4. Click **Add widget property** and create the three following properties in the **General** tab.

| Property | Options to configure |
|---|---|
| `Address` | **Id**: `address`<br><br>**Name**: `Address`<br><br>**Arity**: `ONE`<br><br>**Description**: `Meta(s) containing the address to position the marker`<br><br>**Is evaluated**<br><br>**Check functions**: `isEmpty()`<br><br>**Context menu functions**: `Metas()`<br><br>**Display functions**: `SetType('code', 'js')` |
| `Height` | **Id**: `height`<br><br>**Name**: `Heigth`<br><br>**Arity**: `ONE`<br><br>**Description**: `map height (in pixels but without px extension)`<br><br>**Check functions**: `isInteger()`<br><br>**Display functions**: `SetType('number')` |
| `Width` | **Id**: `width` |

| Property | Options to configure |
|---|---|
| | **Name**: `Width`<br>**Arity**: `ONE`<br>**Description**: `map width (in pixels but without px extension)`<br>**Check functions**: `isInteger()`<br>**Display functions**: `SetType('number')` |

5. From the **Widgets** panel, drag and drop an **HTML** widget.
6. In the **HTML code** field, enter the following code:

   ```
   <img width="%{option.width}" height="%{option.height}"
   src="http://maps.googleapis.com/maps/api/staticmap?center=%{option.address}&zoom=1
   {option.width}x%{option.height}
   &sensor=false&markers=color:blue%7Csize:small%7C%{option.address}" />
   ```

   This code contains:

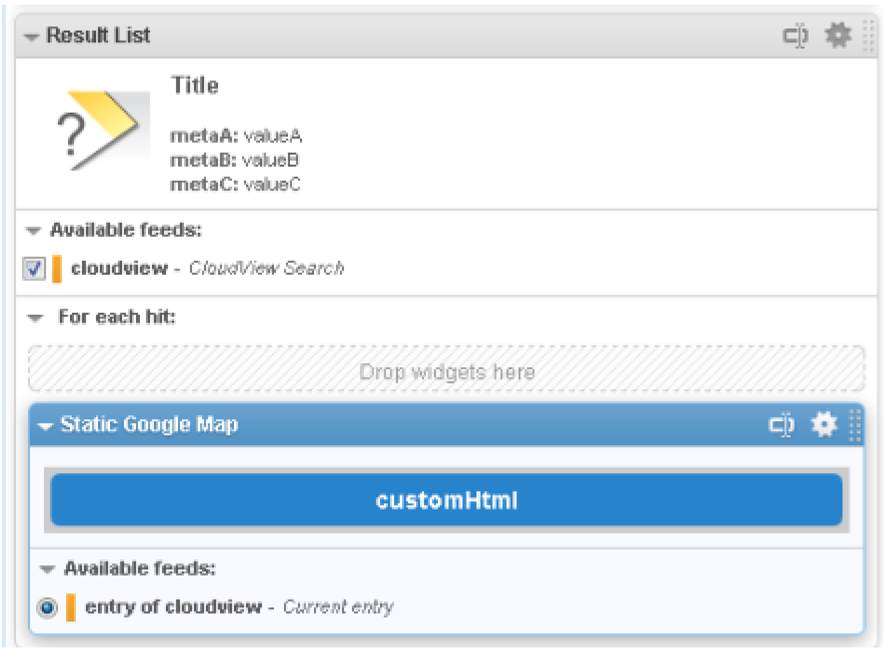
   - Variables made with the custom properties created in step 4 (`option.width`, `option.height`, `option.address`). They are used to configure the width and height of the image, and the address display in the widget.
   - A call to the Google Maps api.
   - A command to center the address in the static map.
   - A command to add a blue marker to flag the address in the static map.
7. Click **Save**.

You then need to add your composite widget on the Design view of your application page, as described in Add and configure your form composite widget on a page .

## Add and configure the static Google Map composite widget

We are now going to add the composite widget within the **Result List** widget to get one image per result. If you do not nest the composite widget within the **Result List** widget, you will get an image for the first result only.

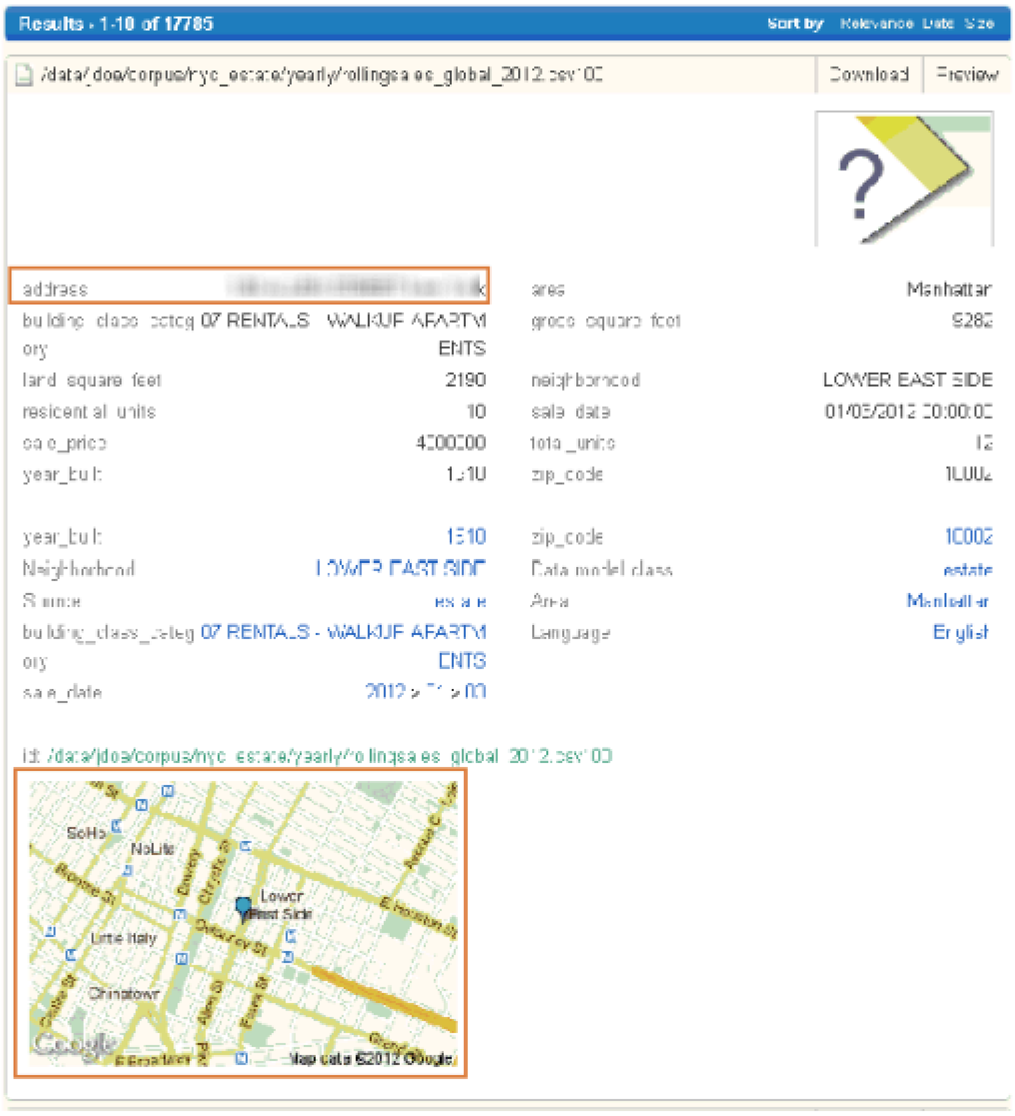
1. Once your composite widget is saved, close the **Widget properties** panel to return to the main **Design** view.
2. Select the page on which you want to add your composite widget, for example `search`.
3. From the **Widgets > Composite Widgets** group, select your composite widget **Static Google Map**, and drag and drop it within the **Result List** widget.



4. Click the widget header to open its properties panel, and for the **Address** property, enter: `${entry.metas['address']}` to link the widget to the address meta of the **Result List** widget.
5. Select the **Preview** to check your configuration changes.

   You should now see the composite widget as it will be displayed on your Mashup Builder page. Each result in the result list displays the postal address as test and the location of this address on a static Google Map.

   *Postal address linked to static Google Map*


Results - 1-10 of 17785

Sort by Relevance Date Size

/data/jdoa/corpus/nyc_estate/yearly/rollingsales_global_2012.csv'03

Download Preview

| | | | |
|---|---|---|---|
| address | [redacted] | area | Manhattan |
| building_class_category | 07 RENTALS - WALKUP APARTMENTS | gross_square_feet | 9282 |
| land_square_feet | 2190 | neighborhood | LOWER EAST SIDE |
| residential_units | 10 | sale_date | 01/03/2012 00:00:00 |
| sale_price | 4300000 | total_units | 12 |
| year_built | 1910 | zip_code | 10002 |

| | | | |
|---|---|---|---|
| year_built | 1910 | zip_code | 10002 |
| Neighborhood | LOWER EAST SIDE | Data model class | estate |
| Source | estate | Area | Manhattan |
| building_class_category | 07 RENTALS - WALKUP APARTMENTS | Language | English |
| sale_date | 2012 ... 00 | | |

id: /data/jdoa/corpus/nyc_estate/yearly/rollingsales_global_2012.csv'03

6. Click **Apply**.

## Create custom widget properties

While creating/editing a composite widget, you can also create the properties and the tabs that will be available in the widget's configuration panel when this widget is added to a page.

To do so, you can click the **Add widget property** button and then click:

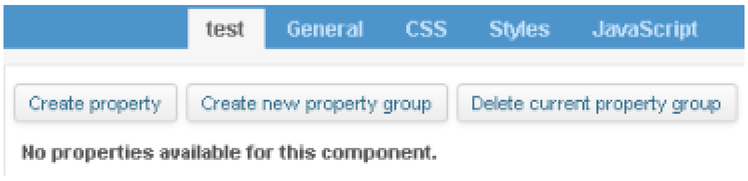
- **Create property** to create new properties, as described in the following procedures.
- **Create new property group** to create a new tab in the panel.

### Create a custom widget property

1. In the **Widget properties** panel, click the **Add widget property** button to create a custom widget property.

   The **Add widget property** panel opens at the bottom of the screen.

2. Click **Create property** and set the options as described in the following table.



No properties available for this component.

*Add widget property panel description*

| Option | Description |
|---|---|
| **ID** | Enter the ID of the widget property. |
| **Name** | Enter the name of the widget property. |
| **Arity** | Specify the arity of the widget property. The arity is the number of arguments that the function takes.<br>• **ONE**: For unary operator<br>• **ZERO_OR_ONE**: For binary operator |
| **Description** | Enter a description for the widget property. |
| **Placeholder** | Specify the placeholder to display to help the user completing the property field. The placeholder disappears as soon as you enter text. |
| **Possible values** | Define the property's values:<br>If one value is declared, it will behave as a default value.<br>If several values are declared, a selection box will be displayed. |
| **Is evaluated** | Choose whether the widget property will be evaluated and have MEL expressions or not. |
| **Is xml escaped** | Choose whether the widget property results will be XML escaped or not, that is to say whether ASCII characters like the angle brackets (<>) will be encoded in XML. |
| **Is url encoded** | Choose whether the widget property results will be URL encoded or not.<br>URL encoding is normally performed to convert data passed via HTML forms, because such data may contain special characters which are not allowed on a valid URL format. |
| **Check functions** | Choose an error checking function to validate user input:<br>• **isInteger**: Checks that the value is an integer and displays an error message if not. |

| Option | Description |
|---|---|
| | • **isAlphanum**: Checks that the value is a chain of alphanumerical characters and displays an error message if not.<br>• **isPageName**: Checks that the value is a page name and displays an error message if not.<br>• **isEmpty**: Checks that the value is NOT empty and displays an error message if it is the case. |
| **Context menu functions** | Specify the types of functions that will be available in the **Value** tab of the contextual menu displayed on the left of the widget properties panel:<br>• **Aggregations()**: Gets all facet aggregations, for example, count, score etc.<br>• **DateFacets()**: Gets all Date facets from the feeds used by the widget.<br>• **Eval()**: Gets all possibilities evaluated by the widget. The possible attributes are: facet, category and meta.<br>• **Facets()**: Gets all facets from the feeds used by the widget.<br>• **Feeds()**: Gets all the feeds used by the widget.<br>• **Fields()**: Gets all the virtual fields, numerical fields and RAM-based fields from the feeds used by the widget.<br>• **GeoFacets()**: Gets only Geographical facets from the feeds used by the widget.<br>• **Hierarchical2DFacets()**: Gets only Hierarchical2D facets from the feeds used by the widget.<br>• **Metas()**: Gets all metas from the feeds used by the widget.<br>• **MultiDimensionFacets()**: Gets Multi-dimension facets from the feeds used by the widget.<br>• **Normal facets()**: Gets only Category facets from the feeds used by the widget.<br>• **Numerical facets()**: Gets Numerical facets from the feeds used by the widget.<br>• **appendOnChange(', ')**: A click of the user on a dynamic list, will append the given string after the current option value.<br>• **emptyOnChange()**: A click of the user on a dynamic list, will remove the current option value before setting the clicked value. |

| Option | Description |
|---|---|
| | • **PageParameters()**: Gets the page parameters names<br>• **Pages()**: Gets the names of available pages.<br>• **Sorts()**: Gets all the feed elements that can be sorted.<br>• **WUIDS()**: Gets the list of this page Widget's unique IDs. |
| **Display functions** | Specify how the property will be displayed. For example, you can force the property to act as a select box, force it to act as an input (default is Text area), force it to act as a password input, etc.<br>• **Code Editor**: Transforms the property's input field into a code editor<br>• **Number**: Transforms the property's input field into a number input field.<br>• **Password**: Transforms the property's input field into a password input field (encrypted).<br>• **Radio**: Transforms the property's input field into a radio button.<br>• **SetHeight**: Sets the minimum height (in terms of lines) of the option's input field.<br>• **TextEditor**: Transforms the property's input field into a rich text editor.<br>• **TextArea**: Transforms the property's input field into a text area field.<br>• **ToggleDisplay**: Shows/hides properties conditionally depending on the selected property value. For example, value1 of PropertyA will display PropertyB and PropertyD, value2 will display PropertyB and PropertyC. You need to set:<br>  ◦ the value to match (using the `valueToMatch` and `ifEquals` attributes)<br>  ◦ the options to hide (in `hideOptions`)<br>  ◦ the options to show (in `showOptions`). |

3. Click **Save**.

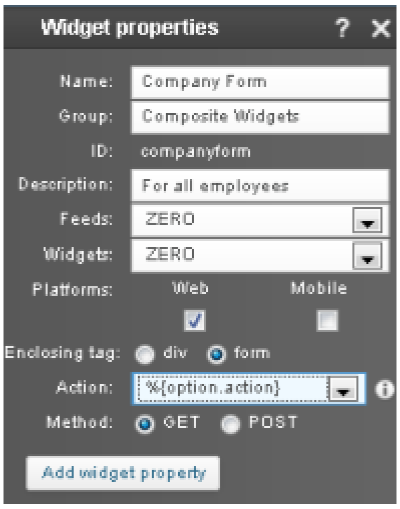
## Use case: Create an Action widget property to reach pages in a Form

To give a simple example of widget property creation, we are going to create an `Action` property that will be available in the contextual menu to select an application page.

1. Click the **Add widget property** button to create a custom widget property.
2. Click **Create property** and configure the widget property as follows:

a. For **ID**, enter `action`
b. For **Name**, enter `Action`
c. For **Check functions**, select `isEmpty()` to check that the value is NOT empty.
d. For **Context menu functions**, select `Pages()`.

3. Now if we go back to the use case described in Create a custom form, for **Widget properties > Action** we can select **%{option.action}** to use the newly created widget property.

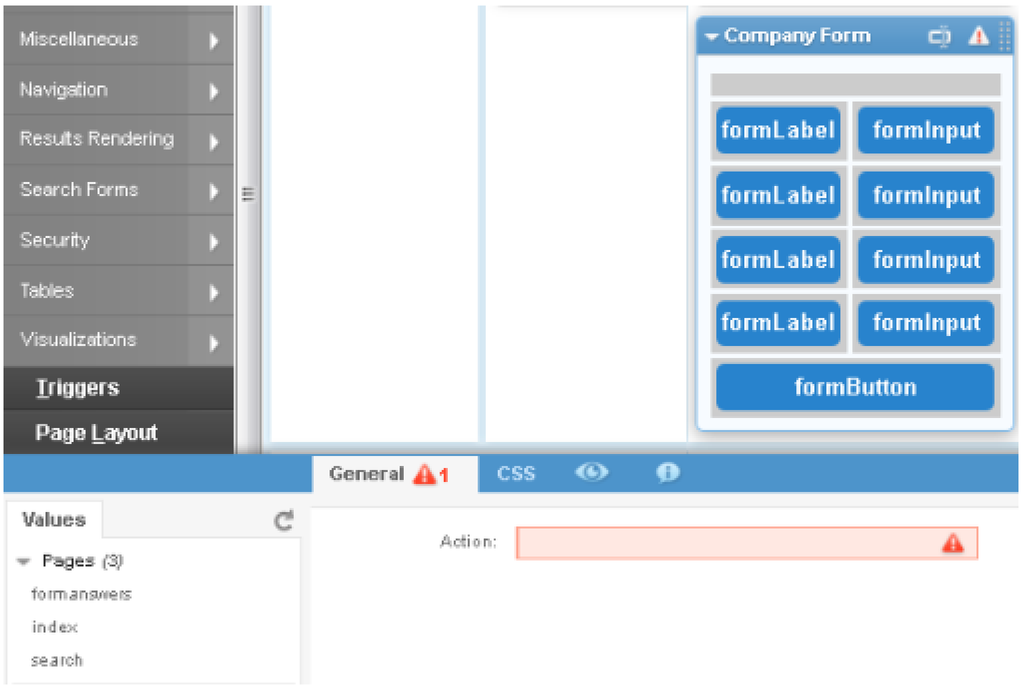

Widget properties
Name: Company Form
Group: Composite Widgets
ID: companyform
Description: For all employees
Feeds: ZERO
Widgets: ZERO
Platforms: Web [✓] Mobile [ ]
Enclosing tag: div [✓ form]
Action: %{option.action}
Method: [✓ GET] POST

4. Go to the page where the composite widget is added and click its header.

   The widget properties panel opens and you can now see the **Action** property.

