Template Builder User Guide

Version 3.0

Template Builder

Disclaimer

Vizrt provides this publication "as is" without warranty of any kind, either expressed or implied. his publication may contain technical inaccuracies or typographical errors. While every precaution has been taken in the preparation of this document to ensure that it contains accurate and up-to-date information, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained in this document. Vizrt's policy is one of continual development, so the content of this document is periodically subject to be modified without notice. These changes will be incorporated in new editions of the publication. Vizrt may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.
Vizrt may have patents or pending patent applications covering subject matters in this document. The furnishing of this document does not give you any license to these patents.

Antivirus

Vizrt does not recommend or test antivirus systems in combination with Vizrt products, as the use of such systems can potentially lead to performance losses. The decision for the use of antivirus software and thus the risk of impairments of the system is solely at the customer's own risk.

There are general best-practice solutions, these include setting the antivirus software to not scan the systems during operating hours and that the Vizrt components, as well as drives on which clips and data are stored, are excluded from their scans (as previously stated, these measures cannot be guaranteed).

Technical Support

For technical support and the latest news of upgrades, documentation, and related products, visit the Vizrt web site at www.vizrt.com.

Created on

2023/10/04

Contents

1   Introduction

Template Builder lets you make customized templates using scene import or existing templates from Viz Pilot's Template Wizard. This user guide shows you how to customize templates.

> ⓘ **Info:** A key feature is that you can add custom HTML panels to templates, giving full control over the template through custom scripting and logic.

1.1   Workflow

A simplified version of the workflow follows below:

· Scenes are made in Viz Artist.
· The scenes are imported into Template Wizard, where templates are made.
· Templates are edited and new templates can be made in Template Builder.
· The template is saved in the Viz Pilot system and is available to newsroom and control room systems for playout.

> ⚠ **Note:** Changes made to a template in Template Builder are not be available when opening the template in Template Wizard.

1.2   Related Documents

The templates customized in Template Builder can be used by other Vizrt products such as Viz Pilot Edge, Viz Story and Viz Multiplay. For more information about all Vizrt products, visit:

- www.vizrt.com
- Vizrt Documentation Center
- Vizrt Training Center
- Vizrt Forum

1.3   Feedback

We welcome your feedback and suggestions regarding Vizrt products and this documentation. Please contact your local Vizrt customer support team at http://www.vizrt.com.

2   Setup And Configuration

This section covers the following topics:

- Opening Template Builder
- Configuring Preview Server
- Specifying a Graphic Hub Endpoint
- Monitoring Graphic Hub Status

For software and hardware requirements, please check the Viz Pilot Edge User Guide.

2.1   Opening Template Builder

Template Builder opens as a web application in your default browser.

The URL to access Template Builder, if hosted on the Pilot Data Server, is: http://pds-hostname:8177/app/templatebuilder/TemplateBuilder.html.

2.2   Configuring Preview Server

Preview Server manages one or more Viz Engines, providing frames for thumbnails and snapshots in an ongoing preview process.

Preview Server must be configured in the Pilot Data Server:

1. Access the Pilot Data Server Web Interface: http://pds-hostname:8177.
2. Click the **Settings** link.
3. Select the *preview_server_uri* setting, and add the URL for the machine on which you installed Preview Server (http://previewserver-hostname:21098).
4. Click **Save**.

> ⚠ **Note:** All applications with a connection to the database will now have access to Preview Server.

2.3   Specifying A Graphic Hub Endpoint

If you are using multiple Graphic Hubs, the one used to store your scenes must be configured in Pilot Data Server:

1. Access the Pilot Data Server Web Interface: http://pds-hostname:8177.
2. Click the **Settings** link.
3. Select the *graphic_hub_url* setting, and add the URL for the machine on which your scenes are stored (http://gh-hostname:19398).
4. Click **Save**.

2.4   Monitoring Graphic Hub Status

Since some users have multiple Graphic Hubs (GHs) for design, distribution, testing and production, **green icons** at the bottom of the interface show you which GH and which database you are currently connected to:

> ⚠ **Note:** GH REST status info is based on the *graphic_hub_url* parameter mentioned above, not Graphic Hub's search provider settings.

3 Creating And Managing Templates

Create templates using the scene import feature.

This section covers the following topics:

- Creating a New Template
- Opening a Template
- Template Management

3.1 Creating A New Template

- Click **Create New** in the **Open Template** menu or the plus icon on the toolbar.

> ⚠ **Note:** If this option is not available, make sure the required programs are up-to-date.

You now need to add a scene to your new, empty template:

- Go to **Scenes** at the top left of the interface.
- Select **+Add Scene**.
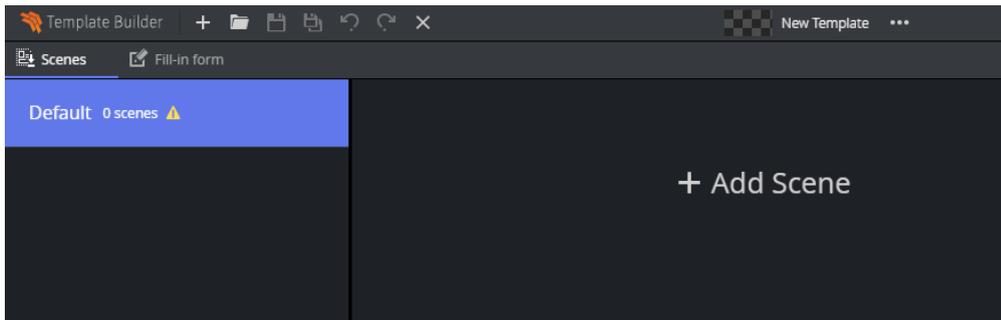
The **Select scenes** menu appears, containing all of the scenes stored in the Graphic Hub to which you are connected:
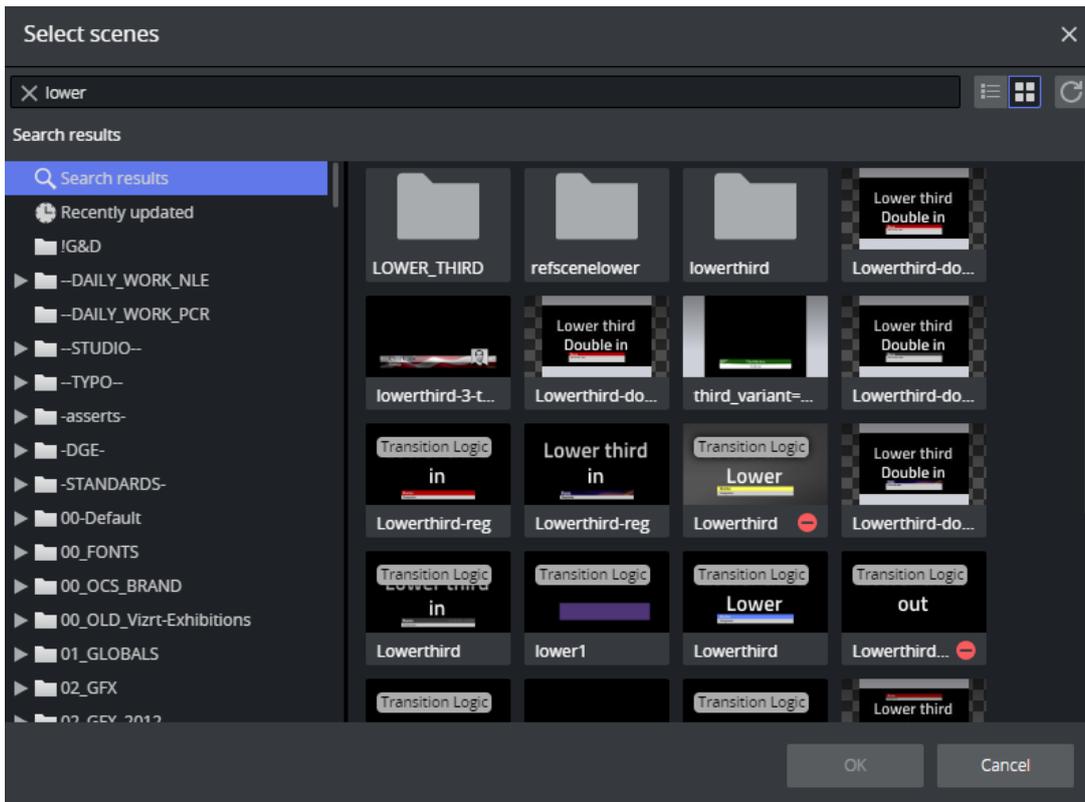
> ⚠ **Note:** The Graphic Hub containing your scenes is specified through the *graphic_hub_url* setting in Pilot Data Server.

- Enter a search term or browse the folder structure. Once you have selected the correct scene or scenes, press **OK** to add them to the template:
- If you want to assign the scene to a different concept, right-click it and select **Replace concept**:
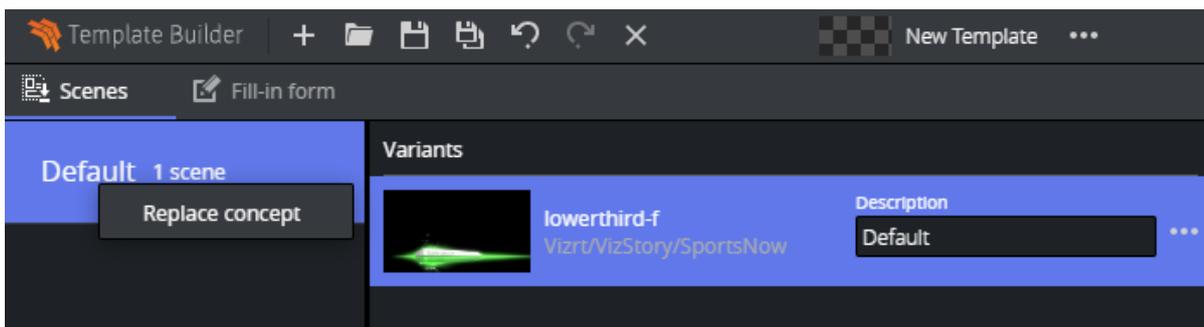
- Select an existing concept or enter a new **Concept name** and click **Ok**.

> ⚠ **Note:** New concepts are inactive by default, which means they will not be visible in other applications. Once the template has been saved, use the **Concept Manager** (see below) at the top right of the interface to activate them.

- Click the **Fill-in form** tab to see a preview of the template with its auto-generated fields.

3.2   Opening A Template

- Click **Open**, or **CTRL + O**, to open a dialog containing templates available within the Pilot system.
- In the **Open Template** dialog, use **Concepts** and **Tags** to filter templates. The search can also be narrowed down by searching for the template name in the **Type to filter...** field at the top of the dialog.
- Select a template and click **OK** or double-click it to open.

> ⚠ **Note:** Template Builder can detect if there is an unsaved state while opening an old template. If this occurs, the following message will show: "The old template was updated and needs saving". Save the template and continue your workflow.
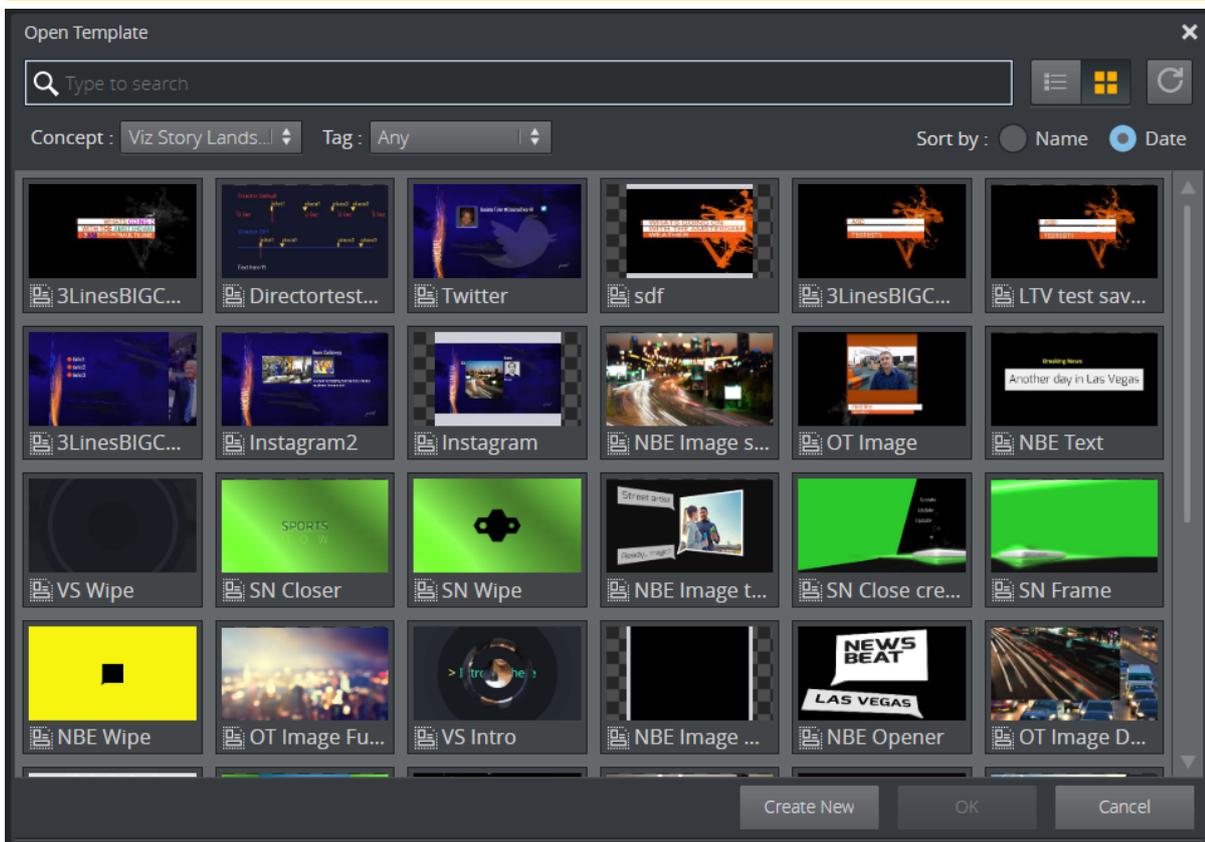
3.3 Template Management

3.3.1 Advanced Dialog
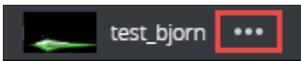
The menu at the top center of the interface allows you to open the **Advanced** dialog:
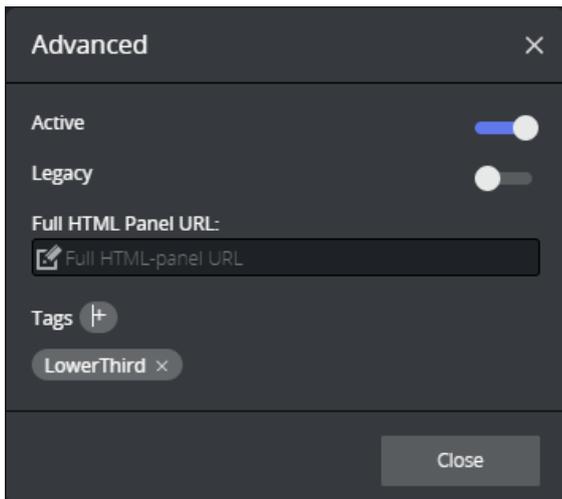
- · **Active:** To mark this template as a draft or not. A template marked as a draft means that this template is not visible in Viz Pilot Edge.
- · **Legacy:** If Viz Pilot Director should open it using Viz Pilot Edge or Viz Pilot News (a legacy setting):
  - · **Enabled:** In Viz Pilot Director. Legacy templates open in the built-in template window.
  - · **Disabled** (default): In Viz Pilot Director 8.6 or later, the template opens in Viz Pilot Edge.
- · View and edit the **Tags** of the template.

3.3.2 Concept Manager

The Concept Manager is available by clicking the icon on the top right toolbar:
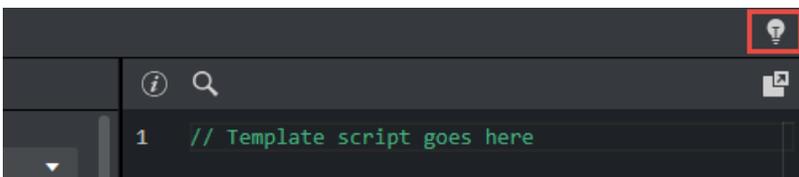
The Concept Manager allows you to mark Concepts as active or inactive (draft):
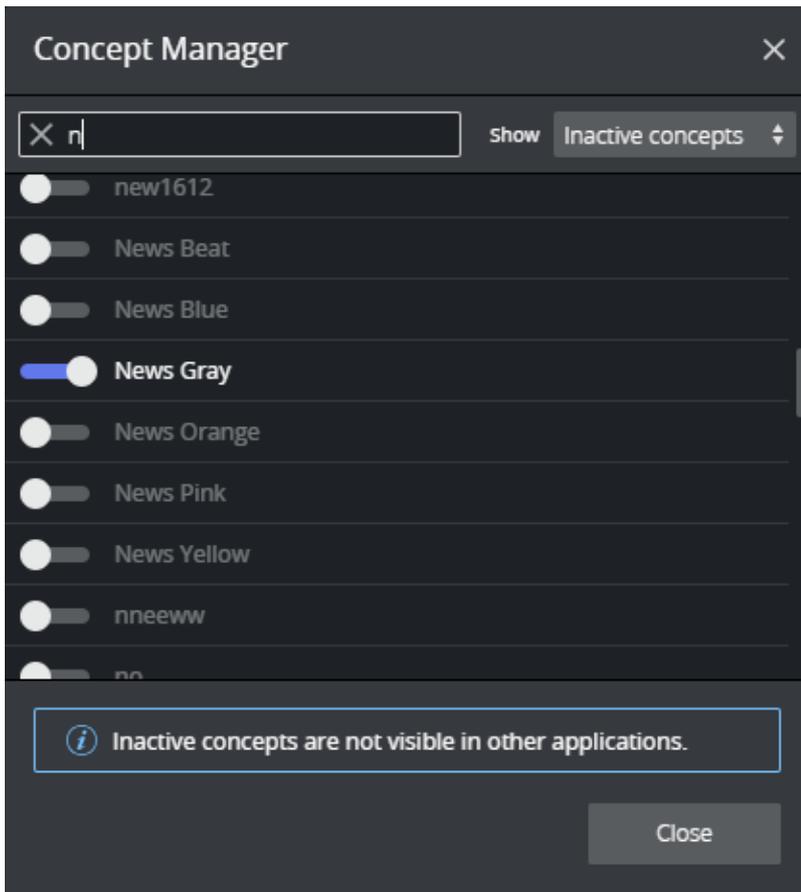
Inactive concepts will not be visible in Viz Pilot Edge. The templates of this concept will also be hidden from the Viz Pilot Edge user. Setting a concept as inactive means it can be in draft mode for the Template Builder user.

Once it is activated, it is visible to the Viz Pilot Edge newsroom users.

See Also:

· Transition Logic and Combo Templates

4 Customizing Templates

For simple use cases, the auto-generated template is often usable out-of-the-box after importing scene(s). Though, in a professional newsroom workflow, advanced customization is needed. There are numerous of ways a template can be customized in TemplateBuilder:

- Template Scripting
- Template Layout
- Template Fields
- Custom HTML Templates
- Auto-generated Title

4.1 Template Scripting

If dynamic or advanced customization of the fill-in form is needed, Viz Pilot Edge allows this through template scripting.

Scripts are written in TypeScript, and access the template via a provided Script API named `vizrt`.

It allows users to customize how templates look and behave, as well as fill in values from external sources. This section describes how to use the script editor, and access the values of the fields supported in Template Builder.

> ⓘ **Info:** Check out the Quick Start Examples for some quick hands-on experience.

> ⚠ **Note:** For testing API endpoints, please use the following:
> - HTTP://<PDS-HOST>:8177/testing/fakepremierleague/
> - HTTP://<PDS-HOST>:8177/testing/fakepersonsearch/

These are the following topics:

- The Script Editor
- Initialization
- Field Access
  - Accessing Concepts & Variants from Scripting
  - External Sources
  - Image Metadata
  - Read Only Fields
  - Unsupported Fields

4.1.1 The Script Editor

The script editor is available in the "Fill-in Form" tab of Template Builder.
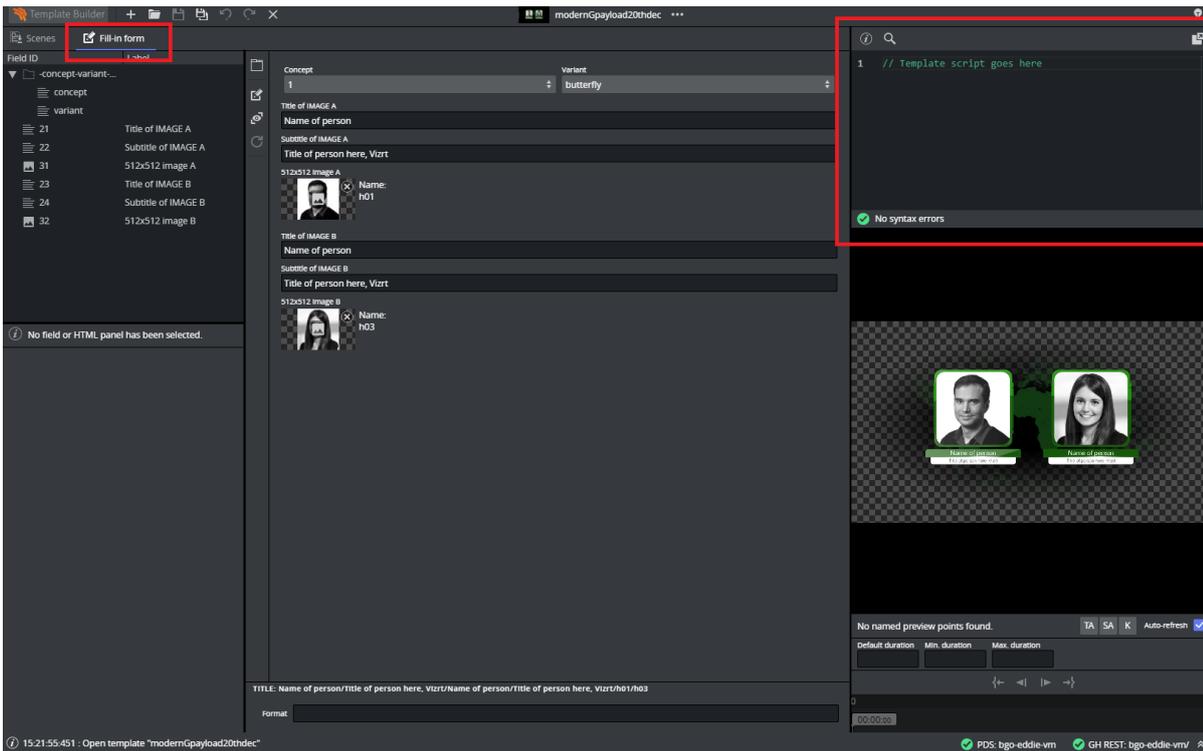
It can be used docked or undocked from the Template Builder. While undocked, you can adjust the size of the window for a smoother experience.

