Viz Arc Script Guide

Version 3.1

Disclaimer

Antivirus Considerations

Vizrt advises customers to use an AV solution that allows for custom exclusions and granular performance tuning to prevent unnecessary interference with our products. If interference is encountered:

- **Real-Time Scanning:** Keep it enabled, but exclude any performance-sensitive operations involving Vizrt-specific folders, files, and processes. For example:
    - C:\Program Files\[Product Name]
    - C:\ProgramData\[Product Name]
    - Any custom directory where [Product Name] stores data, and any specific process related to [Product Name].
- **Risk Acknowledgment:** Excluding certain folders/processes may improve performance, but also create an attack vector.
- **Scan Scheduling:** Run full system scans during off-peak hours.
- **False Positives:** If behavior-based detection flags a false positive, mark that executable as a trusted application.

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

2026/08/20

Contents

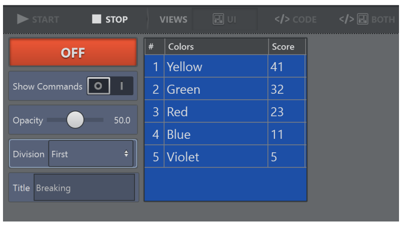
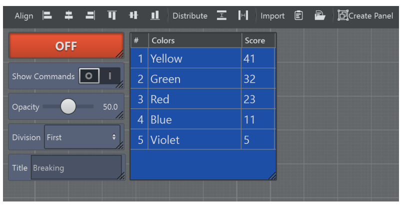
1 Introduction to Viz Arc Script Guide

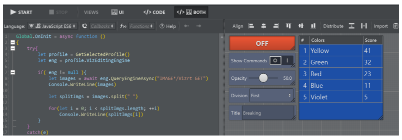
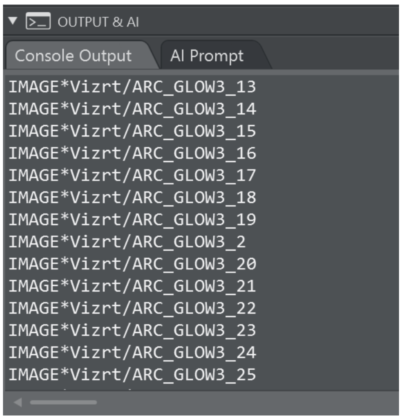
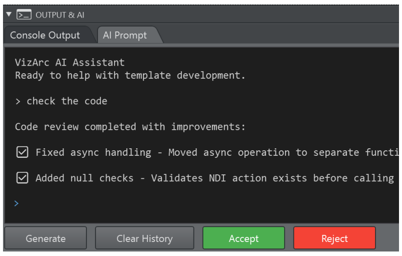
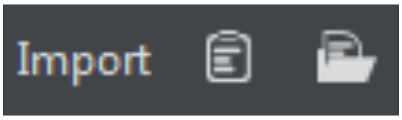
2 View

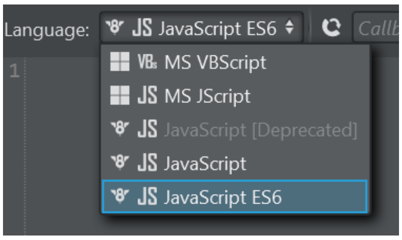
In Script View, you can write your own custom script in JavaScript language through Google's **V8** or Microsoft's **JScript** (ECMAScript3) or in **VBScript** language, as in the following example:

```
Global.OnInit = async function ()
{
    try{
        let profile = GetSelectedProfile()
        let eng = profile.VizEditingEngine

        if( eng != null ){
            let images = await eng.QueryEngineAsync("IMAGE*/Vizrt GET")
            Console.WriteLine(images)

            let splitImgs = images.split(" ")

            for(let i = 0; i < splitImgs.length; ++i)
                Console.WriteLine(splitImgs[i])
        }
    }
    catch(e)
    {
        Console.WriteLine("Error: " + e + " " + e.stack)
    }
}
```

It's possible to create custom forms and components, such as text boxes and buttons.

	To run the script, select the **Start** button on the top left of the window.
	Form Design can be edited by selecting the **UI** button . Every element can be selected and moved, aligned and distributed on the main form. To go back to code editing, select the **CODE** button or select **BOTH** to have code and UI side by side.

	To edit a script, press the **Stop** button ■ Stop on the top of the script main window.
	Console logs and debugs are displayed in the **CONSOLE** pane.
	When configured and licensed the **AI Prompt** can be used to create or modify the global script.
	Import code internally from the clipboard or an external text file in the script pane.

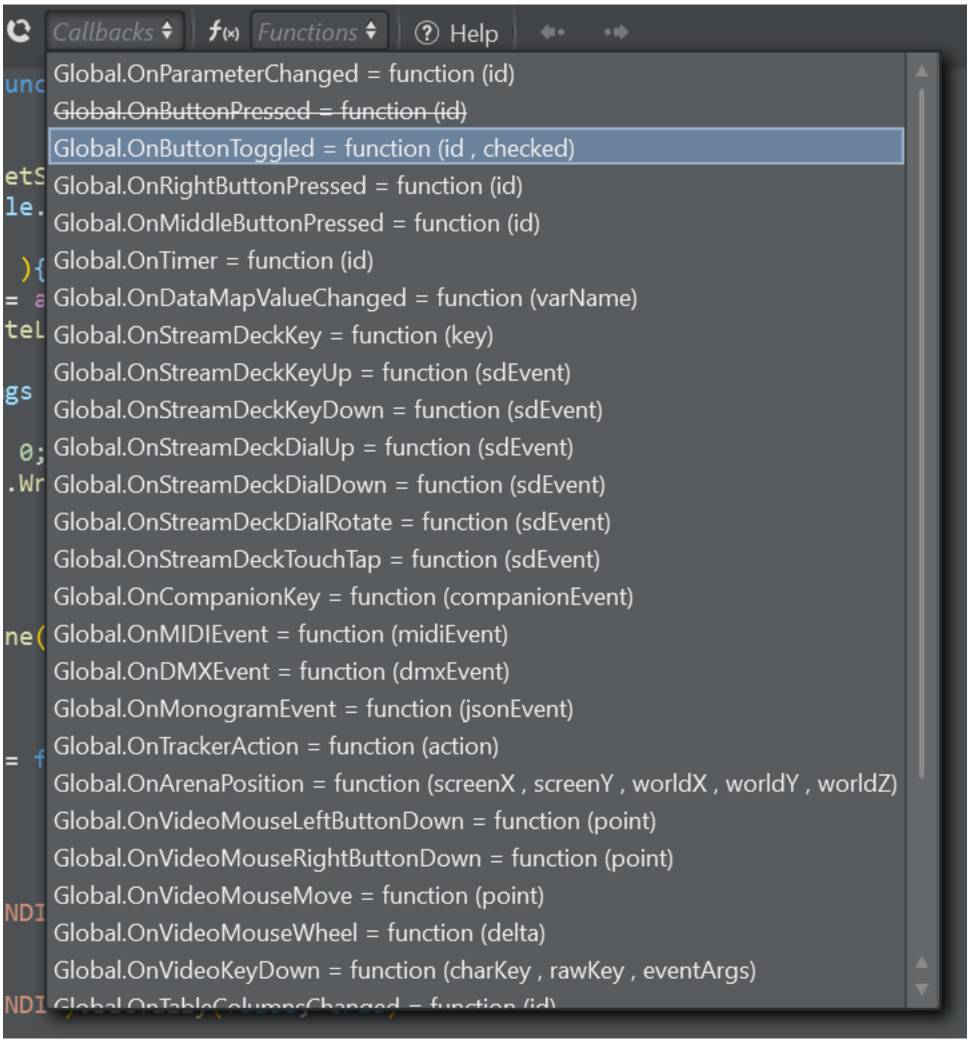


Use the **Language** menu to select a scripting language.

