opentext™

Custom Views and Appearances

OpenText™ Content Server

This document is part of the Content Server User Online Help documentation list. If conflicts exist, the Online Help supersedes this document.

LLESAPP210100-UGD-EN-01

Custom Views and Appearances
OpenText™ Content Server
LLESAPP210100-UGD-EN-01
Rev.: 20. Oct. 2020

This documentation has been created for software version 21.1.
It is also valid for subsequent software releases unless OpenText has made newer documentation available with the product, on an OpenText website, or by any other means.

Open Text Corporation

275 Frank Tompa Drive, Waterloo, Ontario, Canada, N2L 0A1

Tel: +1-519-888-7111
Toll Free Canada/USA: 1-800-499-6544 International: +800-4996-5440
Fax: +1-519-888-0677
Support: https://support.opentext.com
For more information, visit https://www.opentext.com

Table of Contents

Chapter 1

Working with Custom Views

A Custom View lets you apply your own “look and feel” to Content Server. Many users use this feature to make a container or workspace look and act like a typical Web page. Many organizations use Custom Views to make certain containers or workspaces look like other organizational Web sites or publications.

1.1 Creating and Administering Custom Views

A *Custom View* is an item type that contains HTML code. When you add a Custom View to a Workspace, Folder, Project, Task List, or Compound Document, that HTML code is displayed as part of the interface. The Custom View is inserted between the navigation menus and the items in the container or Workspace. You can keep the `customview.html` file in your Folder or Workspace, and prevent the Custom View from appearing by *disabling* it. When you disable a Custom View, it will appear in the Folder list but will not appear as part of the Folder or Workspace's interface.

The first step in adding a Custom View is to create the HTML file that defines the Custom View, along with any other files, such as image files or style sheets. By default, a Custom View can contain the following:

- HTML, JavaScript, VBScript, Java applets or any other code that can be interpreted by a web browser. However, your administrator may restrict your ability to add certain content.
- Links to other items in Content Server or to other Web pages.
- References to image, audio, or other files that are stored either in Content Server or in a directory on any other accessible web server.
- Saved searches in Content Server that other users can add to their Favorites or to reuse your search criteria, or enable you to define the display priority of custom search forms. For more information about using Custom Views with saved searches, see *OpenText Content Server - Search (LLESWBB-UGD).*

Because the contents of a Custom View are inserted within the HTML file that delivers the interface, a web browser ignores tags such as <TITLE>, <BODY>, or <FRAMESET>. This means that there are limits to the types of code you can include. For example, you cannot create frames in a Custom View.

Note: You can add a resource folder to the same location as the Custom View to store images and other files that the Custom View uses. To prevent accidental deletion of these files, and help you maintain an organized workspace, you should consider hiding the Custom View and the resource folder. For more information about hiding items, see *OpenText Content Server - Get Started (LLESRT-UGD).*

The ability to create and manage Custom Views is usually restricted to certain users or groups. To add a Custom View, you must have the object creation privilege for Custom Views, as well as the *Add Items* and *Modify* permissions on the appropriate containers. However, you do not need permission to modify any subitems that the Custom View may affect. To modify a Custom View that was added by another user, you need the *Modify* permission on the Custom View.

Using Templates to Create Custom Views

You can create templates based on existing Custom Views in Content Server or based upon a file you have stored on your computer. When you add a template, it does not automatically become the Custom View, you must then add a Custom View based upon that template and ensure that it is enabled.

If your administrator has added Custom View templates to the Templates Volume, you will be able to access the templates when you create a Custom View. Templates can also be added to any Workspace or Folder that you have the Add Item permission.

Administering Custom Views

You determine which users or groups a Custom View should apply to by setting permissions on the Custom View. For example, you can:

- Apply the Custom View to all users by assigning the Custom View the *See Contents* permission. This enables public access for the Custom View.
- Apply the Custom View only to specific users by granting those users or groups the *See Contents* permission.
- Apply the Custom View to all project participants by granting coordinators, members, *and* guests of a project the *See Contents* permission for that Custom View.
- Apply the Custom View only to project participants with a specific role by granting coordinators, members, *or* guests of a project the *See Contents* permission for that Custom View.

For more information about permissions, see *OpenText Content Server - Get Started (LLESRT-UGD)*.