There is a search option to search within the script code, and you can access it by clicking the icon [search icon] within the script editor.
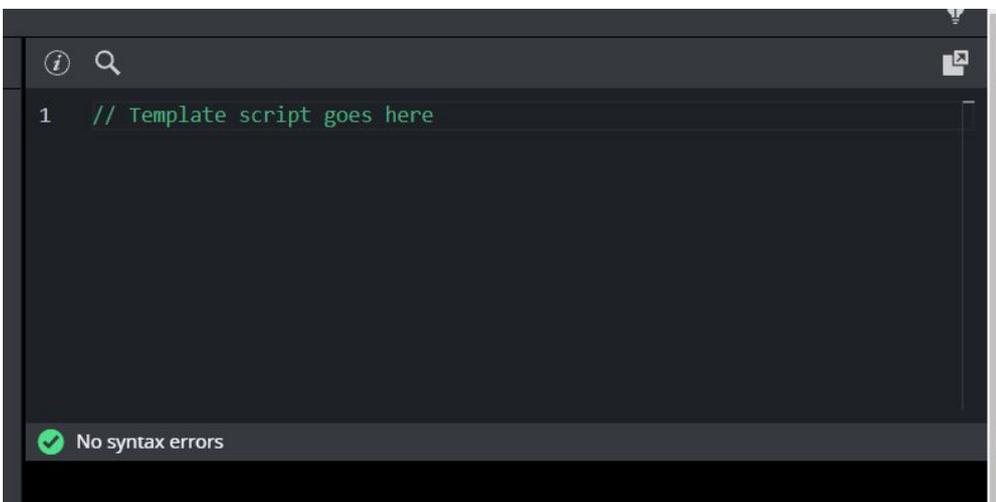
The script editor also provides error messages depending on the problem. It will show compile or run time errors.

Compile Error

Runtime Error

When trying to catch a runtime error, it is recommend that you name a function with a string value. By naming a function, this name is used in the error message to indicate where the error came from.

In the following example, if you catch an error, the string used under **"user defined name"** is shown in the error message:

In this case, you can just add the string value in your function error report so it shows in the error message:

4.1.2   Initialization

There are two global events that can be used to initialize the Fill-in Form:

- **onCreate:** Called when template is opened in Viz Pilot Edge.

> ⚠ **Note:** This event is **not** triggered when creating a new data-element from an existing one.

Example:

```
vizrt.onCreate = () => {
    // For example set startup values, initialize fields visibility, etc.
    Initialize()
}
```

- **onLoad:** Called when an existing data-element is loaded in Viz Pilot Edge. The normal steps are to initiate/refresh data each time you open a data-element.

> ⚠ **Note:** This event can potentially modify the data-element when opening, which means it must be saved before being draggable/sendable to a NRCS.

Example:

```
vizrt.onLoad = () => {
    // For example reset or refresh values, etc.
    ResetForm()
}
```

> ⓘ **Info:** These events must be defined in the Template Builder's script, but they are **not** triggered in Template Builder.

4.1.3   Field Access

When using the scripting tool in the template, the individual fields must be accessed through the global name space *vizrt.fields* (for example, *vizrt.fields.$singleline.value*).

> ⚠️ **Note:** Writing *$singleline.value* instead of *vizrt.fields.$singleline.value* will not work, and will give a Compile Error.

The script executes when a graphic element is opened or created with the scripted template in Viz Pilot Edge.

In Template Builder, the script is also re-loaded and restarted when there are changes made to it.

By typing *vizrt.fields*, the editor's autocomplete will show you the available fields to choose from.
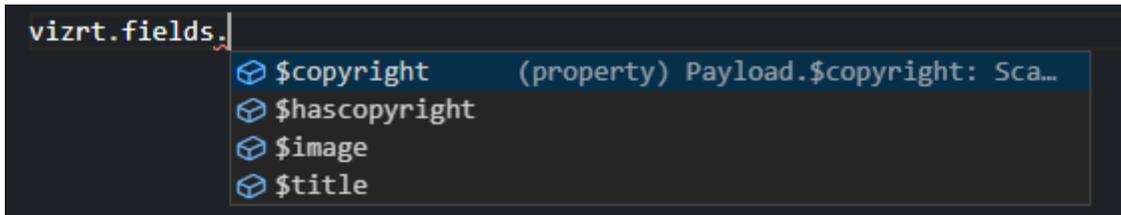
You can read and write field values, as well as react to value changes from outside the script.

You can also access the properties **error**, **hidden**, **readOnly**, and **tip** of the `vizrt` fields.
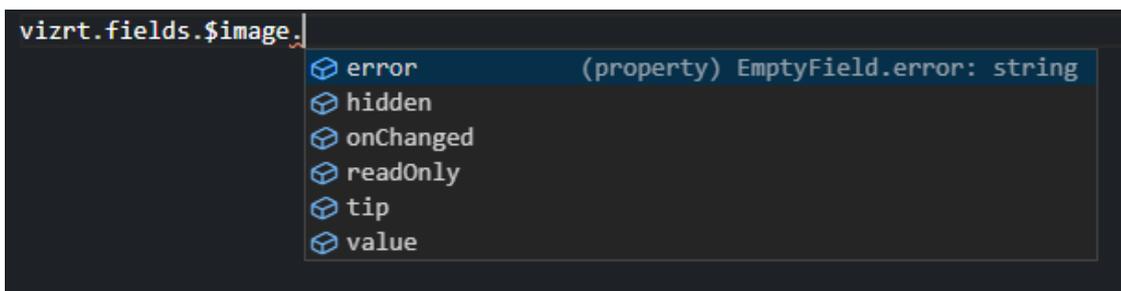
- **onChanged:** A property on fields that you can set as a function, and if you do so, this function is called whenever the value of the fields changes, and gets the new value as an argument. If this is not set, it is *null*.

  > ⚠️ **Note:** Changes done to field values by the template script will *not* trigger the **onChanged** function to be called.

- **readOnly:** Read and write *boolean* access, to whether the field should be editable in the form or not. If *false*, the field and its input elements are editable in the UI. If *true*, they are read-only and greyed out in the UI, but are accessible, saved and loaded as part of the payload.
- **hidden:** Read and write *boolean* access, to whether the field should be editable in the form or not. If *false*, the field and its input elements are present and visible in the UI. If *true*, they are hidden from the UI but are accessible, saved and loaded as part of the payload.
- **error:** Read and write *string* access, to an error to display for this field. It overrides other error messages associated with the field when non-empty (e.g. errors due to input length or regex constraints specified in the field definition).
- **tip:** Read and write *string* access, to a tip to use for this field. It overrides the tip specified in the field definition when non-empty.

> ⚠ **Note:** Dashes cannot be used in Typescript with the dot syntax, instead you can use *vizrt.fields["$01-week"]* syntax to be able to access it.

Typescript lets you access fields using known names (for example, *vizrt.fields["$field_2"]),* but does not allow expressing the field names using variables (for example, *vizrt.fields["$field_" + num]),* as it is unable to determine whether the field name is valid and what type the field is. To overcome this, the type checking associated with *vizrt.fields* can be locally ignored by using the *any* type. You may do so inline using for example *(vizrt.fields **as any**)["$field_" + num],* or storing it as a new variable, as shown in the example below.

```typescript
function resetValues() {
    for (let playerNumber = 1; playerNumber <= 2; playerNumber++) {
        const untypedFields = vizrt.fields as any

        untypedFields["$player" + playerNumber + "_firstname"].value = ""
        untypedFields["$player" + playerNumber + "_lastname"].value = ""
        untypedFields["$player" + playerNumber + "_age"].value = 0
    }
}
```

> ⚠ **Note:** Be aware that when using an object cast to *any*, the script editor will **not** be able to help you catch errors like setting a boolean value to a string field, or even trying to access a field that does not exist..

Accessing Concepts & Variants from Scripting

To access the information from concept and variants fields, because they contain dashes, you must use square brackets, as shown in the example below.
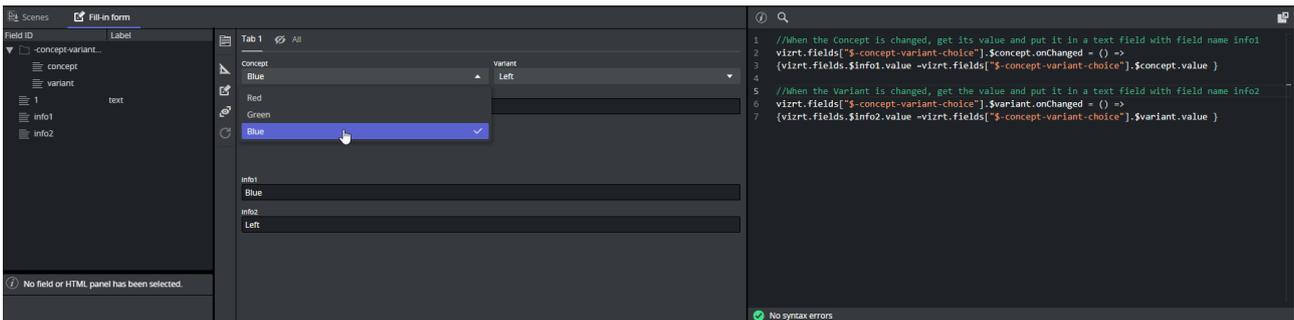

```
//When the Concept is changed, get its value and put it in a text field with field
name info1
vizrt.fields["$-concept-variant-choice"].$concept.onChanged = () =>
{vizrt.fields.$info1.value =vizrt.fields["$-concept-variant-choice"].$concept.value }
```

```
//When the Variant is changed, get the value and put it in a text field with field
name info2
vizrt.fields["$-concept-variant-choice"].$variant.onChanged = () =>
{vizrt.fields.$info2.value =vizrt.fields["$-concept-variant-choice"].$variant.value }
```

External Sources

Whether on template load, or as a reaction to a field change, you can initiate HTTP, HTTPS or REST calls to fetch values from third party or external services.

This can easily be done via the browser's built-in *fetch* API: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.

> ⓘ See the Quick Start Examples section for a short example of a REST call triggered by an *onChanged* event.

Image Metadata

Every image has some amount of metadata attached to it, and with this field you are able to access this metadata by using the script editor.

You can upload images and the corresponding metadata to asset source servers. By accessing an image's metadata, you can auto-fill the name field within a template or alternatively display or hide an image that might be copyrighted.

> ⚠ **Note:** Image scripting metadata currently works for images retrieved from the following asset source servers: Vizone, Vos, GH and OMS.

How to Use the Image Metadata

If you are using a newly created template, simply choose any image, and then query that image's metadata in the script editor by querying the image's metadata map. With an existing template, which already contains an image that was created before the 2.4 version of Template Builder and Pilot Edge, then simply click on the current image and re-select it from the asset selector, or alternatively select another random image and then select back the original one. Once this is done, you can proceed by querying the image's metadata map.

This example checks if *imageName* has a metadata key named *test* and then tries to get the value of that metadata key. You can use the script syntax shown below:

```
let hasMetadataKey: boolean = vizrt.fields.$imageName.metadata.has("test") //true if
the metadata contains the key test
let metadataValue: string = vizrt.fields.$imageName.metadata.get("test") //it is set
to the value it has in the metadata, otherwise it is undefined
```

The following example shows a more realistic code example of the scripting metadata functionality. An image field named *image1* tries to get the metadata value for *description* and put it into a text field called *img1_txt* in case it is found, otherwise a message shows explaining it was not found.

```
if (vizrt.fields.$image1 != undefined && vizrt.fields.$image1.metadata != undefined)
{
    let keyName = "description"
    let a = vizrt.fields.$image1.metadata.get(keyName) // get the metadata value that
has the given key
    if (a != undefined) {
        vizrt.fields.$img1_txt.value = a // if the value is not undefined, then set
it into a string field within the template
    } else {
        vizrt.fields.$img1_txt.value = "The key '" + keyName + "' was not found
inside the metadata map"
        // Alternatively, set it to nothing: vizrt.fields.$img1_txt.value = ""
    }
}
```

This code example will react to any image changes. When the *image2* field gets assigned a new image, it tries to get the description from the metadata associated with the new image, into the text field *img2_txt*. If the description does not exist, a message displays explaining it was not found.

```
vizrt.fields.$image2.onChanged = () => {
    if (vizrt.fields.$image2.metadata != undefined) {
        let keyName = "description"
        let a = vizrt.fields.$image2.metadata.get(keyName) // get the metadata value
that has the given key
        if (a != undefined) {
            vizrt.fields.$img2_txt.value = a // if the value is not undefined, then
set it into a string field within the template
        } else {
            vizrt.fields.$img2_txt.value = "The key '" + keyName + "' was not found
inside the metadata map"
            // Alternatively, set it to nothing: vizrt.fields.$img2_txt.value = ""
        }
    }
}
```

To access the entire unprocessed/unparsed metadata file, use *wholeMetadataString*. This can be useful for debugging, or for finding the keys available in the metadata:

```
let rawMetadata:string = vizrt.fields.$imageName.metadata.get("wholeMetadataString")
```

Image Metadata XML

An image's metadata is stored within asset source servers in XML format, and the XML metadata structure should follow a simple field-value (key-value) structure. This is to guarantee that all the metadata is correctly mapped and made accessible through the script editor.
However, since many image metadata XMLs are disorderly, a parser has been created to handle most XML structures, although this has some consequences that should be made aware of. All these example scripts are based on an image field named "image1".

Empty field-value pairs get added accordingly:

```
<field name="car"/>
<field name="color">
    <value/>
</field>

// Accessing these can be done using:

let a = vizrt.fields.$image1.metadata.get("car")
let b = vizrt.fields.$image1.metadata.get("color")

// Both a and b will be ""
```

Nested structures get stored based on their hierarchy, with "/" being the parent-child separator:

```
<field name="access-rights">
    <field name="user-rights">
        <value>true</value>
    </field>
</field>

// To access the user-rights value:

let a = vizrt.fields.$image1.metadata.get("access-rights/user-rights")

// a will then be set to true.
```

Any fields with duplicate names get assigned a unique name with an incrementing suffix:

```
<field name="file-link-id">
    <value>id123</value>
</field>
<field name="file-link-id">
    <value>id456</value>
</field>
<field name="file-link-id">
    <value>id789</value>
</field>
```

Access duplicate names like this using the incrementing suffix:

```
let a = vizrt.fields.$image1.metadata.get("file-link-id")
let b = vizrt.fields.$image1.metadata.get("file-link-id(2)")
let c = vizrt.fields.$image1.metadata.get("file-link-id(3)")

a will be "id123"
```

```
b will be "id456"
c will be "id789"
```

Read Only Fields

Some fields are currently supported only for read-access by the scripting API. These are the following:

- Duplet
- Triplet
- Map
- Image
- Video

For the **Image** and **Video** fields, the script is able to access some properties (their height, width, etc.) from the file.

The following example shows how to retrieve the image height:

```
vizrt.fields.$ImageInfo.value = "No image info";

vizrt.fields.$image.onChanged =() => {
    vizrt.fields.$ImageInfo.value = 'Image changed';

    var v = vizrt.fields.$image.value;
    if(v != undefined && v.height != undefined)
        vizrt.fields.$ImageInfo.value = v.height.toString();
    else
        vizrt.fields.$ImageInfo.value = "No image info";
}
```

> ⚠ **Note:** Because images and videos can be undefined, they must be checked before they are used.

Unsupported Fields

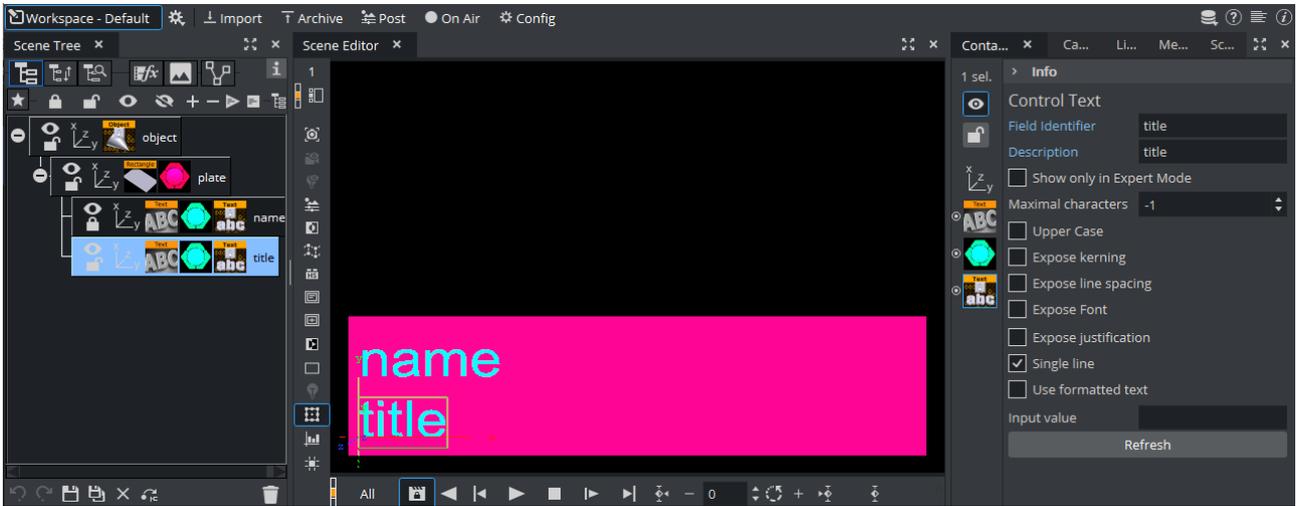As of now, all **List** and **Table** fields are unavailable from the scripting API.

4.1.4   Quick Start Examples

> ⓘ **Info:** In this section you can find short examples of how to use the scripting functionality.

- Automatically Clear Title Field
- Fetch Title from REST Service

In Viz Artist, create and save a regular Viz Pilot template scene with two text control fields:

- name
- title

Make sure to uncheck **Use formatted text** in the Control Text properties for both fields, which is easier to work with.
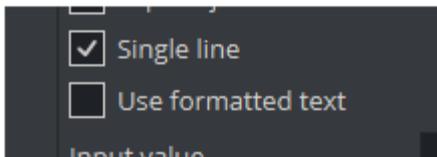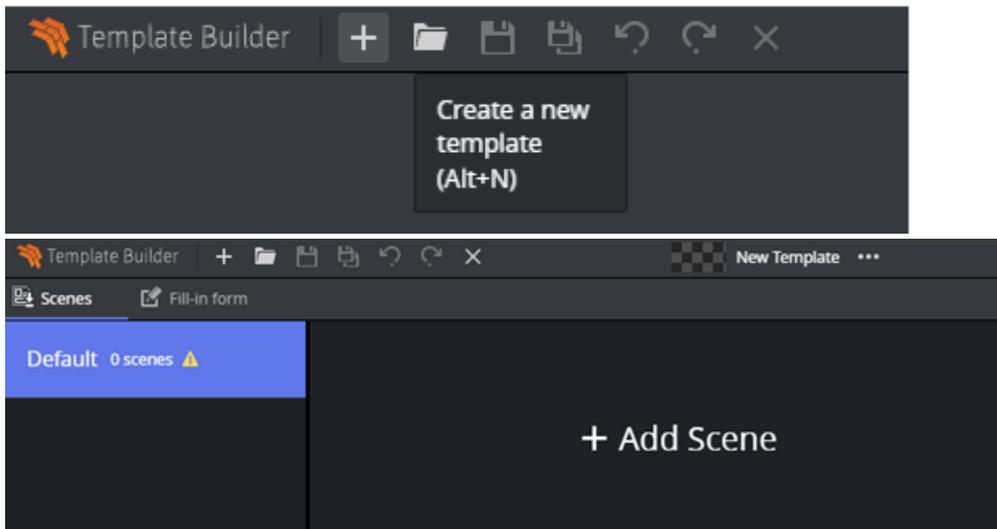


Automatically Clear Title Field

In this example, the following basic features are shown:

- Script that executes on template load.
- Reacting to user changes to the fields.
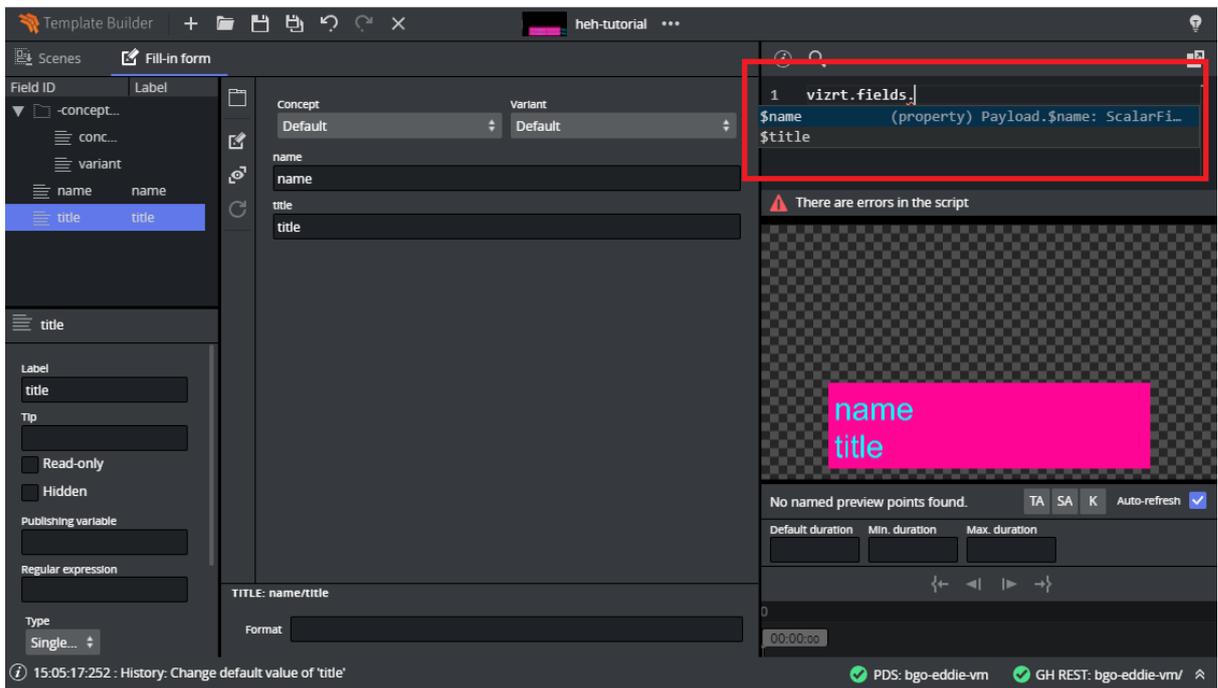- Modifying fields from the script.

1. Create a new template based on this in Template Builder, by choosing **Create a new template** and adding your newly created scene via **Add Scene**.