5. Click in the **Action** property field.

   The contextual menu displays the application pages that can be used as destination pages when the form is submitted.



Values
Pages (3)
formanswers
index
search
Action:

6. Select the destination page you want to link for your custom form, for example `formanswers`.

## Create a composite from a page or widget

You are not forced to create a composite widget from scratch. You can also create a composite widget from a page and reuse that page as a composite widget in other pages, or from an existing widget and reuse it as a composite widget.

### Create a composite widget from a page

1. In **Mashup Builder**, select a page, for example **/search**, and select the **Design** view.
2. From the page editing tools, select **Use for composite widget**.
3. In the **Widgets properties** panel:
    a. Fill in the **Name** field, for example **MyWidget**.
    b. Enter a **description** (optional).
    c. For **Feeds**, define how the widget supports feeds by selecting a cardinality, for example **ONE**.
    d. Select the platforms for which the composite widget will be available, for example **WEB**.
    e. For **Enclosing tag**, select the type of enclosing tag used by the widget, for example **div**.
4. If necessary, add widget properties to your composite widget.
5. Click **Save**.

### Create a composite widget from a widget

1. In **Mashup Builder**, select a page, for example **/search**, and select the **Design** view.
2. From the widget header, click the edit widget properties ⚙ icon, and choose **Use for Composite Widget**.
3. In the **Widgets properties** panel:
    a. Fill in the **Name** field, for example **MyWidget**.
    b. Enter a **description** (optional).
    c. For **Feeds**, define how the widget supports feeds by selecting a cardinality, for example **ONE**.
    d. Select the platforms for which the composite widget will be available, for example **WEB**.
    e. For **Enclosing tag**, select the type of enclosing tag used by the widget, for example **div**.
4. If necessary, add widget properties to your composite widget.
5. Click **Save**.

### Delete a composite widget

As previously stated a composite widget is a plugin. To delete a composite it widget you need to remove it from the list of plugins.

1. In **Mashup Builder**, click the **Application** button at the top left of the screen.
2. In the **Manage plugins** panel, select **Plugins**.
3. From the list of plugins, click the **delete** action at the right of the screen for the composite widget you want to delete.

The composite widget is removed from the list and from the **Widgets** panel.

## Modifying the Search Results Display

You can modify the display of results on the **search** page in many ways. For example, you can filter the metas displayed in the result list, remove thumbnails, and link hits to a new Details page.

### Filter metas in the result list

1. In **Mashup Builder**, go to the **search** page and select the **Design** view.
2. Click the header of the **Result List** widget.
3. On the widget properties panel (at the bottom of the screen), go to the **Hit metas** tab.
4. Set **Meta filtering method** to **Exclude**.
5. Click inside **Meta list**. A list of available metas displays in the **Values** contextual menu on the left. Select the metas to exclude, separating them with commas.
6. Select the **Preview** to check your configuration changes.
7. Click **Apply**.

### Set the facet order

1. In **Mashup Builder**, go to the **search** page and select the **Design** view.
2. Click the header of the **Standard Facets** widget.
3. On the widget properties panel (at the bottom of the screen), set **Facet root list mode** to **Include**.
4. Click inside **Facets list**.

   A list of available facets displays in the **Values** contextual menu on the left.
5. Select the facets to include, separating them with commas.

   The facet order in this field corresponds to the facet order in the UI display.

6. Select the **Preview** to check your configuration changes.
7. Click **Apply**.

## Modify how search results display

You can change how the search hits display in the Mashup Builder. By default, the title that displays for each result shows is based on the metas `title` and `url`, but other metas can be used instead.

For more information about this, see "Modify the display of hit titles" in the *CloudView Getting Started Guide*.

## Display results in a new page

This example explains how to create a ‘details’ page providing more information for each hit. To save time, we will create it by copying another page.

1. Go to the **/search** page, and from the page editing tools, select **Copy**.
2. At the prompt, enter `details` as the new page name.
3. On the new **details** page, select the **Feeds** view.
4. Drag a **CloudView Search** feed on the page and call it `cloudview`.
5. For **Query**, enter `rawurl:${page.params["q"]}`

   This is to make the query fetch the document that matches the returned URL, as we want to display the details of each hit in the new **account** page.
6. Select the **Design** view to customize the display of the **details** page. For example:
   - You can choose to keep only the widgets **Image**, **Spacer**, and **Result List** on this page.
   - You can select the metas to display, that is to say the information given by your details page, by going to the **Hit metas** tab and editing the list of metas to include.
7. Go back to the **search** page, and click the header of the **Result List** widget.
8. Configure the hit URL to open the **/details** page. For **Hit URL**, enter `details?q= ${entry.metas["url"]}`
9. Select the **Preview** to check your configuration changes.
10. Click **Apply**.

## Customize icons in the search results

Default icons used in the search result page are stored in `<DATADIR>/webapps/360-edm-mashup-ui/resources/images/icons`. You may need to change these icons.

1. Add your .GIF file to the `icons` folder in `<DATADIR>/webapps/360-edm-mashup-ui/resources/images/icons`.
2. Edit the `mapping-entry-icon.properties` file located in `<DATADIR>/webapps/360-edm-mashup-ui/WEB-INF/config` and specify the icon name.

## Display hits depending on meta values

The search feed can return different kinds of values for a given meta, and you may want to display these values with specific look and feels in the search hits.

For example, you can have several types of documents - for example an MS Word document, an intranet page, etc. - and you want to associate each type to a specific look and feel in the search hits. In this case, you can customize the display of hits by types conditionally, by creating your own JSP files and referencing them as described in the following procedures.

**Note:** To customize the look-and-feel, the JSP files can either include specific HTML codes or reference CSS files.

### Conditionally display hits with the Result List widget

1. In **Mashup Builder**, go to the /**search** page and select the **Design** view.
2. Click the header of the **Result List** widget.
3. On the widget properties panel, go to the **Hit templates** tab.
4. In the **Hit JSP template** field:
    a. Enter the relative or the absolute path to the directory containing your custom JSP.
    b. Use a dynamic hit meta in your path to add a condition on the document type.

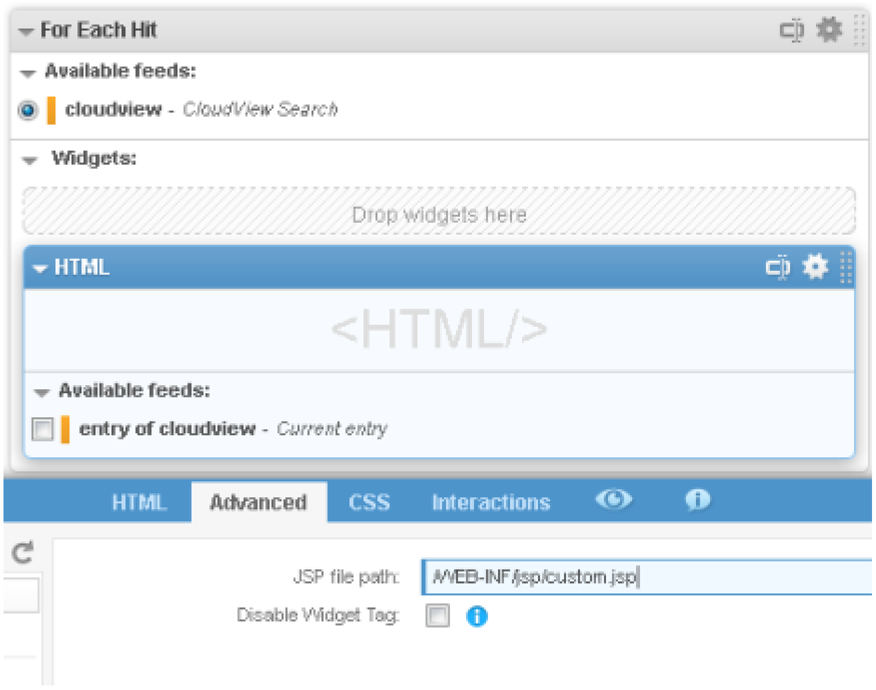
       For example: `mytemplate/${entry.metas["type"]}.jsp`
    c. The dynamic value will apply a specific JSP template for each type of document that has its own dedicated JSP. The default template is applied otherwise. For example, let’s say that you have created the following JSP templates: `filesystem.jsp`, `intranet.jsp`, `mail.jsp`, etc.
5. Select the **Preview** to check your configuration changes.
6. Click **Apply**.

### Conditionally display hits with the For Each Hit widget

1. In **Mashup Builder**, go to the /**search** page and select the **Design** view.

2. In the **Widgets** panel, expand the **Miscellaneous** category and drag a **For Each Hit** widget on the workspace.
3. In the **Widgets** panel, expand the **HTML** category, and drag the **HTML** widget within the **For Each Hit** widget.
4. On the widget properties panel, in the **Advanced > JSP file path** field:
   a. Enter the absolute path to the directory containing your custom JSPs.
   b. To add a condition on the document type, use a dynamic hit meta in the path. For example: `/WEB-INF/jsp/${entry.metas["type"]}.jsp`
   c. The dynamic value will apply a specific JSP template for each type of document that has its own dedicated JSP. The default template is applied otherwise.

   For example, let's say that you have created the following JSP templates: `filesystem.jsp`, `intranet.jsp`, `mail.jsp`, etc.



JSP file path: /WEB-INF/jsp/custom.jsp

5. Select the **Preview** to check your configuration changes.
6. Click **Apply**.

## Using the Google Maps Widget

You can configure your Mashup Builder to display a Google Maps widget for a variety of use cases.

Textual address Vs GPS coordinates

Restrict the search results to a Geographical Area

Link the search results list to a Google map

## Textual address Vs GPS coordinates

In order to use Google Maps widget you must provide either:

- the textual address provided from hit meta such as **city** and **country**, or
- the GPS coordinates via a GPS point meta; this data must be provided from one of your sources and defined in the data model.

**Note:** If you a need a cartographic widget, you can also use the **Maps** widget which gives access to mapquest/openstreetmap OR to BING services.

**Important:** You must pay attention to license issues when using cartographic services! For Google and Bing, you can use the premium versions by defining your API key in the widget configuration. If no key is specified, you will use the light versions of these services.

### Why use textual addresses

- To easily configure the Google Maps widget using the hit meta. This uses Google Maps API for the geocoding therefore, you don't have to supply the GPS coordinates.
- To link the search results to pinpoint on a Google Maps widget. See Link the search results list to a Google map.

### Why use GPS coordinates

- To perform geographical search (both polygon and circle selection), see Restrict the search results to a Geographical Area.
- To perform calculations such as distance (results in proximity) used for the search results in the Mashup UI.

## Restrict the search results to a Geographical Area

You can configure your Mashup Builder to display a Google Maps widget on which the user will be able to draw a polygon or a circle and then launch a standard search on the specified geographical area, for example, a search on the restaurants of this area.

To use the Google Maps widget for geosearch you must provide the GPS coordinates via a GPS point meta; this data must be provided from one of your sources and defined in the data model.

The workflow to implement geosearch in the Mashup Builder is to:

- Prepare your data model.

- Restrict search results to a geographical area.

## Prepare your data model for geographical search

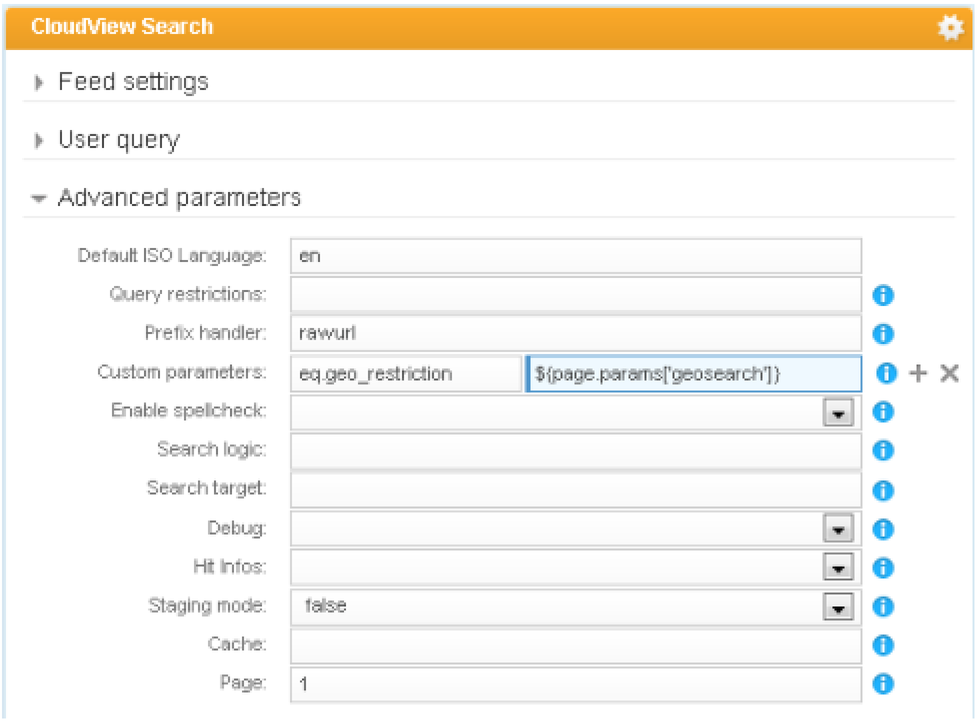
To perform geographical search, you must provide the GPS coordinates via a GPS point meta. This data must be provided from one of your sources and defined in the data model as follows. Therefore, this procedure assumes that the coordinates data has already been indexed.

1. In the Administration Console, select **Data Model > Classes**.
2. Select the property from the data model that will supply GPS coordinates. For example, `location`.
3. For **Data type**, select **GPS point**.
4. Click **Apply**.

   The index field `document_location` can now be referenced in the Mashup Builder as the geographical property to search with.

## Link the Standard Search Form and the Google Maps widgets

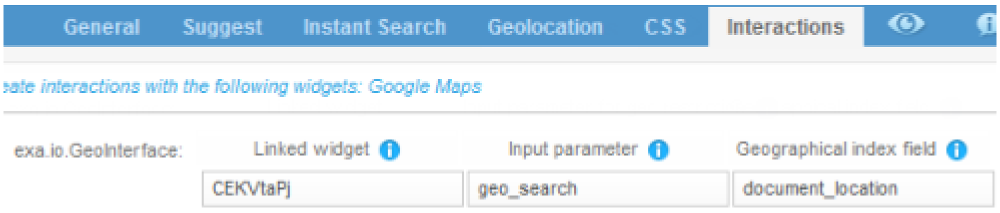
1. In Mashup Builder, select the **Feeds** view, add a parameter in **Page parameters** called `geosearch` that will be used for the geographic restriction.
2. In the Exalead CloudView Search feed, expand **Advanced parameters** and add a custom parameter named `eq.geo_restriction` and the `geosearch` page parameter for the value as shown below.



| | | |
|---|---|---|
| Default ISO Language: | en | |
| Query restrictions: | | |
| Prefix handler: | rawurl | |
| Custom parameters: | eq.geo_restriction | ${page.params['geosearch']} |
| Enable spellcheck: | | |
| Search logic: | | |
| Search target: | | |
| Debug: | | |
| Hit Infos: | | |
| Staging mode: | false | |
| Cache: | | |
| Page: | 1 | |

3. In the **Design** view, drag a **Vizualization > Maps > Google Maps** widget on the **index** page.

4. In the widget properties **General** panel, select the **Enable geo form editing**.
5. Click the header of the **Standard Search Form** widget to display its widget properties panel.
6. Select the **Interactions** tab and configure the **exa.io.GeoInterface** property to restrict the search on a specific geographical area.
   a. For **Linked widget**, select your Google Maps widget from the contextual menu on the left.
   b. For **Input parameter for geoData**, enter the Page parameter you created in step 2, for example `geosearch`.
   c. For **Meta name to search with**, enter the index field name of the geographical property to search with. The format is `<class>_<property>`, for example, `document_location`. See Prepare your data model for geographical search.

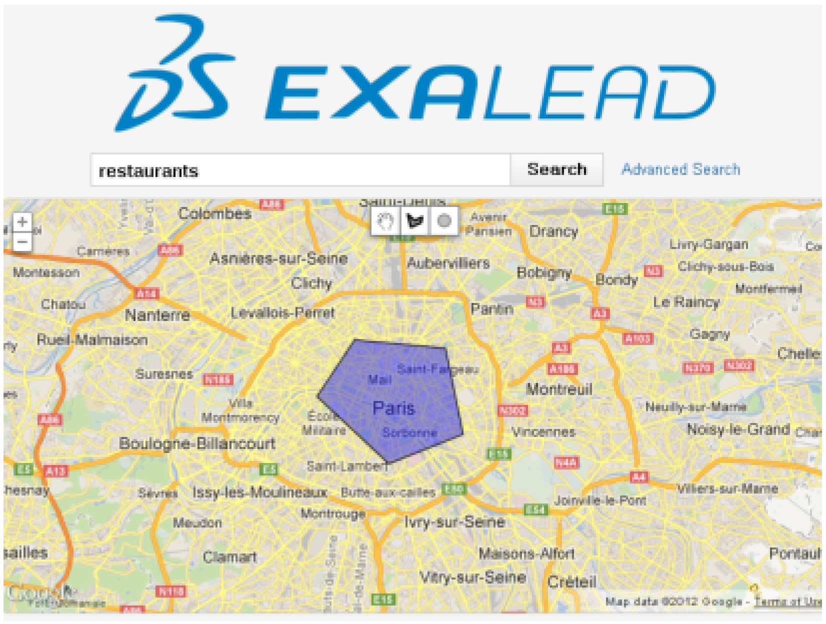


ate interactions with the following widgets: Google Maps

exa.io.GeoInterface: Linked widget: CEKVtaPj; Input parameter: geo_search; Geographical index field: document_location

7. Select the **Preview** to check your configuration changes.

   The Mashup Builder should display a Google Maps widget on which you can draw a polygon or a circle, and a standard search form to search for items in this area. Example of geographical search on Google Maps.



8. Click **Apply**.

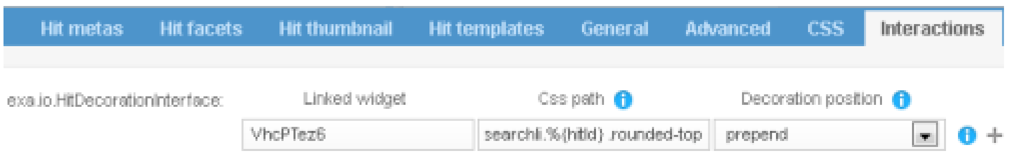
## Link the search results list to a Google map

You can also link up the search results lists to pinpoints on the Google Maps widget using hit meta rather than GPS coordinates. For example, you can view the search results for restaurants and their associated pinpoint location on Google Maps.