Note: New Custom Views are enabled by default, which means they automatically appear without having to enable them. If multiple Custom Views reside in the same container, they will appear based on which name comes first alphabetically. All others will be ignored, unless they are manually enabled.

1.1.1 To Add a Custom View

To add a custom view:

1. On the **Add Item** menu, click **Custom View**.
2. On the **Add: Custom View** page, do one of the following:
   a. Click **Existing**, click **Browse** to navigate to the HTML file you created that defines the Custom View, and then click **Open**.
   b. Click **Template**, click **Browse Content Server...** to navigate to the template file that defines the Custom View, and then click **Open**.
3. Type a name for the Custom View in the **Name** field.

   If your system has multiple languages installed and enabled, click the **click to edit multilingual values** button to edit the names in the other, enabled, languages.
4. Optional In the **Description** field, type a description for the Custom View.

   If your system has multiple languages installed and enabled, click the **click to edit multilingual values** button to edit the descriptions in the other, enabled, languages.
5. Optional In the **Categories** field, click **Edit** to either select or add a Category to apply to this Custom View.
6. Content Server will save your new Custom View, by default, in the **Enterprise Workspace**. To change the default location, in the **Create In** field, click **Browse Content Server**. Choose **Select** next to a new location.
7. Click **Add**.

Note: If the HTML file contains references to other files, such as images, you must add those files separately.

1.1.2 To Enable or Disable a Custom View

To enable or disable a custom view:

- Click the **Functions** icon for the `customview.html`, and then click **Enable** or **Disable**.

1.1.3 To Create a Custom View Template

To create a custom view template:

1. Click the **Functions** icon for the `customview.html` file for which you want to create a template, and then click **Make Template**.
2. On the **Add: Custom View Template** page, type a name for the template in the **Name** field.
3. In the **Source** area, do one of the following:
   a. Click **Custom View**, click **Browse** to navigate to the `customview.html` file you want the template created from, and then click its **Select** link.
   b. Click **File**, click **Browse** to navigate to the `.xml` file you want the template created from, and then click **Open**.
4. Specify any other general item settings.
5. Click **Add**.

Tip: You can also create a template from an existing template by clicking the Custom View template's **Functions** icon, and then clicking **Create From**. When you create a template this way, you must specify a new name for it before clicking the **Add** button.

1.1.4 To Add a Custom View from a Template

To add a custom view from a template:

1. On the **Add Item** menu, click **Custom View**.
2. On the **Add: Custom View** page, click **Template**, click **Browse Content Server**, navigate to the Custom View template you want to add, and then click its **Select** link.
3. Type a name for the Custom View in the **Name** field.

   If your system has multiple languages installed and enabled, click the **click to edit multilingual values** button to edit the names in the other, enabled, languages.
4. Optional In the **Description** field, type a description for the Custom View.

   If your system has multiple languages installed and enabled, click **click to edit multilingual values** to edit the descriptions in the other, enabled, languages.
5. Optional In the **Categories** field, click **Edit** to either select or add a Category to apply to this Custom View.
6. Content Server will save your new Custom View, by default, in the **Enterprise Workspace**. To change the default location, in the **Create In** field, click **Browse Content Server**. Choose **Select** next to a new location.

7. Click **Add**.

Chapter 2

Working with Appearances

An *Appearance* is a container that stores documents and HTML code. Appearances enable you to customize certain locations in Content Server, much as you would design a Web page. Appearances are useful in situations like the following:

- To make the interface conform to the established graphic design standard used by your organization. Customizations can include static content in the form of custom graphics, text, and links to external resources.
- To simplify the interface for certain users or groups by hiding some of the controls or tools in a workspace, project, or folder.
- To embed the interface in another application framework, suppressing the display of everything except the actual content from Content Server.
- To enhance a Workspace, Project, or Folder to display dynamic content that is derived from the page and tailored to the role of the user.
- To enable the personalization of a Workspace, Project, or Folder that is used by a particular group, such as a department.

The ability to add Appearances is usually restricted. For more information, see “Administering Appearances” on page 16.

2.1 Adding and Editing Appearances

There are two types of Appearances you can add to help you customize the interface:

- **Global Appearances** are available system-wide and can only be added to the Appearances Volume. Global Appearances are applied consistently throughout Content Server for specific users.
- **Location-Based Appearances** can be applied the following ways:
  - **Non-Cascading**, which applies the Appearance *only* to the container in which it is added.
  - **Cascading**, which applies the Appearance to the current container in which it is added *and* to any sub-containers. This type of Appearance can also be applied only to a specific set of users or groups.
  - **Cascade to Contents**, which applies the Appearance to the container *and* the immediate contents of the container, but does not apply it to the any of the child containers.