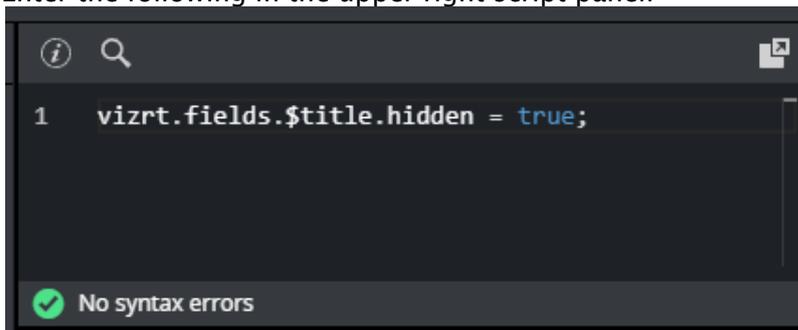2. Go to the **Fill-in form** tab, which should look like this:
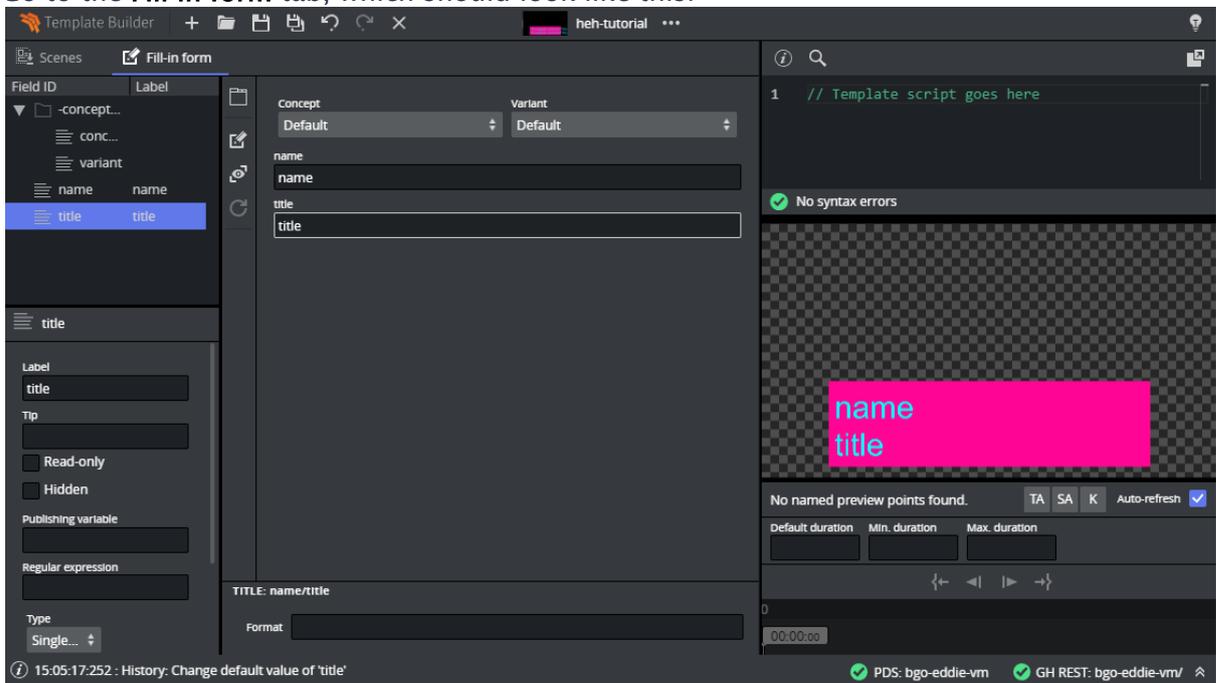
3. In the scripting tab, you can verify that the template fields are available by typing *vizrt.fields* and looking at the autocompletion:
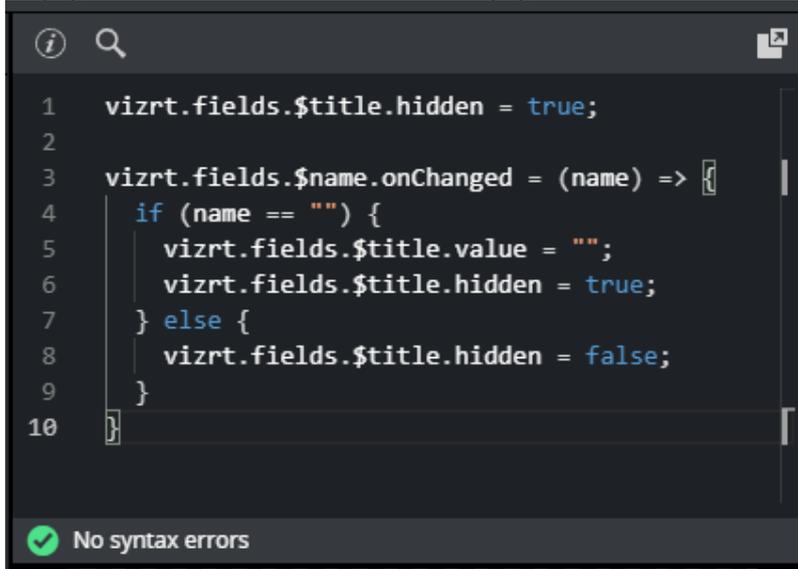
4. Enter the following in the upper right script panel:

This script causes the template to hide the **title** field when the template loads. You should see this change applied immediately on the payload editor preview:
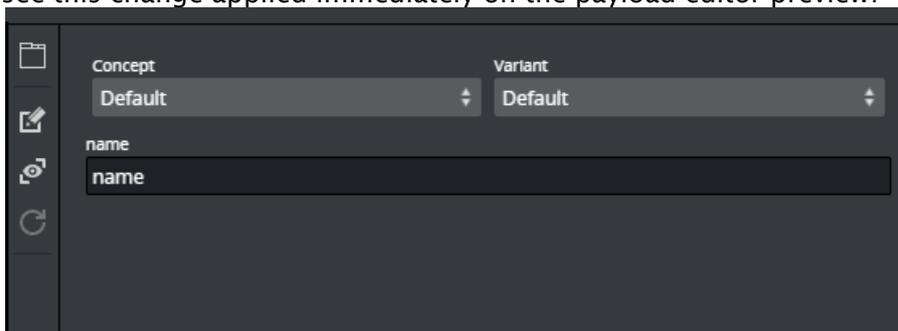
5. The script is now extended to react to changes made in **name**. When *name* is empty, *title* should be hidden and empty, but not otherwise:

```
1    vizrt.fields.$title.hidden = true;
2
3    vizrt.fields.$name.onChanged = (name) => {
4      if (name == "") {
5        vizrt.fields.$title.value = "";
6        vizrt.fields.$title.hidden = true;
7      } else {
8        vizrt.fields.$title.hidden = false;
9      }
10   }
```

✅ No syntax errors

The *title* field will now hide and clear when *name* is cleared and reappear when something is entered into name.

6. **Save** the template and observe this action in the Pilot Edge client.

> ⚠ **Note:** Fields with dashes in their name, cannot be used in Typescript with the dot syntax, instead you can use *vizrt.fields["$01-week"]* to be able to access it. This means, when creating a new scene, you should use camel case notation or underscore (for example, *01thisIsMyField* or *01_week*), to access the field with the dot syntax.

Fetch Title from REST Service

In this example, the script will automatically fill the **title** field by fetching it from a REST endpoint.

This illustrates more advanced features:

· Using the standard browser fetch API.
· Changing field values based on responses from other services.

Using the same template as the example above, or creating a new one from the same scene, delete anything in the script tab and write the following:

```
1   vizrt.fields.$name.onChanged = (name) => {
2     fetch("http://HOSTNAME:8177/testing/fakepersonsearch/" + name)
3     .then(response => response.json())
4     .then(title => {
5       vizrt.fields.$title.value = title;
6     });
7   }
```

No syntax errors

Remember to replace "HOSTNAME" with the hostname of your Pilot Data Server. In a default install, this should be showing in your Template Builder address bar.

In the following example using Google Chrome, the hostname would be "stephanie", marked in blue:
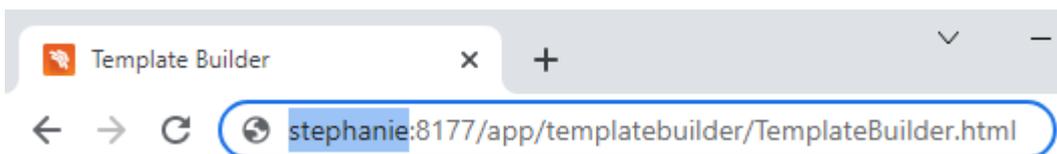
If everything is working correctly, you should see an autogenerated title appear when you set or change the **name** field.
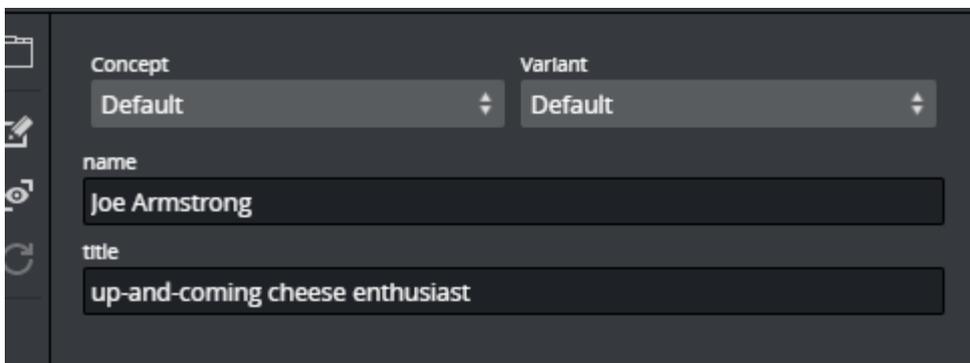
In this case, a specially provided test endpoint was used on the Pilot Data Server, but you can point to any other REST resource. Also, you are not constrained to the fetch API used. All standard JavaScript network mechanisms can be used.

4.2   Template Layout

Editing a template's layout makes it easy to create fill-in forms for journalist. With drag & drop functionality, creating new fill-in forms is quick and easy.

- When a template is created, only the **auto-generated form** is displayed. The layout of this form cannot be modified.

- Adding new tabs enables you to quickly create additional fill-in forms based on selected fields.
- When adding new tabs, the default auto-generated form is accessible in the **All** tab. The auto-generated tab can be hidden from the Pilot Edge user.
- In the additional tabs, it is possible to resize, move, edit, add and delete fields quickly.

This section covers the following topics:

- Creating a Template
- Adding Alternative Layout Forms
    - Adding, Moving, and Resizing Fields
    - Renaming, Deleting and Reordering Tabs
    - Hiding and Showing the Auto-Generated Tab

Follow the steps below to get started.

4.2.1   Creating a Template

1. Open or create a new template and add a scene.
2. Click **Fill-in form**. The default view is then displayed:
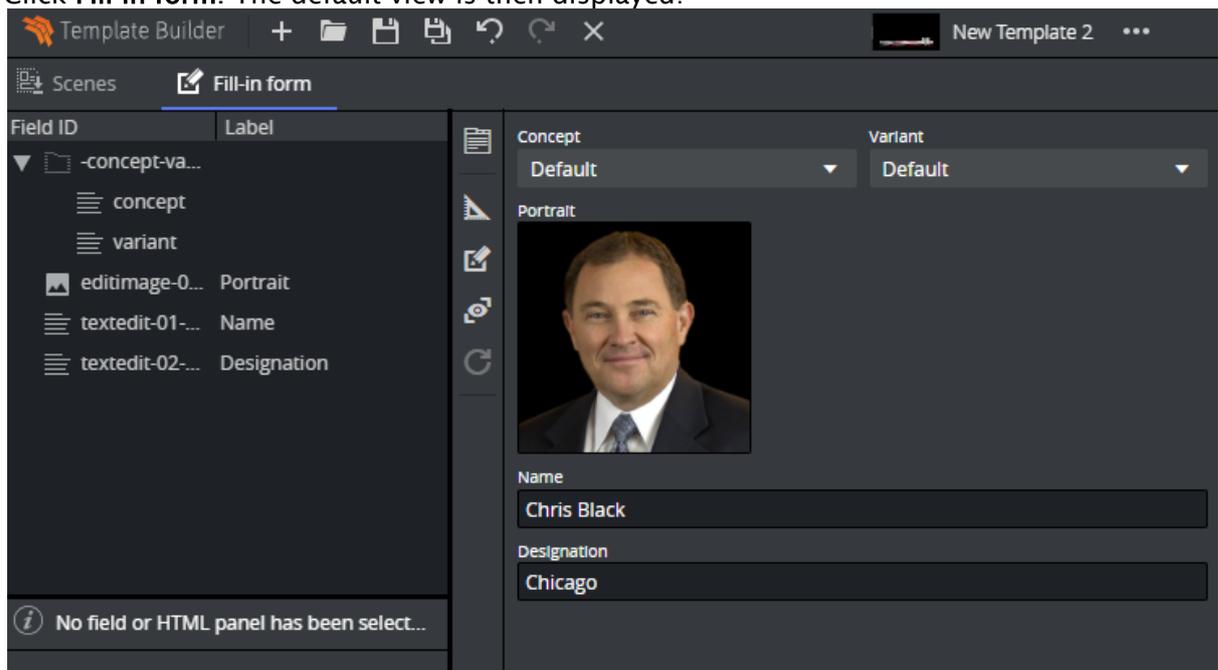
The left window will contain all the fields in this template. These fields are auto-generated based on the exposed control plugins in the scenes on which this template is based.
The middle view contains the fill-in form(s) for this template. The toolbar has the following functions:

Add a new Tab (fill-in form). All tabs are visible in Viz Pilot Edge.

Enter layout edit mode. Only enabled in additional tabs, not enabled for the default auto-generated **All** form. Click this to be able to move and resize fields.

[icon] Allow temporary editing of read-only fields. This is only when working with the template in Template Builder, it is not stored anywhere.

[icon] Reveal hidden fields. In the auto generated **All** mode, also some system fields are revealed, for instance the title, the auto generated title, the resolved concept and variant.
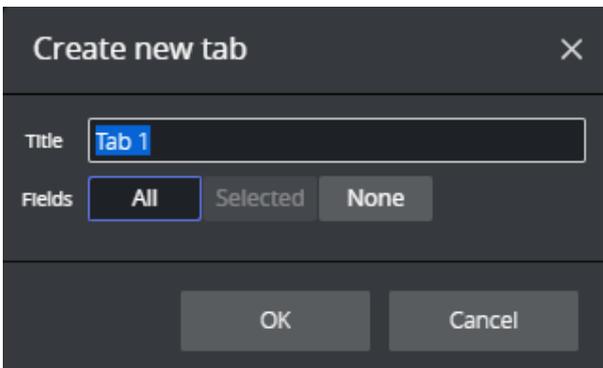
[icon] Refresh HTML panel(s) in the template or the full HTML panel if the template is represented by a custom HTML panel.

4.2.2   Adding Alternative Layout Forms

To create an additional template representation, click the **Create New Tab** button [icon]:

Enter a new title for your new tab and click **OK**:

There are 3 ways of adding fields to the new form by selecting one of the following options:

1. The **All** option in the dialog box, then all fields in the auto-generated form are added to the new form.
2. The **Selected** option will include only selected fields in the new form.
3. **None** will create a new, empty form.

Adding, Moving, and Resizing Fields

1. Click an additional tab (not the **All** tab).

2. Use drag and drop from the field view to **add fields** to the fill-in form:
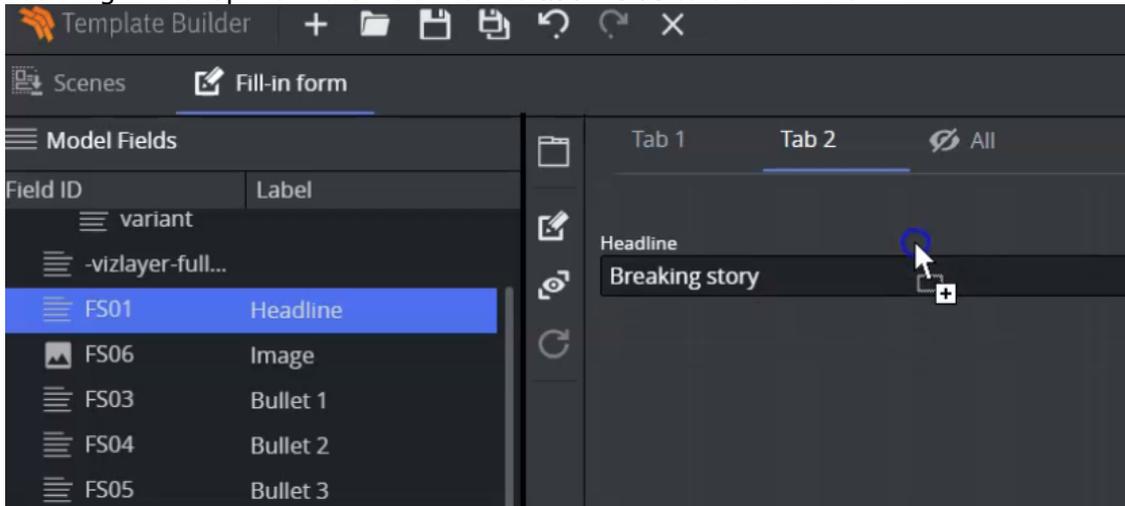
3. Move and resize fields by clicking the ruler button ![ruler] to enter edit layout mode, and then grabbing the edges of each field:
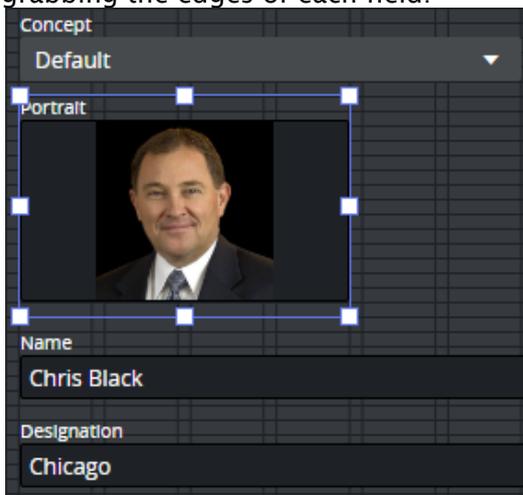
Renaming, Deleting and Reordering Tabs

Right click an additional tab (not the auto-generated **All** tab) to reveal the following functionality:
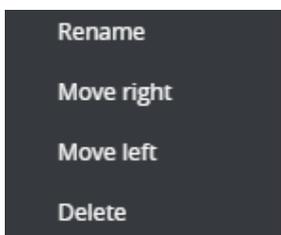
Now you can rename, reorder or delete each tab.

Hiding and Showing the Auto-Generated Tab

You can hide the auto-generated **All** tab from the Viz Pilot Edge user:

- Open the **Advanced** dialog by clicking the breadcrumbs on the top toolbar:

- Select **Hide generated form** and click **Close**:
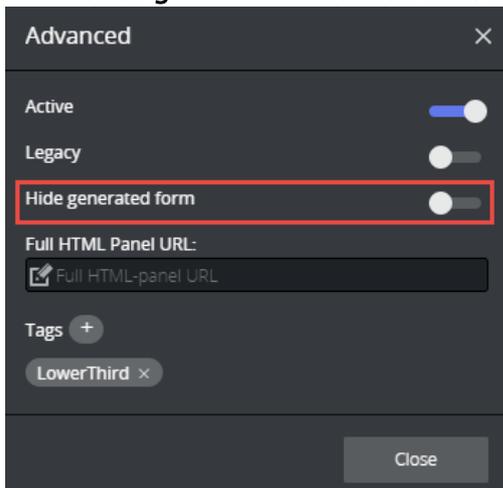
4.3 Template Fields

The left window in Template Builder contains the field tree. The initial fields reflects the exposed Control plugins in the imported scene(s), but it is possible to add fields that are not bound to plugins in the scene. Each field has a type, and numerous properties that alters the behavior of the field.

Fields can be restricted: for example, to only include text with a certain amount of characters, numbers within a specific range, or media placeholders for media assets, or be displayed as options in a drop-down list.

The fields can be manipulated in various ways to decide how data is entered into the field. See Data Entry.

Field representation in the UI can also be replaced with an inline HTML panel.

- Field Tree
    - Sub Fields
    - Text Fields
- Field Properties
    - Image Constraints
    - Default Search Parameters
- Field Types

4.3.1 Field Tree

The Field Tree contains the **Field ID** and **Label**, which are also shown in the Fill-in form. The icon beside each line in the tree indicates the Type of content in the field.

Fields can be rearranged by drag-and-drop within the Field tree. Right-click a field to open a menu where additional fields and HTML panels can be added.
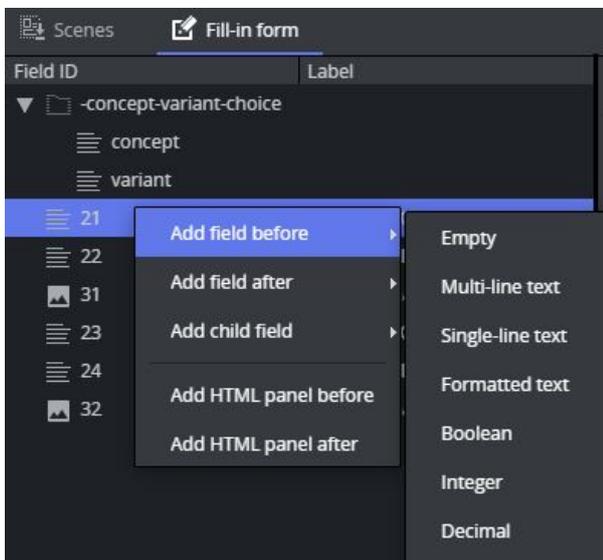
> ⓘ **Info:** Only fields created in Template Builder can be deleted and given a new ID. Fields bound to the scene have a fixed name and cannot be deleted.

Sub Fields

In the Field tree we can find the main fields and the sub fields:
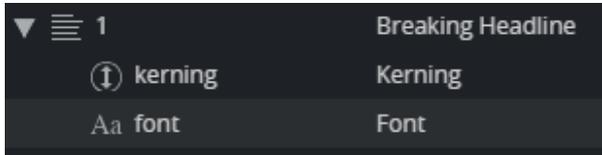
In this case, sub fields are Control Plugin properties of the main field. In Viz Artist, the scene designer has chosen to expose kerning and font for the main text field:
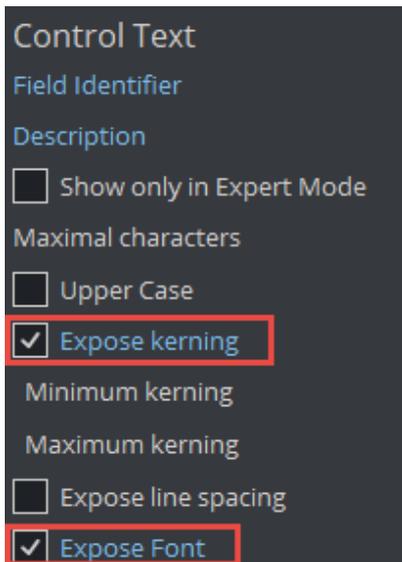
Sub-fields can also be used in Viz Artist and Template Builder to group fields. This is important when using Feed Browser functionality to bind multiple child fields to the parent field:
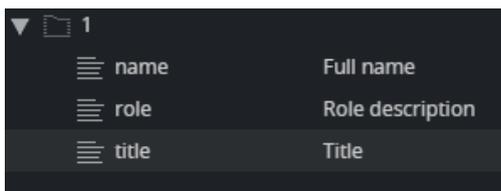
In Viz Artist, the three Control Plugins above are named **1.name**, **1.role** and **1.title** in the Viz Artist scene. When using the dot naming notation in Viz Artist, Graphic Hub and Template Builder will group these fields as parent/children.