1. Go to the **/search** page and select the **Design** view.
2. Drag a **Vizualization > Maps > Google Maps** widget on the page above the **Standard Facets** widget.
3. Click on the **Google Maps** widget header to display its properties.
4. Select the **Based on Entries** tab and then select the hit meta for the restaurant address in **Address (1)**, for example, `${entry.metas['address']}`

   **Note:** Alternatively, the Google Maps widget can reference an index field that contains GPS points for the Location property.

5. Click on the **Result List** widget header to display its properties.
6. Select the **Interactions** tab and configure the **exa.io.HitDecorationInterface** property to add a gmap pinpoint next to the search result title. The aim is to link the search result pinpoint with a corresponding pinpoint on the gmap.
   a. For **Linked widget**, select your Google Maps widget from the contextual menu on the left.
   b. For **Css path**, select **Default template > Header left** from the contextual menu.
   c. For **Decoration position**, choose whether to place the pinpoint before or after the search result title, for our example, **prepend**.

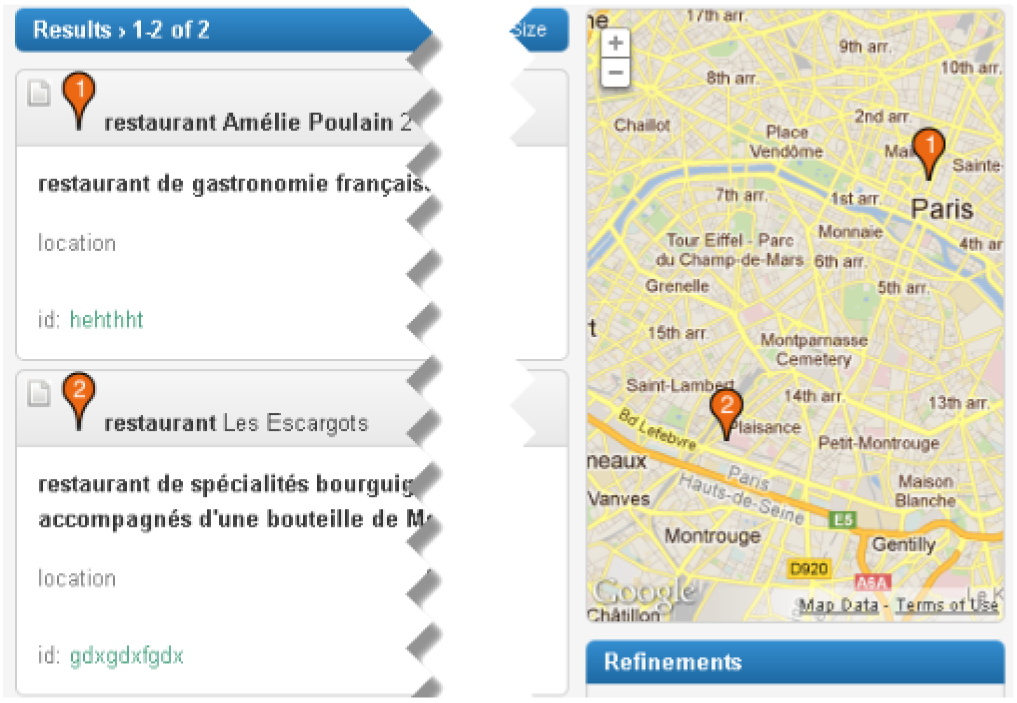


exa.io.HitDecorationInterface:

| Linked widget | Css path | Decoration position |
|---|---|---|
| VhcPTez6 | searchli.%{hitId} .rounded-top | prepend |

7. Select the **Preview** to check your configuration changes.

   The search page of the Mashup Builder should display a list of results with pinpoints before each result title. Hovering on a search result pinpoint should refresh the Google Maps view to display the related entry.


Results › 1-2 of 2
restaurant Amélie Poulain 2
restaurant de gastronomie français
location
id: hehthht
restaurant Les Escargots
restaurant de spécialités bourguig
accompagnés d'une bouteille de M
location
id: gdxgdxfgdx
Paris
Map Data - Terms of Use
Refinements

8. Click **Apply**.

## Adding Trusted Queries

Trusted Queries will guide your end-users through a set of structured suggestions based on your data model classes.

For our example, we’ll set this up for the `file_extension` category facet in the Administration Console. Then, in the Mashup Builder, we will add the Trusted Queries widget on the home page (or `/index` page) of our Mashup Builder application to enable trusted queries for the category facet(s) defined in the Data model.

**Note:** This widget is only available in Mashup Builder Premium.

### Configure Category facets for trusted queries

1. In the Administration Console, select **Data Model** from the menu bar.
2. On the **Classes** tab, from the **Properties for the document class** pane, select a property set as Category Facet. For example, **file_extension**.
3. Expand the **Advanced faceting options**, and select **Trusted queries support**.
4. Repeat steps 2 and 3 for the other category facets on which you want to enable trusted queries.
5. In **Search Logics > Query Language**, make sure that **datamodel_class_hierar-chy(class_hierarchy)** and **trustedqueries** are in the list of prefix handlers.

6. Click **Apply**.

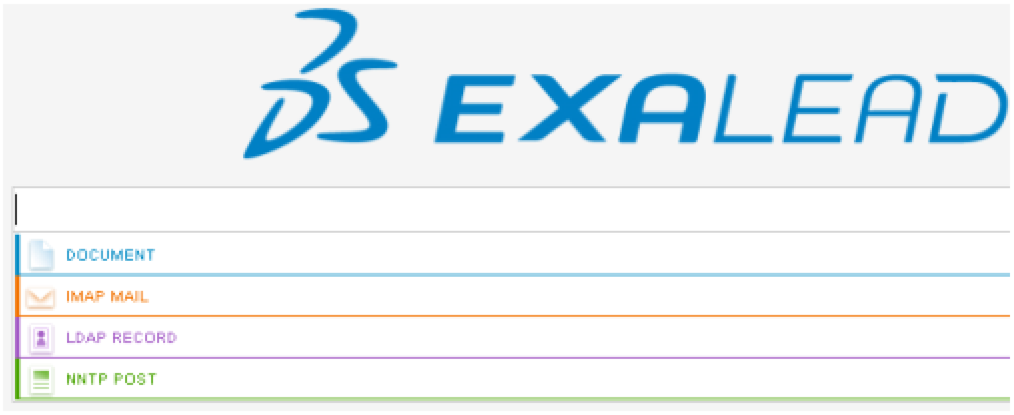
## Add trusted queries in Mashup Builder

1. In **Mashup Builder**, go to the **index** page and select the **Feeds** view.
2. Drag and drop a **CloudView Search** feed.
   a. Give it a name, for example, `cloudview`.
   b. For the **Query** property, enter a query, for example, `#all`.
3. Select the **Design** view.
4. Delete the **Standard Search Form** widget.
5. Add a new **Trusted Queries** search form widget.
   a. In the **Widgets** panel, click the **Search Forms** widget category to expand it.
   b. Drag the **Trusted Queries** widget onto your page.

   The widget properties panel opens.
6. In the **General** tab:
   a. Specify the feed(s) that should be refined when the user query is submitted, for example, `cloudview`.
   b. Configure the **Action** property to submit on the `search` page.
7. In the **Search API** tab, for **Search Logic**, specify the search logic to use. If no search logic is defined, the default one will be used.

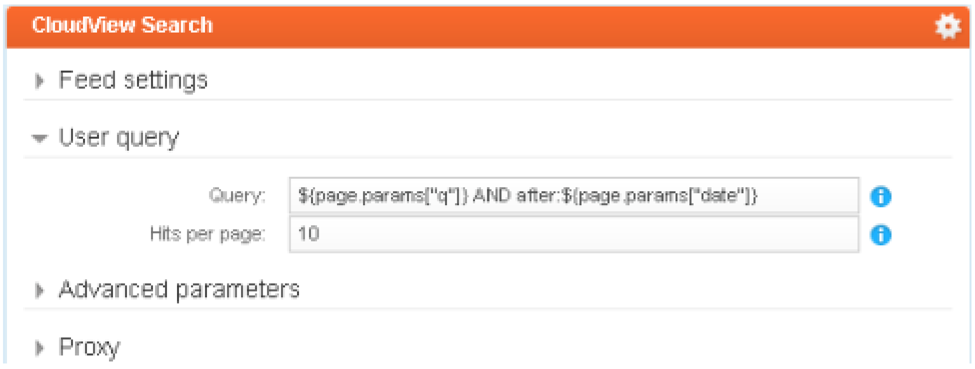
   **Note:** The trusted query widget uses the Search API directly.
8. Select the **Preview** and click in the **Search** field.

   The Trusted Queries search form displays a list of document types to guide you through the content.



9. Click **Apply**.

For example, if your feed query includes a date specification like `${page.params["q"]} after: ${page.params["date"]}`, you can test it with a date value in the **Preview** page to see the feed behavior.



CloudView Search

Feed settings

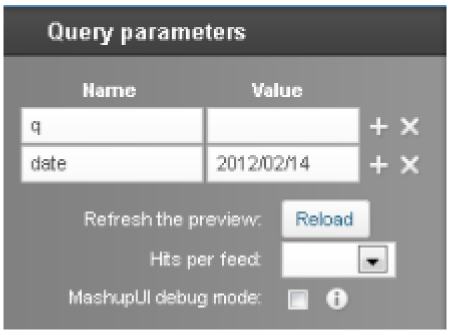
User query

Query: ${page.params["q"]} AND after:${page.params["date"]}

Hits per page: 10

Advanced parameters

Proxy



Query parameters

| Name | Value |
|---|---|
| q | |
| date | 2012/02/14 |

Refresh the preview: Reload

Hits per feed:

MashupUI debug mode:

## Customizing the Look and Feel

Mashup Builder gives you the possibility of customizing the look and feel of your search applications very precisely. You can edit the default look and feel for the whole application and/or customize specific pages only.

### Modify the logo

By default, Mashup Builder is set up to use the Exalead logo. You can easily install your own logo and switch to your logo to customize your application.

1. Copy your logo into `<DATADIR>/webapps/360-edm-mashup-ui/resources/logo/images`.
2. Open Mashup Builder: http://<HOSTNAME>:<BASEPORT+1>/mashup-builder.
3. Select the **/index** page, and then select the **Design** view.
4. Click the header of the **Image** widget.
5. On the widget properties panel at the bottom of the page, modify the **Image URL** path so it points to your logo file.
6. Go to the **/search** page and repeat the previous two steps.
7. Select the **Preview** to check your configuration changes.

**Note:** The placeholder image inside the **image** widget does not change.

8. Click **Apply**.

## Switch to another theme

By default, Mashup Builder is set up to use the standard **theme_enterprise** theme. You can easily install your own themes and switch to another theme to customize your application.

1. You must customize your own theme in a `theme.less` file.
2. Add your custom theme in a new directory (for example `theme_MyCompany`) under `<DATADIR>/webapps/360-edm-mashup-ui/resources/themes/`
3. Repeat steps 1 and 2 to add more custom themes.
4. In **Mashup Builder**, click the **Application** button at the top left of the screen.
5. In **Developer Area**, click **Reload components**.
6. In **Application properties**, select another **Theme**, for example, `theme_MyCompany`.
7. Select the **Preview** to check your configuration changes.
8. Click **Apply**.

## Customize the look and feel for a whole application

You may want to go further than applying themes to customize the look and feel of your Mashup Builder application. This can be performed either by referencing CSS and JavaScript files, or by entering inlined CSS or JavaScript code.

1. In **Mashup Builder**, click the **Application** button at the top left of the screen.
2. From the **Styles** menu, specify:
   - **Style files**: Enter the path(s) of the CSS files that you want to use on your application.
   - **Style inlined**: Enter the inlined CSS code that you want to use on your application (to use CSS directly within your HTML code).
3. From the **JavaScript** menu, specify:
   - **JavaScript files**: Enter the path(s) of the JavaScript files that you want to use on your application.
   - **JavaScript inlined**: Enter the inlined JavaScript code that you want to use on your application (to use JavaScript directly within your HTML code).
4. Click **Apply**.

## Customize the look and feel of a page

You may want to have a different look and feel for one or several pages of your application, and override the theme that is applied all throughout, see Switch to another theme. This can be performed by customizing the CSS of the page(s).

1. In **Mashup Builder**, select the page you want to customize, for example **/search**.
2. Select **Edit page settings** at the right of the screen.
3. From the **General** section, you can customize the page title with variables and meta headers.
4. From the **Resources** section, specify:
    - **Style files**: Enter the path(s) of the CSS files that you want to use on your page.
    - **Styles inlined**: Enter the inlined CSS code that you want to use on your page (to use CSS directly within your HTML code).
5. From the **Style** section, specify:
    - **CSS ‘id=’ of the body page**: The page unique id that will be used in the CSS.
    - **CSS ‘class=’ of the body page**: The page class name that will be used in the CSS.
6. Select the **Preview** to check your configuration changes.
7. Click **Apply**.

## Add custom code to a page

You can change the look and feel or the behavior of a page by specifying custom code.

1. In **Mashup Builder**, select the page you want to customize, for example **/search**.
2. Select the **Code** view and in:
    - **Styles**: Enter the inlined CSS code that you want to use on the page.
    - **JavaScript**: Enter the JavaScript code that you want to use on the page.
3. Select the **Preview** to check your configuration changes.
4. Click **Apply**.

## Customize the layout of a widget with a CSS call

You can change the standard style of widgets by customizing their CSS

1. In **Mashup Builder**, select the page containing the widget you want to customize, for example **/search**, and select the **Design** view.
2. Click the header of the widget, for example **Result List**.

3. On the widget properties panel, select the **CSS** tab and specify the following:
   - **Widget CSS ‘id=’**: Specify the widget’s unique id that will be used in the CSS.
   - **Widget CSS ‘class=’**: Specify the widget’s class name that will be used in the CSS.
4. Select the **Preview** to check your configuration changes.
5. Click **Apply**.

## Customize the layout of a widget with JavaScript

For several chart widgets, you can also set the look and feel from the **JavaScript** tab by adding custom JavaScript code.

**Note:** Most chart widgets come from the highcharts and highstock libraries, we recommend reading their documentation on `http://www.highcharts.com/` for advanced configuration in the **Javascript** tab of the chart widget properties.

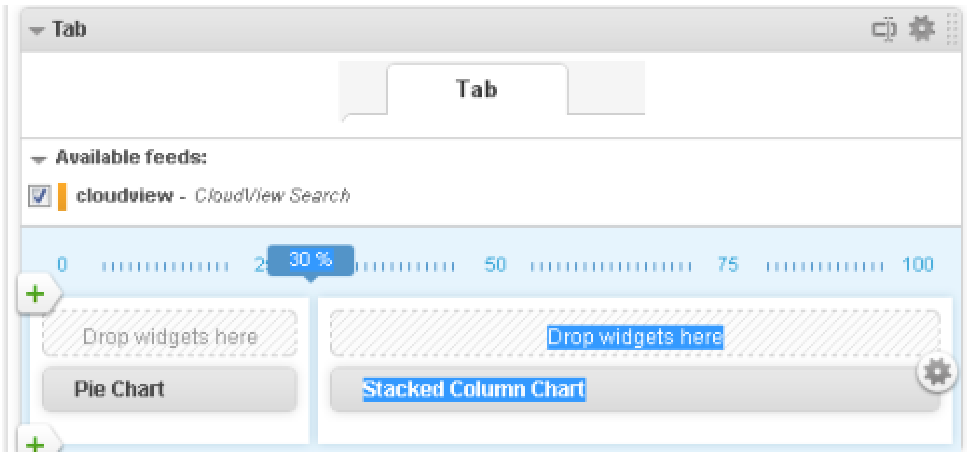
1. For example, if you want to customize the colors used by a pie chart widget:
   - Go to the **JavaScript** tab.
   - Uncomment the `colors` line by removing `/*` and `*/`
   - Specify your favorite colors using their HTML hexadecimal codes. For example: `colors: ['#058DC7', '#50B432', '#ED561B', '#FF9900', '#CC33CC', '#660099', '#FF00FF']`
2. Select the **Preview** to check your configuration changes.
3. Click **Apply**.

## Edit the layout of widgets within a widget container

For several widgets acting as containers (Tab, Table widgets), you can configure the layout of sub-widgets by adding or merging cells with the cell designer, resize the width of cells, etc.

1. For example, add two charts within a **Tab** container widget (the Tab widget must be in a **Tabs Container** widget).
2. Click the Edit widget properties icon ⚙ and select **Edit widget layout**.
3. Hover your cursor over the cells (the sub-widgets) to display the cell designer, or between two cells to modify the width of the cells.

   *In this example, the width of the Stacked Column Chart widget has been increased to use 70% of the Tab’s width*



4. Select the **Preview** to check your configuration changes.
5. Click **Apply**.

# Managing Applications

Mashup Builder lets you create and manage one or several search applications in a single Exalead CloudView installation.

By default, the Mashup Builder menu lets you work on a **default** application presenting a standard configuration, the **Index** and **Search** pages being organized in a very simple way. You can customize this template and give it a more elaborate display.

You can also create new application instances using predefined templates, that is to say, **template_web** for web applications and **template_mobile** for mobile applications (with Mashup Builder Premium only). These new applications will actually have different entry points or context paths in your Exalead CloudView installation.

This section describes how to manage applications within Mashup Builder

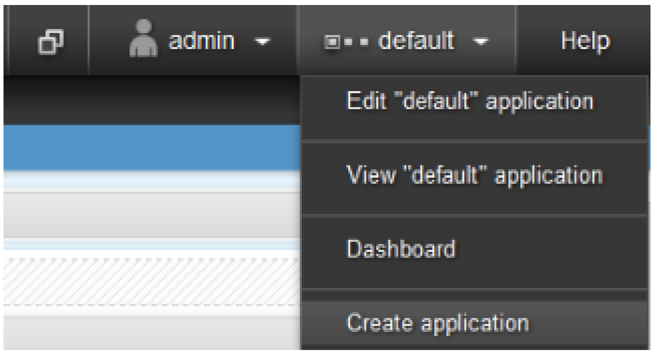
## Creating New Applications

With both Mashup Builder and Mashup Builder Premium, you can create several web applications.