- **Cascade to Level Below Only**, which applies the Appearance to the immediate sub-containers, but not the container itself or any descendants of the container or sub-containers.
- **Cascade to all Levels Below Only**, which applies the Appearance to all of the sub-containers and levels below it, but *not* the container itself.

An Appearance is separated into an **Overview** page and a **Workspace** page, on which you define or edit the Appearance. After you add an Appearance, you define settings and layout on its **Overview** page, and add content, such as images, to its **Workspace** page. The ability to add and edit Appearances is usually restricted. For more information, see "Administering Appearances" on page 16.

The **Overview** page of an Appearance is divided into the following sections:

- Settings
- Layout & Content
- Workspace

Settings

This section displays the type and status of the Appearance. The Appearance type is one of: Global, Cascading, or Non-Cascading. Cascading or Non-Cascading can only be set for Location-based Appearances. The Appearance status can be set to either Enabled or Disabled.

Layout & Content

This section contains text boxes for each custom content area specified by the selected layout. Each text box contains the HTML code that defines the custom area.

Note: You can also include JavaScript, VBScript, or any other code that can be interpreted by a web browser.

Appearance layouts are defined by a number of pre-configured tables that surround the content area in the center of a page. You can choose from five different layouts:

- **Layout 1** lets you specify three custom areas around the content area: top left, top center, and center left.
- **Layout 2** lets you specify eight custom areas around the content area: top left, top center, top right, center left, center right, bottom left, bottom center, and bottom right.
- **Layout 3** lets you specify two custom areas around the content area: top center and bottom center.
- **Layout 4** lets you specify one custom area around the content area: center left.
- **Custom** allows you to define any table structure you want. This layout does not put any areas inside table cells.

You edit each content area using the built-in text editor. A text editor is provided for each content area defined by the layout. Each text document is stored in a **Layout Content** folder, which is added to the Appearance workspace page by default, and bears the name of its associated section. Content Server retains versions of each text document used to maintain custom content areas.

This section also includes **Header** and **Content Server Components** areas. The **Content Server Components** area, lists any of the standard interface elements of a Content Server page that are enabled for the Appearance. These elements are:

- Header
- Search Bar
- Enterprise Menu
- News Player
- Add Item
- Navigation
- Sidebar
- Footer

Workspace

This section provides a detailed view of the items contained in the Appearance's workspace. You must open the workspace itself in order to work with the items. Unlike with other workspaces, you can only add folders, documents, and text documents to an Appearance workspace. Items on an Appearance workspace page are always sorted by name, you cannot change the sort order.

2.1.1 To Open the Appearances Volume

To open the Appearances volume:

- In the **Appearances Administration** section of the **Content Server Administration** page, click the **Open the Appearances Volume** link.

2.1.2 To Add an Appearance

To add an Appearance:

1. Do one of the following:
   a. To add a Location-Based Appearance, click **Add Item**. Select **Appearance Folder** from the list.
   b. To add a Global Appearance, click **Add Item**. Select **Global Appearance** from the list.
2. On the **Add: Appearance** page, type a name for the Appearance in the **Name** field.

If your system has multiple languages installed and enabled, click the **click to edit multilingual values** button to edit the names in the other, enabled, languages.

3. Optional In the **Description** field, type a description for the Appearance.

   If your system has multiple languages installed and enabled, click the **click to edit multilingual values** button to edit the descriptions in the other, enabled, languages.

4. Optional In the **Categories** field, click **Edit** to either select or add a Category to apply to this Appearance.

5. Content Server will save your new Appearance, by default, in the **Enterprise Workspace**. To change the default location, in the **Create In** field, click **Browse Content Server**. Choose **Select** next to a new location.

6. Click **Add**.

2.1.3 To Edit Appearance Settings

To edit Appearance settings:

1. Click the Appearance's **Functions** icon, and then choose **Open**.
2. On the **Appearance Overview** page, click **Settings**.
3. If you are editing a Location-Based Appearance, click one of the following in the **Type** list:
   - **Non-Cascading**
   - **Cascading**
4. Click either the **Enabled** or **Disabled** radio button.
5. Click **Submit**.