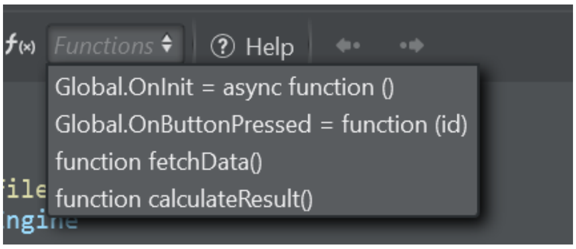
- **MS VBScript:** Microsoft Visual Basic Scripting language
- **MS JScript:** Microsoft implementation of the ECMA 262 JavaScript
- **V8 JavaScript:** Google's open source high-performance V8 JavaScript
    - Using CommonJS syntax to import modules (using **require**).
- **JavaScript ES6 (ECMAScript 2015)**: Open source high-performance JavaScript
    - Using ES Modules syntax to import modules (using **import**)

It is recommended to use **JavaScript ES6** language when possible.

In edit mode, **Script Callbacks** can be selected from the list and added:



You can locate a custom function by selecting it from the **Functions** list:



See Also

- Scripting Classes

3 Properties

- Action

3.1 Action

All Actions in the project can be accessed and modified via scripting. Use the **GetAction** function to get a reference to the action:

- BaseAction **GetAction**(string actionName)

Note: It's possible for a project to contain multiple actions that have the same name. If that is the case for your project, the first Action created with a name is returned. Make sure to use unique names when accessing actions through scripting.

Every Action type has these generic properties:

Action type	Description
string **Name**	The title of the action.
string **Description**	By default this is the type of action (for example, "Key", "Chroma", "Image" etc.). When assigned to a TransformationAction and visible on the SET view, this field is used as a tooltip when the mouse hovers over the element.
int **ExecutionDelay**	Expressed in milliseconds, minimum delay is `0` (default), maximum delay is `10000` (10 seconds).

Every Action type has these generic methods:

- void **Execute():** Executes the action.
- void **Preview():** Executes the action on the preview channel.
- void **QueryState():** Queries the current state of the action from the **Editing Engine**. For example, if the action is a transformation action, it retrieves the current transformation from the editing engine's scene tree and updates the UI accordingly.

The example below shows how to set the alpha value to 75% of an Alpha Action called *AlphaText* and execute the action from scripting:

3.1.1 Sample

```
var alphaAction = GetAction("AlphaText");
alphaAction.Alpha = 75.0;
alphaAction.Execute();
```

There are specific properties/functions for each action type:

- Alpha
- Chroma
- Command

3.1.2 Alpha

Properties:

- double **Alpha**

3.1.3 Chroma

Properties:

- ChromaPrecisionContent **Precision**
    - double **hueAdjust**

- double **saturationAdjust**
- int **edgeBlur**
- double **despillScale**
- double **backingPlateR**
- double **backingPlateR**
- double **backingPlateR**
- double **yellowGain**
- double **cyanGain**
- int **denoiseRadius**
- int **denoiseSharpen**
- double **opacityPoint**
- doube **transparencyPoint**
- double **bgEdgeGain**
- double **bgSpillGain**
- double **bgLWBlur**
- double **colorEdgeGain**
- double **colorSpillGain**
- double **colorLightwrapR**
- double **colorLightwrapG**
- double **colorLightwrapB**
- bool **addShadows**
- double **innerShadows**
- double **shadowsGain**
- bool **addHighlights**
- double **innerHighlights**
- double **highlightsGain**
- double **masterLiftR**
- double **masterLiftG**
- double **masterLiftB**
- double **masterGammaR**
- double **masterGammaG**
- double **masterGammaB**
- double **masterGainR**
- double **masterGainG**
- double **masterGainB**
- double **masterSaturation**

Sample

```
var action = GetAction("Chroma");
// sample for setting some color Precision Keyer settings
action.Precision.hueAdjust = -1140;
action.Precision.saturationAdjust = 2.0;
```

3.1.4 Command

Properties:

- string **Command**

3.1.5 ControlObject

See Control Object Classes.

Director

The Director action drives a scene (or stage) director — the same control offered by the Director action edit panel. The **DirectorType** property selects the director type (e.g. START, CONTINUE) and the remaining properties configure the keyframes/stops it acts on. Set the mode first, then only the properties that mode uses.

Properties

Property	Type	Description
DirectorType	`string`	Transport mode. One of `START`, `STOP`, `CONTINUE`, `START_REVERSE`, `CONTINUE_REVERSE`, `PLAY_FROM`, `PLAY_FROM_REVERSE`, `FROM_TO`, `GO_TO`, `PAUSE`, `LOOP_FROM_TO`, `PLAY_FROM_TO`.
SelectedDirector	`string`	The target director, addressed by name or full tree path. Reading returns the director name. An unknown value is ignored.
SelectedDirectorIndex	`int`	The target director as a zero-based index into the director list. Authoritative alternative to **SelectedDirector** when several directors share a name.
From	`string`	Start of the interval for `FROM_TO`, addressed by stop/tag name (e.g. `"stop2"`).
To	`string`	End of the interval for `FROM_TO`, addressed by stop/tag name.
FromIndex / ToIndex	`int`	The `FROM_TO` interval start/end as zero-based keyframe indices. Use instead of **From**/**To** for direct addressing, or when a keyframe has no name.

Property	Type	Description
StopTag	string	The single Stop/Tag used by `PLAY_FROM`, `PLAY_FROM_REVERSE` and `GO_TO`, addressed by stop/tag name.
StopTagIndex	int	The single Stop/Tag as a zero-based keyframe index.
CustomFrameTime	double	Custom frame time in seconds, used when the `<CUSTOM>` keyframe is selected (e.g. with `PLAY_FROM` or `GO_TO`).
IsStageAction	bool	When `true`, the command targets the stage director instead of a scene director.

The name-based properties (**From**, **To**, **StopTag**, **SelectedDirector**) resolve against the action's currently loaded keyframe/director lists, which are populated once the director has been parsed. Names are matched case-insensitively and an unknown name is ignored. Use the `*Index` variants for direct addressing.