### Create a new application

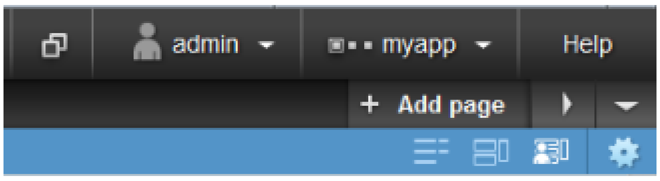
The creation of mobile applications is possible with Mashup Builder Premium only.

1. In **Mashup Builder**, select **Create application** from the top menu bar.



2. In the **Create a New Application** dialog box:
   a. Enter an application name, for example `myapp`
   b. Select a mashup base template from the pull-down menu: **template_web**, **template_mobile** (with Mashup Builder Premium only), or any application already created.
   c. Select the host on which you want to deploy the application.
   d. Click **OK**.

   The new application is added to the top menu bar.



**Note:** The Mashup API and Mashup UI roles used by the host to manage applications, are defined in the Administration Console (in **Deploy > Roles**).

## Select the application to edit

1. If there are more than one application, you can select the application you want to edit by clicking the application menu in the menu bar.
2. A list of available Applications displays, and you can click one of the **Edit > “application name” > application** menu item.

## Deploy an application to another Exalead CloudView instance

1. For details on deploying an application to another Exalead CloudView instance, see "Deploying Applications on Another Instance" in the Exalead CloudView Administration Guide.

# Managing Custom Components

Mashup Builder Premium gives you the possibility of adding plugins for feeds, widgets and triggers. You can therefore install and deploy custom components on your applications.

## Install plugins

The following procedures explain how to install and manage a plugin in your application. It is basically a .zip file with custom code for a feed, widget, or trigger, that must be imported within Mashup Builder.

### Install a plugin

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage components > Plugins**.
3. Click **Upload plugin**.
4. From the **Upload Plugin** dialog box, click **Browse** and select the plugin `zip` file.

The plugin is added to the list of plugins in the **Installed plugins** pane.

### Uninstall a plugin

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage components > Plugins**.
3. Click the **delete** icon corresponding to the plugin to uninstall.

### Uninstall a plugin manually

This procedure describes how to uninstall a plugin manually and cleanup all its references. This can be useful if a plugin breaks the search server or the gateway. In the following examples, the plugin to uninstall is called `MyPlugin1`.

1. Go to your `<DATADIR>/gct/registeredPlugins/` directory and remove all references of MyPlugin1.

```
gct/registeredPlugins/
|-- default|
   |-- 360-administration-service|
   |    |-- MyPlugin1.cvplugin.properties|
   |-- 360-edm-mashup-ui.mu0|
   |    `-- MyPlugin1.cvplugin.properties
```

2. Go to your `<DATADIR>/extrajava/` directory and remove all its references of MyPlugin1.

```
extrajava/
|-- 360-plugins
|    `-- applications
|         `-- default
|              `-- MyPlugin1
```

## Synchronize the plugins with another application

The following procedure details how to synchronize the plugins of a given application with another application. This action compares the plugins of your current application with the plugins of a source application. It then adds or deletes plugins on your current application so as to get exactly the same as those of the source application.

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage components > Plugins**.
3. Click **Synchronize plugins**.

   A **Synchronize plugins** dialog box opens.
4. Select the source application from the drop-down list.
5. Click **Continue**.

   **Note:** If you synchronize a plugin that already exists on the current application (a plugin with the same name), an error message will be displayed.

## Import custom components

You can import custom components for the:

- MashupAPI: feeds, triggers, etc., from `<DATADIR>/javabin/`
- MashupUI: triggers, security providers, etc., from `<DATADIR>/webapps/360-edm-mashup-ui/WEB-INF/lib`

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage component** > **Register Components**.
3. Click **Add new custom component** and enter the full path of the custom class you want to import.
4. Click **Save** and apply the configuration.

**Note:** If you change the code of your registered components, do not forget to save, apply and restart the search servers to get your components correctly updated. The **Restart search server processes** option is accessible when you click **Developer area > Reload components**.

## Use plugin controllers

Mashup Builder is based on a Spring Framework and uses an MVC (Model View Controller) model.

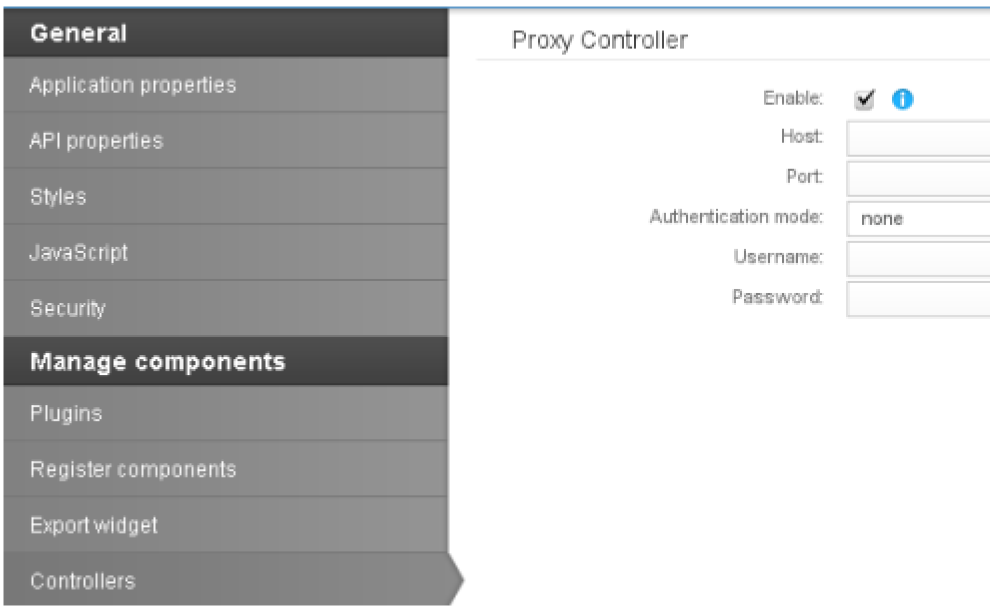
In Mashup Builder, several plugins embed Spring controllers. A controller can send commands to its associated view to change the view's presentation of the model. For example, a controller can launch queries to the Suggest service, take a URL and return a screenshot of the page, send an email, etc.

Once installed, controllers can be configured through the Mashup Builder interface. This avoids configuring the plugin XML file manually.

**Note:** For information on Controllers' creation and packaging, see "Creating Controllers" in the *Exalead CloudView Mashup Programmer's Guide*.

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage component** > **Controllers**.
3. Configure the plugins as needed using the controllers.

   On the following screenshot, a Proxy plugin containing a controller has been installed and can therefore be seen under **Controllers**.



Enable: ✓
Host:
Port:
Authentication mode: none
Username:
Password:

4. Restart the search servers to get your components correctly updated. The **Restart search server processes** option is accessible when you click **Developer Area > Reload Components**.

### Export widget

If you have created custom widgets, either through the Widget Builder or through custom code, it can be useful to export them. You can do so to share your widgets as a plugins with other Exalead CloudView users, or simply to backup your widgets if necessary.

For more information about widget creation, see "Creating Widgets" in the Exalead CloudView Mashup Programmer's Guide.

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **Manage component** > **Export Widget**.
3. Configure the widget to export as follows:
    a. **Widget**: Select the widget to export.
    b. **Name**: Give a name to your widget plugin.
    c. **Description**: Enter a description.
    d. **Author**: Enter the name of the widget’s creator.
4. Click **Download this widget** and save the widget plugin zip file.

## Adding Security to Your Application

You can add authentication to your application by defining a security provider.

Once a security provider has been applied to an application, it is possible to enable the security on one or several pages of this application.

**Note:** The Mashup framework provides built-in support for standard Exalead CloudView security sources but you can also implement your own front-end security provider using the Mashup Builder SDK.

### Add a CloudView Security Provider

The following procedure describes the setup of a standard Exalead CloudView security provider to use Exalead CloudView security sources.

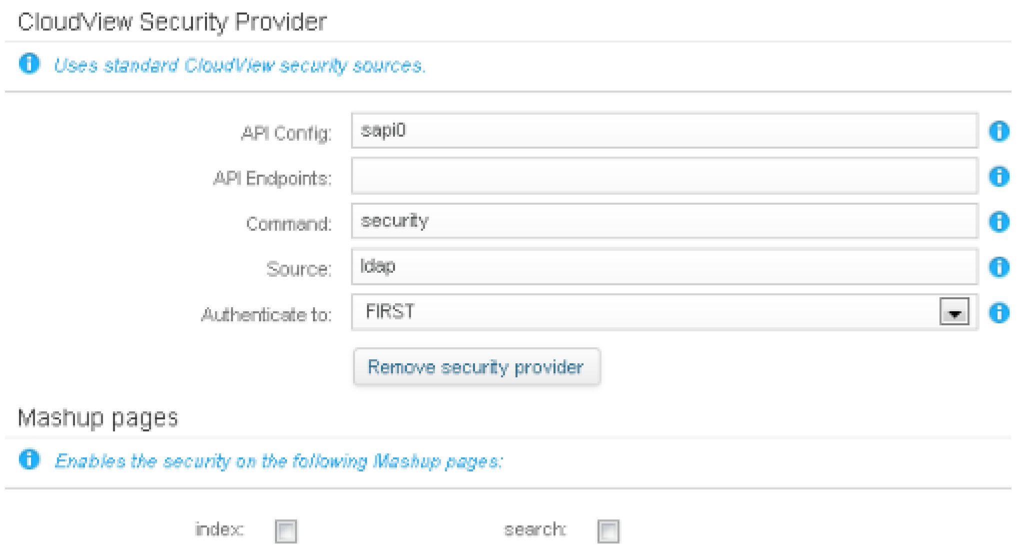
We assume that a security source named `ldap` has been properly configured in the Administration Console. For more information, see "Configuring security sources" in the Exalead CloudView Administration Guide

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Select **General** > **Security**.
3. Click **Add a security provider**.
4. From the **Add security provider** dialog box, select the Exalead CloudView **Security Provider**.
5. Configure the security provider.

   In **Source**, select a security source that has previously been configured in the Administration Console, for example `ldap`.

   See the following table if you want to configure the other properties.

| **Property** | **Description** |
|---|---|
| **API Config** | Indicates the name of the default Search API as defined in the **Applications** menu, for example `sapi0`. |
| **API Endpoints** | Defines the URL that will be used by the Search API. For example: `http://<HOST>:<PORT+10>` |
| **Command** | Specifies the authentication command. Default is `security`. |
| **Source** | Enter the name(s) of your CloudView security source(s). |
| **Authenticate to** | Select the authentication behavior of your security source(s). By default, the first source will be used for authentication. |


CloudView Security Provider

*Uses standard CloudView security sources.*

| | |
|---|---|
| API Config: | sapi0 |
| API Endpoints: | |
| Command: | security |
| Source: | ldap |
| Authenticate to: | FIRST |

Mashup pages

*Enables the security on the following Mashup pages:*

index: ☐ search: ☐

6. In the **Mashup pages** section, select the pages of your application on which you want to enable security.

7. Click **Save**.

   An **Authentication Required** dialog box asking for your credentials is added to the page(s) on which security was enabled. These credentials are the ones defined in the security source.

## Add a Kerberos Security Provider

This section describes the setup of a Kerberos security provider to secure the access to your application pages.

The installation procedure requires to:

- Pre-authenticate as a host, using a HOST set of credentials and a keytab.
- Install the spnego plugin (ask it to the Exalead CloudView Support team).

If you want to secure the application sources, you can define a security source in the Administration Console. For more information, see "Configuring security sources" in the Exalead CloudView Administration Guide.

### Prepare the Kerberos configuration files

You first need to generate the kerberos configuration files and copy them to a folder on the host running Exalead CloudView.

1. Ask your system administrator to generate the files required to connect to the Kerberos server on the host on which Exalead CloudView is running. These are:

   - the `krb5.conf`,
   - the `login.conf`,
   - and the `krb5.keytab` files.

   `login.conf` sample file

```
spnego-client {
        com.sun.security.auth.module.Krb5LoginModule required;
};
spnego-server {
        com.sun.security.auth.module.Krb5LoginModule required
        storeKey=true
        isInitiator=false
        useKeyTab=true
        principal="HOST/<hostname>@OFFICE.EXAMPLE.COM"
        keyTab="/johndoe/kerberos/krb5.keytab";
};
```

   `krb5.conf` sample file

```
[libdefaults]
        default_realm = DOMAIN.EXAMPLE.COM
        dns_lookup_kdc = on
        dns_lookup_realm = on
[domain_realm]
        site1.example.com = DOMAIN.EXAMPLE.COM
        .site1.example.com = DOMAIN.EXAMPLE.COM
        site2.example.com = DOMAIN.EXAMPLE.COM
        .site2.example.com = DOMAIN.EXAMPLE.COM
[realms]
        DOMAIN.EXAMPLE.COM = {
                kdc = domain.example.com
                admin_server = domain.example.com
                kpasswd_server = domain.example.com
}[logging]
        kdc = SYSLOG
        admin_server = SYSLOG
        default = SYSLOG
```

**Important:** The `login.conf` file must reference the KeyTab file (see the line highlighted in green in the `login.conf` sample file above).

**Note:** For information about the KeyTab file generation, see the SPNEGO documentation: `http://spnego.sourceforge.net/client_keytab.html`

2. Your system administrator must also add a principal name, using the following commands:
   - `setspn` for Active Directory,
   - or `kadmin add_principal` command for MIT Kerberos.

   The principal name must have the following format: `HTTP/<Server name as shown in the browser URL>`

   **Note:** See your browser's documentation to enable Kerberos authentication.

3. In the Administration Console, add a security source (for instance a unix security source) to fetch security tokens.

## Install the spnego plugin

1. Ask the Exalead CloudView Support team for the spnego plugin.
2. In **Mashup Builder**, select **Application** from the top left menu bar.
3. Select **Manage components** > **Plugins**.
4. Click **Upload plugin** and select the spnego plugin.

## Add the Kerberos Security Provider to your application

1. In **Mashup Builder**, select **Application** from the top left menu bar.

2. Select **General** > **Security**.
3. Click **Add a security provider**.
4. From the **Add security provider** dialog box, select the Kerberos **Security Provider**.
5. Configure the Kerberos security provider.
    - For **spnego.login.conf**, enter the relative or absolute path of the `login.conf` file.
    - For **spnego.krb5.conf**, enter the relative or absolute path of the `Krb5.conf` file.
    - For **Source**, enter the name of your Exalead CloudView security source.
    - See the following table if you want to configure the other properties.

| Property | Description |
|---|---|
| **spnego.login.conf** | [Required] Path to `login.conf` file (relative or absolute path) |
| **spnego.krb5.conf** | [Required] Path to `krb5.conf` file (relative or absolute path) |
| **spnego.preauth.usernan** | Enter the Network Domain user name. For Windows, this is sometimes referred to as the Windows NT user name. |
| **spnego.preauth.passwo** | Enter the Network Domain password. For Windows, this is sometimes referred to as the Windows NT password. |
| **spnego.login.server.moc** | Enter the server module name specified in the `login.conf` file. |
| **spnego.login.client.mod** | Enter the client module name specified in the `login.conf` file. |
| **spnego.logger.level** | Specify a logging level to define the amount of details to display. Valid values go from `1` to `7` (`1` = FINEST; `7` = SEVERE). Set value to `1` for debugging/verbose logging. |
| **spnego.prompt.ntlm** | The SPNEGO Filter does not support NTLM.<br>Set this value to `true` if clients who wish to authenticate via NTLM should be offered Basic Authentication (assuming `spnego.allow.basic=true`).<br>Set this value to false if NTLM Authentication should be rejected. |
| **spnego.allow.unsecure.b** | With respect to Basic Authentication, specify if HTTPS is required. If Basic Authentication is not allowed, this operation is a no-op.<br>Set this value to `false` if you do not want to offer Basic Authentication for non-SSL connections. |
| **spnego.allow.localhost** | This property is set to false by default, which means that requests coming from local host will not require authentication. |

| Property | Description |
|---|---|
| | Set this value to `true` if you run a local instance of the server and you want to avoid having to register an SPN for your workstation. |
| **spnego.allow.basic** | Valid values are true or false.<br>Offer HTTP Basic Authentication in addition to Kerberos Authentication.<br>Consider this option if an HTTP client cannot negotiate SPNEGO token(s).<br>Set this value to false if you only want to allow Kerberos Authentication. |
| **API Config** | Indicates the name of the default Search API as defined in the **Applications** menu, for example `sapi0`. |
| **API Endpoints** | Enter the URL that will be used by the Search API. For example: `http://<HOST>:<PORT+10>` |
| **Command** | Specifies the authentication command. Default is `security`. |
| **Source** | [Required] Enter the name(s) of your CloudView security source(s) if any. The security source is defined in the Administration Console. It fetches user security tokens.<br>For more information, see "Configuring security sources" in the Exalead CloudView Administration Guide. |
| **Authenticate to** | Select the authentication behavior of your security source(s). By default, the first source will be used for authentication. |

6. In the **Mashup pages** section, select the pages of your application on which you want to enable security.
7. Click **Save** and **apply** your configuration.

On your mashup application, the secured page(s) should not ask for your credentials as Kerberos is an SSO protocol. The retrieved credentials are the ones defined in the security source.

## Enabling the Reporting Services

In Mashup Builder **> Application**, you can enable reporting for your Mashup Builder applications and for the Mashup API. In both cases, reporting relies on reporters which can be configured in the Administration Console **> Reporting** menu.

**Note:** To configure the reporters available in Mashup Builder, see "Analyzing User Queries with Reporters" in the Exalead CloudView Configuration Guide.

## Enable reporting on your Mashup Builder applications

The edm-mashup-ui-reporting reporter collects data relative to task execution and to CPU activity on the Mashup UI. For example, when a user queries a page, the reporter retrieves data such as the execution and CPU time of pages, widgets and triggers.

Once configured in the Administration Console, this reporter must be enabled in the Mashup Builder **> Application > Application Properties** menu.

As for all reporters, the information collected by the edm-mashup-ui-reporting reporter can be exported to CSV, JDBC and SQLite repositories, to make reports with external tools. It can also be sent to a Push API publisher if you want to index this data and use it like other indexed Exalead CloudView data.