2.1.4 To Edit Appearance Layout

To edit an Appearance layout:

1. Click the Appearance's **Functions** icon, and then choose **Open**.
2. On the **Appearance Overview** page, click **Layout & Content**.
3. In the **Layout** field, click one of the following radio buttons:
   - **Layout 1**
   - **Layout 2**
   - **Layout 3**
   - **Layout 4**

- **Custom**

4. Click **Submit**.

2.1.5 To Edit Appearance Components

To edit Appearance components:

1. Click the Appearance's **Functions** icon, and then choose **Open**.
2. On the **Appearance Overview** page, click **Content Server Components**.
3. On the **Edit Appearance Components** page, clear any of the following to exclude the associated interface component from the Appearance:
   - **Header**
   - **Search Bar**

     If the **Search Bar** check box is selected, you must also make a selection from the **Search Bar** list.
   - **Enterprise Menu**
   - **News Player**
   - **Add Item**

     If the **Add Item** check box is selected, you also have the option of selecting the **Add Document Button** and / or the **Add Folder Button** check boxes.
   - **Navigation**
   - **Sidebar**
   - **Footer**
4. Click **Submit**.

2.1.6 To Open an Appearance Workspace

To open an Appearance Workspace:

1. Click the Appearance's **Functions** icon, and then choose **Open**.
2. On the **Appearance Overview** page, click the **Open Workspace** icon, ✎.

2.2 Administering Appearances

The ability to add and modify Appearances is restricted to certain users or groups known as *Branding Administrators*. As a Branding Administrator you set permissions on the Appearance. In setting the permissions, you determine to which users or groups an Appearance should apply. For example, you can:

- Apply the Appearance to all users by assigning the Appearance the *See Contents* permission. This enables public access for the Appearance.
- Apply the Appearance only to specific users by granting those users or groups the *See Contents* permission.
- Apply the Appearance to all project participants by granting coordinators, members, *and* guests of a project the *See Contents* permission for that Appearance.
- Apply the Appearance only to project participants with a specific role by granting coordinators, members, *or* guests of a project the *See Contents* permission for that Appearance.

Note: When a user has permissions on more than one Appearance, the Appearance whose name comes first alphabetically takes precedence.

To become a Branding Administrator, you must have the following:

- The object creation privilege for Appearances.
- The *Add Items* and *Modify* permissions on the appropriate containers.

To add Global Appearances, you also need the *Add Items* permissions on the Appearances Volume. In addition, to modify an Appearance that was added by another user, you need the Modify permission on the Appearance. You do not need permission to modify any subitems that an Appearance may affect.

For more information about permissions, see *OpenText Content Server - Get Started (LLESRT-UGD)*. For information about adding and editing Appearances, see “Adding and Editing Appearances” on page 11.

Chapter 3

Working with JavaScript Functions for Custom Views and Appearances

Content Server provides a group of JavaScript classes that can be used to help customize Appearances and Custom Views. The following sections describe the classes, variables, and formats used by the classes.

3.1 OTvar Replacement Variables

Replacement variables are used to replace the variable class with actual data from Content Server once the page is loaded. You can use replacement variables in HTML and in JavaScript.

OTvar Replacement Variables in HTML

In HTML a replacement tag has a format of: `<ot:OTvar_component_variablename/>`.

The `livelink` component, for example, contains replacement variables with information about the Content Server system, including the `<ot:OTvar_livelink_supportPath/>` tag. This tag is replaced by the relative path to the Content Server `support` virtual folder when Content Server displays the page.

Example: If you add `Location of my graphic file: <ot:OTvar_livelink_supportPath>home/mygraphic.gif<br>` to a Custom View, a Content Server instance with an `img` virtual folder displays it as `Location of my graphic file: /img/home/mygraphic.png`

To add an image that is located in the Content Server `support` folder, include `<img src="{ot:OTvar_livelink_supportPath}myImage.jpg"></img>` in your Custom View. It automatically becomes `<img src="/img/myImage.jp"></img>`, presuming that the URL Prefix for the `/support` folder is set to `img`. The URL Prefix is specified by the Content Server administrator.