Text Fields
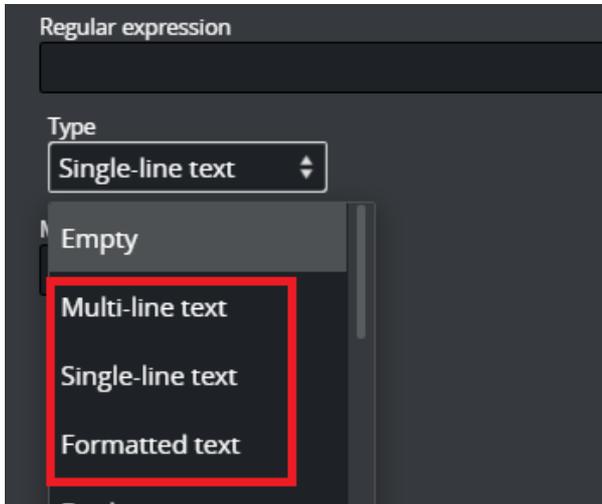
- **Multi-line text:** Multi-line text supports standard ASCII characters. It does not support any type of text formatting and does not convert any text. It keeps the text as it was typed.
- **Single-line text:** A single-line text field converts any white-space to space. White-space includes space, tab, newline, etc. Single-line text converts any white-space to the space character you get by pressing the spacebar KEY only. For Template Builder, this is also a text field with a single-line entry, unlike multi-line text.
- **Formatted text:** Formatted text refers to the ability to hold formatted text. For example, a formatted text field can show that some of the texts are bold or italic, for example, when a field has Rich-Text functionality.
  Although such a display is not yet completely supported (no Rich-text support yet) on our payload text field, formatted text is used so that if a field has a formatted type text created in Viz Artist, the field type can also be selected in Template Builder.

4.3.2   Field Properties

The **Field Properties** window is located below the Field Tree window. It displays the properties of a selected fields in the Field Tree.

Multi-selection: If several fields are selected in the Field Tree (CTRL + click), a subset of the field properties is displayed. If the selected fields have different field property values, the Field Properties window displays a multiple values state. Changes made in the Field Properties window are immediately applied to all the selected fields.

Note that the set of properties displayed depends on the type of the field. The following properties are available:

- **Label:** Specifies the label of the field in the Fill-In Form.
- **Tip:** A tooltip text can be entered to provide more information about the field.
- **Read-only:** The field remains visible, but is grayed out in the Fill-In Form.
- **Hidden:** Hides the field in the Fill-In Form.

- **Publishing variable:** Viz Story specific property. Can be used to link the field to a system field affecting the playout or publishing of the template.
- **Regular expression:** Defines constraints for the value in the field, using **Regex**. See the table below for examples.
- **Preview point link:** If set, clicking on/selecting the field will also show the given preview point in the Preview.
- **Type:** The field type. might be changed here.
- **Max length:** Text fields only, see comments in table below.
- **Single-line:** Formatted text fields only, see comments in table below.
- **Data entry:** Set how data is entered into the field. Drop-down list of all field types. For more information, see Data Entry.
- **Read-only** and **Hidden expression:** Basic Javascript eval expression that decides whether a field should be hidden or read only. This can be used instead of scripting to build conditions based on values in other fields. See the table below for examples.

ⓘ **Info:** A Regular Expression (or **Regex**) is a pattern (or filter) that describes a set of strings that match the pattern. In other words, a regex accepts a certain set of strings and rejects the rest.

Regex	Description	Example
[a-zA-Z]+	Match a word containing only letters.	MyLongWord
^[A-Z][a-z\s]*$	Match a string starting with capital letter containing only lowercase letters and space.	This is a sentence
\d+	Match a sequence of digits.	123489
\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\b	Match a typical email address.	joe@microsoft.com

---

ⓘ **Info: Read-only** and **Hidden expressions** support javascript notation to evaluate an expression. It supports field lookup, arithmetic and logical operators.
- Field references must be enclosed in double curly brackets, for example {{field01}}.
- Text fields in the expression must be of type Single Line.

Expression	Description
{{L301}}=="HIDE" || {{selector}}<2	Match if the **L301** field value is "no-logo" and the numeric field **selector** is less than 2.
({{V01}} / {{V02}}) *100 < 50	Match if the numeric field **V01** is less than 50% of **V02**.

Image Constraints

For fields of type **Image**, it is possible to set image constraints to force the Viz Pilot Edge user to select an image having the correct aspect or a minimum resolution.

Specify standards for image quality and size using either or both of the following:

- **Minimum size of the image (pixels):** Number of pixels (width or height), irrespective of aspect ratio.
- **Aspect ratio:** Any number describing the proportional relationship between the image with and height.
- **Allowed error**: Specifies a margin for the constraints when an image is selected.

Default Search Parameters

For the types **Image**, **Video**, **Font**, **Geometry** and **Material**, it is possible to define default search parameters that will be used by the media search that is launched when you click on the field:

1. Click the **Set** button to open a media search window.
2. Enter text in the search field, selecting whether to show all items or to limit by time from the **Show** drop-down list, and/or selecting a tag from the **Tags** drop-down.
3. Save by clicking **Use current search parameters**.
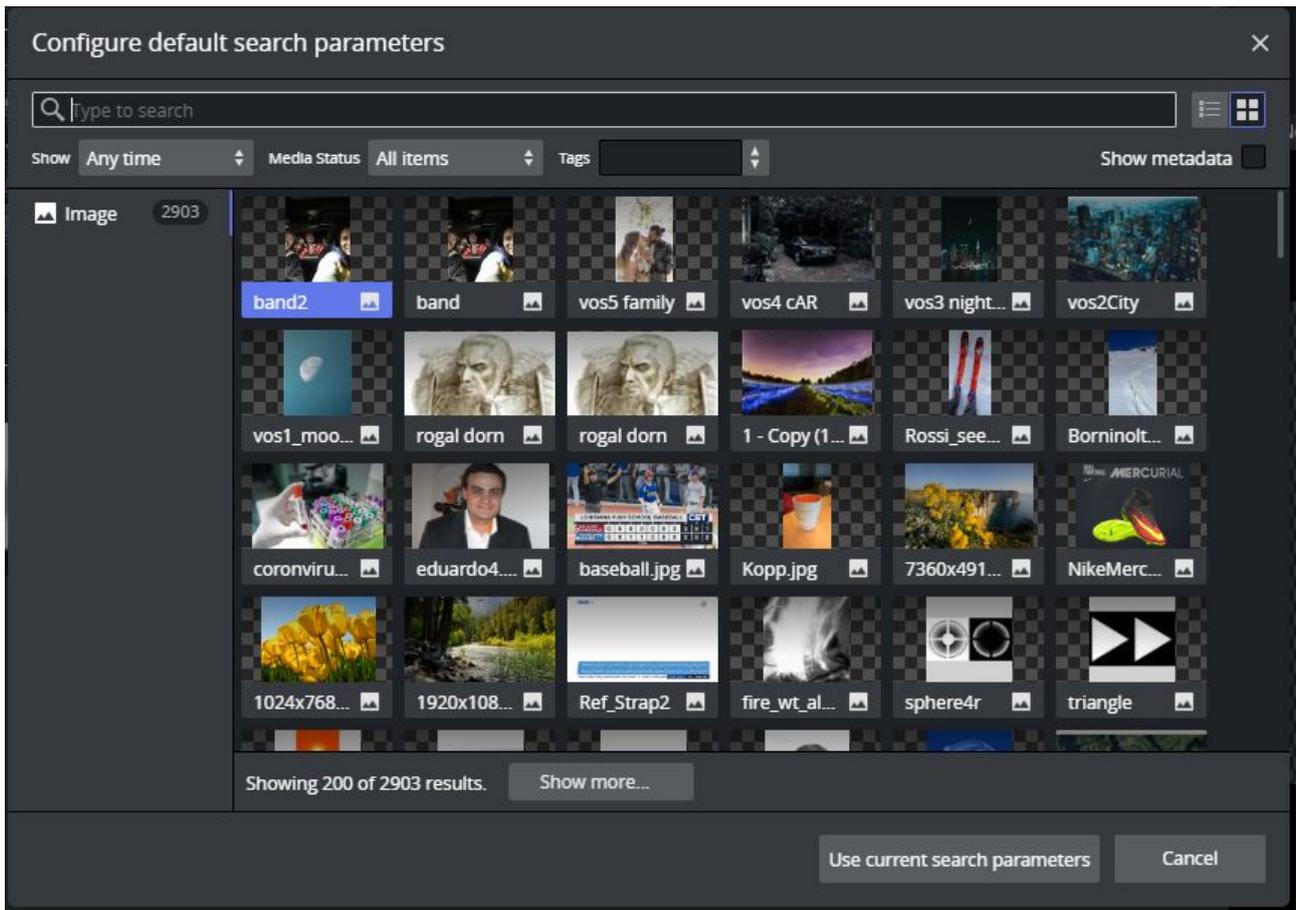
4.3.3 Field Types

The type of content allowed in the field in **Default Values** is set by using the drop-down list under **Type**. Depending on the type selected, different sub-options become available, as specified in the table below.

There are two main field type categories: *scalar* and *list*. Fields of all types apart from the list type are referred to as *scalar fields*. Fields using the list type are referred to as list fields.

The following types are available:

Type	Icon	Media Type (XSD Type)*	Comments
Empty			Makes the field unavailable in the Fill-in form. Typically used as a container for other fields.
Multi-line text		text/plain (string)	**Max length**: Sets the maximum number of characters allowed in the field.

| Type	Icon	Media Type (XSD Type)*	Comments
Single-line text		text/plain (normalizedString)	**Max length**: Sets the maximum number of characters allowed in the field.
Formatted text		application/ vnd.vizrt.richtext+xml	**Max length**: Sets the maximum number of characters allowed in the field. **Single-line**: Check this box to specify that the rich-text editor allows one line of text only.
Boolean		text/plain (boolean)	Creates a checkbox that has two states: true and false.
Integer		text/plain (integer)	This field is an integer field. **Minimum**: Sets the minimum value allowed in the field. **Maximum**: Sets the maximum value allowed in the field.
Decimal		text/plain (decimal)	This field allows decimal numbers. **Minimum**: Sets the minimum value allowed in the field. **Maximum**: Sets the maximum value allowed in the field.
Date and time		text/plain (dateTime)	Use the **Date Chooser** in **Default Values** to select date and time in this field.
Date		text/plain (date)	Use the **Date Chooser**, or the individual editors for day, month and year in **Default Values**, to select the date in this field.
Two numbers (duplet)		application/ vnd.vizrt.duplet	This field allows two numbers (decimal numbers are allowed). **Minimum**: Sets the minimum value allowed for both numbers. **Maximum**: Sets the maximum value allowed for both numbers.
Three numbers (triplet)		application/ vnd.vizrt.triplet	This field allows three numbers (decimal numbers are allowed). **Minimum**: Sets the minimum value allowed for all three numbers. **Maximum**: Sets the maximum value allowed for all three numbers.

Type	Icon	Media Type (XSD Type)*	Comments
Image		application/ atom+xml; type=entry;media=image	Makes the field available for an image. **Image Constraints**: Enable this option if you want to set constraints on the image. **Minimum Size of the image (pixels)**: Specifies the minimum allowed image dimensions in pixels. Both width and height must be at least this big. **Aspect Ratio (width x height)**: Specifies the aspect ratio of the image. **Allowed Error (%)**: Specifies the maximum stretch limit for both the width and height of the image, in relation to its defined aspect ratio.
Video		application/ atom+xml; type=entry;media=video	Makes the field available for a video.
Font		application/ vnd.vizrt.viz.font	Makes the field available for a font.
Geometry		application/ vnd.vizrt.viz.geom	Makes the field available for a geometry.
Material		application/ vnd.vizrt.viz.material	Makes the field available for a material.
Map		application/ vnd.vizrt.curious.map	Makes the field available to present and edit a map.
Custom			Lets you freely specify the media and XSD type.

Type	Icon	Media Type (XSD Type)*	Comments
List			Lists may be modified by adding and removing columns in the Field Tree. · To add columns to a list - right-click the columns node under the list field node in the Field Tree and select **Add column**. · To remove a column - select the column field in the Field Tree and press **Delete**, or right-click it and select **Delete field**. ⚠ **Note:** List fields are fundamentally different from scalar fields. It's therefore not possible to change a list type to a scalar type and vice versa. **Minimum number of rows**: Defines the minimum allowed number of rows in the list. **Maximum number of rows**: Defines the maximum allowed number of rows in the list.
Color		text/ vnd.vizrt.color	Text (for example: #140E7E or rgba(255, 0, 0, 1)).

* For more information on media types, see: Overview of Media Types.

4.3.4   Data Entry

The Data Entry field property specifies how users should fill in field values.
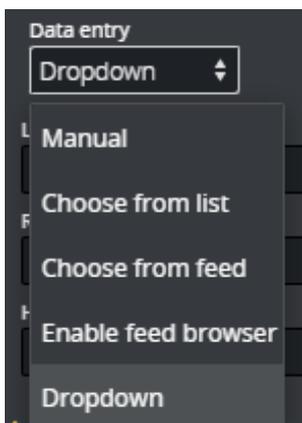

· Manual
· Choose from List

Manual

Selecting **Manual** in the Data entry drop-down list does not give access to any additional settings for the field. The default is for the input to the field to be a text box with manual input.

Choose from List

Selecting **Choose from list** lets you see the content in a drop-down list, which may in some cases make it easier and less error-prone to fill the template in with the right content.

For example, when a Control Object moving (Omo) plugin is accessible in the template, scenes using Omo plugins are originally presented as integer values for the different elements in the Fill In Form. The **Choose from list** option can assign text to these values to make it easier to select the right element.

The example below contains a scene with a sponsor logo having different display options in the graphics. For the Omo plugin, these options correspond to the values 0, 1 and 2 respectively. To assign text to these values:
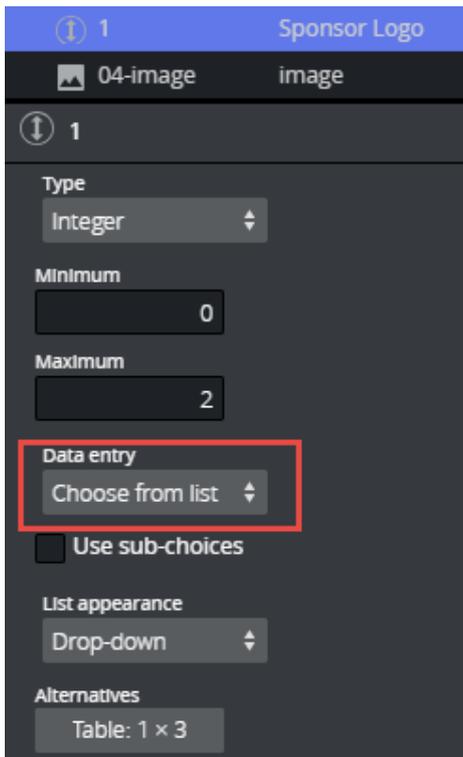
- Select the **Omo** field in the Field Tree.
- Select **Choose from list** in the **Data entry** drop-down list.
- Click **Alternatives**. A new window appears.
- Click a row or press F2 to do inline edits.
- Insert **Name**, **Tip**, and **Value**.
- **Right-click** to insert or remove rows.

- Close the editor.
- The **Omo** field in the Fill In Form now contains a drop-down list or radio buttons (depending on the **List appearance** option), containing the alternatives created above as text, as opposed to an integer field where the user would have to remember which integer corresponds to which position.
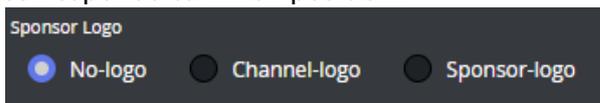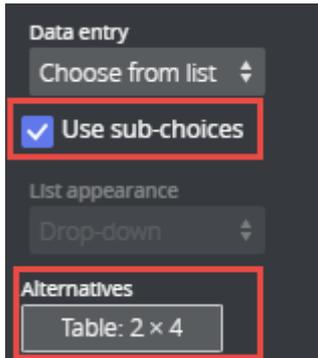
Sub-Choices

If you select the **Choose from list** option, a checkbox called **Use sub-choices** becomes available, letting you set multiple sub-choices for each choice.

For example, if the choices list different companies, sub-choices could list persons in each company.

- Mark the desired Field ID in the Field Tree.
- Select **Choose from list** and check the **Use sub-choices** check box.

- When clicking on the **Alternatives** button, a new window appears. Right-click to insert or remove rows.
- Double-click the table to insert **Name**, **Tip** and **Sub-choice Label**, in this example **Person:**

- Double-click the **Sub Alternatives** field in a row to add the sub-alternatives.

- A new table for sub alternatives appear. Fill the table and exit when complete.
- Instead of a text field in the Fill In Form, the field now contains two drop-down lists: the main choices, which in this case is a list of countries, and sub-choices, with
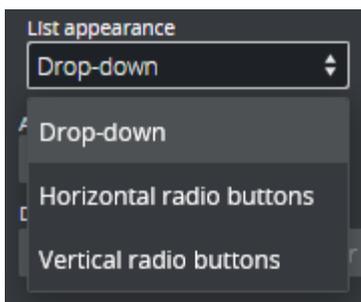
corresponding cities.

List Appearance

When **Choose from list** is selected in the data entry field, a **List appearance** option is shown:
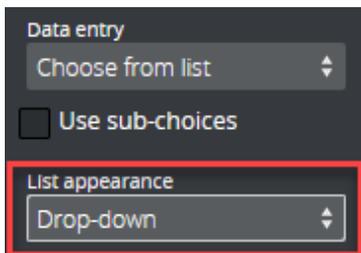
This options allows you to select the radio button appearance for single level choice fields:

Enable Feed Browser

This option specifies that the field should get its value from a property of an Atom or RSS feed entry. If the field is a sub-field of another field that has enabled feed browser, the option is named *Parent feed browser*. Otherwise, it is named *Enable feed browser*.

· If the Enable feed browser option is selected, a **Browse** button appears next to the field in the fill-in form.
· Click **Browse** to open the **Feed Browser** dialog.
· In the Feed Browser, the items of the feed are presented (with thumbnails, if available), and one of the entries can be selected.
· Information from the selected item is used to fill in the feed browser enabling field and its subfields.

⚠ **Note:** In order to be able to fill in multiple fields from a single selection in the feed browser, fields must be subfields of the field that enables the feed browser.

Feed URL

Specify the feed URL for the field. The URL must be accessible from the Viz Pilot Edge browser and lead to a valid Atom XML or RSS feed:

⚠ **Note:** Internet servers can have strict CORS policies denying access to their feed from within Viz Pilot Edge.

Select from Feed Item

This option binds the element of a feed item (title, link, content etc.) to the value of the current field. This is a 1:1 relation between the feed item and the field value. Though, it is fully possible to bind a feed item to multiple fields in the template. For example, if you select a story from a feed and title, content, author and image is applied to the template. To accomplish this, the fields in the template need to be grouped under a parent field. See Sub Fields on more information on this.

⚠ **Note:** The options available for a given field depend on the type of the field (the atom namespace prefix represents the http://www.w3.org/2005/Atom namespace, and the media namespace represents the http://search.yahoo.com/mrss/ namespace).

- · **<Not linked>:** Not linked to the feed item, and must be filled in manually.
- · **Content:** Linked to the content of the *atom:content* element in the atom entry.

- **Title:** Linked to the content of the *atom:title* element in the atom entry.
- **Link:** Linked to the *href* attribute of the *atom:link* element in the atom entry. Which link entry to pick depends on the *Link-rel in atom entry* property and the type of the field - the first link with a correct rel attribute and a type that matches the type of the field is chosen.
- **Entry:** Linked to the atom entry itself.
- **Author name:** Linked to the content of the *atom:name* element inside the relevant *atom:author* element. If the entry itself contains an *atom:author* element, that is used. Otherwise the *atom:author* element of the feed is used.
- **Author e-mail:** Linked to the content of the *atom:email* element inside the relevant *atom:author* element. If the entry itself contains an *atom:author* element, that is used. Otherwise the *atom:author* element of the feed is used.
- **Author URI:** Linked to the content of the *atom:uri* element inside the relevant *atom:author* element. If the entry itself contains an *atom:author* element, that is used. Otherwise the *atom:author* element of the feed is used.
- **Contributor name:** Linked to the content of the *atom:name* element inside the *atom:contributor* element in the atom entry.
- **Published:** Linked to the content of the *atom:published* element in the atom entry.
- **Updated:** Linked to the content of the *atom:updated* element in the atom entry.
- **Thumbnail:** Linked to the *url* attribute of the *media:thumbnail* element in the atom entry.
- **Summary:** Linked to the content of the *atom:summary* element in the atom entry.
- **Link-rel in Atom Entry:** Only available if **Link** is selected in the Select from atom entry property; it specifies the `rel` attribute of the link element in the atom entry.

> ⚠️ **Note:** A linked field may also be filled in manually if it's not hidden or read-only.

RSS Mapping

The feedbrowser in TemplateBuilder works with Atom XML, but also supports a minimal mapping from standard RSS items according to the RSS 2.0 specification: https://www.rssboard.org/rss-specification.

The media namespace *xmlns:media="http://search.yahoo.com/mrss/* is specified in https://www.rssboard.org/media-rss.

RSS	TemplateBuilder	Comment
<title>	Title	
<description>	Summary	
<pubDate>	Updated, Published	RSS has only one field when the item was published.
<author>	Author e-mail	In RSS the <author> element is strictly specified as an e-mail address.