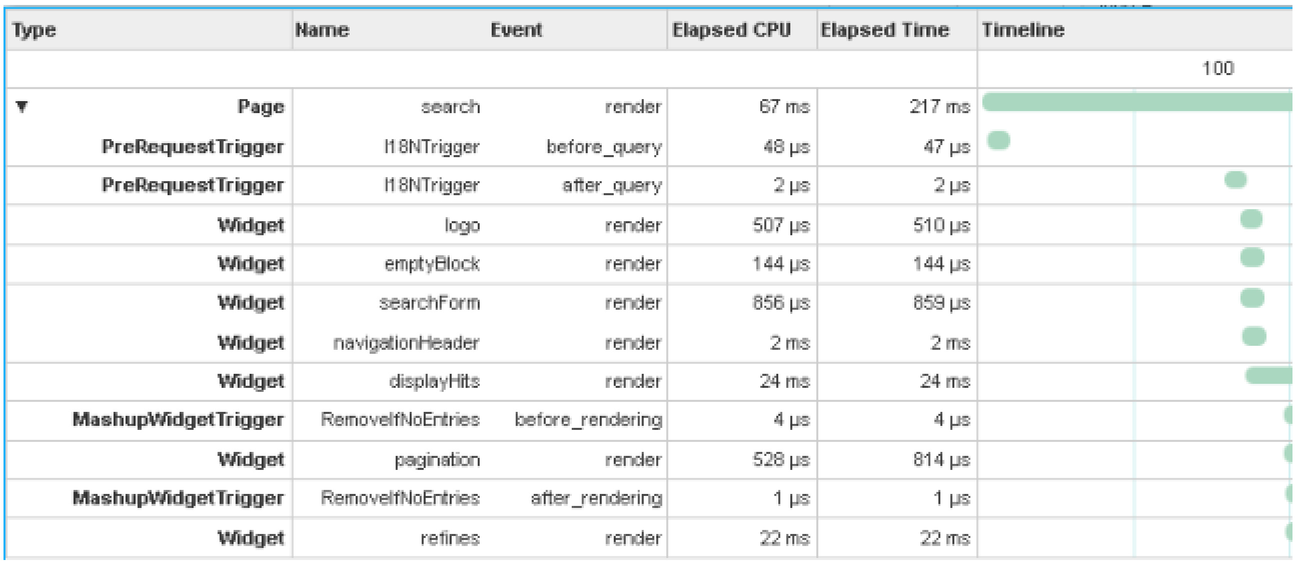
The difference with the other reporters, is that you can directly display collected information into a timeline chart, accessible when the debug mode is enabled.

Once reporting is enabled for a given application, the debug mode displays a timeline tab, allowing you to get a graphical view of collected data.

1. In Mashup Builder, select **Application** from the top left menu bar.
2. Select **General** > **API Properties**.
3. In the **Reporting** section:
    a. Select the **Enable reporting** checkbox.
    b. Click in the **Reporter** field and select the **edm-mashup-ui-reporting** reporter.
4. Go to **Developer area** and select the **Mashup UI debug mode**.
5. Open your Mashup Builder application.

    The **Debug** bar displays at the bottom of your search application pages.
6. Click the **Timeline** tab.

    A window displaying a timeline chart opens (see the following screenshot example).



| Type | Name | Event | Elapsed CPU | Elapsed Time | Timeline |
|---|---|---|---|---|---|
| | | | | | 100 |
| ▼ Page | search | render | 67 ms | 217 ms | |
| PreRequestTrigger | I18NTrigger | before_query | 48 µs | 47 µs | |
| PreRequestTrigger | I18NTrigger | after_query | 2 µs | 2 µs | |
| Widget | logo | render | 507 µs | 510 µs | |
| Widget | emptyBlock | render | 144 µs | 144 µs | |
| Widget | searchForm | render | 856 µs | 859 µs | |
| Widget | navigationHeader | render | 2 ms | 2 ms | |
| Widget | displayHits | render | 24 ms | 24 ms | |
| MashupWidgetTrigger | RemoveIfNoEntries | before_rendering | 4 µs | 4 µs | |
| Widget | pagination | render | 528 µs | 814 µs | |
| MashupWidgetTrigger | RemoveIfNoEntries | after_rendering | 1 µs | 1 µs | |
| Widget | refines | render | 22 ms | 22 ms | |

## Enable reporting for the Mashup API

The edm-mashup-api-reporting reporter collects data relative to feeds, subfeeds and triggers execution. This reporter allows you to understand explicitly the feed execution process, with subfeeds and triggers and to identify possible problematic issues.

Once configured in the Administration Console, this reporter must be enabled in the Mashup Builder **> Application > API Properties** menu.

As for all reporters, the information collected by the edm-mashup-api-reporting reporter can be exported to CSV, JDBC and SQLites repositories, to make reports with external tools. It can also be sent to a Push API publisher if you want to index this data and use it like other indexed Exalead CloudView data.

The difference with the other reporters, is that you can directly display collected information into a timeline chart, accessible when the debug mode is enabled.

**Note:** To get the count of unique users in your edm-mashup-api-reporting (this is displayed in the **Business Console > Query Reporting** graph), you must select the **Enable Security** option on your feeds. Otherwise, the count will return 0. For more information, see Enable security on a Exalead CloudView Search feed.

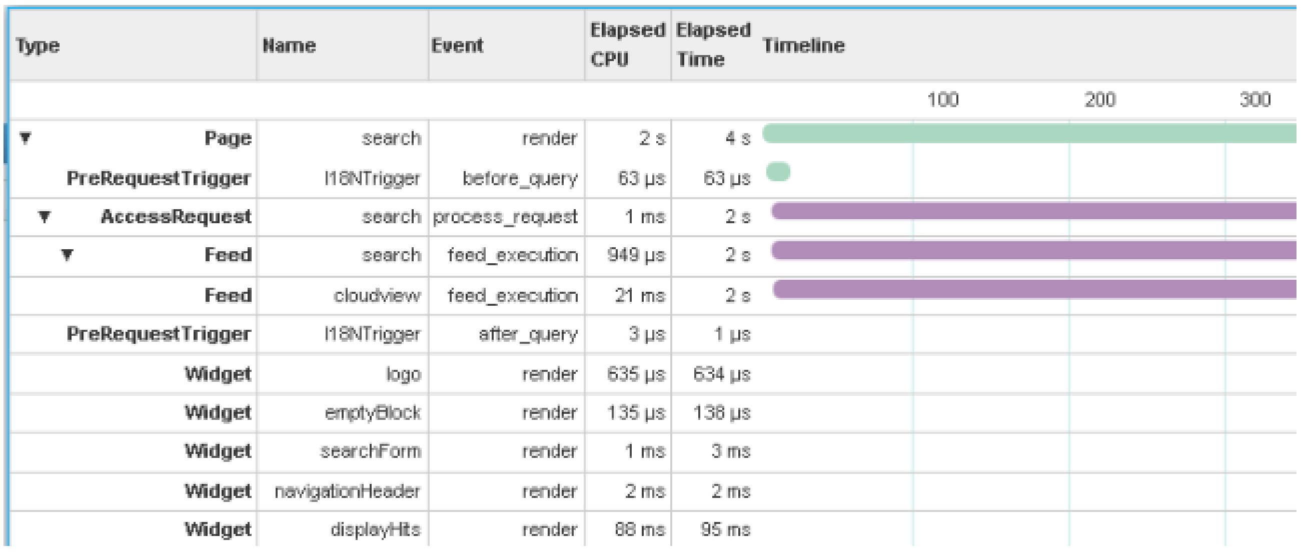
1. In Mashup Builder, select **Application** from the top left menu bar.
2. Select **General** > **API Properties**.
3. In the **Reporting** section:
    a. Select the **Enable reporting** checkbox.
    b. Click in the **Reporter** field and select the **edm-mashup-api-reporting** reporter.
4. Go to **Developer area** and select the **Mashup UI debug mode**.

5. Open your Mashup Builder application.

   The **Debug** bar displays at the bottom of your search application pages.

6. Click the **Timeline** tab.

   A window displaying a timeline chart opens (see the following screenshot example). In the **Type** column, the edm-mashup-api-reporting information can be seen for the **Feed** and **Feed triggers** entries.



| Type | Name | Event | Elapsed CPU | Elapsed Time | Timeline |
|---|---|---|---|---|---|
| | | | | | 100 200 300 |
| ▼ Page | search | render | 2 s | 4 s | |
| PreRequestTrigger | I18NTrigger | before_query | 63 µs | 63 µs | |
| ▼ AccessRequest | search | process_request | 1 ms | 2 s | |
| ▼ Feed | search | feed_execution | 949 µs | 2 s | |
| Feed | cloudview | feed_execution | 21 ms | 2 s | |
| PreRequestTrigger | I18NTrigger | after_query | 3 µs | 1 µs | |
| Widget | logo | render | 635 µs | 634 µs | |
| Widget | emptyBlock | render | 135 µs | 138 µs | |
| Widget | searchForm | render | 1 ms | 3 ms | |
| Widget | navigationHeader | render | 2 ms | 2 ms | |
| Widget | displayHits | render | 88 ms | 95 ms | |

## Enhancing Application Response Time with Gzip Compression

Gzip compression is activated by default to improve the application response time. It applies to the HTML, CSS, and JS code returned by the web server to the web browser.

**Recommendation:** It is better to leave Gzip compression activated. This section explains how to deactivate Gzip compression if required.

1. Go to the `<DATADIR>/webapps/360-mashup-ui/WEB-INF/` directory.
2. Edit the `web.xml` file and comment the content of `Gzip filter: content compression`
3. Repeat these actions for your other Mashup Builder applications, if any.
4. Restart Exalead CloudView.

## Clearing Application or Widget Storage

The following procedure details how to clear the data storage for a given application or for a specific widget.

1. In Mashup Builder, select **Application** from the top left menu bar.

2. Select **Developer area**.
3. In the **Actions** pane, click **Clear storage**.

   A **Clear storage** dialog box opens.
4. Either select:
   - the **All application data** radio button if you want to clear all data including widget data.
   - or the **Specific widget** radio button if you want to select and clear the data of a specific widget only. This can be useful to clear the stored data of a collaborative widget.
5. Click **OK**.

## Deleting an Application

The following procedure details how to delete a given application.

1. In **Mashup Builder**, select **Application** from the top left menu bar.
2. Click the ⚙ icon at the right the application menu bar.
3. Click **Delete application**.

   A **Confirm** dialog box opens.
4. Type in **YES** and click **OK**.

## Troubleshooting your application

This section lists the most common errors and how to troubleshoot them.

### Check the Mashup logs

Checking the logs is a very good starting point for troubleshooting your Mashup application. To check the Mashup logs:

1. In the Administration Console, select **Troubleshooting > Logs** from the menu bar.
2. From the **Processes** pull-down menu, select **searchserver-ss0**.
3. Click **Add**.

The **searchserver-ss0** log displays in a new tab.

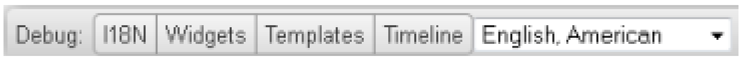
### I can’t see any data for a specific widget in my Mashup Builder

Don’t panic, this is certainly related to the most common error: you forgot to select a feed in your widget and therefore, you don’t have any source data.

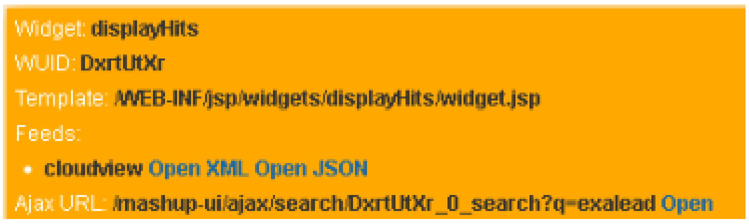
Otherwise, it means that the query is incorrect. To find the error, use the debug mode as follows:

1. Click **Application** and select **Developer area**.
2. In the **Mashup UI** section, select **Mashup UI debug mode**.
3. Open your Mashup Builder.

   A **Debug** bar is displayed at the bottom of your search application pages.



4. Click **Widgets** to highlight the various widgets used in the page.
5. Hover over the buggy widget and use either the Open XML or the Open JSON links to see the Mashup API output and check out feed errors.



Widget: displayHits
WUID: DxrtUtXr
Template: /WEB-INF/jsp/widgets/displayHits/widget.jsp
Feeds:
• cloudview Open XML Open JSON
Ajax URL: /mashup-ui/ajax/search/DxrtUtXr_0_search?q=exalead Open

## My charts don’t display correctly

1. Check that the correct facets are selected. Otherwise, check that the query is correct.

## How to test my MEL expressions and calculations?

We recommend using the **HTML** widget to test your MEL expressions and calculations.

1. In the **HTML code** field, enter your MEL expressions/calculations.

   **Note:** You can press F11 to expand the editor.

2. To validate your MEL syntax, go to the contextual menu, select the **Debug** tab and click **Validate**.
3. Click the Preview icon to check that your code calculates properly.

   **Note:** For more details on MEL expressions, see Appendix - Mashup Expression Language.

## How to change IE compatibility to a higher version?

By default, Mashup Builder enforces Internet Explorer compatibility to IE9. You can change this behavior for each mashup page, by overriding compatible versions in the `http-equiv` meta.

1. In the Mashup Builder, click **Edit Page settings**.

2. In the **Page properties > General** section, edit the **Meta http-equiv** as follows:
   a. For **name**, enter `X-UA-Compatible`
   b. For **Value**, enter the compatible version(s) as you would do manually in the `content` attribute, for example `IE=EDGE`.
3. Repeat the previous steps for other pages if necessary and click **Apply**.

# Running an Application in 3DDashboard

This section describes the main steps to install and configure an application in 3DDashboard.

## Overview

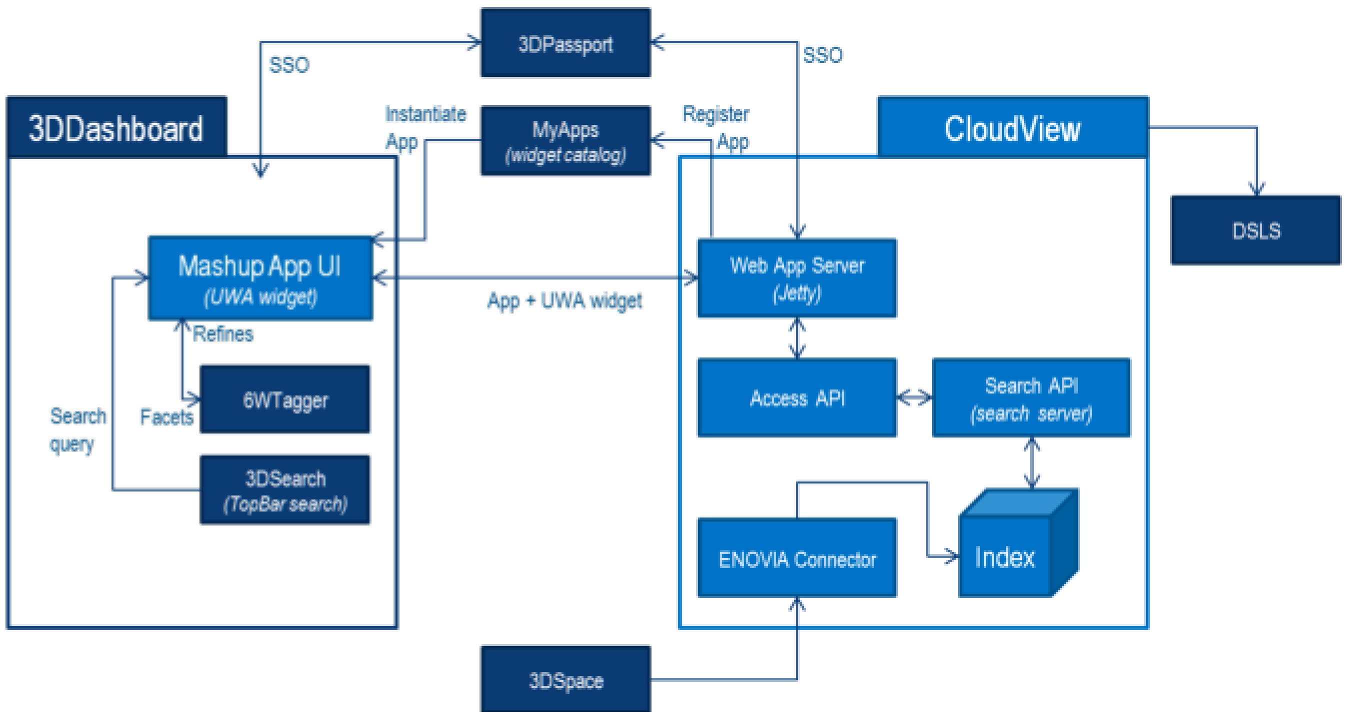
### What is Mashup Builder?

Exalead Mashup Builder allows you to build applications on top of Exalead CloudView indices. In context, customers expect all their applications to be available in a seamless user experience inside the 3DDashboard. Exalead CloudView Mashup Builder allows you to create Apps that can run inside a 3DDashboard using plugins.

The Mashup Application needs to be configured so that it can run properly inside a 3DDashboard and leverage components from the 3D#EXPERIENCE platform. When deploying the Application, you must use the Mashup Builder user interface (to upload plugins and configure SSO security) and 3D#EXPERIENCE environment (to setup reverse proxy or register Apps).