Sample

```
// FROM_TO: play a named director between two stops
var dir = GetAction("Default")      // select the action by name
dir.SelectedDirector = "Default"    // director by name (or full tree path)
dir.DirectorType = "FROM_TO"
dir.From = "stop2"                  // From Stop/Tag by name; or dir.FromIndex = 3
dir.To   = "stop3"                  // To Stop/Tag by name;   or dir.ToIndex = 5
dir.Execute()

// PLAY_FROM / PLAY_FROM_REVERSE / GO_TO use the single Stop/Tag instead of From/To
dir.DirectorType = "GO_TO"
dir.StopTag = "stop2"               // by name; or dir.StopTagIndex = 3
dir.CustomFrameTime = 1.5           // seconds, when the <CUSTOM> keyframe is selected
dir.Execute()
```

3.1.6 Group

This action has no additional public properties.

3.1.7 Image

Properties:

- string **Image**

    - Value should be a Graphic Hub path that starts with "IMAGE*"
- bool **IsBuiltin**
- string **Builtin**
    - Possible values: LIVE1, LIVE2, CLIP1, etc.
- double **PosX**
- double **PosY**
- double **RotX**
- double **RotY**
- double **RotZ**
- double **ScaX**
- double **ScaY**

The Image parameter can be assigned to a Graphic Hub path when the string starts with "IMAGE*"; when it starts with "http" it will be assumed to be a web link (or a Media Service link), otherwise it will be interpreted as a local file path, see the samples below:

Sample

```
var imageAction = GetAction("Image");
imageAction.Image = "IMAGE*/VizArc/arcLogo";
// or
imageAction.Image = "http://127.0.0.1:21099/serve/original/AR_03.jpg";
// or
imageAction.Image = "C:/Users/admin/Desktop/CAKE.jpg";
```

3.1.8 Light

Properties:

- string **LightType** [read only]
    - Possible values: NONE, SPOTLIGHT, DIRECTIONAL, AREA, POINT
- string **LightColor**
- double **LightIntensity**
- double **DiffuseIntensity**
- double **SpecularIntensity**
- double **LightRadius**
- double **OuterConeAngle**
- double **InnerConeAngle**
- int **LightLayer**
- double **DirectionalSpread**
- double **RadiosityMultiplier**

3.1.9 Key

Properties:

- bool **KeyEnabled**
- bool **CombineBackground**
- bool **DepthInfoOnly**
- bool **DrawKey**
- bool **DrawRGB**

3.1.10 Material

Properties:

- string **ColorHex** [#RRGGBB]
- string **Diffuse** [#RRGGBB]
- string **Emission** [#RRGGBB]
- string **Specular** [#RRGGBB]
- string **Ambient** [#RRGGBB]
- double **Alpha** [0…100]
- double **Shiniess** [0…100]
- bool **UseSimpleColor**

Functions:

- SetColorRBG(int r, int g, int b)

3.1.11 MSE

Properties:

- string **Page**
- string **DirectorType**
  - Possible values: TAKE, CONTINUE, TAKE_OUT

3.1.12 Multizone Chroma Key

Properties:

- string **ZoneName**
- double **Height**
- double **Altitude**
- double **Luminance**
- double **MinLuminance**
- double **MinGrad**
- double **MaxLuminance**
- double **MaxGrad**
- double **Blend**
- double **U**
- double **V**
- double **UVDiameter**
- double **UVGradient**

- bool **IsFullscreen**
- bool **PickLuma**
- bool **PickChroma**
- bool **PickInViz**

3.1.13 NDI

Properties:

- int **Preset**
    - Value must be between 0 and 99
- float **Velocity**
    - Value must be between 0 and 1

Functions:

- void **SetSource**(string name)
    - Sets the source to a different NDI source called *name*.
- void **GotoPreset**(int preset, float velocity)
    - Goes to a stored preset *preset*. The value range is from 1 to 100. Move the camera using *velocity* (from 0.0 to 1.0), where velocity of 1 is maximum speed.
- void **GotoPreset**(int preset)
    - Goes to a stored preset *preset*. The value range is from 1 to 100. The velocity is determined on the action's current velocity property value.
- void **StorePreset**(int preset)
    - Stores the current camera position as *preset*. The value range is from 1 to 100.
- void **SetTally**(bool program, bool preview)
    - Sets or unsets the tally of the current NDI source to program or preview, or both.
- bool **IsOnProgram**()
    - Returns whether the source is in program.
- bool **IsOnPreview**()
    - Returns whether the source is in preview.
- void **SetZoom**(double value)
    - Sets the zoom value of the NDI camera, the range is from 0.0 (fully zoomed out) to 1.0 (fully zoomed in).
- void **SetFocus**(double value)
    - Set the focus value of the NDI camera, the range is from 0.0 to 1.0.
- void **SetPanTiltValue**(double pan, double tilt)
    - Sets the absolute pan and tilt values of the NDI camera, the ranges are for both from 0.0 to 1.0.
- void **SetPanTiltSpeed**(double pan, double tilt)
    - Sets the pan and tilt speed values of the NDI camera, the ranges are for both from -1.0 to 1.0. If the values are non zero, the camera moves continuously along the pan/tilt axis with the given speed and direction.

3.1.14 Omo

Properties:

- int **ElementIndex**
- bool **ShowUntil**

3.1.15 PBR

Properties:

- Modes
    - bool **IsPreload**
    - bool **IsGHMode**
- GH Mode
    - string **PhongMaterialAsset**
        - Value should be a Graphic Hub path that starts with "MATERIAL_DEFINITION*"
- Values Mode
    - Material Settings
        - string **ColorTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - |string **ColorTint**
        - bool **ColorIsSRGB**
        - string **EmissiveTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **EmissiveColor**
        - double **EmissiveIntensity**
        - string **NormalTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **RoughnessTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - double **RoughnessFactor**
        - string **MetallicTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - double **MetallicFactor**
        - string **AmbientOcclusionTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **HeightTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - double **HeightDepth**
        - string **EnvironmentTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - double **EnvironmentRotation**
    - Texture Settings
        - double **TillingU**
        - double **TillingV**
        - double **UvAngle**
        - double **UvScaleU**
        - double **UvScaleV**
        - double **UvOffsetU**

- double **UvOffsetV**

3.1.16 Phong

Properties:

- Modes
    - bool **IsPreload**
    - bool **IsGHMode**
- GH Mode
    - string **PbrMaterialAsset**
        - Value should be a Graphic Hub path that starts with "MATERIAL_DEFINITION*"
- Values Mode
    - Material Settings
        - string **ColorTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - |string **ColorTint**
        - bool **ColorIsSRGB**
        - string **AmbientTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **AmbientColor**
            - For example: "#FF00A0"
        - double **AmbientIntensity**
        - string **DiffuseTexture**
        -
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **DiffuseColor**
            - For example: "#FF00A0"
        - double **DiffuseIntensity**
        - string **SpecularTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **SpecularColor**
            - For example: "#FF00A0"
        - double **SpecularIntensity**
        - string **EmissiveTexture**
            - Value should be a Graphic Hub path that starts with "IMAGE*"
        - string **EmissiveColor**
            - For example: "#FF00A0"
        - double **EmissiveIntensity**
        - double **Shininess**
        - bool **Lit**
            - Whether the material should be lit
    - Texture Settings
        - double **UvAngle**
        - double **UvScaleU**
        - double **UvScaleV**