RSS	TemplateBuilder	Comment
<enclosure type="image/jpeg">	Thumbnail, Link rel="enclosure"	Both Thumbnail and Link with link relation "enclosure" will map to the <enclosure> element in RSS.
<media:content type="image/jpeg">	Thumbnail, Link rel="content"	Both Thumbnail and Link with link relation "content" will map to the <media:content> element in RSS.
<media:thumbnail>	Thumbnail	

Choose from Feed

The **Choose from feed** option works exactly like the Enable feed browser option, but displays the results in a dropdown instead of in the feed browser window:
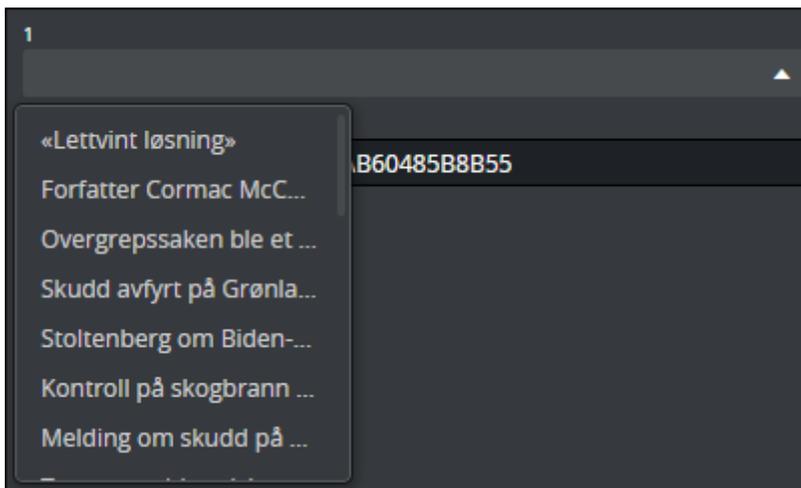
Dropdown

The **Dropdown** option allows you to create a dynamic dropdown with items read from the value of another field. Whenever the (hidden) source field is changed, the dropdown items are updated. See Dynamic Dropdown.

Dynamic Dropdown

The entries of a dropdown can be dynamically populated based on the value from another (hidden) field in the template.

· The dropdown is populated with a JSON string from the source field.
· Whenever the source field is changed, the dropdown is re-generated.

- Use case: A natural workflow is to use the `onLoad()` event to fetch data and populate the source field whenever data elements based on this template is opened in Viz Pilot Edge.

Example – onLoad() Populating a Dropdown

Follow these steps to create a dynamic, data driven dropdown that is populated each time a data element is opened:

1. Create a new single-line text field that may be called *source*. This acts as the source of the drop-down item (this field can be hidden).
2. Secondly, create another single-line text field, for instance called *dropdown*. This is where your drop-down will be displayed.
3. In the dropdown single-line text field, click on it to have the properties displayed.
4. In the **Data entry** property, click on the **Dropdown** alternative.
5. Once the alternative is added, a new text field right below called **Linked source field** will appear.
6. Fill the **Linked source field** with the name of your first single text field, which in this example is *source*.
7. Within the **source** field, you can add data, either manually through the source field directly, or from the script editor like shown below.
8. The data must be in JSON arrayformat: [{"label": "my label1", "value": "my value1"}, {"label": "my label2", "value": "my value2"}, {"label": "my label3", "value": "my value3"}]
   a. The **label** property is what is displayed in the drop-down.
   b. The **value** property is what is being stored in the data element (this is eventually sent to Viz Engine).

Populate Dropdown Source Field in onLoad() Event

```
 1   // onLoad() is executed each time a data element based on this
 2   // template is loaded in Pilot Edge
 3   vizrt.onLoad = () => {
 4       fetch("http://myhost/app/teams.json")
 5           .then(r => r.text())
 6           .then(result => {
 7               vizrt.fields.$source.value = result
 8               console.log("Fetched: ", result)
 9           }
10       ).catch(e => console.log("Error: ",e))
11   }
```

In the example above, the *teams.json* file looks like this:

```
[{"label": "Tottenham Hotspur", "value": "tottenham"}, {"label": "Liverpool",
"value": "liverpool"}, {"label": "Manchester City", "value": "mancity"}]
```

The format is a JSON array. This line can be pasted directly into the dropdown source field for testing purposes.

The result looks like this (with the source field visible):
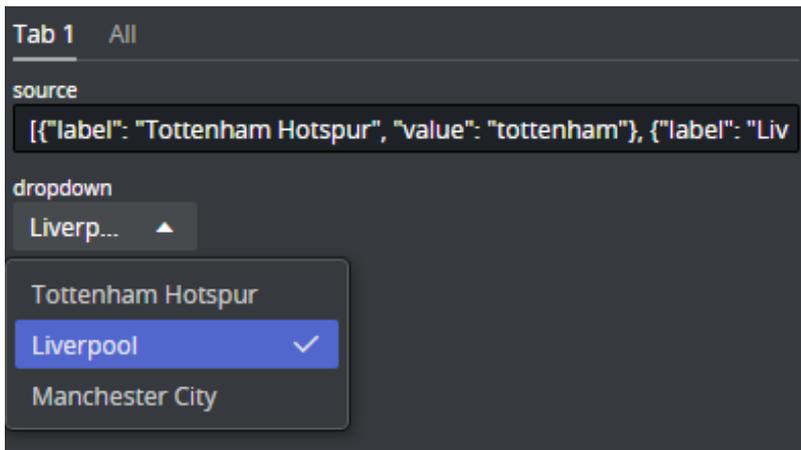
4.3.5   Inline HTML Panel

You can add an HTML panel to the template as part of the template customization workflow, giving you full control through custom scripting and logic when building the template.

> ⚠ **Note:** The template can host a web page if you enter a web address in Template Builder. The aim is usually not to host a single web page, but rather to make a data integration HTML template.

> ⚠ **Note:** Custom HTML Templates contains examples of how to use HTML templates.

> ⚠ **Note:** HTML panels should refer to their own Javascript libraries instead of Edge libraries. This is to prevent templates from breaking when Viz Pilot Edge is upgraded to a new version.
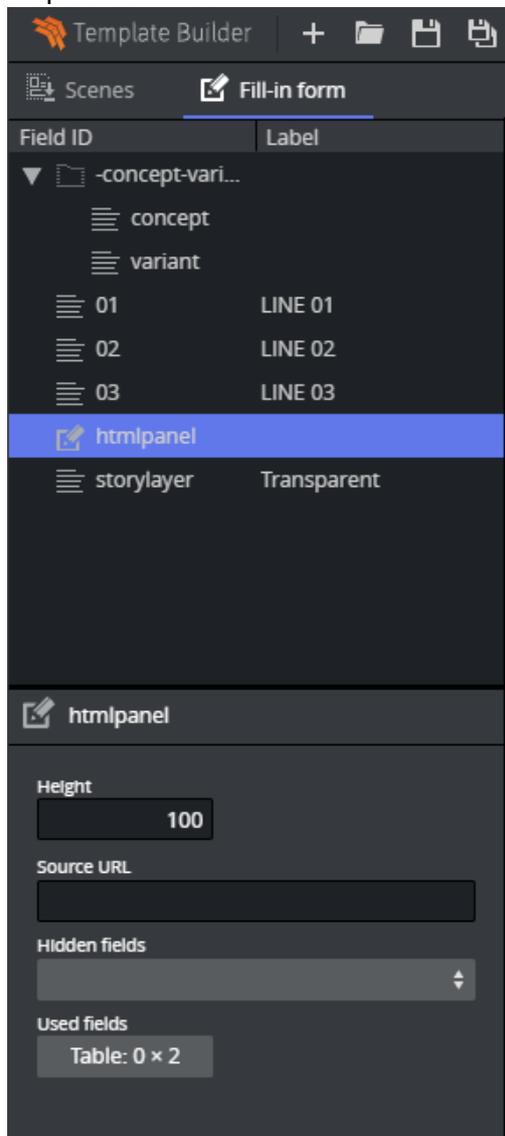
Adding an HTML Panel

· Right-click the field ID list and select **Add HTML panel before/after**:
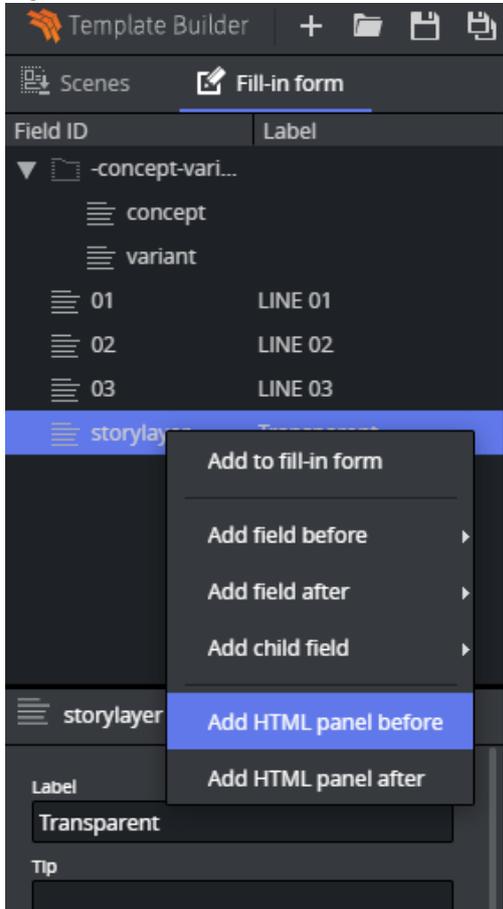
· Enter the ID of the of the new panel in the dialog that appears:
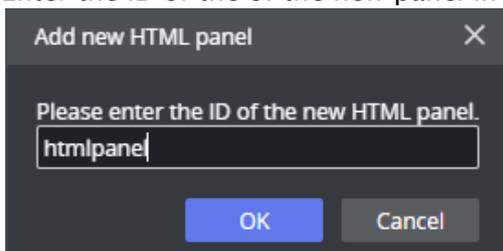
· Select the new **html-panel** field ID in the field tree to show its properties in the Field
  Properties window:

· Adjust the size of the HTML panel shown in the Fill In Form using the **Height** field.
· In **Source URL**, enter the web address.
· The **Hidden fields** drop-down list allows you to hide available fields in the Fill In Form.

Browser Caching

You may experience browser caching behavior when trying to update and display changes in the
custom HTML template in Template Builder, this is standard behavior. Template Builder does not
control caching resources included in the HTML file itself.

To prevent caching:

1. Ensure the URLs to the resources are unique upon reload.
2. Optionally configure the web server serving the resources to send Expiry headers set to 0.
3. Disable caching on the browser side.

> ⚠ **Note:** A detailed description of how caching works is unfortunately beyond the scope of this documentation.

4.4 Custom HTML Templates

A few examples of how to create HTML templates are shown below:

- Setting Up a Simple Custom HTML Template
- Full Screen for Custom HTML Panels
- Connecting a Custom HTML Template to a Viz Pilot Template
- Connecting a Custom HTML Template to a Viz Pilot Template - Advanced
- Creating a List of Functions Where You Can Bind Fields
- Redesigning Concept/Variant Fields
- Controlling the Auto-generated Fill In Form from the HTML Template

> ⚠ **Note:** To fully understand the workflow in these examples, some basic knowledge about web technology (javascript, HTML, CSS) is required. Although detailed description of the content in the files used will not be provided, API documentation bundled with the product describes the API to which a web developer creating custom HTML templates has access. This can be found at *http://<pilotdataserverhost>:8177/app/templatebuilder/js/doc/index.html.*

> ⚠ **Note:** jQuery is used in the examples for brevity, but is not mandatory when creating your own HTML template.

> ⚠ **Note:** The following examples are a proof of concept and show only some of what can be done using the customized workflow; more advanced use allows full control of the template using custom scripting and logic. More samples can be found at *http://<pilotdataserverhost>:8177/app/templatebuilder/samples/html_panels/README.html.*

- Use simple HTML <input> or <textarea> fields that contain an id="field_<fieldid>" that will automatically have a bi-directional connection.
- Take control of function mapping, and use JavaScript to do virtually whatever you wish.

> ⚠ **Note:** HTML panels should refer to their own Javascript libraries instead of Edge libraries. This is to prevent templates from breaking when Pilot Edge is upgraded to a new version.

4.4.1 Setting Up a Simple Custom HTML Template

The example below uses a template that shows the message **Hello world** when opened in a browser.

The following three files, including the HTML template, must be located in the same folder (*C: \Program Files\Vizrt\Pilot Data Server\app\mytemplates*):
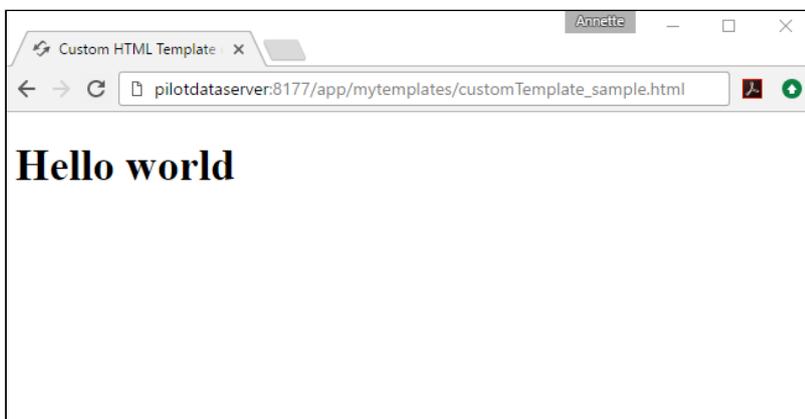
1. **customTemplate_sample.html:** The custom HTML template.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js">
</script>
<script type="text/javascript" src="./payloadhosting.js"></script>
<script type="module" src="./customTemplate_sample.js"></script>
<head></head>
<body>
    <h1>Hello world</h1>
</body>
```

2. **customTemplate_sample.js:** The JavaScript part of the template.

```
$(document).ready(function () {
    console.log("Hello world");
});
```

3. **payloadhosting.js:** This connects everything. (Get it from http://*<pilotdataserverhost>*:8177/ app/templatebuilder/js/payloadhosting.js).

The URL *http://<pilotdataserverhost>:8177/app/mytemplates/customTemplate_sample.html* in the image below points to the location of the *.html* file mentioned above:

Viewing a Custom HTML Template in Template Builder

· Open a template and add an HTML panel as described here.
· In the URL field, enter the URL of the custom HTML template. In this example, the URL from the picture above.
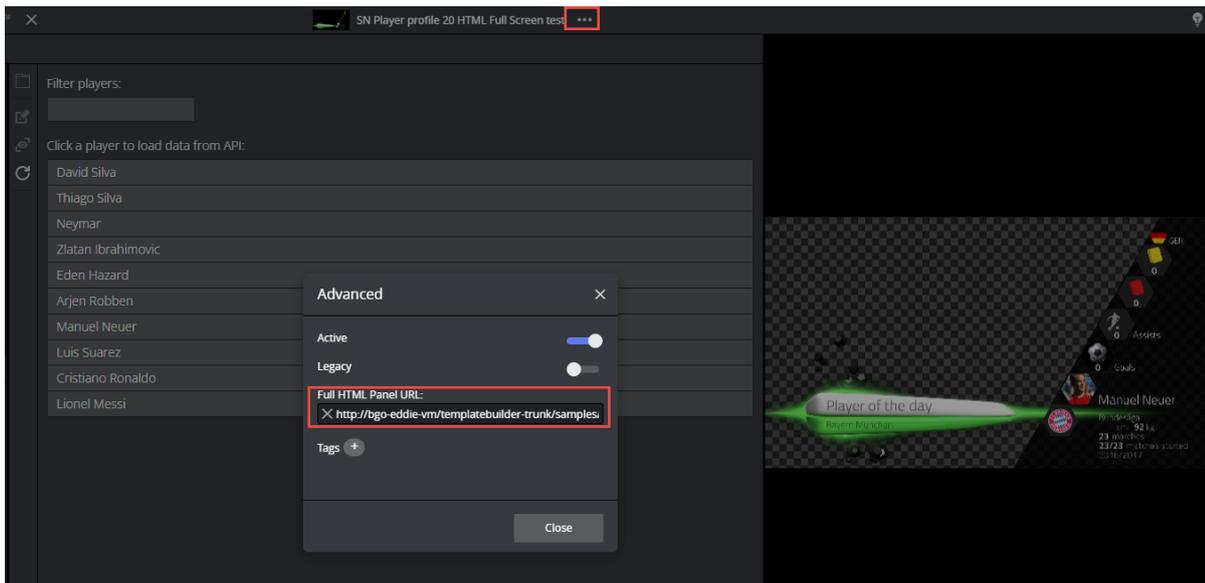
· *Hello world* now appears in the Fill In Form:
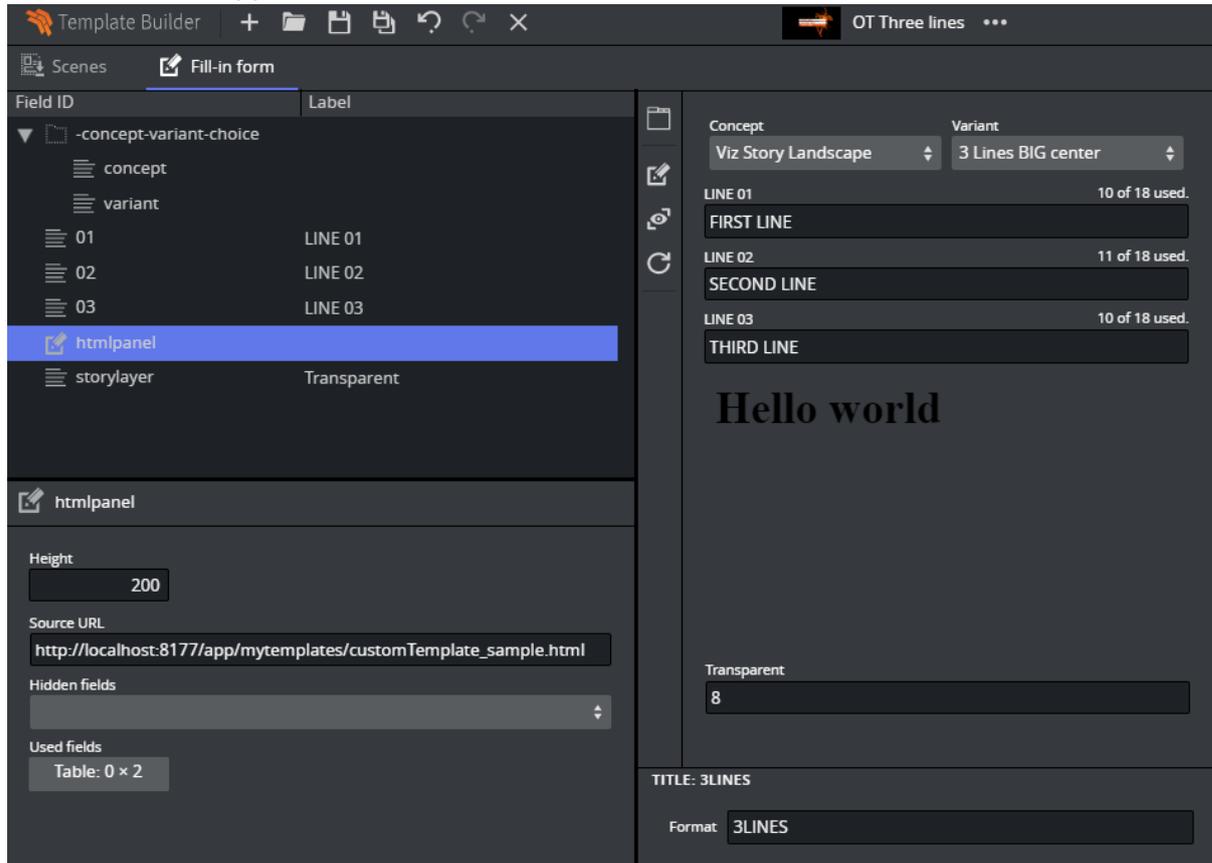
4.4.2   Full Screen for Custom HTML Panels

With full screen HTML, it is possible to replace the built-in template with a custom HTML representation that replaces the whole template.

The URL to the custom HTML page must be set in the **Advanced** menu:

Copy the URL that is in the html field of the template, paste it in the **Advanced** menu and click **Close**. The template must be saved in order to have the custom HTML open in full screen the next time.

4.4.3   Connecting a Custom HTML Template to a Viz Pilot Template

Following the example above, we can establish a two-way communication, or *bind fields*, between the HTML template and the opened pilot template. This provides a simple way of setting up a binding field. Add a new field to the template:

- Right-click in the HTML panel field, choose **Add field before/after** and select **Multi-line text.**
- Give it the ID **50**.

- A new field appears:
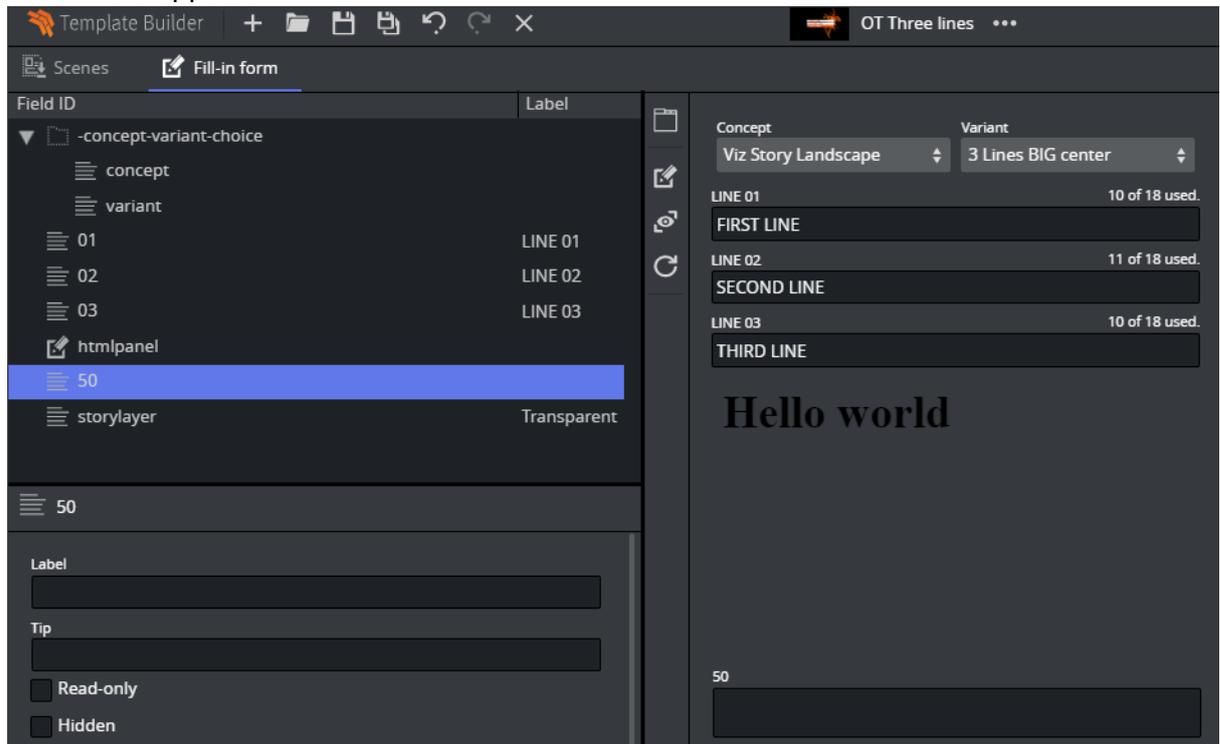
The <body> block in the custom HTML template (*Template_sample.html*) that previously contained:

```
<body>
    <h1>Hello world</h1>
</body>
```

must then be replaced with:

```
<body onload="vizrt.payloadhosting.initialize()">
    <label for="field_50">My input field:</label>
    <input name="field_50" type="text" id="field_50">
</body>
```

Saving the HTML file and clicking **Refresh HTML panels** reloads the custom HTML template with the changes just made. A bi-directional connection between the custom template and pilot template has now been established. If you now type inside either the template or the field with ID 50, both fields are updated at the same time.

> ⚠ **Note:** This way of binding fields works for any HTML fields that have value support, typically `<input>` types and `<textarea>` .
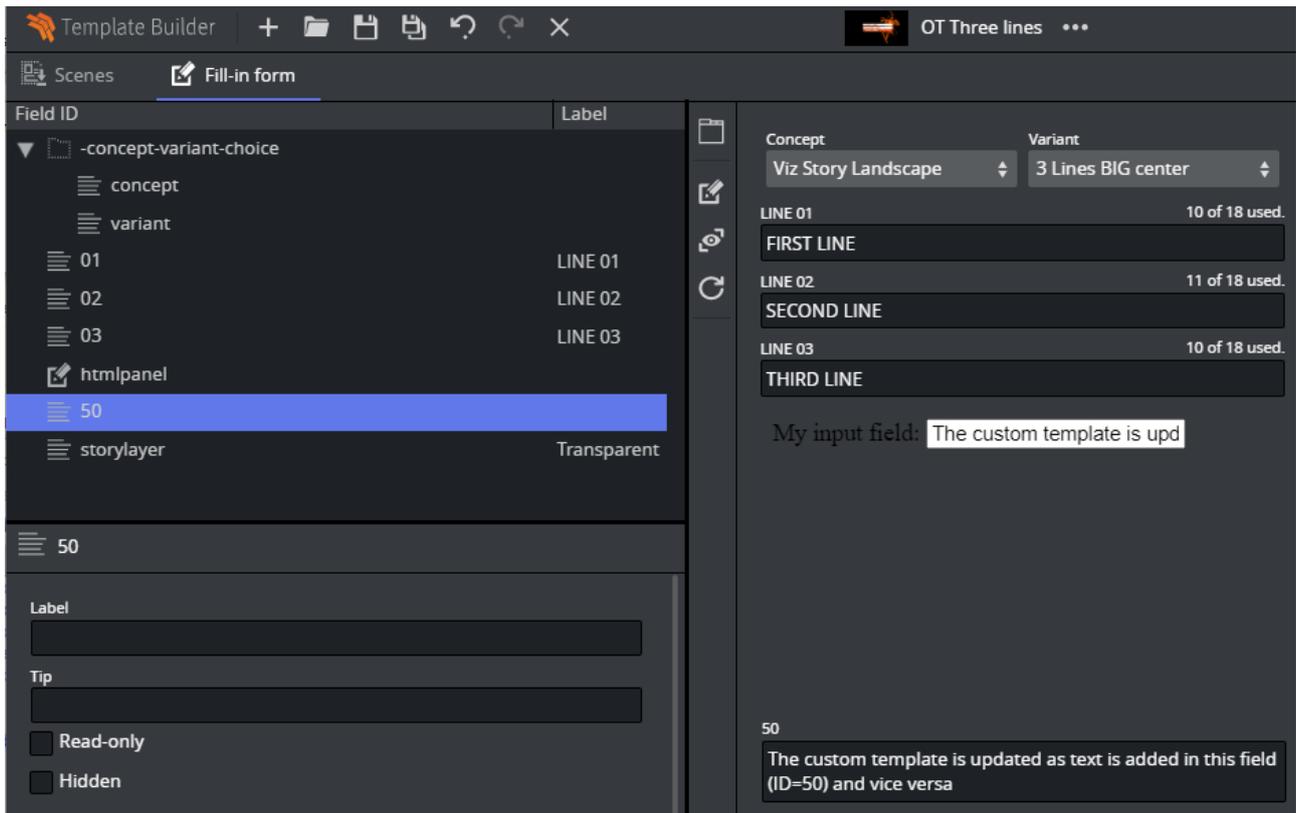
The JavaScript file automatically seeks input elements in the HTML that match the ID of fields inside the template. Adding the `id="field_50"` to the <input> element inside the HTML template is all that is needed for the two-way communication to be set up since a field with ID 50 was added above. An unlimited number of these binding fields can be established in the exact same way, since they are mapped via ID.

> ⚠ **Note:** Updating the `<input>` elements programmatically still sends data back and forth, which is useful for automated data integration such as fetching live sports data.

> ✅ **Tip:** Use the **Hidden fields** setting inside the HTML panel settings to prevent two editors for the same field being visible at the same time.

4.4.4 Connecting a Custom HTML Template to a Viz Pilot Template – Advanced

The following example will go more into detail than the example above, and use a more scripting to give you 100% control over the template. The three files mentioned in the Setup a simple custom HTML template example are also used here.

4.4.5   Creating a List of Functions Where You Can Bind Fields

By adding the following above the *document.ready()* function in the *customTemplate_sample.js* file:

```
// Will be called when the field with id "50" changes
function on50Changed(value) {

}
```

And the following inside the $(document).ready function:

```
var pl = vizrt.payloadhosting;
pl.initialize();
pl.setFieldValueCallbacks({ "50": on50Changed });
```

You set up a way for a custom JavaScript function to be called upon detecting a change. When field_50 receives a change from the host, the function will be called with its new value as a parameter.

Some changes will be made to the HTML file below to demonstrate that we can use custom HTML/ JavaScript to do something with these values.

Inside the HTML file, the entire body is replaced with:

```
<body>
    <span id="myfield" class="sample red">My custom 50 field
</body>
```

To add some CSS to style the text, add the style tag after the closing </head> tag and before the <body> tag:

```
<style>
    .sample {
        padding:5px;
        color:white;
        border-radius:5px;
        text-shadow:0 1px 0 black;
        background:red;
    }
    .green {
        background:green;
    }
</style>
```

This provides the following output in Template Builder:
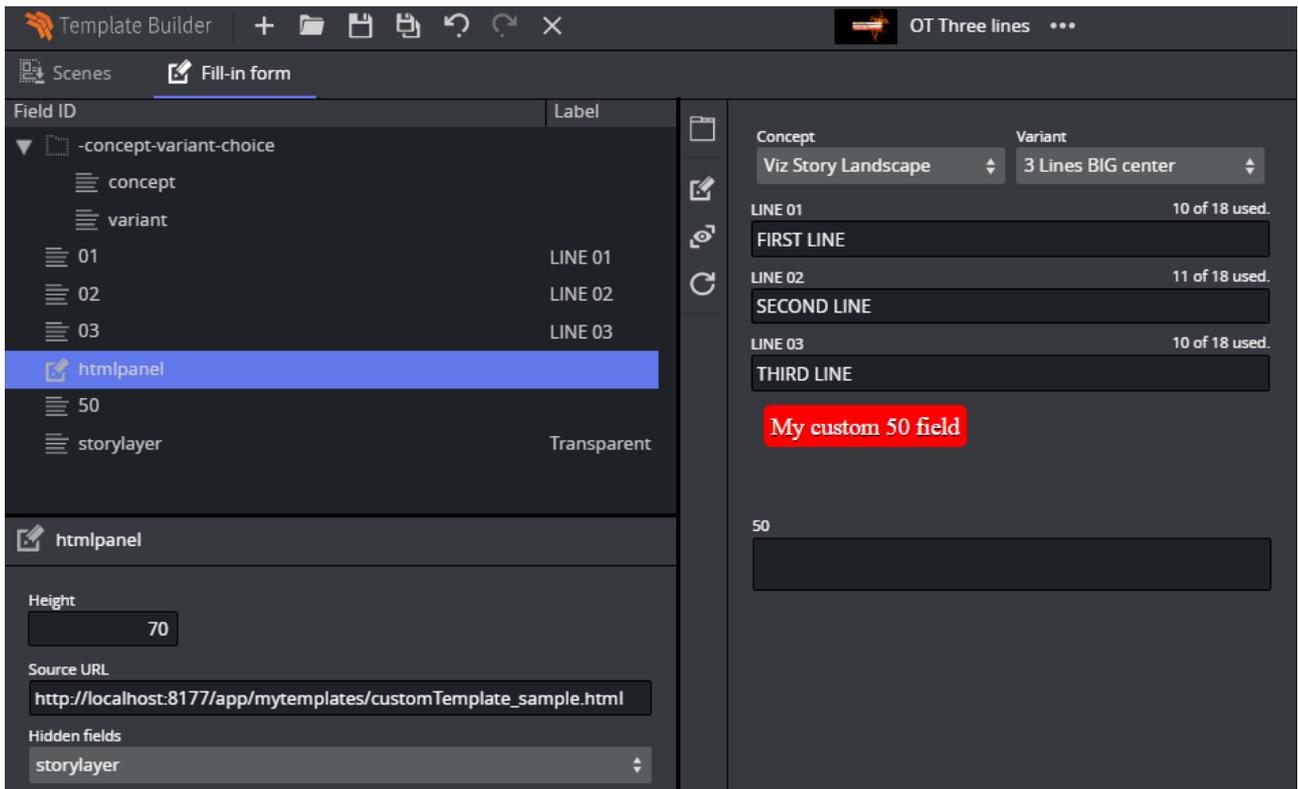
Adding a bit more custom logic, we will make the background color green when there is a text value that is longer than five or shorter than 20 characters. The function is expanded by adding the following function:

```
function on50Changed(value) {
    var myField = $("#myfield");
    myField.text(value);
    if (value.length > 5 && value.length < 20) {
        myField.addClass("green");
    } else {
        myField.removeClass("green");
    }
}
```

After refreshing the HTML panel, the background color should change to green dynamically when typing.
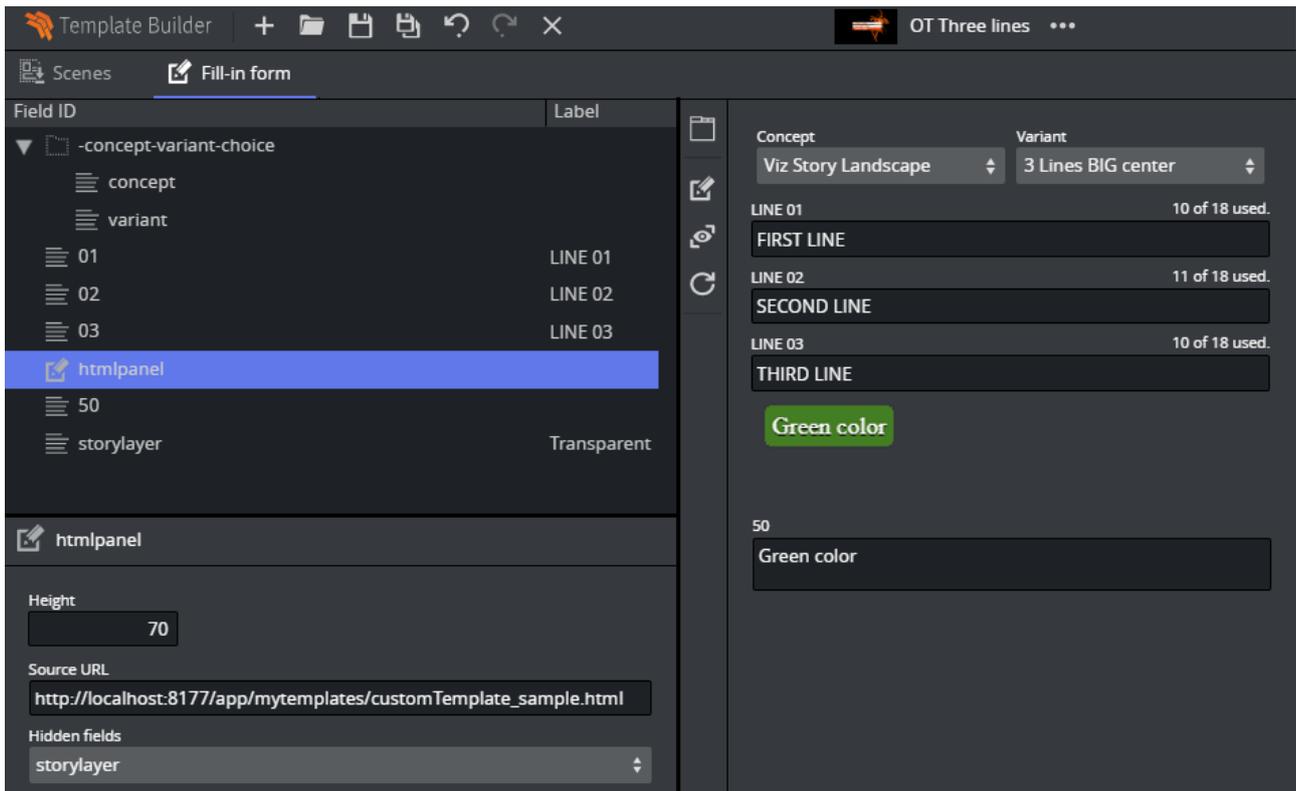
4.4.6 Redesigning Concept/Variant Fields

This example shows how to present the concepts and variants in a template in a different way.

The full HTML / JavaScript code is available at *http://<pilotdataserverhost>:8177/app/ templatebuilder/samples/html_panels/concept_variant.*

Let's consider a template with concepts **Fullscreen, Lower Third, OTS** and variants **Red, Green, Blue** available as drop-down lists in the Fill In Form:
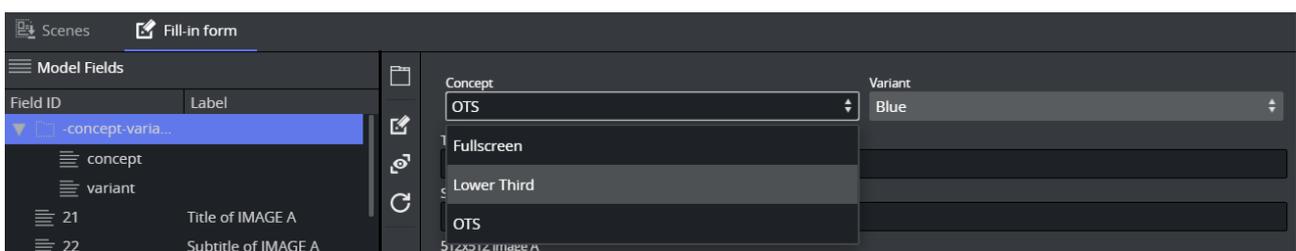

The field **-concept-variant-choice** actually contains 2 subfields, **concept** and **variant**.
You can access their value using slash "/" to navigate in the list. For example, to access the concept use **-concept-variant-choice/concept**.

By setting up the HTML panel hosted at *http://<pilotdataserverhost>:8177/app/templatebuilder/ samples/html_panels/concept_variant*, the concepts and variants are now presented as buttons. This example has mutual binding support for both concept and variant - clicking on the new buttons updates the original drop-downs and vice versa:
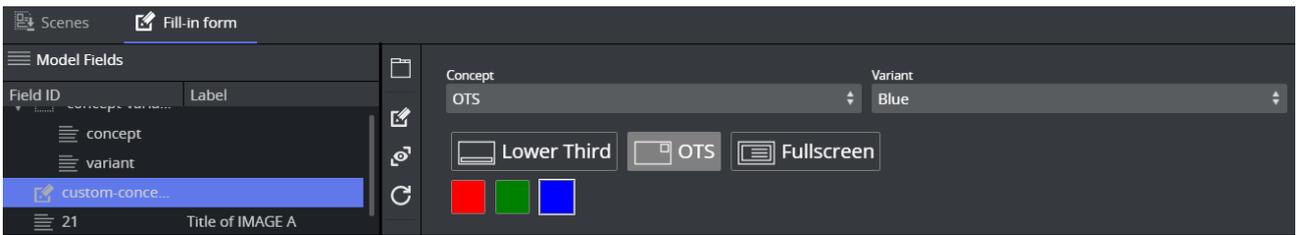
The drop-down lists are no longer needed and can be set as a **Hidden field** in the HTML panel properties window:
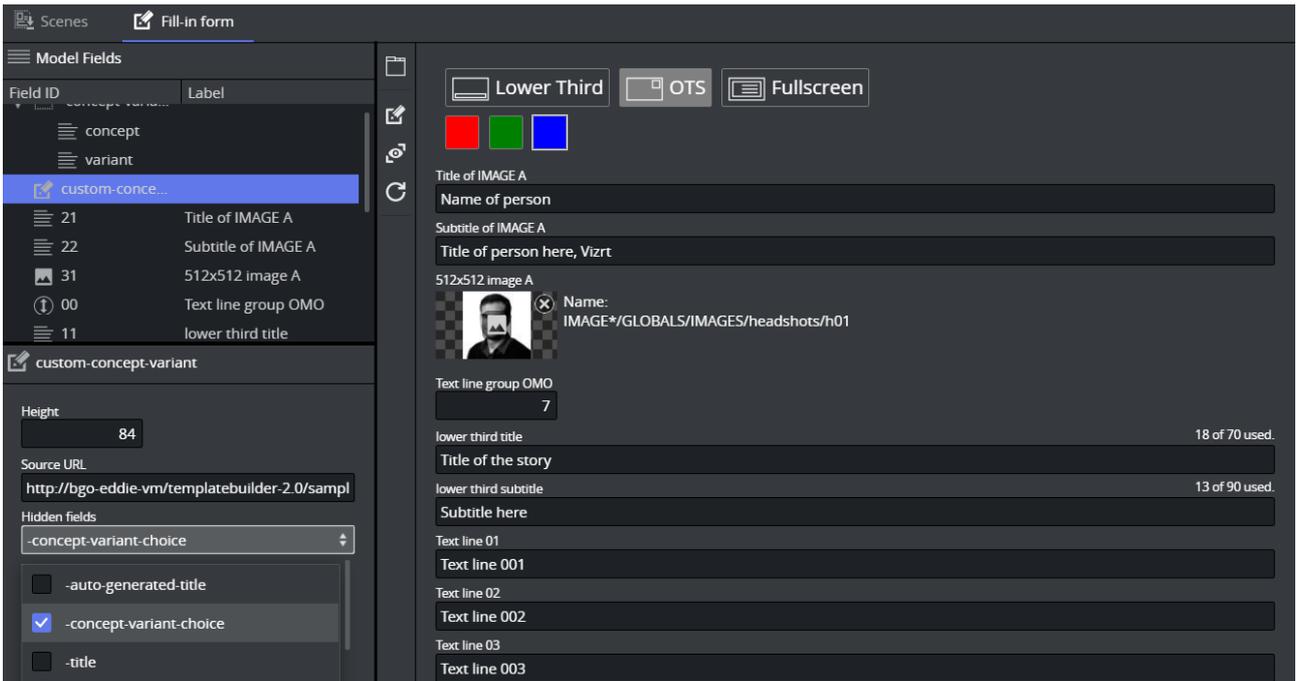
4.4.7   Controlling the Auto-generated Fill In Form from the HTML Template

It's possible to dynamically set visibility and read-only attributes, so you can filter the auto-generated form based on the custom HTML template.
In the following example, the **31** image field should only be visible when the **Fullscreen** or **OTS** concept is active:
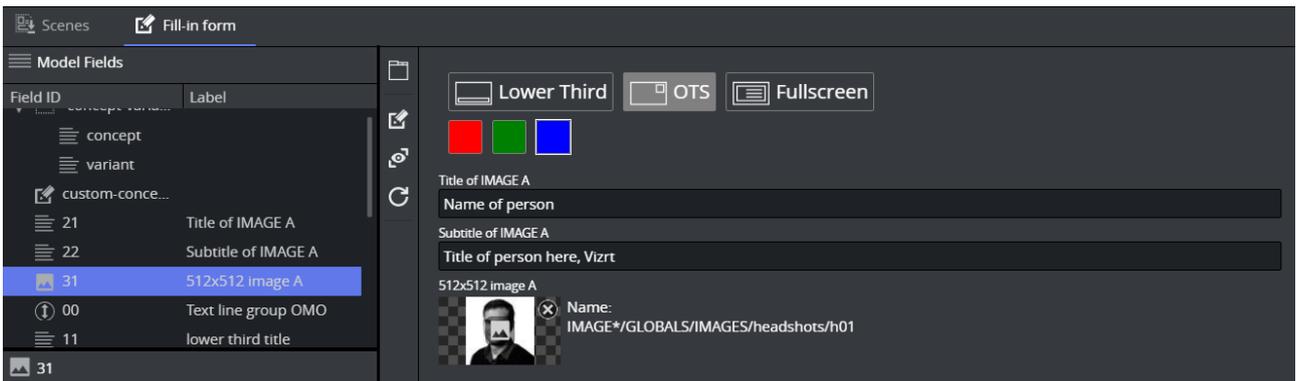
In the JavaScript used in the example, there is a function called *updateActiveConcept* which is called when the concept changes.

Adding the following line inside the updateActiveConcept method block, it checks which concept is chosen. If it isn't **Lower Third**, it displays the field with ID **31** in the Fill In Form:

```
pl.setFieldVisibility("31", conceptValue != "Lower Third");
```

If you now click on the **Lower Third**, the image field with ID **31** disappears, but is displayed if the **OTS** or **Fullscreen** concept is selected.

> ⚠ **Note**: This is a powerful feature that lets you customize available editing options based on certain conditions set in the template.

Sub Fields

Sub fields can be addressed by using a " / ", which is read as **field/subfield**.

When using duplets or triplets, the example below shows how you are able to control the value of *image_scaling* by using *1/image_scaling*.
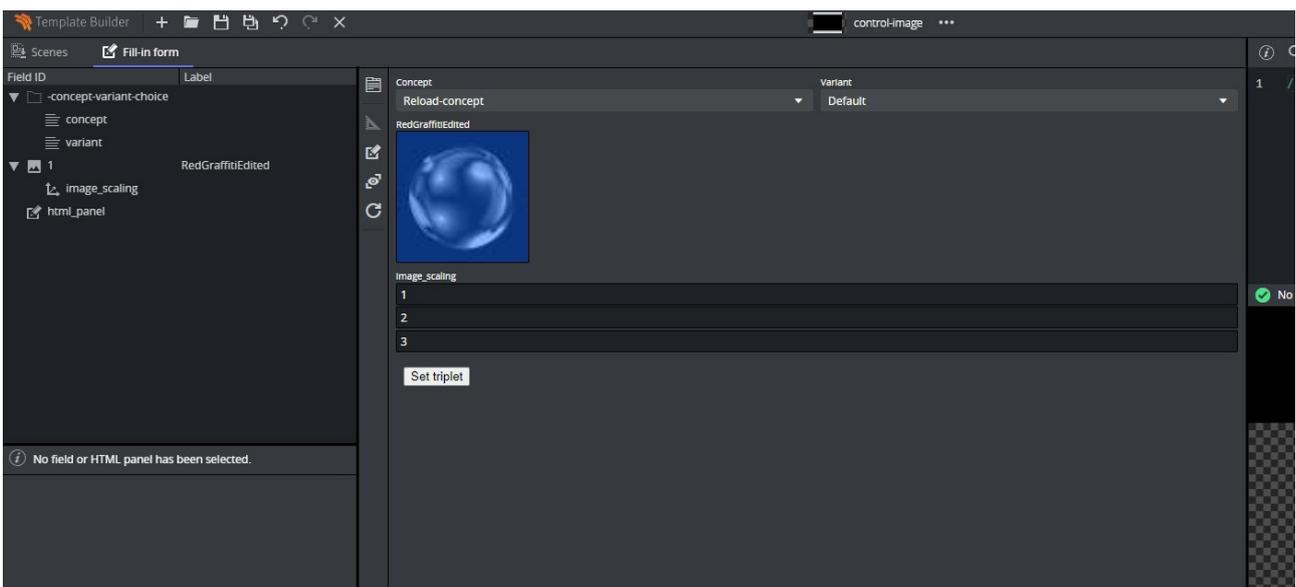
> ⓘ **Info:** It remains possible to use *setFieldText(field, value)* where **field** is *field/subfield* and **value** is *1 2 3*.
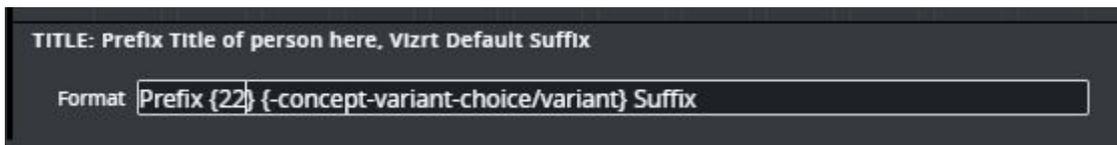
```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <script type="text/javascript" src="./payloadhosting.js"></script>
5      </head>
6      <body onload="vizrt.payloadhosting.initialize()">
7        <button onclick="vizrt.payloadhosting.setFieldText('1/image_scaling', '1 2 3')">
8            Set triplet
9        </button>
10     </body>
11   </html>
```

4.5   Auto–Generated Title

The **Title** setting provides an auto-generation of the title. The title can be plain text or it can be a placeholder for one or several field values, or it can be a combination of these. The placeholder is the *{Field ID}*, the example below shows a combination of plain text, field name, and sub-field name:

TITLE: Prefix Title of person here, Vizrt Default Suffix

Format Prefix {22} {-concept-variant-choice/variant} Suffix

A template title can be auto-generated by combining one or several of these options:

- · **Normal text:** Plain text (red).
- · **{Field ID}:** Substituted with the value of the field (green).
- · **{Field ID/subfield ID}:** Substituted with the value of the subfield (purple).
- · **{listfieldname/#index/cellname}:** Substituted with the value of the field in a row in a list. Note that the index is zero-based.

> ⬥ **Warning:** The auto-generated title's length is not shortened in Vizrt web clients. However, if the title is longer than 128 characters it will be reduced when dragging out the MOS XML file due to size constraints. This affects the element title in the newsroom system.

> ⚠ **Note:** When the template uses **custom HTML** representation, it is highly recommended to set a pattern for Auto-generated title.

5   Transition Logic And Combo Templates

This section covers transition logic and combo templates, and contains the following topics:

5.1   What Is Transition Logic (TL)?

Transition Logic (TL) is a way of designing a graphics package that lets you maintain the look and feel of the graphics while letting journalists add graphics items to a rundown, without the need for technical knowledge. TL lets you independently control any number of graphics layers, providing a code-free and design-based method to build graphics that gracefully animate in and out, and transitions from one to another automatically.

> ⓘ **Info:** Transition Logic (TL) can be played out by most Vizrt control applications such as Viz Trio, Viz Pilot, Viz Multiplay and Viz Multichannel.

5.2   How Does TL Work?

5.2.1   Master Scenes

This is accomplished by using a Master Scene (aka Background Scene) that coordinates the animation of independently controlled objects which make up the whole. The master scene commonly contains the background items of the graphics package. Such items can be looping backgrounds or the design items of the lower third, over the shoulders, and full-screen graphics. The *variable* or changing content, such as the text in a lower third, is stored separately in Object Scenes.

5.2.2   Object Scenes

When a lower third is played On Air, the object scene for the lower third is triggered. This tells the engine to load the master scene, place the object scene inside the master, and animate the timelines. TL handles all of this automatically.

5.2.3   Combo Templates

These are templates that contain multiple layers of TL scenes.

5.2.4   TL Terminology

- **Combo Templates:** A TL template that contains more than one layer of scenes.
- **Master Scenes:** A TL scene is not a single scene, but a set of Viz graphics scenes that consist of a master scene that may have multiple layers of graphics that can be On Air at the same time and independently controlled.
- **Object Scenes:** Each layer in the master scene may have multiple referring object scenes. However, only one object scene per layer can be active at any given time.
- **Layers:** Layers in the transition logic scene define how many scenes can be on air at the same time. TL layers are conceptual, not spatial.

> ⚠ **Note:** With Transition Logic scene design, *take in* and *take out* commands are still used as with standalone scene design. Where standalone scene design demands that only a single scene can be On Air at a time, however, Transition Logic allows for more than one scene to be On Air simultaneously. This means that using Transition Logic lets you have a graphic covering the lower third of the screen and another graphic covering the left and/or right side of the screen for over the shoulder graphics On Air at the same time.

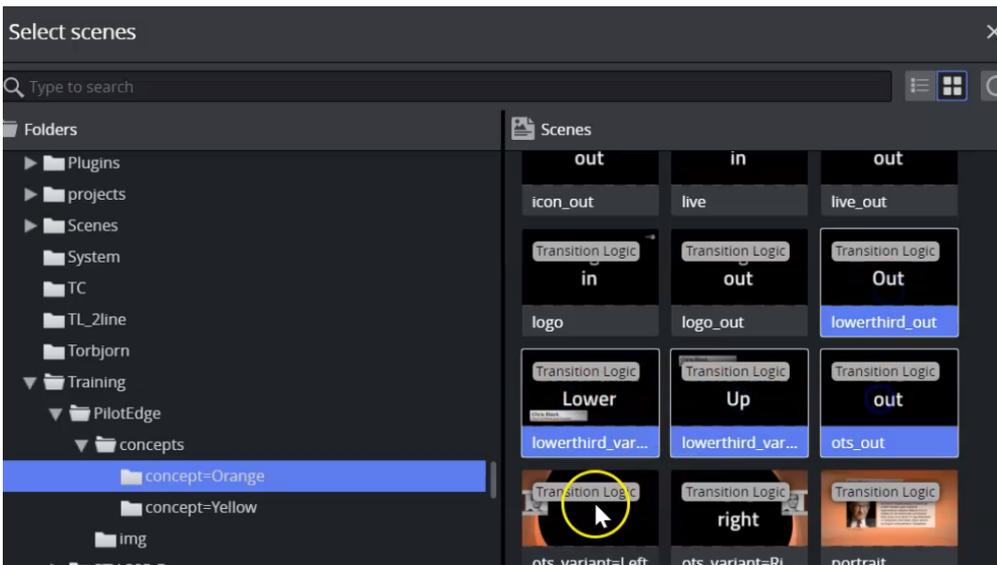## 5.3   Working With Transition Logic And Combo Templates

Follow the steps below to get started.

> ⚠ **Note:** Transition logic and combo templates require Viz Engine 4.3 or above.
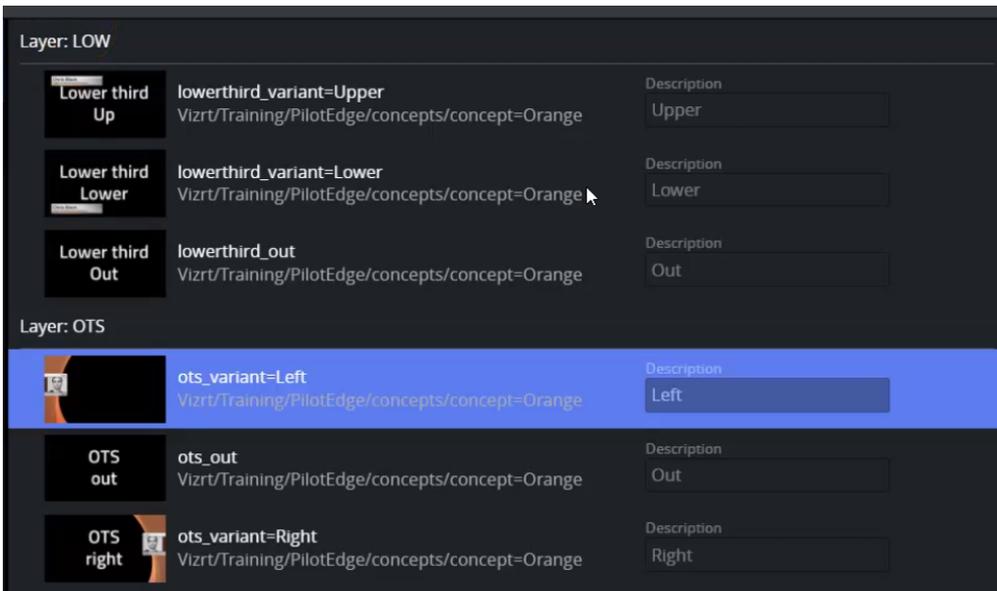
5.3.1   Creating a New Combo Template

Create a new template and add transition logic scenes. The following example uses **Orange** and **Yellow** concepts.
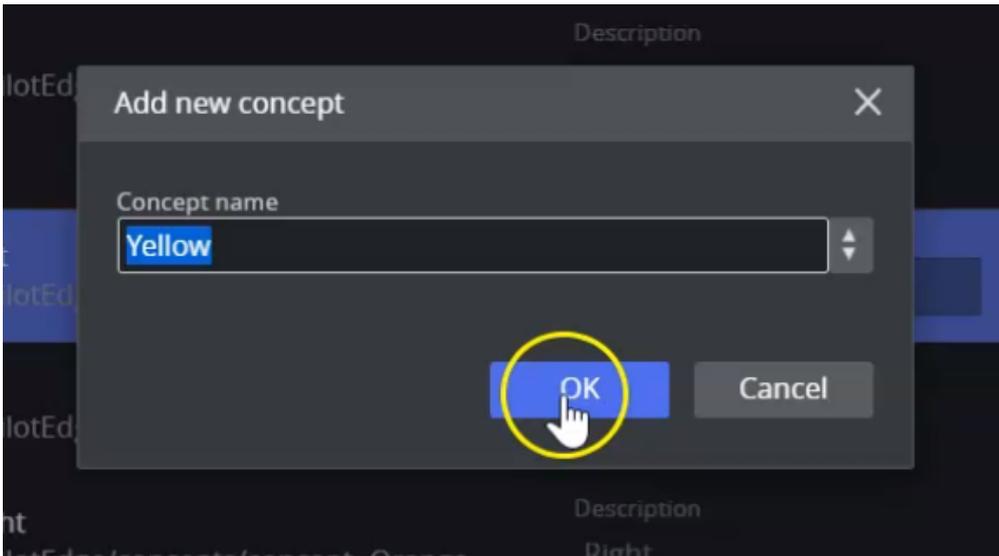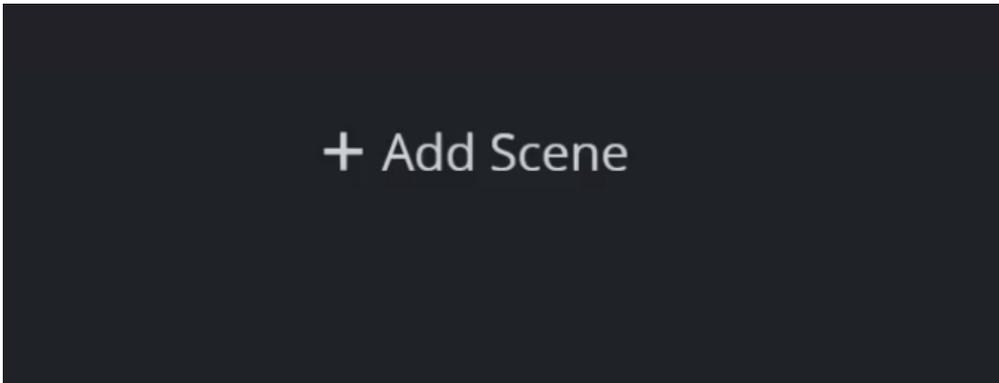
Select scenes and click **OK**.

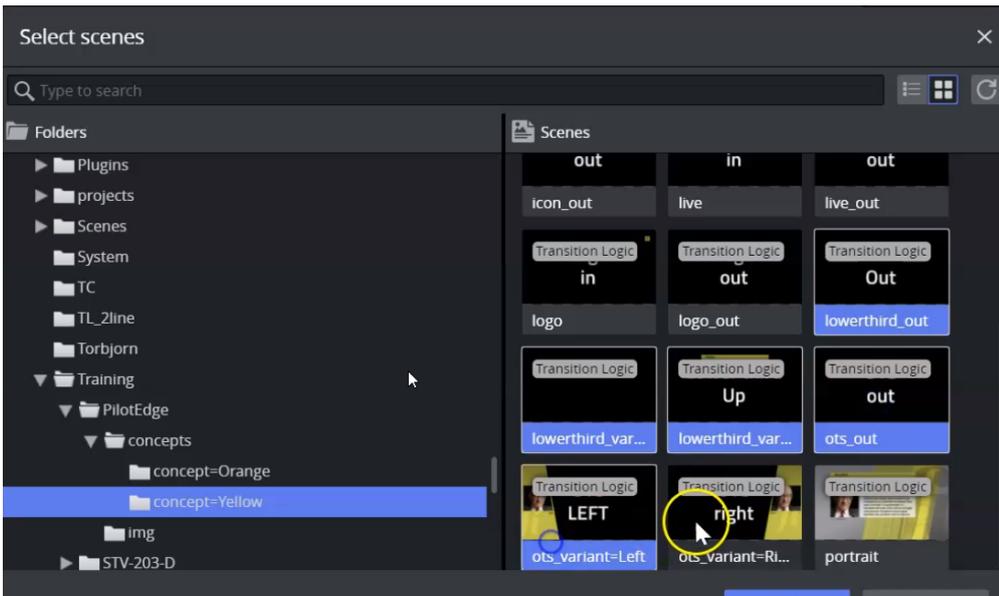The new template contains transition logic and two layers, and is therefore a combo template:

Click **+Add Concept** at the lower left corner of the screen. Enter a name for your new concept and click **OK**.
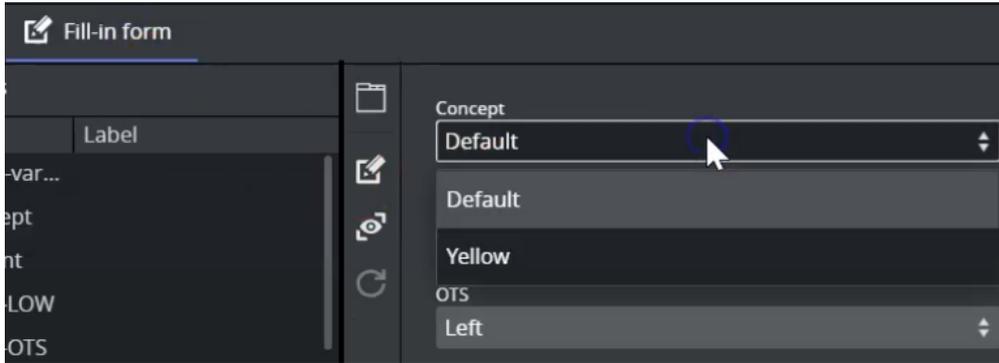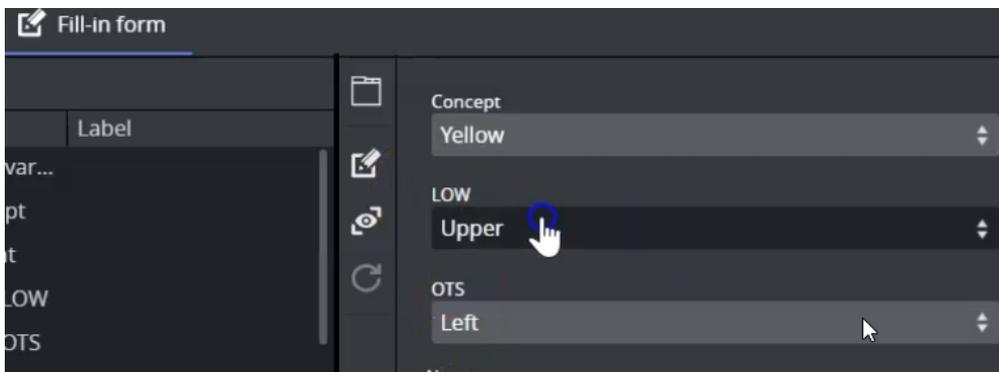
Click the **+Add Scene** button:

Select the same scenes as those you selected for **Orange**, above:

In the **Fill-in form**, you can now see that the template contains two concepts:

And two layers:

6   Previewing Content

The **Graphics Preview** window is located to the right of the interface. It displays snapshots of the final output in an ongoing preview process, and provides an indication of how the graphics look when played out in high resolution on a Viz Engine.

> ⚠ **Note:** Template Builder sends requests to Preview Server which manages the Viz Engines that provide the snapshots.
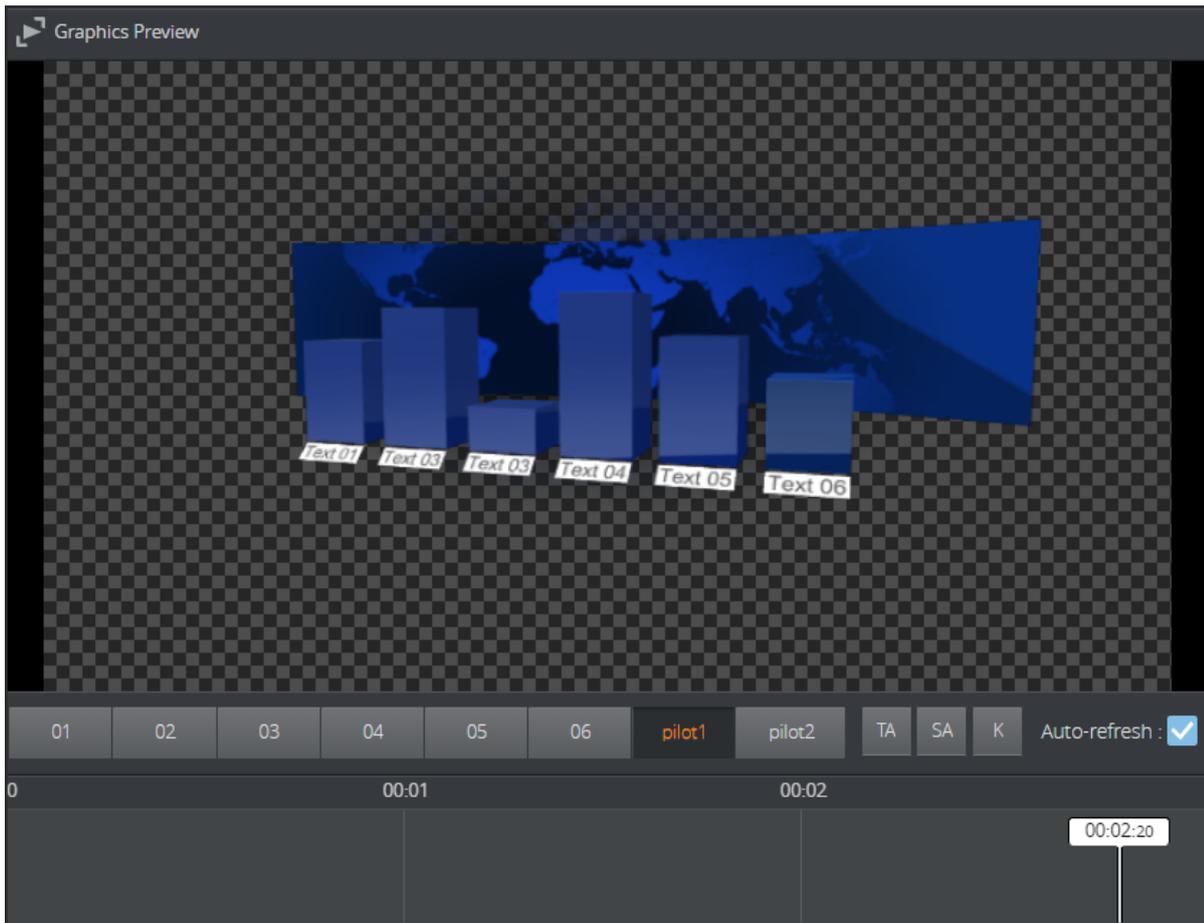
Adjust preview settings using the toolbar at the bottom of the Preview Window:

- **Preview points:** If the scene contains named preview points, such as stop points and/or tags in the Default director, these are displayed as buttons on the toolbar. If there is not enough space for the buttons, they appear in a drop-down list instead. Pressing the buttons or selecting an entry from the drop-down list shows a preview of the scene at the given preview point, and the playhead on the timeline jumps to where the preview point is set.
- **TA:** Show/hide the Title Area.
- **SA:** Show/hide the Safe Area.
- **K:** Show the key signal for the graphics.
- **Scrub:** Back and forth by clicking on the timeline or moving the playhead. If the scene does not have a director called Default or the Default director does not have a duration,

the timeline is disabled. Setting the minimum duration in Template Builder enables the timeline. Setting the default duration in Template Builder changes the duration of the timeline if it is enabled.

· **Auto-refresh:** Enabled by default.

> ⓘ **Info:** Clicking on a preview point to request a preview sends a snapshot request with a named position to Preview Server. Clicking on the timeline sends a snapshot request with an absolute position to Preview Server. For more information, see the Preview Server REST API documentation.

6.1   Viz Scene – OnPreview()

In the Viz Scene Script, you can use the *OnPreview()* function to customize the preview shown in Template Builder and Viz Pilot Edge.
Here is a sample code for Viz Scene Script:

```
//If AlwaysRunPreviewScript is set to True in Preview Server, this will get the
hostname of the Viz Engine rendering the preview and sets the returned value in the
text container from JustForPreview->Hostname.
//It also sets the container visibility to True, thus revealing the hostname and
background image, but only in Preview.