### How do Exalead CloudView and 3DDashboard communicate?

The diagram below shows how 3D#EXPERIENCE components (in dark blue) and Exalead CloudView components (in light blue) communicate:

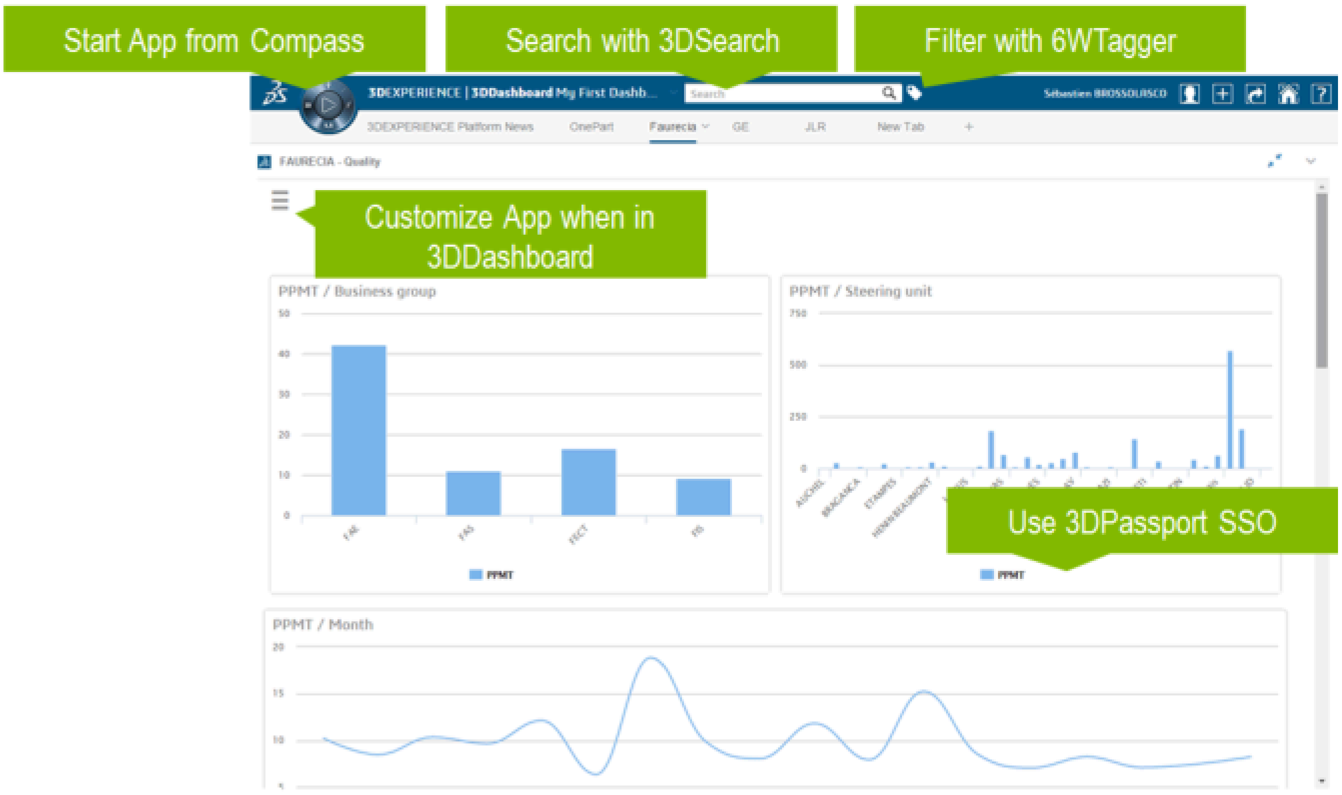


The main Exalead CloudView components include:

- Connectors such as ENOVIA Connector for crawling data sources and get data that will be stored into the index;
- Indices, which are optimized data structures to ensure fast access to indexed data and provide advanced search and aggregation capabilities;
- Search server web service that allows querying indices to retrieve data, create aggregates (facets, groups…) and compute analytics. This service is accessed via the low level Search API and the high level Access API. The Access API provides additional features on top of the Search API including security and multi-query orchestration;
- Web app server that serves JSP-based web applications. These Apps are created using the Mashup Builder IDE.

## What can I do with my Mashup App?

The diagram below shows a sample Mashup application running in 3DDashboard:



You can:

1. Start the Mashup Application from the Compass (available in MyApps catalog)
2. Authenticate to the Mashup Application using the Single Sign On mechanism provided by 3DPassport
3. Search the Mashup Application from the 3DSearch field when using the 'Search In This Tab' mode
4. Filter the Mashup Application content with 6WTags
5. Adapt the look, feel and behavior of the Mashup Application to match your 3D#EXPERIENCE interface

## Components

The following Mashup Builder components are provided to run Mashup Apps in the 3DDashboard:

| Component | Description |
|---|---|
| 3D#EXPERIENCE - Widget Builder | Mashup controller (web service) that gives access to a Mashup page as a UWA widget. This widget can be registered into MyApps as any Third Party App or added to a packaging definition to make it a trusted widget. |

| Component | Description |
|---|---|
| 3D#EXPERIENCE - Search and Refine | Mashup widget (HTML/Javascript page fragment) that defines the behavior of the Mashup Application when:<br>• a search is performed from the 3DSearch TopBar search field (in 'Search In This Tab' mode)<br>• a facet must be registered into the 6WTags<br>• a refinement event is received from the 6WTags. |
| CAS Security Provider | Mashup Security Provider for authenticating a user using 3DPassport SSO |
| 3D#EXPERIENCE - Detect Context | Mashup trigger (server side Java code executed before page rendering) that enables the definition of specific Mashup App behaviors when run inside a 3DDashboard context |
| 3D#EXPERIENCE - Show / Hide widget | Mashup triggers that allow to customize the Mashup App by either showing or hiding widgets when in 3DDashboard context |
| experience:is3DXP | MEL (Mashup Expression Language) function that does the same thing as the Show / Hide widget triggers, but anywhere in the Mashup where MEL functions can be used |

## Security

Security controls rely on Exalead CloudView infrastructure (Security Providers for authorization on the Mashup side and Security Source on the search server side for authorization). When integrated into the 3DPassport / CAS single sign on mechanism, the app will delegate authentication checks to the 3DPassport server. Regarding data access security, the solution depends on Exalead CloudView ability to handle the associated security (such as LDAP, Active Directory or ENOVIA security).

## Limitations

### Facets

Facets are published to the 6WTags only when a Mashup page is loaded. It means that if the facets available in the Mashup page change without reloading the page, the 6WTags will not be updated accordingly. This limitation impacts the Mashup Apps in which widgets and associated feeds are refreshed/executed using asynchronous queries (ie. AJAX).

### 6W Vocabulary

The Mashup App doesn't communicate or use the 6W vocabulary services from the 3D#EXPERIENCE platform. As a result, even if the Mashup App is configured to export its facets to the 6WTags, it doesn't mean that all facets shown inside the Mashup App pages will be displayed with the same name (especially localization-wise) as in the 6WTags.

## Installing your Mashup App

This section describes how to install your Mashup app.

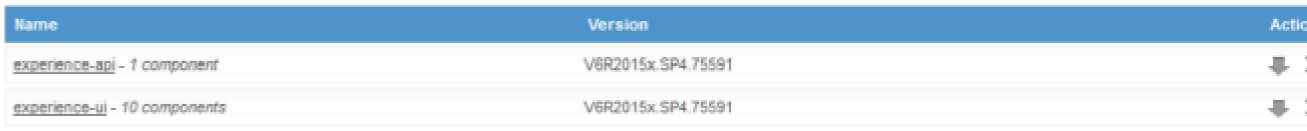
**Note:** You must have access to 3DDashboard with administrator rights (VPLMAdminUser or any user who is granted with Admin rights in 3DSpace) to register the new App.

### Install 3DExperience Mashup Builder plugin

1. In the Mashup Builder, select **Application** from the top left menu bar.
2. Select **Manage components > Plugins**.
3. Click **Upload plugin**.
4. From the **Upload Plugin** dialog box, click **Browse** and select the plugin `experience-api.zip` file.
5. Repeat step 4 to upload the plugin `experience-ui.zip`.

   The following plugins must be installed:

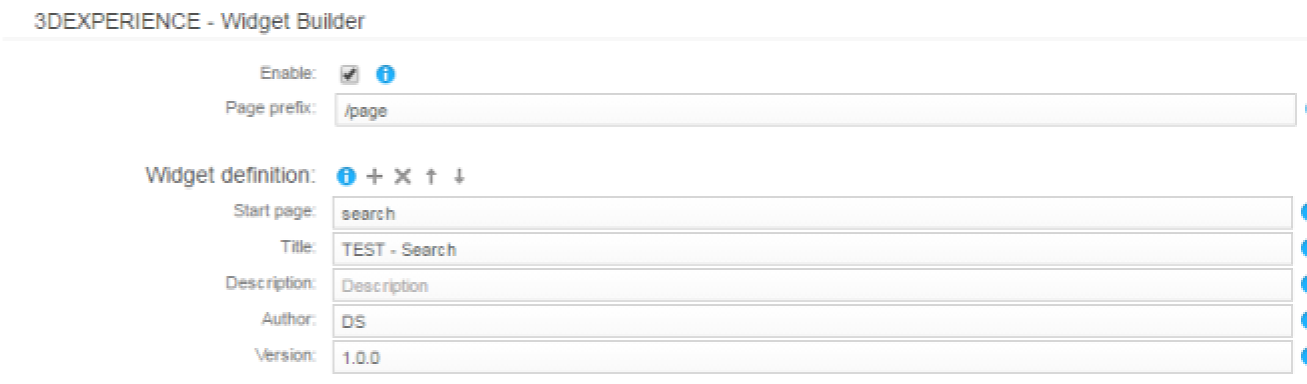
   - `experience-api` (contains MEL expression)
   - `experience-ui` (contains widgets, controllers and triggers)
6. Restart the search server process as asked by the application.

Example: 

| Name | Version | Actio |
|---|---|---|
| experience-api - *1 component* | V6R2015x.SP4.75591 | |
| experience-ui - *10 components* | V6R2015x.SP4.75591 | |

## Generate UWA widget

1. In the Mashup Builder, select **Application** from the top left menu bar.
2. Select **Manage components > Controllers**.
3. In **3DExperience - Widget Builder**:
   a. Select **Enable** to generate 3DExperience widgets.
   b. In **Page prefix**, enter **/page** if pages are accessed from `/page/PAGENAME` or leave blank if pages are accessed directly from `/PAGENAME`.
   c. In **Start page**, specify the Mashup page to be opened when the widget is started (widget will be accessed from `MASHUP/uwa/widget/START-PAGE`)
   d. In **Title**, specify the title displayed in the widget header.
4. Save and apply configuration.

Example: 

3DEXPERIENCE - Widget Builder

| | |
|---|---|
| Enable: | ✓ |
| Page prefix: | /page |
| Widget definition: | |
| Start page: | search |
| Title: | TEST - Search |
| Description: | Description |
| Author: | DS |
| Version: | 1.0.0 |

## Setup reverse proxy

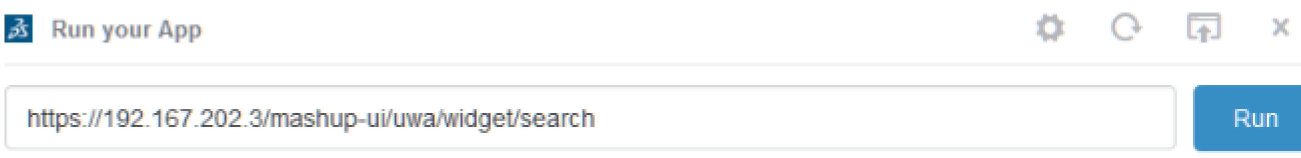
1. Log in as root with SSH on 3DExperience.
2. Edit `httpd.conf` file in `/usr/local/reverseproxy/conf/`:
   a. On virtual host on port *:443
   b. Add "`ProxyPass /mashup-ui http://cloudview:port/mashup-ui`"
   c. Add "`ProxyPassReverse /mashup-ui http://cloudview:port/mashup-ui`"
3. Restart the reverse proxy:

```
/usr/local/reverseproxy/bin/httpd -k restart
```

## Run Mashup App in 3DDashboard

You need administrator rights (VPLMAdminUser or any user who is granted with Admin rights in 3DSpace) to register the new App.

1. In the 3DDashboard, add a **Run Your App** widget from the Compass to a Dashboard.
2. Enter the **Run Your App** URL using the following syntax: `https://HOST/mashup-ui/uwa/widget/search`.

   Example: 

   https://192.167.202.3/mashup-ui/uwa/widget/search
3. Click **Run**.

   The **Run Your App** widget displays the search page inside a widget.
4. Open the **Platform Manager** dashboard.
5. Scroll down the **Members & Roles** tab to access the **Third Party Apps** section.
6. Click **Create Third Party Apps**.
7. In the **Create Third Party Apps**:
   a. Enter a short name
   b. Specify the compass quadrant
   c. Set an icon URL
   d. Select the **Widget Type**
   e. Enter the **Source code URL** as specified at step 2 (`https://HOST/mashup-ui/uwa/widget/search`)
   f. Save
8. Open the compass in the quadrant specified previously.

   Your widget should be displayed in the list.
9. Drag your widget to the dashboard.

   The Mashup App should be loaded on the 'search' page.

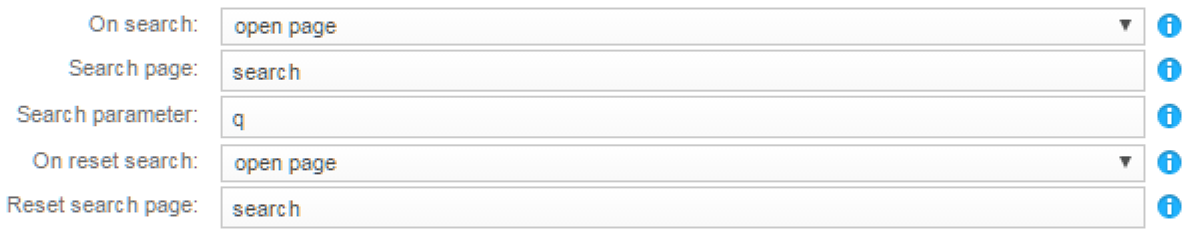
## Configuring your Mashup App

This section describes how to configure a Mashup application to work in the 3D#EXPERIENCE.

## Configure the 3DSearch Behavior

To configure the search behavior when users run queries in the 3DSearch field, add the **3DEXPERIENCE - Search and Refine** widget to the **search** Mashup page.

1. In the Mashup Builder, select the '/search' page.
2. Go to **Widgets > 3DEXPERIENCE**.
3. Select the **3DEXPERIENCE - Search and Refine** widget and add it to the page.
4. Click the widget header to open the **3DSearch** tab of the widget properties.
5. For **On search**, select:
    - **open page** to open the Mashup page specified in the **Search page** parameter and set the page parameter defined in **Search parameter** to the query string input in the 3DSearch field.
    - Or **run custom code** (advanced) to define the Javascript code called when 3DSearch receives the query.
6. In **On reset search**, specify the behavior when using the red cross or clearing the search in the 3DSearch field (same options as for **On search**).

   Example:
   

   On search: open page
   Search page: search
   Search parameter: q
   On reset search: open page
   Reset search page: search

7. Save and apply configuration.

## Define the mapping between Facets and 6WTags

You can refine the result list of a Mashup App using the 6WTags.

In Mashup Builder, you can define the mapping between facets and 6WTags:

- Locally, on a specific Mashup page.
- Globally, at the application level.

The following table summarizes the different mapping strategies:

| Define mapping on | Mapping definition with | Possibility to share mapping | Tag publishing |
|---|---|---|---|
| widget | list of facets on the UI | none (page-specific) | on page loading |
| widget | file | app or page | on page loading |

| Define mapping on | Mapping definition with | Possibility to share mapping | Tag publishing |
|---|---|---|---|
| tag controller | file | app or page | on-the-fly (AJAX) |

## Mapping locally using the widget

You can map facets to the 6WTags widget, using the widget configuration parameters.

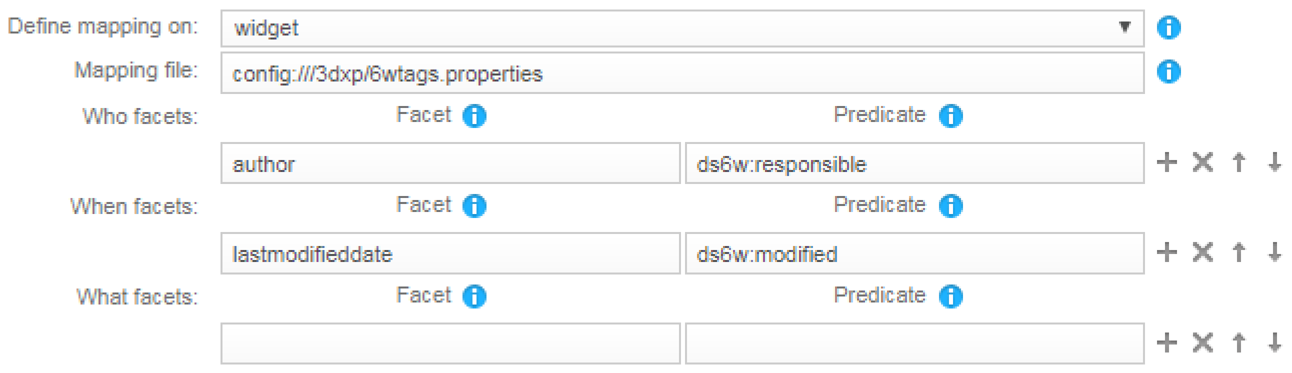
1. In Mashup Builder, select a page, for example, /search page.
2. Select **Widgets > 3DEXPERIENCE**.
3. Select the **3DEXPERIENCE - Search and Refine** widget and add it to the mashup page.
4. Click the **3DEXPERIENCE - Search and Refine** widget header and select the **6WTagger** tab.
5. For **Define mapping on**, select the **widget** option.

   **Important:** With this option, facets are published in the 6WTags only when a Mashup page is loaded. It means that if the Mashup page facets change without reloading the page, the 6WTags do not upload. This limitation impacts Mashup Apps in which widgets and associated feeds are refreshed or executed using asynchronous queries (with AJAX). To update tags dynamically after page loading, use the **tag controller** option.

6. Specify a mapping file if you want to upload a configuration and override only a few facets using the UI.

   For more information, see Mapping locally using a mapping file.

7. Map each facet to a 6W predicate.



Define mapping on: widget

Mapping file: config:///3dxp/6wtags.properties

| | Facet | Predicate |
|---|---|---|
| Who facets: | author | ds6w:responsible |
| When facets: | lastmodifieddate | ds6w:modified |
| What facets: | | |

8. Click **Apply**.

## Mapping locally using a mapping file

You can map facets to the **6WTags** widget using a mapping file.