OTvar Replacement Variables in JavaScript

In JavaScript, a replacement tag is used as follows:

`var myVar = OTvar.component.variablename;`

Example: The following declares a variable named `imgDir` that points to the Content Server `support` virtual directory (`img`, by default)

```
<script>
...
```

```
                var imgDir = OTvar.livelink.supportPath;
                ...
```

`OTvar_component_variablename`.

3.2 Replacement Variables by Component

There are three replacement variable components. Each one contains a number of replacement variables.

node
Replacement variables in the `node` component provide information about the current node. See "Replacement Variables with Information about the Current Node" on page 18.

currentUser
Replacement variables in the `currentUser` component provide information about the current user. See "Replacement Variables with Information about the Current User" on page 19.

livelink
Replacement variables in the `livelink` component provide information about the Content Server system. See "Replacement Variables with Information about the Content Server System" on page 19.

3.2.1 Replacement Variables with Information about the Current Node

Replacement Variables with Information about the Current Node

`Variable: <ot:OTvar_node_childCount/>`
Description: Number of children in this node.

`Variable: <ot:OTvar_node_description/>`
Description: Description for this node.

`Variable: <ot:OTvar_node_createDate/>`
Description: Date and time that this node was created. Appears in the following format: `2008-06-17 16:36:26`.

`Variable: <ot:OTvar_node_creator/>`
Description: Login name of the creator of this node.

`Variable: <ot:OTvar_node_ID/>`
Description: ID of this node.

`Variable: <ot:OTvar_node_parentID/>`
Description: ID of this node's parent.

`Variable: <ot:OTvar_node_modifyDate/>`
Description: Date and time that this node was last modified. Appears in the following format: `2008-08-25 13:58:30`.

Variable: `<ot:OTvar_node_name/>`
Description: Name of this node.

Variable: `<ot:OTvar_node_type/>`
Description: Type number of this node.

3.2.2 Replacement Variables with Information about the Current User

Variable: `<ot:OTvar_currentUser_birthday/>`
Description: Birthdate of the current user. Appears in the following format: `1974-08-19 00:00:00`.

Variable: `<ot:OTvar_currentUser_email/>`
Description: Email address of the current user.

Variable: `<ot:OTvar_currentUser_firstName/>`
Description: Current user's first name.

Variable: `<ot:OTvar_currentUser_ID/>`
Description: Current user's ID.

Variable: `<ot:OTvar_currentUser_lastName/>`
Description: Current user's last name.

Variable: `<ot:OTvar_currentUser_login/>`
Description: Current user's login name.

3.2.3 Replacement Variables with Information about the Content Server System

Variable: `<ot:OTvar_livelink_cgiPath/>`
Description: Relative path to the Content Server cgi.

Variable: `<ot:OTvar_livelink_name/>`
Description: Display name for this Content Server install.

Variable: `<ot:OTvar_livelink_supportPath/>`
Description: Relative path to the Content Server support directory.

3.3 OTmacro: Macros

The Macros group of classes will have their variable content replaced with actual Content Server data when the page is loaded. Macros are in the following format: `<ot:variable_name options=""></ot:variable_name>`

Variable options can appear in any order. You only need to add options if you want to alter them.

OTmacro: Macros

Class: OTmacro_debug

Purpose: For debugging only. This class will print out all variables, functions, and macros available on the current page.

Options

- none

Examples

- `<ot:OTmacro_debug></ot:OTmacro_debug>`

Class: OTmacro_foldersWithDescriptions

Purpose: Creates a list of Folders that are children of the specified node (defaults to the current node). Contains descriptions (optionally filtering on the content of the description).

Options

- contains: a string that must be present in the Folder description in order for it to be included in the list.
- max: an integer indicating the maximum number of items to include in the list. The maximum default number of items is 24.
- parentID: an integer indicating the ID of the parent container from which to get children. Default: current node.
- functionMenus: a boolean that indicates if function menus should be made available with each item in the list. Default: True.
- icons: a boolean that indicates if icons indicating the item's type should be included in the list. Default: True.
- columns: an integer between 1 and 4 that indicates the number of columns in which to display the results. Default: 3 columns.
- classes: a string containing a space separated list of CSS classes to apply to the resulting list.
- nameFilter: a string that must be present in the name of the item in order for it to be included in the list.