sub OnPreview(active As Integer)
    FindContainer("JustForPreview$Hostname").Geometry.Text = "Hostname: "&
System.Hostname
    FindContainer("JustForPreview").active = true
end Sub
```
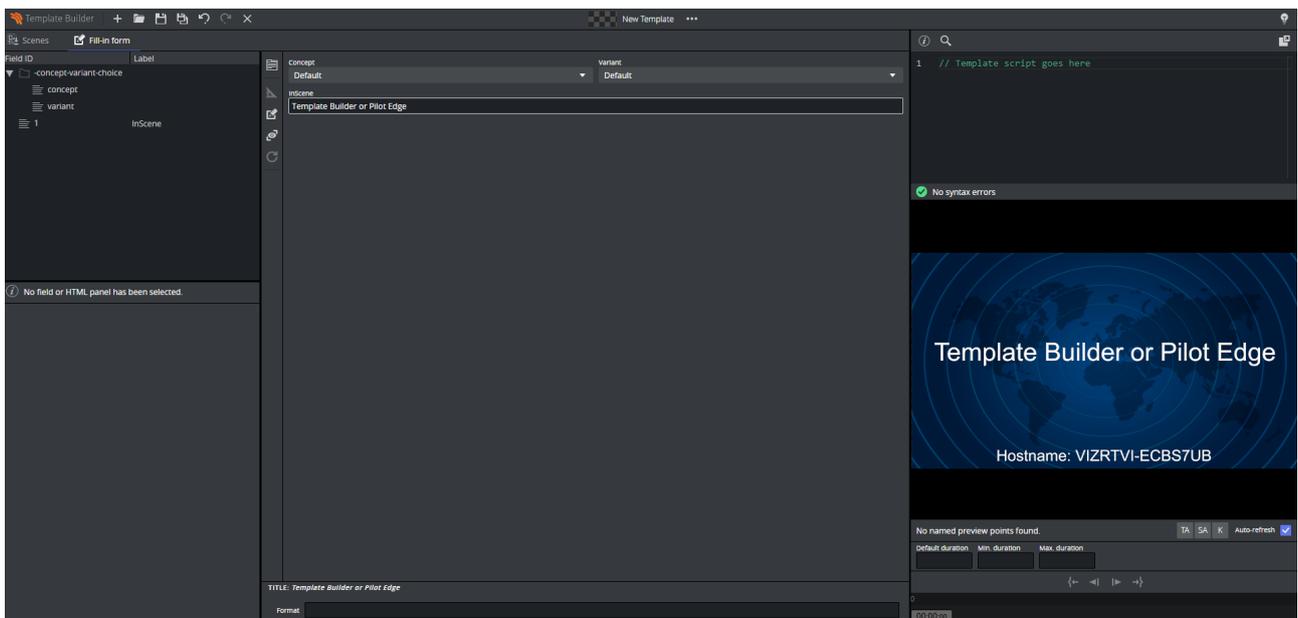
Hidden Container
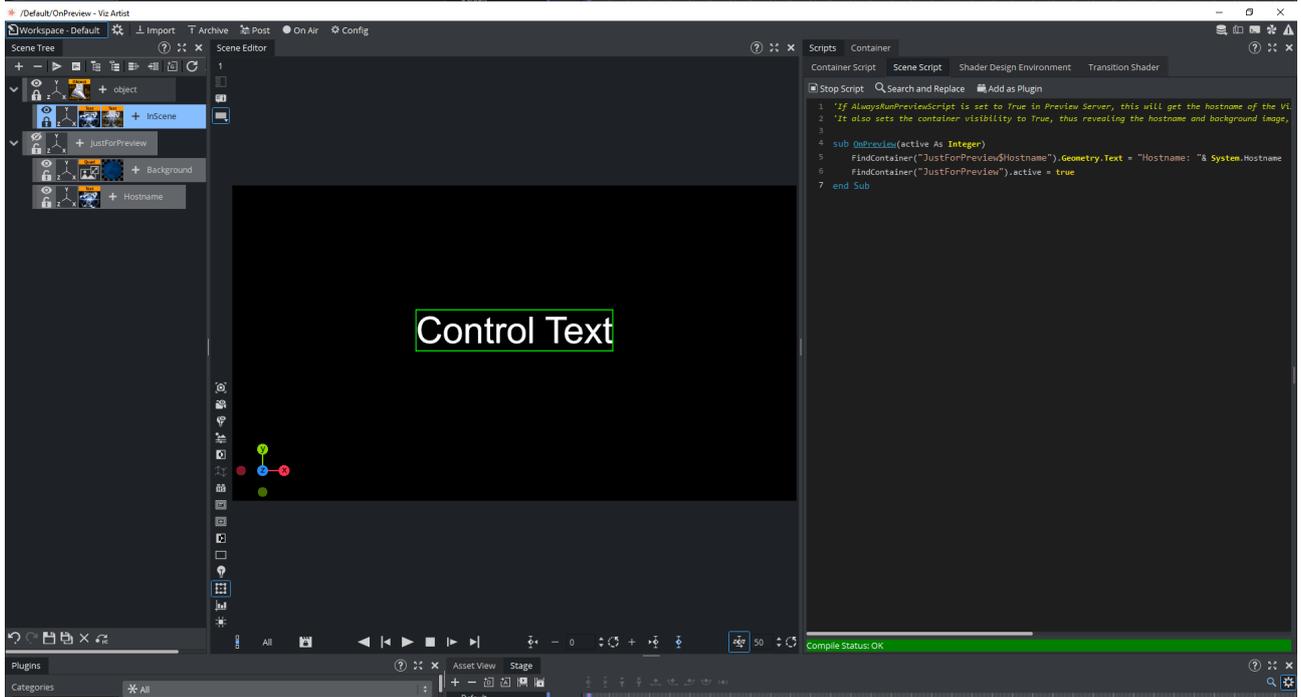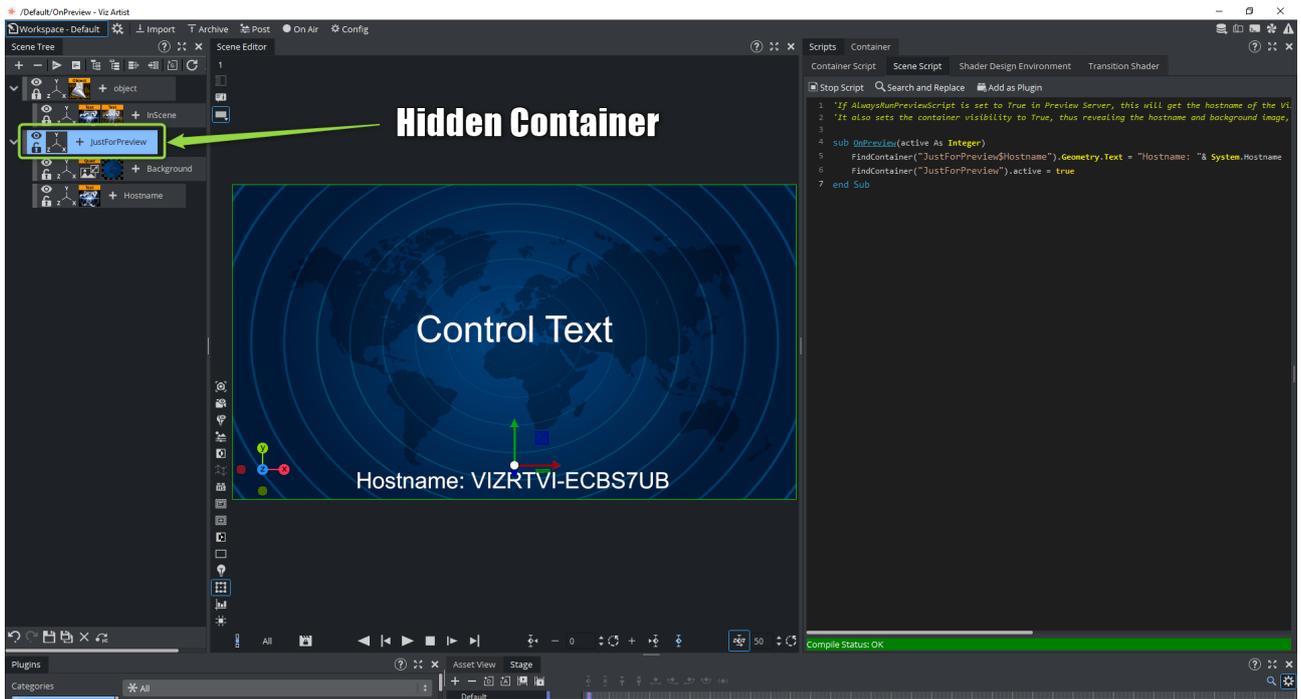
To enable this functionality in Preview Server, navigate to the location of the *PreviewServer.exe.config* and set the value **True** for **AlwaysRunPreviewScript**:

You will then have to restart the Preview Server Windows Service for this change to take effect.

7   Appendix

The appendix contains the following pages:

- Keyboard Shortcuts
- Troubleshooting
- Overview of Media Types
- Overview of Control Plugins

7.1   Keyboard Shortcuts

This page lists available keyboard shortcuts in Template Builder.

Shortcut	Description
CTRL + O	Open the **Open Template** dialog where you can select a template to open.
CTRL + S	Save a template.
CTRL + Z	Undo.
CTRL + Y	Redo.

> ⛔ **Warning:** The shortcut **CTRL + O** does not work properly in Firefox version 65.0.1 and later.

7.1.1   Graphics Preview Player Shortcuts

Use the following shortcuts for the **Graphics Preview** player:

Shortcut	Description
SPACE or **CTRL + SPACE**	Play/pause.
SHIFT + I	Go to the in-point.
SHIFT + O	Go to the out-point.
, (comma)	Move one frame back.
. (period)	Move one frame forward.

7.2   Troubleshooting

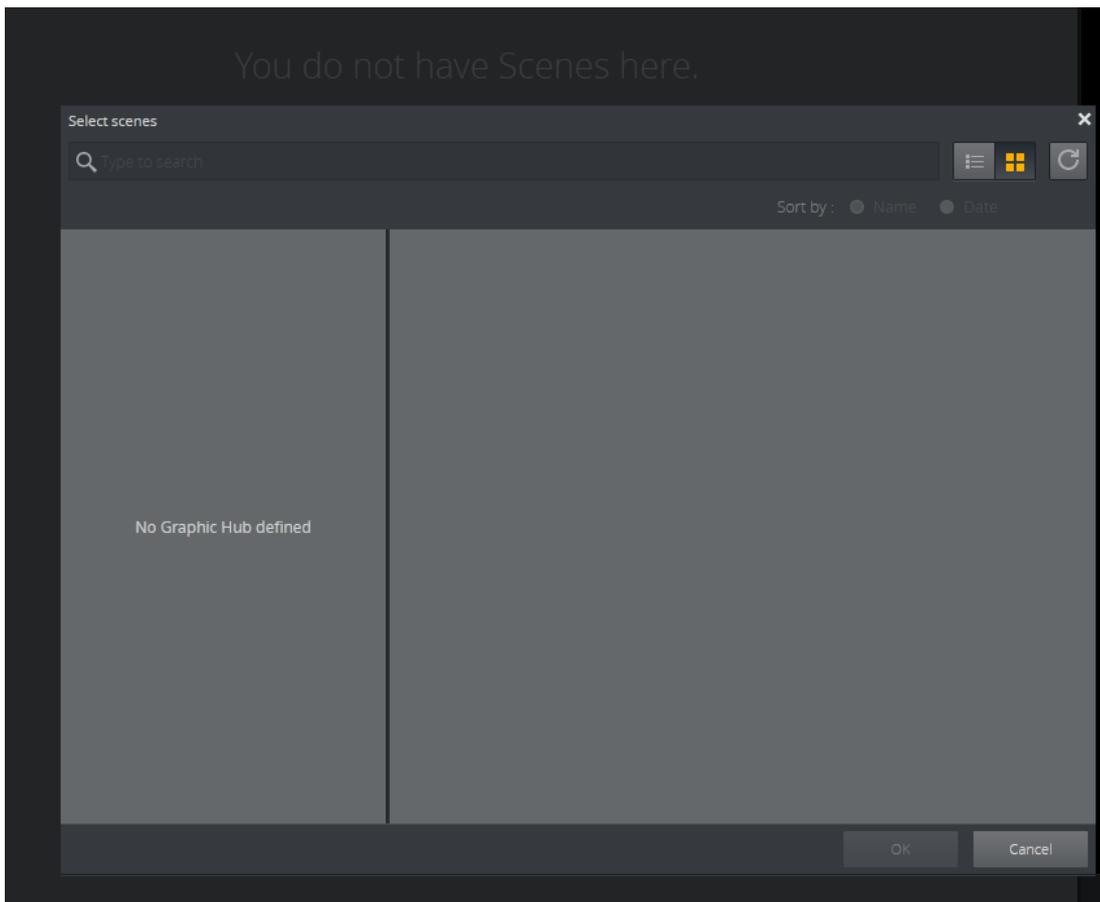A list of known issues and their fixes are listed below.

7.2.1   Create New Button Not Displayed on UI

An outdated PDS version (<8.5) is installed. Install version 8.5 or above. 8.6 is mandatory for Transition Logic support.
Preview Server must also be updated to 4.4.1 or above.
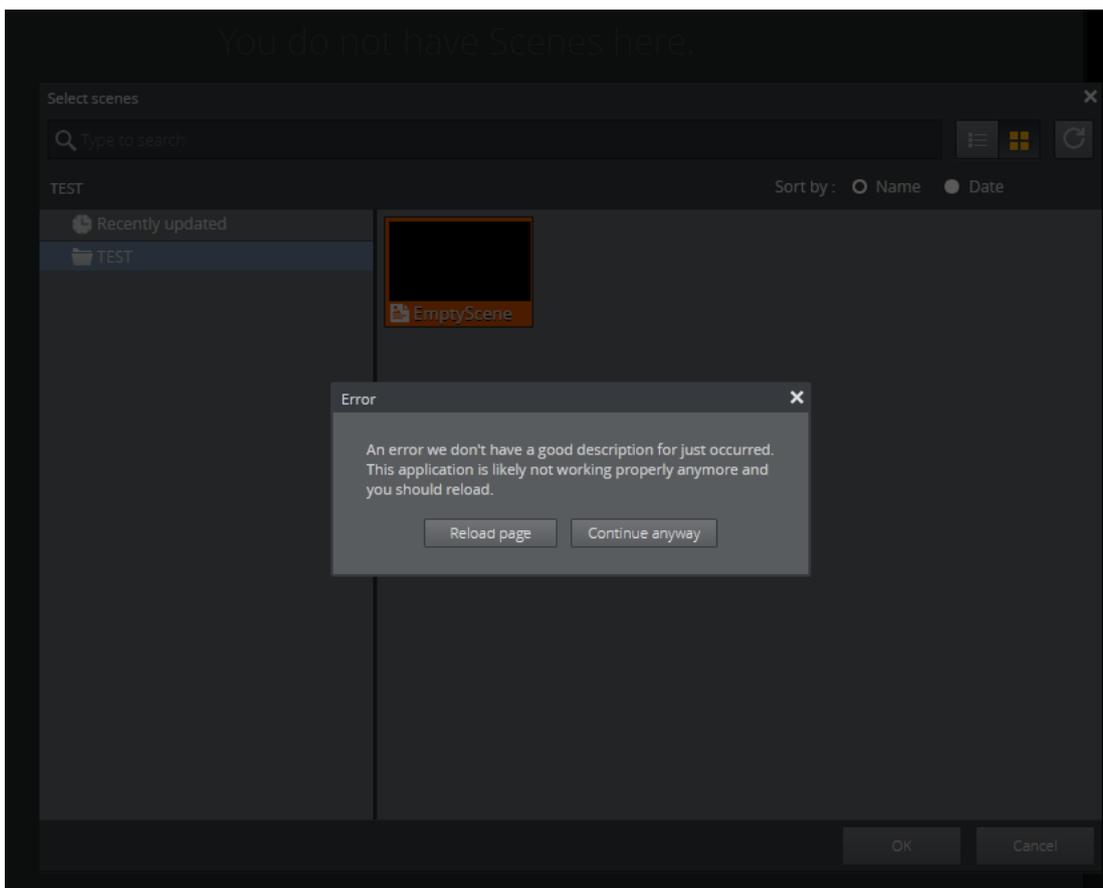
7.2.2   GH Scenes Tree Not Displayed When Pressing **Create New**

Make sure that `http://<PDS server>:8177/app/DataServerConfig/DataServerConfig.html`
→ **graphic_hub_url** is properly set:

7.2.3 An Error Message is Shown When Attempting to Open a Scene

An outdated GH REST version (<3.4.2) is installed. Install version 3.4.2 or later.

7.2.4   Preview Server Error Message Shown When Trying to Open a Scene
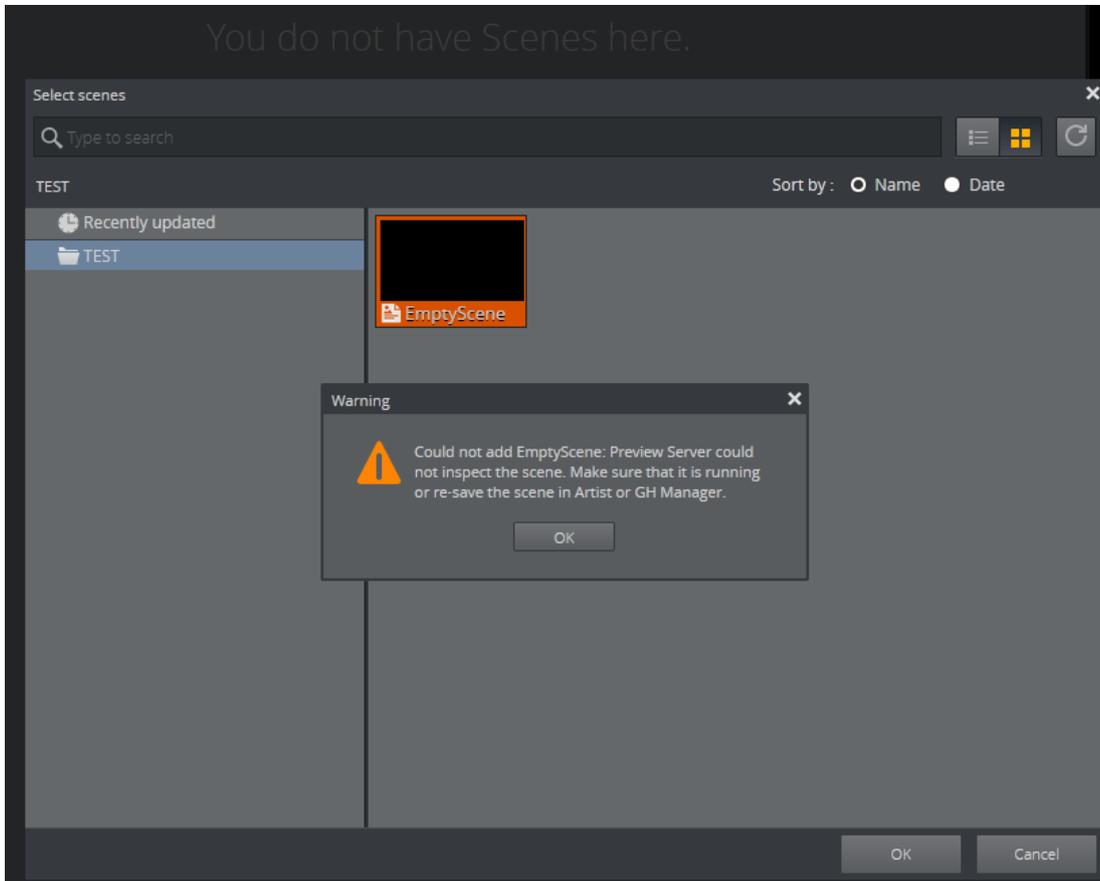
Check that the http://<PDS server>:8177/app/DataServerConfig/DataServerConfig.html → **preview_server_uri** property is set:

7.2.5   Scene Blocked Due to Outdated or Empty Geom

If the Geom of a scene is outdated or empty when creating a transition logic template, Template Builder will block use of the scene.
To fix this, save or update the scene in Viz Artist 4.2.

> 🔴 **Important:** The feature below must be enabled in the Viz Artist config file.

- Enable automatic creation of merged geometries when saving a transition logic scene: AutoExportTransitionLogicGeometries = 1

See the Viz Artist User Guide for more information on editing the Artist config file.

7.2.6   Support

Support is available at the Vizrt Support Portal.

7.3   Overview Of Media Types

The following media types are available for single value fields in Template Builder (click the links for W3C definitions):

Type		Media Type (XSD type)	Content of field/value element
Multi-line text		text/plain (string)	text
Single-line text		text/plain (normalizedString)	text
Formatted text		application/ vnd.vizrt.richtext+xml	XML (accepts plain text if unformatted)
Boolean		text/plain (boolean)	text (`true` or `false`)
Integer		text/plain (integer)	text (for example, `-42`)
Decimal		text/plain (decimal)	text using period as decimal point (for example, `123.456`)
Date and time		text/plain (dateTime)	text (for example, `2021-04-06T13:35:00Z`)

Type		Media Type (XSD type)	Content of field/value element
Date		text/plain (date)	text (for example, `2021-04-14`)
Two numbers (duplet)		application/ vnd.vizrt.duplet	text containing two decimal numbers separated by a space (for example, `0.6 0.8`)
Three numbers (triplet)		application/ vnd.vizrt.triplet	text containing three decimal numbers separated by spaces (for example, `3 4.5 5`)
Image		application/atom+xml; type=entry;media=image	The image path on GH (for example, `IMAGE*images/ flags/denmark)`
Geometry		application/ vnd.vizrt.viz.geom	The geometry path on GH (for example, `GEOM*objects/ my-geom`)
Material		application/ vnd.vizrt.viz.material	The material path on GH (for example, `MATERIAL*objects/my- material`)
Map		application/ vnd.vizrt.curious.map	Proprietary format
Color		text/vnd.vizrt.color	text (for example, #140E7E or rgba(255, 0, 0, 1)).

7.4   Overview Of Control Plugins

7.4.1   Supported Viz Artist Control Plugins

Plugin Name	Comments
Control Chart	The Control Chart plug-in exposes control of chart data from the Visual Data Tools plugins.
Control Text	From Viz Engine 5.0.1 and above, Control Text exposes text from both the Classic and Viz Engine renderer pipeline.
Control Geom	The Control Geom plug-in exposes the control of geometry objects to the user.
Control Image	The Control Image plug-in creates an image control in the control clients.
Control List	The Control List plug-in allows you to create table controls. Normally there should be a Control Object for each row.
Control Material	The Control Material plug-in exposes the material control to the user. Remember to set up a search provider towards Graphic Hub, to use this.
Control Number	The Control Number plug-in (also known as Control Num) is used to be able to decide how a number input is to be formatted. It can be a value given by the control client user or by any external source. It should be used instead of Control Text when numbers are the input value.
Control Object	Control Object should always be added to a scene when you add other control plugins. It is in this plugin that you mark if a scene is part of a transition logic set or not. There should normally only be one control object in the scene. The exception is when you use a control list when there is a control object for every line.
Control OMO	The Control Object Moving (Omo) plug-in gives you the possibility to add a group of containers and reveal one at the time. This control plugin exposes an integer. It is typically nice to use a dropdown for these with proper field names.
Control Video	The Control Video plug-in exposes control over a video codec channel (ClipChannel). Does not support clips in soft clip players.
Control WoC	A replacement of the Control Maps plugin with more options and on-the-fly feedback from Viz Artist/Engine.

ⓘ **Info:** If a Control Plugin is not listed in the table above, it is not supported by Graphic Hub and/or Template Builder.