1. Create a mapping file with a `.properties` extension, mapping facets to predicates as in the following sample:

```
closuredate=ds6w:when/ds6w:actualEnd
```

```
current=ds6w:what/ds6w:status
issue_priority=ds6w:why/ds6w:priority
issue_to_assigned_issue=ds6w:who/ds6w:assignee
Language=ds6w:what/ds6w:language
related_project=ds6w:where/ds6w:project
Source=ds6w:where/ds6w:dataSource
type=ds6w:what/ds6w:type
```

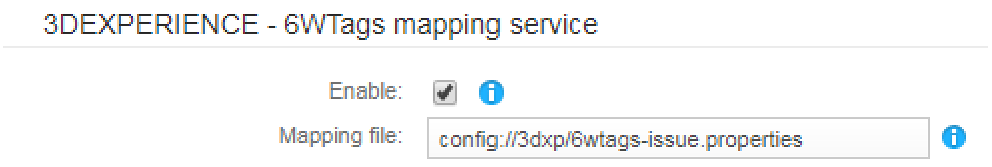
2. Put your `.properties` file in `<DATADIR>/config` (or a subfolder).
3. In Mashup Builder, declare the `.properties` file path in the **Mapping file** parameter, using `config://`
4. Click **Apply**.
5. From the command line, go to `<DATADIR>/bin` and force the product configuration update by running `buildgct`
6. In Mashup Builder, restart the Search Server:
   a. Go to **Application > Developer area**.
   b. Click **Reload components**.
   c. Select the **Restart search server processes** option.

**Note:** Repeat these steps whenever the file is modified to take changes into account and apply them on secondary servers.

## Mapping globally at the application level

You can define a tag controller to map facets and 6WTags globally, for all application pages.

1. Go to **Application > Manage components > Controllers**.
2. In **3DEXPERIENCE - 6WTags mapping service**, specify a mapping file to apply to all mashup



3DEXPERIENCE - 6WTags mapping service

Enable: ✓

Mapping file: config://3dxp/6wtags-issue.properties

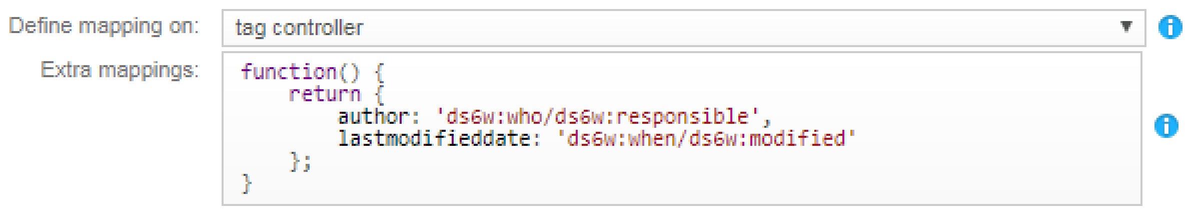
   pages.

   For a mapping file sample, see step 1 in Mapping locally using a mapping file.

3. Drag the **3DEXPERIENCE - Search and Refine** widget on your mashup page.
4. For each page, click the **3DEXPERIENCE - Search and Refine** widget header and select the **6WTagger** tab.
5. For **Define mapping on**, select the **tag controller** option.

   This option uses AJAX to update tags dynamically after page loading.

To extend the global **3DEXPERIENCE - 6WTags mapping service**, enter the additional JavaScript code in **Extra**


Define mapping on: tag controller

Extra mappings:

```
function() {
    return {
        author: 'ds6w:who/ds6w:responsible',
        lastmodifieddate: 'ds6w:when/ds6w:modified'
    };
}
```

**mappings**.

6. Click **Apply**.

## Trigger the App Display in the 3DDashboard

You can trigger the display of your Mashup app (show or hide) in the 3DDashboard.

**Warning:** The HTML preview of multiple MS Excel sheets is not compatible with the embedding of a mashup app in a 3D#EXPERIENCE Platform widget.

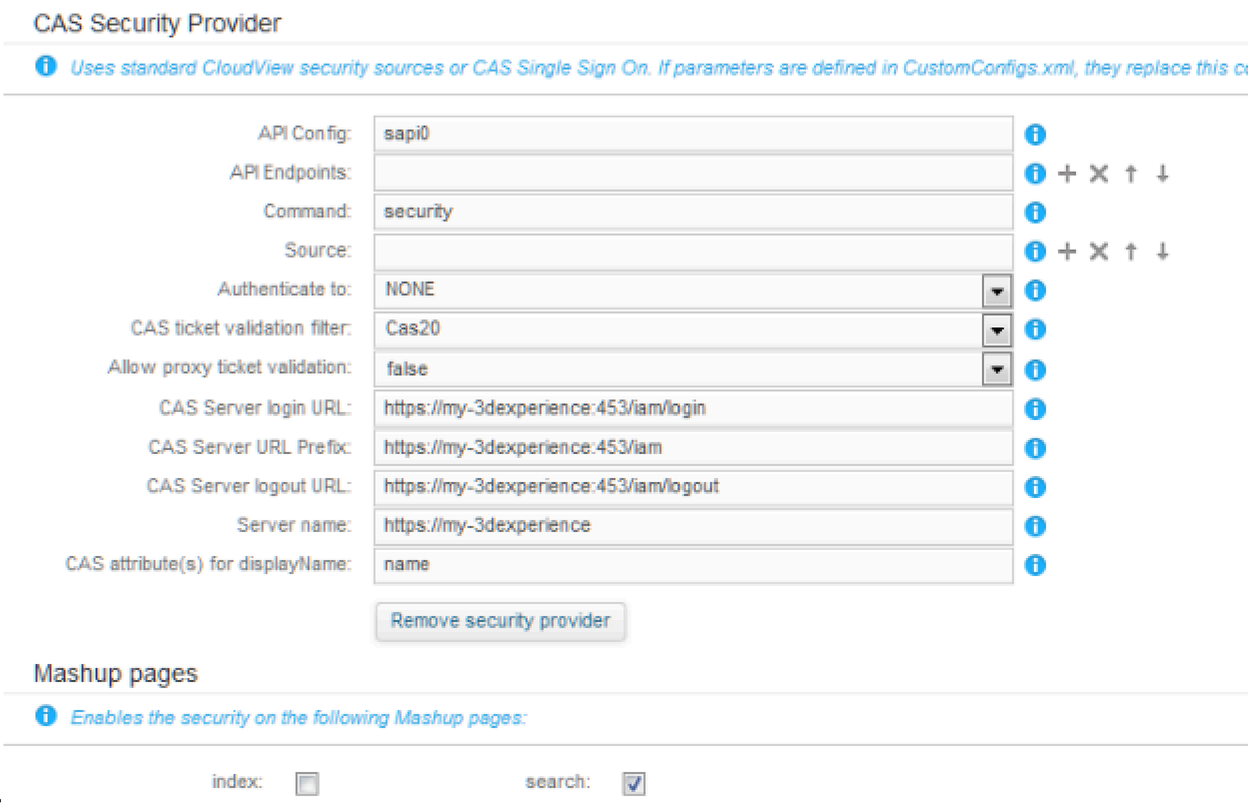
1. In the Mashup Builder, select the '/search' page.
2. Go to **Triggers > Pre Request Trigger**.
3. Select **3DEXPERIENCE - Detect Context** and add it at the top of the **/search** page.
4. In **Propagate 3DExperience context using**, select either:
   - **redirect** to use the HTTP referrer to propagate the context. It allows you to run an App inside the 3DDashboard in one browser tab, while running the standalone App in a separate browser tab (the cookies do not store the context.)
   - **session** to store the detected context in the user session (HTTP cookie). It prevents you to run the Mashup App in the 3DDashboard and the standalone App in the same browser (all tabs share the same cookies).
5. Go to **Triggers > Mashup Widget Trigger**.
6. Select either:
   - **3DEXPERIENCE - Hide in context** to hide the Mashup App when inside 3D#EXPERIENCE (for example, in a 3DDashboard).
   - **3DEXPERIENCE - Show in context** to show the Mashup App when inside 3D#EXPERIENCE (for example, in a 3DDashboard).
7. Save and apply configuration.

## User Authentication from 3DPassport

To authenticate users from the 3DPassport using an SSO mechanism, first configure the security source in the Mashup Builder and then add the 3DPassport SSL certificate to the Exalead CloudView trusted keystore.

### Configure the security source

1. In the Mashup Builder, go to **Application > General > Security**.
2. Click **Add a security provider**.
3. Select **CAS Security Provider** and click **OK**.
4. In the **CAS Security Provider** section, configure the following parameters:
    a. In **Authenticate to**, select **None**.
    b. In **CAS ticket validation filter**, select **Cas20**.
    c. In **Allow proxy ticket validation**, select **false**.
    d. In **CAS Server login URL**, enter `https://<HOSTNAME>:<PORT>/iam/login`
    e. In **CAS Server URL Prefix**, enter `https://<HOSTNAME>:<PORT>/iam`
    f. In **CAS Server logout URL**, enter `https://<HOSTNAME>:<PORT>/logout`
    g. In **Server Name**, enter `https://<HOSTNAME>`.
    h. In **CAS attribute(s) for displayName**, enter **name**.
5. In the **Mashup pages** section, select the check-box corresponding to the search page.



CAS Security Provider

Uses standard CloudView security sources or CAS Single Sign On. If parameters are defined in CustomConfigs.xml, they replace this c

|  |  |
|---|---|
| API Config: | sapi0 |
| API Endpoints: |  |
| Command: | security |
| Source: |  |
| Authenticate to: | NONE |
| CAS ticket validation filter: | Cas20 |
| Allow proxy ticket validation: | false |
| CAS Server login URL: | https://my-3dexperience:453/iam/login |
| CAS Server URL Prefix: | https://my-3dexperience:453/iam |
| CAS Server logout URL: | https://my-3dexperience:453/iam/logout |
| Server name: | https://my-3dexperience |
| CAS attribute(s) for displayName: | name |

Mashup pages

Enables the security on the following Mashup pages:

index: ☐ search: ☑

Configuration:

6. Click **Apply**.

## Add 3DPassport SSL certificate to Exalead CloudView

1. Verify the certificate format using the following command:

```
openssl x509 -in <infile.cert> -text -inform <format>
```

   Where the format is `DER` or `PEM` depending on your needs.

2. Encrypted `PEM` files usually store private keys. Convert them to a file that is not encrypted.

   You can use openssl on the command line:

```
openssl pkcs8 -topk8 -in <key> -out <hostname>-<instance>.key -nocrypt
```

3. Verify that the certificate is stored using UNIX LF end of line characters:
   a. On Windows, you can use the following tool:
   b. On UNIX, you can alternatively use dos2unix.
4. For every product instance, overwrite the key and certificate files generated at installation time in `DATADIR/security`.

   If you use an alias, the private key name must use the alias and not the default `<hostname>-<instance>`.

5. For every product instance, add the server certificate to the truststore:

```
keytool -import -file <.cert file (DER)> -alias
 <jetty> -keystore DATADIR/security/trusted.servers.ks -storepass
```

```
<exalead>
```

6. Restart Exalead CloudView.

   **Note:** For more information, see *Exalead CloudView Administration Guide: Securing Exalead CloudView: Securing your installation with HTTPS and SSL*.

## Troubleshooting

This section explains common issues encountered when installing and configuring 3DDashboard Apps.

- Check that **Search in current dashboard** is selected in the search bar options before searching.
- Check that script execution on your page is not blocked (shield icon in browser location bar)
- Check the web console/Firebug for specific errors

# Appendix - Mashup Expression Language

This appendix describes the Mashup Expression Language (MEL) Functions.

## About Mashup Expression Language

The configuration of the Mashup UI through the Mashup Builder makes an extensive use of expressions such as `${feeds["cloudview"].metas["name"]}` that allows you to construct text that contains dynamic content from your feeds.

These expressions, known as the *Mashup Expression Language* (MEL), actually provide much more than just dynamic variables and support common operations that would usually require editing JSP files.

MEL Functions appear in a contextual menu on the left when you click in widget and property fields. The elements displayed in this contextual menu correspond to Mashup API elements in Atom format, for example:

- `${feed.*}` corresponds to `<feed>`
- `${entry.*}` corresponds to `<entry>`
- etc.

  For more information about the Mashup API and its Atom elements, see "Using the Mashup API" in the Exalead CloudView Mashup Programmer's Guide.

Tip:
When you add MEL expressions, you can verify their syntax by selecting the **Debug** tab and clicking **Validate**. You can also use the autocomplete function by pressing Ctrl+space.

**Important:** MEL is always computed server-side whereas JS is computed client-side. JS cannot be used inside MEL, but you can use MEL inside JS. For example, you can use MEL to iterate over a list and save values as an array/object init in JS, and then use this array/object in your JS.

## Syntax

The MEL supports all the key syntactic constructions: variables, operations, loop statements, etc.

The language itself is entirely lower case.

The following examples are based on a result feed named `persons` that has an entry with the following metas:

- `name`: roger
- `age`: 42
- `${feeds["person"].metas["name"]}`: roger, bruce

## Simple variables

A variable represents a specific data item, or value, and acts as a placeholder for that value. When a formula encounters a variable, the formula searches for the value of the variable and uses it in the formula.

The basic syntax is: `${my.variable}`

*Simple expression examples*

| Expression | Result |
|---|---|
| `hello ${feeds["person"].metas["name"]}` | hello roger |
| `hello ${feeds["person"].metas["unknown_meta"]}` | hello |
| `hello \${feeds["person"].metas["name"]}` | hello ${feeds["person"].metas["name"]} |

## Fallbacks

You can declare variable fallbacks using pipes `|`

- `${my.variable|other.variable}`
- `${my.variable|other.variable|last.variable}`
- `${my.variable|"hardcoded fallback text"}`

*Fallback/Advanced expression examples*

| Expression | Result |
|---|---|
| `hello ${feeds["person"].metas["name"]|"world"}` | hello roger |
| `hello ${feeds["person"].metas["unknown_meta"]|"w` | hello world |

| Expression | Result |
|---|---|
| `hello ${feeds["person"].metas["unknown_meta"]\|feeds["person"].metas["name"]\|"world"}` | hello roger |
| `hello ${feeds["person"].metas["unknown_meta"]\|feeds["person"].metas["unknown_meta2"]\|"wo` | hello world |
| `hello ${feeds["person"].metas["unknown_meta"]\|feeds["person"].metas["unknown_meta2"]}` | hello |

## Dynamic variables and functions

Dynamic variables are variables whose values are determined when the program is run.

- `${i18n["machine_"+robot.id]}`
- `${math:${op.sum}(1,1)}`

*Dynamic expression examples*

| Expression | Result |
|---|---|
| `${fn:${feeds["person"].metas["name"]}(${feeds["person"].metas["age"]})} = ${fn:roger(42)}` | result of the function `roger` passing the parameter `42` |

## Operations

- Strings: `${my.age == "42"}`
- Supported numerical operators: `- + * / % == != < > ( ) ^ >= <=` Example: `${(2+3)*4} = 20`
- Supported string operators: `== !=`
- Logical operators: `${(2 <= 3 && 42 != 21) || 3 >= 3}` where `&&` = and `||` = OR
- Brackets support:
  - `${page.params["q"][1]}`
  - `${math:sum(1,1)[0]}`
- Concatenation and sum: + is the concatenation operator (as in Java) and also the sum operator for numerical operations.

- `${"5" + "6"}` --> 11
- `${"a" + "b"}` --> ab
- `${"a" + "5"}` --> '' (it will be resolved to NaN+5)
- `${str:join('',"a","5")}` --> a5

*Operation examples*

| Expression | Result |
|---|---|
| `i am ${feeds["person"].metas["age"]}` | i am 42 |
| `i am ${feeds["person"].metas["age"] + 10}` | i am 52 |
| `i am ${feeds["person"].metas["age"] / 2}` | i am 21 |
| `i am ${feeds["person"].metas["age"] % 40}` | i am 2 |
| `i am ${feeds["person"].metas["age"] * 2}` | i am 84 |
| `i am ${feeds["person"].metas["age"] - 20}` | i am 22 |
| `i am ${feeds["person"].metas["age"] + 10.0}` | i am 52.0 |
| `i am ${feeds["person"].metas["age"] / 2.0}` | i am 21.0 |
| `i am ${feeds["person"].metas["age"] % 40.0}` | i am 2.0 |
| `i am ${feeds["person"].metas["age"] * 2.0}` | i am 84.0 |
| `i am ${feeds["person"].metas["age"] - 20.0}` | i am 22.0 |
| `i am ${feeds["person"].metas["age"] == "42"}` | i am true |
| `i am ${feeds["person"].metas["age"] == "24"}` | i am |
| `i am ${feeds["person"].metas["age"] != "42"}` | i am |
| `i am ${feeds["person"].metas["age"] != "24"}` | i am true |

| Expression | Result |
|---|---|
| `i am ${feeds["person"].metas["age"] == "roger"}` | i am true |
| `i am ${feeds["person"].metas["age"] == "robert"}` | i am |
| `i am ${feeds["person"].metas["age"] != "roger"}` | i am |
| `i am ${feeds["person"].metas["age"] != "robert"}` | i am true |
| `${feeds["person"].metas["age"][0]}` | roger |
| `${feeds["person"].metas["age"][1]}` | bruce |
| `${feeds["person"].metas["age"][1][0]}` | b |
| `${feeds["person"].metas["age"][1][2]}` | u |
| `${"world"[1]}` | o |
| `${("hello" + "world")[8]}` | l |

*Logical operation examples*

| Expression | Result |
|---|---|
| `I am ${Roger && 42}` | false |
| `I am ${Roger \|\| 42}` | true |
| `I am ${Roger \|\| (42 && Roger)}` | false |

## Functions

MEL Functions are classified by categories under the following namespaces:

- category functions: `className`, `url`.
- currency: `format`.
- date: `addDays`, `addYears`, `addMilliseconds`, `addMonths`, `addMinutes`, `addHours`, `addSeconds`, `addWeeks`, `now`, `format`, `elapsedTime`.
- entry: `cleanId`, `previewHtmlUrl`, `thumbnailUrl`, `downloadUrl`, `previewImageUrl`.
- list: `join`, `size`.

- math: `min`, `atan`, `max`, `pow`, `asin`, `cos`, `ceil`, `sqrt`, `random`, `log2`, `sum`, `round`, `divide`, `multiply`, `log`, `subtract`, `exp`, `abs`, `floor`, `sin`, `avg`, `tan`, `acos`.
- number: `format`.

  **Note:** For number approximation, we use the rounding modes described in https://docs.oracle.com/javase/7/docs/api/java/math/RoundingMode.html

- str (string): `replace`, `lowerCase`, `abbreviate`, `upperCase`, `length`, `hashCode`, `split`, `contains`, `trim`, `substr`.
- var: `get`, `set`.