Examples

- `<ot:OTmacro_foldersWithDescriptions/>`
- `<ot:OTmacro_foldersWithDescriptions options=""></ot:OTmacro_ foldersWithDescriptions>`
- `<ot:OTmacro_foldersWithDescriptions options=""></ot:OTmacro_ foldersWithDescriptions>`

Class: OTmacro_recentlyUpdatedDocuments

Purpose: Creates a list of documents that are children of the specified node (defaults to the current node) that are the most recently updated.

Options

- max: an integer indicating the maximum number of items to include in the list. The maximum default number of items is 24.
- parentID: an integer indicating the ID of the parent container from which to get children. Default: current node.
- functionMenus: a boolean that indicates if function menus should be made available with each item in the list. Default: True.
- icons: a boolean that indicates if icons indicating the item's type should be included in the list. Default: True.
- columns: an integer between 1 and 4 that indicates the number of columns in which to display the results. Default: 3 columns.
- classes: a string containing a space separated list of CSS classes to apply to the resulting list.
- nameFilter: a string that must be present in the name of the item in order for it to be included in the list.

Examples

- `<ot:OTmacro_recentlyUpdatedDocuments/>`
- `<ot:OTmacro_recentlyUpdatedDocuments options=""></ot:OTmacro_ recentlyUpdatedDocuments>`

Class: OTmacro_listFolders

Purpose: Creates a list of Folders that are children of the specified node (defaults to the current node).

Options

- max: an integer indicating the maximum number of items to include in the list. The maximum default number of items is 24.
- parentID: an integer indicating the ID of the parent container from which to get children. Default: current node.

- functionMenus: a boolean that indicates if function menus should be made available with each item in the list. Default: True.
- icons: a boolean that indicates if icons indicating the item's type should be included in the list. Default: True.
- columns: an integer between 1 and 4 that indicates the number of columns in which to display the results. Default: 3 columns.
- classes: a string containing a space separated list of CSS classes to apply to the resulting list.
- nameFilter: a string that must be present in the name of the item in order for it to be included in the list.

Examples

- `<ot:OTmacro_listFolders/>`
- `<ot:OTmacro_listFolders options="max:30"/>`
- `<ot:OTmacro_listFolders options="max:30, parentID:2000"/>`

Class: OTmacro_listContainers

Purpose: Creates a list of containers that are children of the specified node (defaults to the current node).

Options:

- max: an integer indicating the maximum number of items to include in the list. The maximum default number of items is 24.
- parentID: an integer indicating the ID of the parent container from which to get children. Default: current node.
- functionMenus: a boolean that indicates if function menus should be made available with each item in the list. Default: True.
- icons: a boolean that indicates if icons indicating the item's type should be included in the list. Default: True.
- columns: an integer between 1 and 4 that indicates the number of columns in which to display the results. Default: 3 columns.
- classes: a string containing a space separated list of CSS classes to apply to the resulting list.
- nameFilter: a string that must be present in the name of the item in order for it to be included in the list.

Examples

- `<ot:OTmacro_listContainers/>`
- `<ot:OTmacro_listContainers options="max:30"/>`
- `<ot:OTmacro_listContainers options="max:30, parentID:2000"/>`

Class: OTmacro_listItems

Purpose: Creates a list of items that are children of the specified node (defaults to the current node).

Options

- max: an integer indicating the maximum number of items to include in the list. The maximum default number of items is 24.
- type: an integer indicating the only item type to show. Default: Show All.
- parentID: an integer indicating the ID of the parent container from which to get children. Default: current node.
- functionMenus: a boolean that indicates if function menus should be made available with each item in the list. Default: True.
- icons: a boolean that indicates if icons indicating the item's type should be included in the list. Default: True.
- columns: an integer between 1 and 4 that indicates the number of columns in which to display the results. Default: 3 columns.
- classes: a string containing a space separated list of CSS classes to apply to the resulting list.
- nameFilter: a string that must be present in the name of the item in order for it to be included in the list.

Examples

- `<ot:OTmacro_listItems/>`
- `<ot:OTmacro_listItems options="max:30"/></ot:OTmacro_listItems>`
- `<ot:OTmacro_listItems options="max:30, parentID:2000"/></ot:OTmacro_listItems>`
- `<ot:OTmacro_listItems options="max:30, parentID:2000, type:202"/></ot:OTmacro_listItems>`

Example showing classes

- `<ot:OTmacro_listitems options="max:30, parentID:2000, classes:'firstClass,another'"/>`

  The options specified in this example result in the unordered lists to have the classes: `"firstClass, another"`.