- double **UvOffsetU**
- double **UvOffsetV**

3.1.17 Scene Loader

Properties:

- bool **UseGUID**
- string **FrontUUID**
- string **MainUUID**
- string **BackUUID**
- string **GfxUUID**
- string **SubSceneUUID**
- bool **FrontClear**
- bool **MainClear**
- bool **BackClear**
- bool **GfxClear**
- bool **SubSceneClear**
- bool **FrontResetStage**
- bool **MainResetStage**
- bool **BackResetStage**
- bool **GfxResetStage**
- int **GfxLayerNumber** [0,…,17]
- bool **SubSceneResetStage**

3.1.18 Script

Functions:

- dynamic **GetParameterValue**(string name)
  - Returns the value of a Script UI element. The *name* is the name of the UI parameter as specified in the viz script by the **RegisterParameter*** function.
- bool **SetParameterValue**(string name, dynamic value)
  - Sets the value for a UI parameter. The *name* is the name of the UI parameter as specified in the viz script by the **RegisterParameter*** function. Returns true on success.

Example

```
// get the script action with the name "scriptA"
let action = GetAction("scriptA")

// set some parameter values
action.SetParameterValue("aDouble", 3.3)
action.SetParameterValue("aInteger", 2)
action.SetParameterValue("aString", "another string")
action.SetParameterValue("aMultiString", "another\\nmultistring")
```

```
action.SetParameterValue("aBool", false)
action.SetParameterValue("aImage", "c:/tmp/test.jpg")

// read the parameter values
Console.WriteLine("aDouble value is " + action.GetParameterValue("aDouble"))
```

3.1.19 Shared Memory

Functions:

- string[] **GetKeys**()
  Returns the list of keys present in the shared memory action.
- string[] **GetValues**()
  Returns the list of values present in the shared memory action.
- string[] **GetDestinations**()
  Returns the list of destinations present in the shared memory action.
- string **GetKeyValue**(string key)
  Returns the value of *key*. Returns null if *key* is not present int the shared memory action.
- string **GetKeyDesitnation**(string key)
  Returns the destination of *key*. Returns null if *key* is not present int the shared memory action.
- void **AddKeyValue** (string key, string value)
  Add *key/value* pair to the shared memory action.
- void **AddKeyValue** (string key, string value, string destination)
  Add *key/value* pair to the shared memory action and set it to *destination*.
- void **SetKeyValue** (string key, string value)
  Set a new *value* to to the *key* entry. Adds the pair if *key* is not present int shared memory action.
- void **SetKeyValue** (string key, string value, string destination)
  Set a new *value* to to the *key* entry and set it to *destination*. Adds the pair if *key* is not present int shared memory action.
- void **SetKeyDestination** (string key, string destination)
  Change the *destination* to the *key* entry.
- void **InsertKeyValue** (int index, string key, string value)
  Insert *key/value* pair to the shared memory action at position *index*.
- bool **Remove** (string key)
  Remove *key* from shared memory action.
- void **RemoveAt** (int index)
  Remove key at *index* position from shared memory action.

destination can be either "SYSTEM", "COMMUNICATION" or "DISTRIBUTED"

3.1.20 Telemetrics

Properties:

- int **Program**
- int **Scene**

3.1.21 Text

Properties:

- string **Text**

3.1.22 Tracking Hub Command

Properties:

- string **Command**

3.1.23 Transformation

Properties:

- double **PosX**
- double **PosY**
- double **PosZ**
- bool **PosEnabled**
- double **RotX**
- double **RotY**
- double **RotZ**
- bool **RotEnabled**
- double **ScaX**
- double **ScaY**
- double **ScaZ**
- bool **ScaEnabled**

3.1.24 Utah Router

Properties:

- int **Source**
- int **Desitnation**

3.1.25 Unreal Animation

Properties:

- string **AnimationMode**
  - Possible values: LOAD, CONTINUE, PAUSE
- bool **IsLooping**
- double **PlayRate**
- double **BlendTime**
- string **SelectedAnimation**

3.1.26 Unreal Blueprint

See Control Object Classes.

3.1.27 Unreal Dispatcher

This action has no additional public properties.

3.1.28 Unreal Scene Loader

- bool **GetStreamingLevelVisibility**(int index)
    - Returns true when the Streaming Level at position *index* is visible
- void **SetStreamingLevelVisibility**(int index, bool value)
    - Sets the Visibility to *value*, at position *index*.
- string[] **GetStreamingLevels**()
    - Returns the list of Streaming Levels present in the action.
- int **GetNumberOfStreamingLevels**()
    - Returns the number of Streaming Levels present in the action.

3.1.29 Unreal Sequencer

Properties:

- string **DirectorType**
    - Possible values: START, STOP, CONTINUE, START_REVERSE, CONTINUE_REVERSE, PLAY_FROM, PLAY_FROM_REVERSE, GO_TO, PAUSE
- int **LoopCount**
- double **PlayRate**

3.1.30 Unreal Text

Properties:

- string **Text**
- double **ScaleX**
- double **ScaleY**

3.1.31 Vinten

This action has no additional public properties.

3.1.32 Virtual Studio

Properties:

- int **SelectedSceneIndex**

- bool **SendPosition**
- string **SetName**
- double **PosX**
- double **PosY**
- double **PosZ**
- double **RotY**

3.1.33 Visibility

Properties:

- bool **Visibility**
- string **VisibilityMode**
    - Possible values: ON, OFF, ONOFF, DUAL_MODE

3.1.34 Viz Camera

Properties:

- int **SelectedCamera**
- bool **RemoteEnabled**
- bool **IsRemote**
- bool **AngleEnabled**
- double **Angle**
- bool **PosEnabled**
- double **PosX**
- double **PosY**
- double **PosZ**
- bool **DirEnabled**
- double **Pan**
- double **Tilt**
- double **Twist**

3.1.35 Viz Clip

Properties:

- string **ClipName**
- bool **IsLoader**
- string **ControlType**
    - possible values: **START, STOP, CONTINUE, PAUSE**
- string **SelectedClipChannel**
- bool **PlayOnLoad**
- bool **HasLoop**
- bool **ShouldQueue**

3.1.36 Viz PBR Material

Properties:

- bool **IsPreLoad**
- bool **IsGHMode**
- string **PbrMaterialAsset**
- string **ColorTexture**
- string **ColorTint**
  - For example: "#FF00A0"
- bool **ColorIsSRGB**
- string **EmissiveTexture**
- string **EmissiveColor**
  - For example: "#FF00A0"
- double **EmissiveIntensity**
- string **NormalTexture**
- string **RoughnessTexture**
- double **RoughnessFactor**
- string **MetallicTexture**
- double **MetallicFactor**
- string **AmbientOcclusionTexture**
- string **HeightTexture**
- double **HeightDepth**
- string **EnvironmentTexture**
- double **EnvironmentRotation**
- double **TillingU**
- double **TillingV**
- double **UvAngle**
- double **UvScaleU**
- double **UvScaleV**
- double **UvOffsetU**
- double **UvOffsetV**

4 Classes

4.1 Scripting

This section covers the following topics:

4.1.1 General

Viz Arc's scripting has many classes and types that are exposed and accessible via code. The script's main class is called **arc** and it exposes all the functions that are capable of interacting with the remaining parts of Viz Arc as well as many helper functions. All of **arc**'s functions can be accessed via scripting by calling them directly, since they are all exposed directly to the global script, or via the **arc** keyword.