  **Note:** The `fn` namespace is used as the standard namespace.

Function calls:

- Simple: `${date:now()}`
- With arguments: `${str:substr("toto", 2)}`
- With variable arguments: `${math:max(${my.age}, ${your.age})}`

*Function expression examples*

| Expression | Result |
|---|---|
| `I am ${math:substr(page.params["name"], 0, 3)}` | I am rog |
| `${number:format(0.25, 1, 4, 1, 1, true, "HALF_UP")}` | 0.3 |

## Internationalization functions

- `${user:locale(MELHttpServletRequest request)}` returns the locale used by the current user from the HTTP query.
- `${date:format(MELString date, MELString pattern, MELString locale, MELString timeZone)}` returns the date formatted as the output pattern using the specified locale and timezone.
- `${i18n:message(MELHttpServletRequest request, MELType code)` translates a string like `${i18n[code]}` but using the locale from the HTTP query.
- `${i18n:message(MELHttpServletRequest request, MELType code, MELType[]... parameters)` translates a string containing variable parts (specified by `{0}`, `{1}`, etc.) which are replaced by the values in parameters.

Sample usage:

```
${user:locale(request)}
> en
${date:format(${entry.metas['date']}, 'dd/MM/yyyy hh:mm:ss', 'en_EN', 'GMT')
> 10/02/2014 12:30:10
${date:format(${entry.metas['date']}, "MM/dd/yyyy hh:mm:ss", "en_US", "GMT-5")}
> 02/10/2014 07:30:10
```

In localization file (`*.properties`):

```
error.message=Error code: {0}
document.summary=created by {0} on {1} at {2}
```

In the Mashup Builder:

```
${i18n:message(request, 'error.message', '404')}
> Error code: 404
${i18n:message(request, 'document.summary', author, date, time)}
> created by John Doe on February 10, 2014 at 10:30am
```

## Combinations

- Operations and functions: `${math:max(${my.age}, ${your.age}) + 10}`

## Ternaries

- Simple: `?{true?yes:no}`
- Advanced: `?{${persons.age}?i am ${persons.age / 2}:age not found}`
- Short syntax: `?{true?:no}`
- Nested: `?{true?{true?nested yes:nested no}:no}`

*Ternary expression examples*

| Expression | Result |
|---|---|
| `?{feeds["person"].metas["name"]?meta exists:meta not found}` | meta exists |
| `?{feeds["person"].metas["unknown_meta"]?meta exists:meta not found}` | meta not found |
| `?{feeds["person"].metas["age"]?i am feeds["person"].metas["age"]/2:age not found}` | i am 21 |
| `?{true?yes:no}` | yes |

| Expression | Result |
|---|---|
| `?{yes:no}` | no |
| `?{true?:no}` | - |
| `?{true?{true?nested yes:nested no}:no}` | nested yes |
| `?{true?{?nested yes:nested no}:no}` | nested no |
| `${if feeds["person"].metas["name"] != "roger"} wrong ${elseif math:sum(1,1) ==3} wrong ${elseif math:sum(1,1) == 2} happy ${else} wrong ${/if}` | happy |

*Special ternary expression examples*

| Expression | Result |
|---|---|
| `hello ${page.params["name"]}` | hello roger |
| `hello ${page.params["unknown_meta"]}` | hello |
| `hello ?{page.params["name"]?no otherwise}` | hello no otherwise |
| `hello ?{page.params["unknown_meta"]?no otherwise}` | - |

## If statements

The `If` statement enables you to evaluate a sequence of statements if a condition is true and evaluate a different sequence of statements if it is not true.

- `You are ${if page.params["n"] != 42} wrong ${elseif math:sum(1,1) ==3} wrong ${elseif math:sum(1,1) == 2} happy ${else} wrong ${/if}`

## Foreach loop statements

`Foreach` loops enable you to evaluate a sequence of statements multiple times.

- `${foreach 1,2,3} ${loop.element}?{$loop.hasNext}?,} ${/foreach} go!`
- `${foreach number in 1,2,3} ${number}?{${loop.hasNext}?,} ${/foreach} go!`

*Loop expression examples*

| Expression | Result |
|---|---|
| `${foreach page.params["name"]}I am ${loop.element}${/foreach}` | I am roger |
| `${foreach "a", "b", "c"}${loop.element}${/foreach}` | abc |
| `${foreach 1,2,3}and ${loop.index} ${/foreach}` | and 0 and 1 and 2 |
| `${foreach 1,2,3}and ${loop.element} ${/foreach}` | and 1 and 2 and 3 |
| `${foreach number in 1,2,3} and ${number} ${/foreach}` | and 1 and 2 and 3 |

## Flags

- `!m` --> HTML escaping
- `!x` --> XML escaping. For example: `${entry.metas['text']!+x}`
- `!u` --> URL encoding. This flag is useful to declare URLs.
- `!h` --> Highlighting (only available for metas). For example: `${entry.metas["title"]!h}`

**Important:** We recommend declaring whether flags must be used or not specifically, with the + and - signs. For example: `!+h` or `!+h-x` Without the + and - signs, the default behavior is inverted.

## Code samples

The two following code samples using MEL can be pasted in an HTML widget. They both contain typical iteration made with `foreach` statements.

```
<h1>Iterate over entries</h1>
<ul>
${foreach hit in feed.entries}
    <li>
        <h1>${hit.title}</h1>
        <a href="${entry:downloadUrl(hit, request, '<Page name>')}">Download</a>
        <iframe src="${entry:previewHtmlUrl(hit, request, '<Page name>')}"></iframe>
        <img src="${entry:previewImageUrl(hit, request, '<Page name>')}" />
    </li>
${/foreach}
</ul>
<h1>Iterate over the author facet</h1>
```

```
<ul>
${foreach catElement in feeds['cloudview'].facets['author'].leaves}
    <li>
        <a href="${category:url(catElement, request, feeds['cloudview'])}"
class="${category:className(catElement)}">
            ${i18n[catElement]}
        </a>
    </li>
${/foreach}
</ul>
```

**Note:** `request` refers to the request variable available in any context.

## Handling Categories, Facets and entries

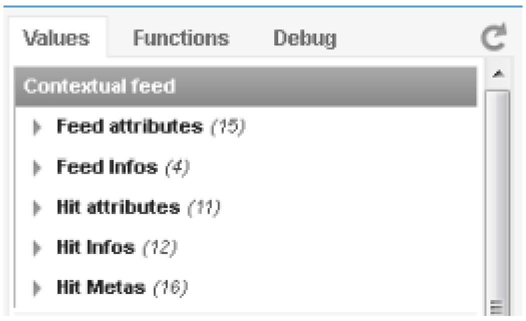
The MEL syntax lets you handle categories, facets and entries to return values. You can access these in both:

- an absolute way, for example, in a custom HTML widget;
- and a relative way, within a specific context set by the widget, for example, in a chart widget that requires the setting of a facet.

### About Relative access in a given context

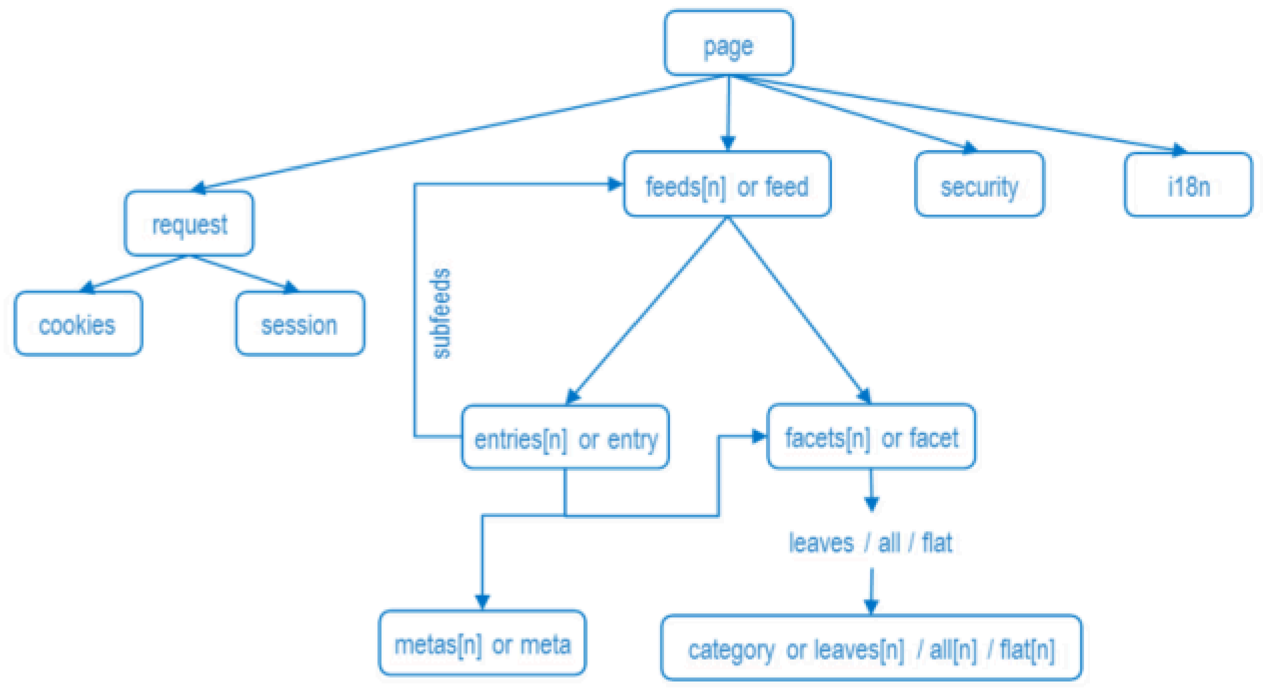
When manipulating MEL, you are in a given context. For example, in widgets, you are most of the time in a "feed" context, so you don't need to give the beginning of the context in your MEL expression as the widget already has set the context for you.

In the contextual menu listing all available Values on the left, select elements from **Contextual feed** to add them in a "feed" context in your MEL expression.



For example, a **Result List** widget iterates over each entry (hit) for each feed, so you can use the relative access to the entries meta as you are scoped at the "entry" level. In MEL, this is written as `${entry.***}` rather than `${page.feed.entry}` as written in an absolute way.

The following diagram represents a tree view of the MEL relative context.



**Note:** You can enter at any level in the context, but you cannot skip sub-context elements.

## Sample data

The following data will be used to illustrate the use of the MEL syntax for facet aggregation.

### First Result Hit

| Name | Type | Value |
|---|---|---|
| title | Meta | The hitchhiker's guide to the galaxy |
| author | Meta | Douglas Adams |
| price | Meta | 42 |
| date | Category | Top/date/2011/02/01 |

### Facet synthesis sample

| Type | | Count | Income |
|---|---|---|---|
| Date | Facet | 25 | |
| 2011 | Category | 16 | |
| 01 | Category | 10 | |

| Type | | Count | Income |
|---|---|---|---|
| 01 | Category | 2 | |
| 03 | Category | 3 | |
| 05 | Category | 5 | |
| 02 | Category | 6 | |
| 01 | Category | 1 | |
| 06 | Category | 2 | |
| 07 | Category | 3 | |
| 2012 | Category | 9 | 125000 |
| 02 | Category | 9 | 125000 |
| 09 | Category | 9 | 125000 |

## Facet and Category access

### Absolute access

- Access to a specific facet
  - Syntax: `${feeds["feedName"].facets[facetId].attributes}`
  - Example: `${feeds["cloudview"].facets["Date"].path` --> `Top/date`

  **Note:** `feeds["feedName"]` can be replaced by `feed` to use the context of the current feed.

- Access to a specific category
  - Syntax:

    ```
    ${feed.facets[facetId]["2012"].attributes}
    ${feed.facets[facetId]["Top/facet_id/rest/of/the/
    path"].attributes} // Accessing through a path
    ```

  - Example:

    ```
    ${feed.facets["Date"]["2012"].count} --> 9
    ${feed.facets["Date"]["Top/2011/01"].count --> 10
    ```

- Access to the 5th item of facet leaves
  - Syntax: `${feed.facets[facetId].leaves[5].attributes}`
  - Example:

`${feed.facets["Date"].leaves[5].description}` --> `07`
`${feed.facets["Date"].leaves[5].path}` --> `Top/date/2011/02/07`
`${feed.facets["Date"].leaves[5].count}` --> `3`

## Contextual (Relative) access

- Access to a category when in a category context
  - Syntax: `${category.attributes}`
  - Example: `${category.description}` --> `2011`
- Access to a facet attributes
  - Syntax: `${facet.attributes}`
  - Example: `${facet.id}` --> `date`
- Access to a category in a Facet context
  - Syntax: `${facet["mycat"].attributes}`
  - Example: `${facet["2012"].count}` --> 9
  - Categories can also be returned from a Facet: `${facet.flat}` --> 2011,2012 (comma-separated list of categories description)
- Access to an aggregation function of a facet in a Facet context
  - Syntax: `${facet["2012"].aggrs["aggregation_id"]}`
  - Example: `${facet["2012"].aggrs["income"]}` --> `125000`

## Available attributes

The available attributes come from a direct mapping of the Access API attributes. These are:

- `id`
- `description`
- `path`
- `state`
- `refinable` (facet only)
- `flat` (iteration on first level)
- `all` (recursive iteration)
- `leaves` (iteration over leaves)
- `count`
- `score` (category only)

- `aggrs["aggregation_function_name"]`

Elements that return list items can also have a list manipulation variable:

- `length`
- `first`
- `last`

Example: `${facet["2012"].length}` --> 1

## Entry access

### Absolute access

- Access to a given meta of a given entry
  - Syntax:

    `${feed.entries[n].metas["metaName"]}`
    `${entries[n].metas["metaName"]}` // (as a shortcut for `feed.entries[n]`...)
  - Example: `${feed.entries[0].metas["title"]}` --> `The hitchhiker's guide to the galaxy`
- Access to a given facet or category of a given entry
  - Syntax: `${feed.entries[n].facets["facetId"].attributes}`
  - Example: `${feed.entries[n].facets["Date"].leaves[0].count}` --> `1`

### Contextual (Relative) access

- Access to a given meta
  - Syntax:

    `${entry.metas["text"]}`
  - Example: `entry.author` --> `Douglas Adams`
- Access to a given facet/category
  - Syntax: `${entry.facets["facetId"].attributes}`
  - Example: `${entry.facets["Date"].path}` --> `Top/date/2011/02/01`
- Access to entry info
  - Syntax:

    `${entry.infos["hitInfos"]}` //new way

- Example:

  `${entry.infos["did"]}` --> `1`

## Available attributes

The attributes available for the INFOS are directly mapped to Access API.

They can be one of the following:

- `id`
- `self`
- `image`
- `builGroup`
- `buildGroupSlice`
- `score`
- `did`
- `slice`
- `url`
- `nCollapsed`

## How to use the entry scope with an HTML widget

If you want to use an HTML widget and iterate on multiple feeds to retrieve their results, you must target entry positions.

For example, to make an HTML widget retrieve the results of 3 feeds called `clients`, `technicians` and `users`, we could use the following syntax:

```
${feeds['clients'].entries[entry.index].metas['address']}
/ ${feeds['technicians'].entries[entry.index].metas['name']}
/ ${feeds['users'].entries[entry.index].metas['userid']}
```

## Result set access

This section shows the syntax for accessing global information regarding the result set.

- Access to result set infos
  - Syntax: `${feed.infos["resultSetInfo"]}`
  - Example: `${feed.infos["nhits"]}` --> `1`

## Available attributes

`resultSetInfo` are directly mapped to the Access API. They can be one of the following:

- `id`
- `totalResults`
- `startIndex`
- `itemsPerPage`
- `query`
- `context`
- `last`
- `start`
- `estimated`
- `nhits`
- `autocorrected`
- `ellql`
- `nmatches`
- unknown parameter provided by a specific feed

## Feed access

- Access to parent parameters in a subfeed
  - Syntax: `${feed.entries[2].subfeeds["subfeedname"].entries[0].metas["title"]` --> Title of the subfeed's first entry of the parent feed's third entry.

## Request, Cookie and Session MEL Manipulation

- `request` gathers all the parameters of the HTTP request.
  - `${request["contextPath"]}`
  - `${request["queryString"]}`
- `cookies` gathers all the parameters available in the Mashup UI cookies.
  - `${cookies["name"]}`
- `session` gathers all the parameters available in the user session.

- `${session["username"]}`
- `${session["name"]}`

## Use cases

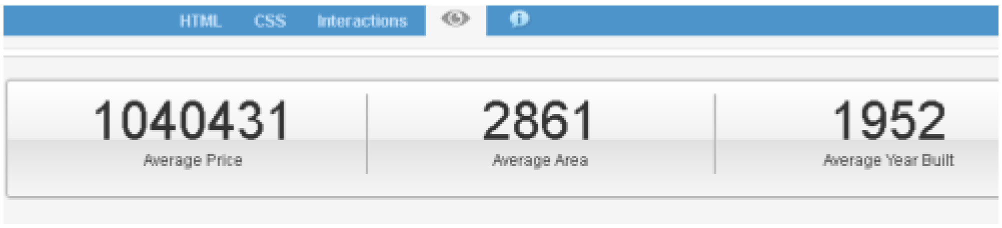
The following examples show typical uses of facet manipulations.

### Create a metric-like widget

You can create a metric-like widget using an **HTML** widget with custom HTML code. MEL allows the widget to interact with facets and calculate aggregations values. In the following example, the HTML code retrieves facet descriptions and aggregations to display average values in a dashboard.

```
<div class="metric_container">
  <div class="metric_single metric_separator">
    <span class="metric_value">${feed.facets["area"].aggrs["avg_price"]}</span><br />
    <span class="metric_descr">Average Price</span>
  </div>
  <div class="metric_single metric_separator metric_separator_light">
    <span class="metric_value">${feed.facets["area"].aggrs["avg_land_sq_feet"]}</span
    <span class="metric_descr">Average Area</span>
  </div>
  <div class="metric_single metric_separator_light">
    <span class="metric_value">${feed.facets["area"].aggrs["avg_year"]}</span><br />
    <span class="metric_descr">Average Year Built</span>
  </div>
</div>
```

The HTML widget should display results as shown in the following screenshot.



### Format standard facet values

The **Standard Facets** widget can display categories and facets with an aggregation function.

You can interact with the facet of a category that is defined in the widget.

### Iterate over facets in an HTML widget

Iterate over facets in an HTML widget

You can iterate over facets in an HTML widget using custom MEL. For example, you can iterate on a category/ facet to replicate the **Standard Facets** widget behavior.