  `<ul start="8" class="firstClass, another">`

3.4 OTfunc: Helper Functions

The OTfunc classes are JavaScript functions that may be useful when designing customized macros. The are not frequently used.

Otfunc: Helper Functions

Class: OTfunc.toDate

Purpose: Returns a date string as is returned by Content Server into a JavaScript date object.

Input: dateString:<*the_string_to_be_converted*>.

Output: A JavaScript date object set to the date represented by the string.

Class: OTfunc.substitueAttributeVariable

Purpose: To exchange all occurrences of a given variable founds within a given attribute with the given value

Input

- attributeName:<*the_attribute_to_search_in*>
- variableName:<*the_name_of_the_variable_being_searched*>
- variableValue:<*the_value_to_be_substituted*>

Output

- none

3.5 User Customizations

It is possible to create custom JavaScript files, replacement variables and macros for use with Content Server.

3.5.1 Customized JavaScript

You can add your own custom JavaScript file, which will be available within all Appearances and Custom Views. If you want to add a JavaScript function that appears in the entire Content Server system, you can add the script to the `custom.js` file, located in the `root` folder of the Content Server `support` directory.

Note: You must have the proper permissions to access the `support` directory. For more information about these permissions, contact your Administrator.

3.5.2 Customvar: User Created Replacement Variables

You can create your own replacement variables, which will be automatically processed by adding them to the CUSTOMvar JavaScript object.

Example: If you want to create a replacement tag that always shows the most important objective, you can add the following to the `custom.js` file:

```
Customvar.mainObjective='Sell! Sell! Sell!';
```

Note: When you define a replacement variable, you must use a *period (.)* after `Customvar` instead of an *underscore (_)*.

The `Customvar` object can be extended inside a script block within an Appearance. The replacement variable defined will *only be available with that Appearance*, and will appear with any Custom Views that display in conjunction with that particular Appearance.

3.5.3 CUSTOMmacro: User Created Macros

You can create your own macros, which will be automatically processed by adding them to the `CUSTOMmacro` JavaScript object.

Example: If you want to create a macro that wishes users happy birthday, you can add the following to the `custom.js` file:

```
CUSTOMmacro.birthdayGreeting = function( options, element ){

 var BirthdayString;
    var birthday = OTfunc.toDate(OTvar.currentUser.birthday);
    var today = new Date();
    var age;

 if (birthday.getUTCDate() === today.getUTCDate()
    && birthday.getUTCMonth() === today.getUTCMonth())
    {
        age = today.getUTCFullYear() - birthday.getUTCFullYear();
        BirthdayString = (" Happy Birthday! You are now " + age + " years old.");
        $(element).after(BirthdayString);
        // $(element).before(BirthdayString);
    }
}
```

Note: The `OTVar` and `OTfunc` objects are referenced in order to retrieve the user's birthday and convert it to a JavaScript date for comparison.

Each macro function is passed in two variables: `options` and `element`. The `options` variable contains a JavaScript object that contains a named property for each of the options set in the `options` attribute of the macro tag.

Example: If you had created the following macro:

```
                  CUSTOMmacro.test = function( options, element ){

                      if ( typeof(options.test) != 'undefined' )
                      {
                          $(element).after(options.test);
                          // $(element).before(options.test);
```

```
                    } else {

                        $(element).after("No test option found");
                        // $(element).before(options.test);
                    }

                }
```

and then entered `<ot:CUSTOMmacro_test options=""></ot:CUSTOMmacro_test>` in an Appearance or Custom View, when the page is viewed, the word *hello* will appear inside the macro tag. If the `test` option is not found, *No test option found* would be inserted into the macro tag.

3.6 Adding a JavaScript Function to a Custom View or Global Appearance

To add a JavaScript function to a custom view:

- Open the `customview.html` file in an editor, and add the JavaScript functions that you want to appear when the Custom View is enabled.

Note: You can add the JavaScript functions to a text file and add the text file as a Custom View.

To add a JavaScript function to a global appearance:

1. Click a Global Appearance **Functions** icon, and then click **Open**.
2. On the Global Appearance page, click the **Edit** icon for the section to which you want to add a JavaScript function.
3. On the Edit page, type the JavaScript functions that you want to appear in the text field, and then click **Add Version**.