The following samples and codes snippets are all written using the **V8 JavaScript** syntax:

Accessing Viz Arc Functions

```
// Getting a reference to an action called VersusTemplate
var versus = arc.GetAction("VersusTemplate")
var versus = GetAction("VersusTemplate")
```

> **Note:** You can also find this section in Viz Arc by selecting the **Help** button in the script section when in edit mode.

4.1.2 Action

All actions in the current project can be accessed using the **GetAction** method, whose content can be manipulated. See Action Properties for more details.

- BaseAction **GetAction** (string actionNameOrGUID)
    - Returns the first action found with the name provided. When a valid GUID is provided as a string, it returns the action found with the provided GUID.
- BaseAction **GetAction** (string actionName, string tabName)
    - Returns the first action found with the name provided inside the action tab named tabName.
- BaseAction **GetActionByName** (string actionName)
    - Returns the first action found with the name provided.
- BaseAction **GetActionByName** (string actionName, string tabName)
    - Returns the first action found with the name provided inside the action tab named tabName.
- BaseAction **GetActionByUUID** (Guid actionUUID)
    - Returns the first action found with the provided GUID.
- BaseAction[] **GetSelectedActions** ( )
    - Returns an array containing all the actions that are selected on the action canvas
- string[] **GetActionsOfTab** (string tabName, string actionType = "ALL")
    - Returns an array with the **names** of the actions inside the tab *tabName*, optionally filtered by action type. It does not descend into Group Actions. Use `GetActions` to retrieve action objects.
- BaseAction[] **GetActions** (string actionType = "ALL")
    - Returns an array with all the actions of the entire project of type equal to the one provided in *actionType* input. Default value ("ALL") includes all actions.
- BaseAction **GetChildAction** (string nameOrUUID)

- Method available on **Group Action objects**: returns the child action with the given name or UUID, for example `GetAction("myGroup").GetChildAction("headline")`.

GetAction Example

```
// Getting a reference to an action explicitly by its name "VersusTemplate"
var versus = GetActionByName("VersusTemplate")

// Getting a reference using a GUID
var versus = GetAction("e87a8031-a86b-4997-a169-c6f791920449")

// Getting a reference to an action called VersusTemplate
var versus = GetAction("VersusTemplate")

// Getting all NDI actions
var nidActions = GetActions("NDI")

// get the action "PrecisionKeyer" within the "INITIALIZE" group action
var chromaAction = GetAction("INITIALIZE").GetChildAction("PrecisionKeyer")
```

4.1.3 Playlist

Playback and Navigation

These methods control the currently loaded playlist and its selected row.

> **Note:** The functions in this section (including the Coloring functions and `GetTemplatePayload`) are available in **template scripts** only. The Playback and Navigation functions above are available in both the main script and template scripts.

- void **ExecuteSelectedPlaylistRow** ( )
  - Executes the selected row on the playlist.
- void **ExecuteSelectedPlaylistRowAndNext** ( )
  - Executes the selected row on the playlist and changes selection to the next row.
- void **PreviewSelectedPlaylistRow** ( )
  - Previews the selected row on the playlist.
- BaseAction **GetSelectedPlaylistRowAction** ( )
  - Returns the action that's attached to the selected row on the playlist.
- void **SetSelectedPlaylistRow** (params string[] path)
  - Tries to find the row at path (path should contain a string per depth level) and makes it the selected row.
- void **PlayPlaylistByName** (string tabName)
  - Highlights the playlist with the name *tabName,* and plays it from the beginning.
- void **PlayPlaylistByIndex** (int tabIndex)
  - Highlights the playlist with 0-based index *tabIndex,* and plays it from the beginning.

- void **StopPlaylist** ( )
  - Stops the currently playing playlist.
- void **ChangePlaylistTab** (string tabName)
  - Selects and highlights the playlist with the name *tabName*.

Playlist Example

```
// Selects the row at "StatsDisplayGroup/AwayTeam/Show" and then previews and
executes it.
SetSelectedPlaylistRow("StatsDisplayGroup", "AwayTeam", "Show")
PreviewSelectedPlaylistRow()
ExecuteSelectedPlaylistRowAndNext()
```

Building and Managing Playlists

A script can build and maintain playlists autonomously: generate one payload item per data row, group them into subfolders, and regenerate or delete them on demand. All of these methods run on the UI thread and refresh the playlist tree automatically. The edits are **not** added to the undo stack (same behaviour as the REST/MCP playlist API).

Two kinds of item can be created. **CreatePlaylistItem** creates a *payload item*: a snapshot of a template's linked control-object values, captured with **GetTemplatePayload** (see the Payloads section). Selecting a payload item fires the template's **OnSetPayload** (isPreview = true) callback and executing it fires **OnSetPayload** (isPreview = false); if no **OnSetPayload** is implemented, Viz Arc auto-restores the linked parameter values. **CreatePlaylistActionItem** instead creates a normal (non-payload) *action item* that references a canvas action, behaving like an ordinary canvas-action playlist row.

Tabs

- bool **CreatePlaylistTab** (string name)
- bool **CreatePlaylistTab** (string name, bool selectByDefault)
  - Creates a playlist tab named *name*. If a tab with that name already exists no new tab is created. When *selectByDefault* is true, the existing tab is selected. Returns false when the name is empty.
- bool **SelectPlaylistTab** (string name)
  - Switches to the existing tab named *name*. Returns false if no tab with that name is found.

Creating Items

- string **CreatePlaylistItem** (string label, string payload, string path = "", string tabName = null, int callup = 0)
  - Adds a payload item with the given *label* and *payload* string. *path* is optional (empty string targets the tab root; folder names such as "Forwards" are created on demand). *tabName* is optional (null or empty targets the current tab, which is created if missing). Returns the new item's UUID string, or null on failure, so it can be removed later. The optional *callup* assigns a Trio-style callup code to the created item (0 = no callup code).
- string **CreatePlaylistActionItem** ( )
- string **CreatePlaylistActionItem** (action, string path = "", string tabName = null, int callup = 0)

- Adds a normal (non-payload) item that references a canvas action rather than a stored payload. *action* identifies the action to reference and may be its name or UUID (string), an action object (such as `ThisAction` or the result of `GetActionByName` / `GetActionByUUID`), or omitted to use the current template's own action. *path* and *tabName* follow the same conventions as **CreatePlaylistItem**. Returns the new row's UUID string, or null on failure (action or tab not found). The optional *callup* assigns a Trio-style callup code to the created item (0 = no callup code).

Inspecting

- string **ListPlaylist** (string tabName = null)
  - Returns a JSON listing of the tab named *tabName* (when omitted, the currently selected tab). Each entry contains `{ uuid, label, type, path, duration, callup }`.

Removing

- bool **RemovePlaylistItem** (string uuidOrLabel)
  - Removes one or more rows. The argument is auto-detected: a UUID removes that single row (searched across all tabs); otherwise it is treated as a payload-item label and every matching item is removed. Returns true on success, false otherwise.
- bool **RemovePlaylistItem** (string label, string folder, string tabName = null)
  - Removes every payload item with the given *label*, scoped to *folder* within *tabName* (current tab when omitted). Returns true on success, false otherwise.
- bool **ClearPlaylistFolder** (string folder, string tabName = null)
  - Empties the folder named *folder* (the folder itself is kept). *tabName* targets the current tab when omitted. Returns true on success, false otherwise.
- bool **ClearPlaylist** (string tabName = null)
  - Empties the tab named *tabName*. When omitted, the currently selected tab.

Coloring

Override the color bar of an item or folder. By default a payload or action row's bar reflects its source action's color, and folders have no bar until one is set. In every method *hexColor* is `"#RRGGBB"` (or `"#AARRGGBB"` with alpha); pass an empty string or null to clear the override and revert to the action color. Color overrides are saved with the project. All of these methods return true on success, false otherwise.

- bool **SetPlaylistItemColor** (string uuidOrLabel, string hexColor)
  - Sets the color bar of one or more rows. The first argument is auto-detected: a UUID colors that single row; otherwise it is treated as a payload-item label and every matching item is colored.
- bool **SetPlaylistItemColor** (string label, string hexColor, string folder, string tabName = null)
  - As above, scoped to *folder* within *tabName* (current tab when omitted).
- bool **SetPlaylistFolderColor** (string folder, string hexColor, string tabName = null)
  - Sets the color bar of the folder named *folder*. *tabName* targets the current tab when omitted.

Coloring Example

```
// Highlight a specific row red, all "Messi" rows green, and a folder blue.
```

```
SetPlaylistItemColor(id, "#E03131")
SetPlaylistItemColor("Lionel Messi", "#2F9E44")
SetPlaylistFolderColor("Forwards", "#1971C2", "Match Day")

// Clear an override and revert to the source action's color.
SetPlaylistItemColor(id, "")
```

Playlist Authoring Example

```
// `squad` would typically come from a linked parameter, a REST/HTTP call, or MQTT.
var squad = [
    { name: "Alisson",         number: 1,  position: "Goalkeepers" },
    { name: "Virgil van Dijk", number: 4,  position: "Defenders" },
    { name: "Mohamed Salah",   number: 11, position: "Forwards" },
    // ...
]

function buildSquadPlaylist()
{
    var tab = "Match Day"
    CreatePlaylistTab(tab, false)   // ensure the tab exists, don't switch to it
    ClearPlaylist(tab)              // start clean so re-runs don't duplicate

    for (var i = 0; i < squad.length; i++)
    {
        var p = squad[i]

        // Populate the template's parameters (or linked control objects) for this
        // player, then snapshot the linked control-object values as the payload.
        SetParameterValue("playerName", p.name)
        SetParameterValue("shirtNo", p.number)
        var payload = GetTemplatePayload()

        // One item per player, filed under a per-position subfolder.
        CreatePlaylistItem(p.name, payload, p.position, tab)
    }
}

buildSquadPlaylist()
```

Callup Code

- int **NextFreeCallupCode** (int start = 1000)
    - Returns the next free callup code in the playlist, searching upwards from *start*.
- bool **SetPlaylistItemCallup** (string idOrLabel, int code, string path = "", string tabName = null)
    - Assigns the callup code *code* to the playlist item identified by UUID or label. The optional folder *path* and *tabName* narrow the search. Returns true on success.

4.1.4 Control Object

arc provides an alternative way of getting a BaseControlObject from a ControlObject/Blueprint action.

- BaseControlObject **GetControlObject** (ControlObjectAction action, string id)
    - Returns the control object with a specific ID from ControlObject action.
- BaseControlObject **GetControlObject** (UnrealBlueprintAction action, string id)
    - Returns the variable with name *id* of the given Blueprint action (used by the Blueprint example below).

Getting a Specific ControlObject from a ControlObjectAction

```
// Get the ControlObject Action
var MatchDayAction = GetAction("MatchdayTable")
// Get Title ControlObject (ControlText) and change its value
GetControlObject(MatchDayAction, "Title").Value = "Sunday Fixtures"


// Get the Blueprint Action
var HeadlineBp = GetAction("HeadlineBp")
// Get Title ControlObject (String Variable) and change its value
GetControlObject(HeadlineBp, "Title").Value = "Lorem Ipsum"
```

4.1.5 MIDI

Attached and configured MIDI devices can be used to receive MIDI events using the **OnMIDIEvent** callback. It's also possible to send MIDI events to an attached device using the following methods:

- bool **SendMIDIControlMessage** (string DeviceName, int Channel, int Number, int Value)
    - Sends a MIDI control message to a the device named DeviceName, using Channel, Number and Value.
- bool **SendMIDINoteMessage** (string DeviceName, bool On, int Channel, int Note, int Velocity)
    - Sends a MIDI note message to a the device named DeviceName, using Channel, Note and Velocity. The parameter **On** determines whether the event is a note on or note off event.

⚠ **Note:** Both of the methods above return true on successful completion and false if not successful.

MIDI Sample

```
Global.OnButtonPressed = function (id)
{
  SendMIDIControlMessage("Midi Fighter Twister", 1, 1, 127) // send control message
to Midi Fighter Twister on channel 1, number 1, value 127
  SendMIDINoteMessage("nanoPAD2", true, 1, 5, 100) // send note down event to
nanoPAD2 device
```

```
}

Global.OnMIDIEvent = function (midiEvent)
{
  // just print the midi event on the console
  Console.WriteLine("midi event |" + midiEvent.DeviceName + "| " +
midiEvent.EventType + "\n" + midiEvent.ToString())
}
```

4.1.6 Art-Net DMX

A sample on how to use the OnDMXEvent callback in a template or global script.

Art-Net Script Sample

```
Global.OnDMXEvent = function (dmxEvent)
{
  Console.WriteLine( "Universe " + dmxEvent.Universe + " first change at channel " +
dmxEvent.firstDiff )
  Console.WriteLine( "Channel 7 has changed: " + dmxEvent.HasChanged(7) )
  Console.WriteLine( "Channel 7 value is: " + dmxEvent.DMXData[7])
}
```

You can enable or disable the DMX signals using the following methods

- void **EnableDMX** ( )
  Enables callbacks of the connected Art-Net devices to be sent to connected actions and script callbacks.
- void **DisableDMX** ( )
  Disables callbacks of the connected Art-Net devices to be sent to connected actions and script callbacks.
- bool **IsDMXEnabled** ( )
  Returns whether the Art-Net callbacks are enabled.

4.1.7 MQTT

Message Queuing Telemetry Transport is supported through the possibility to instantiate a MQTT client and send/receive messages

- ArcMqttClient **createMQTTClient** (string server, int port, string protocolVersion = "3.1.1")
  - Creates an MQTT client connected to *server:port*. *protocolVersion* accepts `"3.1"`, `"3.1.1"` (default) or `"5.0"` — use `"5.0"` when connecting to MQTT 5 brokers.

The returned client ArcMqttClient supports the following methods

- void **Subscribe** (string topic, int qos = 0)
  Subscribes the client to the given topic with the specified quality of service (default 0).
- void **Unsubscribe** (string topic)
  Unsubscribes the client from the given topic.

- void **sendMessage** (string topic, string payload)
  Sends a message payloadto topic.
- void **Dispose** ( )
  Disconnects and deletes the client.

Whenever a message is received from a topic a client has subscribed to, the new data is set to the global DataMap using the topic as key and the payload as value. Payload data in JSON format is passed as a JSON object, anything else is passed as a string object.

MQTT Sample

```
var mqttClient

Global.OnInit = function ()
{
  mqttClient = createMQTTClient("localhost", 6548)
  mqttClient.Subscribe("hello/world/news")

  SubscribeDataMap("hello/world/news")
}

Global.OnDataMapValueChanged = function (varName)
{
  if( varName == "hello/world/news" )
    Console.WriteLine("breaking news alert: " + GetData(varName))
}
```

A sample server written in C# illustrating the server side code using the MQTTnet library.

MQTT Server Sample

```
using MQTTnet;
using MQTTnet.Extensions.ManagedClient;
using MQTTnet.Server;
using System;
using System.Threading;

namespace testmqtt
{
    class Program
    {
        static void Main(string[] args)
        {
            var optionsBuilder = new MqttServerOptionsBuilder()
             .WithConnectionBacklog(100)
             .WithDefaultEndpointPort(6548);

            var mqttServer = new MqttFactory().CreateMqttServer();
```

```
            mqttServer.StartAsync(optionsBuilder.Build());

            int i = 0;
            MqttApplicationMessage message = null;
            while (true)
            {
                message = new MqttApplicationMessageBuilder()
                        .WithTopic("hello/world/news")
                        .WithPayload("Temperatures below " + i + " !")
                        .WithExactlyOnceQoS()
                        .Build();

                mqttServer.PublishAsync(message);
                Thread.Sleep(1000);
                Console.WriteLine(i + "");
                i--;
            }
        }
    }
}
```

4.1.8 RabbitMQ

RabbitMQ is an open-source message broker that enables Viz Arc and other applications to communicate asynchronously by sending and receiving messages through queues.

Create a client

- ArcRabbitMqClient **createRabbitMQClient**(string host, int port = 5672, string username = ‘guest’, string password = ‘guest’, string virtualHost = '/')
  Creates a RabbitMQ client connecting to *host* on *port* using credentials *username* and *password* while *virtualHost* is the virtual host of the broker instance.

The following methods are support on an instance of **ArcRabbitMqClient**:

- void **DeclareQueue**(string queueName, bool durable = false, bool exclusive = false, bool autoDelete = false)
  - Creates queue if it doesn't exist.
- void **ConsumeQueue**(string queueName, bool autoAck = true, bool declareQueue = true)
  - Starts consuming. Messages are pushed to the DataMap with queueName as key.
- void **StopConsuming**(string queueName)
  - Stops consuming from the specified queue.
- void **PublishToQueue**(string queueName, string message)
  - Publishes message directly to a queue.
- void **Publish**(string exchange, string routingKey, string message)
  - Publishes to an exchange with routing key.
- bool **IsConnected** ()
  - Returns true while the client is connected to the broker.
- void **Dispose**()
  - Closes connection and releases resources.

Example usage:

```
var rmqClient = null;
var QUEUE_NAME = "vizarc_demo_queue";

Global.OnCreated = function() {
  // Create RabbitMQ client - connects to localhost with default credentials
  // Parameters: host, port (default 5672), username (default "guest"),
  //             password (default "guest"), virtualHost (default "/")
  rmqClient = createRabbitMQClient("localhost", 5672, "guest", "guest", "/");

  // Declare and start consuming from a queue
  // Messages will be pushed to the DataMap with queue name as the key
  rmqClient.ConsumeQueue(QUEUE_NAME);

  // Subscribe to DataMap to receive messages
  SubscribeDataMap(QUEUE_NAME);

  Console.WriteLine("RabbitMQ client connected and consuming from: " + QUEUE_NAME);
}

Global.OnDestroyed = function() {
  // Clean up when script stops
  if (rmqClient != null) {
      rmqClient.StopConsuming(QUEUE_NAME);
      rmqClient.Dispose();
      rmqClient = null;
  }
  Console.WriteLine("RabbitMQ client disconnected");
}

// Called when DataMap variable changes (message received)
Global.OnDataMapValueChanged = function(varName) {
  if (varName === QUEUE_NAME) {
      var message = GetData(QUEUE_NAME);
      Console.WriteLine("Received message: " + message);

      // Process the message here
      processMessage(message);
  }
}

function processMessage(message) {
  // Example: Parse JSON message and update a parameter
  try {
      var data = JSON.parse(message);
      if (data.action === "update") {
          SetParameterValue("myTextParam", data.value);
      }
  } catch (e) {
      Console.WriteLine("Message is not JSON: " + message);
  }
}
```

```
// Button click handler to publish a test message
Global.OnButtonPressed = function(param) {
  if (param === "btnPublish") {
      var testMessage = JSON.stringify({
          action: "update",
          value: "Hello from VizArc!",
          timestamp: new Date().toISOString()
      });

      // Publish to queue (uses default exchange with queue name as routing key)
      rmqClient.PublishToQueue(QUEUE_NAME, testMessage);
      Console.WriteLine("Published: " + testMessage);
  }
}
```

4.1.9 Object Tracker

The script exposes some useful functions that allows customization and remoting of the Object Tracker. For example, the **StopTracker** and **TakeOutTracker** function could be used to quickly remove tracking or On Air graphics.

- int **GetActiveTracker** ( )
  - Gets the currently active tracker index (starting from 1).
- int **SetActiveTracker** (int tracker)
  - Sets the currently active tracker index (starting from 1). Returns the active tracker index.
- void **TakeTracker** ( )
  - Takes tracker On Air all trackers.
- void **TakeTracker** (int trackerIndex)
  - Takes the tracker with the given index on air (1 = first tracker), without needing the action name.
- void **TakeOutTracker** ( )
  - Takes tracker Off Air all trackers.
- void **TakeOutTracker** (int trackerIndex)
  - Takes the tracker with the given index off air.
- void **PreviewTracker** ( )
  - Previews all trackers.
- void **PreviewOutTracker** ( )
  - Removes all trackers from preview.
- void **StopTracker** ( )
  - Stops all trackers.
- void **StopTracker** (int index)
  - Stops the tracker with index (starting from 1).
- void **ResetPointerOffset**(int index)
  - Reset the pointer offset for tracker with index (starting from 1).
- bool **ActivateSimpleTracking** (string trackingType, string filterType = "", int filterSize = -1)
  - Activates simple (non-DNN) tracking of the given type, optionally with a smoothing filter. Returns true on success.
- bool **ActivateDetectionAndTracking** (string dnnModel, float confidence, string trackingType, string filterType = "", int filterSize = -1, int filterBBSize = -1)

    - Activates DNN-based detection and tracking using the model dnnModel and the minimum detection confidence. Returns true on success.
- void **ResetStandings** ( )
    - Resets the tracker standings.
- void **SetAutoTracking** (int mode, int size, bool retake)
    - Configures automatic tracking.

4.1.10 Viz Arena

The script exposes some useful functions concerning the **Viz Arena** integration.

- bool **DetectArenaCalibration** ( )
    - Redetects the camera calibration (same as the **D** shortcut in Viz Arena).
- bool **ClearArenaCalibration** ( )
    - Clears the camera calibration (same as the **BACKSPACE** shortcut in Viz Arena).
- bool **ClearArenaKeyer** ( )
    - Clears the Keyer mask (same as the **C** shortcut in Viz Arena).
- string[] **GetArenaCameraList** ( )
    - Returns a string-list of available cameras.
- string **GetCurrentArenaCamera** ( )
    - Returns the name of the current camera.
- int **GetCurrentArenaCameraIndex** ( )
    - Returns the zero based index of the current camera.
- bool **SetArenaCamera** (int index)
    - Selects the Arena camera with the given index (see GetArenaCameraList). Returns true on success.
- string **GetCurrentArenaTimeCode** ( )
    - Returns the current Viz Arena timecode.
- bool **IsArenaConnected** ( )
    - Returns whether Viz Arena is running and connected to Viz Arc.

4.1.11 Parameter

All parameters are exposed to the global script and can be accessed via their unique ID.

arc provides an alternative way of getting them.

- BaseParameter **GetParameter** (string id)
    - Gets the parameter identified by the unique id that was input.

It's also possible to get and set a parameter's value directly from **arc**.

- dynamic **GetParameterValue** (string id)
    - Gets the value of the parameter identified by the unique id that was input. [dynamic] The returned value's type depends on the parameter type.
- void **SetParameterValue** (string id, dynamic value)
    - Sets the value of the parameter identified by the unique id that was input. [dynamic] Input variable value can be of any type, see parameters for valid types.

In case you don't want the callback function **OnParameterChanged** to be triggered when changing a value using SetParameterValue, you can use **SetParameterRawValue**. This method does not trigger any calls to **OnParameterChanged**.

- void **SetParameterRawValue** (string id, dynamic value)
    - Sets the value of the parameter identified by the unique id that was input. [dynamic] Input variable value can be of any type, see parameters for valid types.

Buttons are a special case in the sense that they don't hold a value, and therefore have a separate method for triggering their *click*.

- void **PushParameterButton** (string id)
    - Triggers a pressed event on the button identified by the unique id that was input.

Note: Button presses trigger the global script's callback **OnButtonPressed.**

Note: Parameter value changes trigger the global script's callback **OnParameterChanged.**

Parameter Examples

```
// Setting the value of a bool parameter (id = ShowHighlights) to false
// direct assigment
ShowHighlights.Value = false
// Get parameter via arc and then assign to Value
GetParameter("ShowHighlights").Value = false
// Set Parameter value via arc without interacting with the actual parameter
SetParameterValue("ShowHighlights", false)

// Getting the value of a bool parameter (id = ShowHighlights)
var highlightState = GetParameterValue("ShowHighlights")

// Push LoadFixtures button
PushParameterButton("LoadFixtures")
```

4.1.12 Channel

arc grants access to Viz Arc profiles. This is useful whenever more precise control is required for communicating with the Engines.

- ScriptingProfile **GetSelectedProfile** ( )
    - Returns the currently selected profile.
- int **GetChannelCount** ( )
    - Returns the number of channels on the currently selected profile.
- ScriptingChannel **GetChannel** (int index)
    - Returns the channel at the *index* position on the currently selected profile.
- ScriptingChannel **GetChannel** (string channelName)
    - Returns the channel named *channelName* on the currently selected profile.
- ScriptingChannel **GetPreviewChannel** ( )

    - Returns the preview channel of the currently selected profile.
- ScriptingChannel **GetProgramChannel** ( )
    - Returns the program channel of the currently selected profile.
- ScriptingChannel **GetSelectedChannel** ( )
    - In a template script it returns the currently selected channel of the Template Action. In the global script it returns the program channel of the currently selected profile.

Channel Handling Examples

```
// Clear main layer on all channels using GetChannelCount() and GetChannel(int)
for (var i = 0; i < GetChannelCount(); i++) {
  GetChannel(i).SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT")
}

// Send message to VideoWallchannel via GetSelectedProfile () and GetChannel(string)
GetChannel("VideoWall").SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT")
GetSelectedProfile().GetChannel("VideoWall").SendSingleCommand("RENDERER*MAIN_LAYER
SET_OBJECT")
```

Profile
The object returned by **GetSelectedProfile**():

- string **Name** [Get] — The profile's name.
- int **NumChannels** [Get] — The number of channels in the profile.
- ScriptingChannel **GetChannel** (int index) / ScriptingChannel **GetChannel** (string name)
    - Returns a channel by index or name (also available through the indexer, e.g. profile[0]).
- ScriptingEngine **VizEditingEngine** / **UnrealEditingEngine** / **FlowicsEditingEngine**
    - The profile's editing engines.
- ScriptingChannel **VizPreviewChannel** / **UnrealPreviewChannel** / **FlowicsPreviewChannel**
    - The preview channel per engine type.
- ScriptingChannel **VizProgramChannel** / **UnrealProgramChannel** / **FlowicsProgramChannel**
    - The program channel per engine type.

Engine
Individual engines are reached through a channel (**GetChannel**(...).**GetEngine**(...)) or the profile's editing-engine properties:

- void **SendCommands** (object[] commands)
    - Sends multiple commands to this engine.
- string **SendSingleCommand** (string command)
    - Sends a single command to this engine.
- string **QueryEngine** (string command) / string **QueryEngine** (string command, int timeout)
    - Sends command and returns the engine's answer, optionally with a timeout in milliseconds.
- async Task<string> **QueryEngineAsync** (string command) / **QueryEngineAsync** (string command, int timeout)
    - Awaitable versions of the above.
- string **GetFlowicsOutput** ( )

- Returns the Flowics output URL of this engine.

4.1.13 Viz Engine/Unreal Engine Communication

arc provides quick access functions for sending messages to specific channels/Engines.

- void **SendSingleCommand** (string command, string channelName)
    - Sends *command* to all the Engines in the specified channel channelName.
- void **SendMultipleCommands** (string[] commands, string channelName)
    - Sends all the input *commands* to all the Engines in the specified channel *channelName*.
- string **GetFromEngine** (string command, string channelName)
    - Sends *command* to all the Engines in the specified channel *channelName*. Returns the answer to the sent *command*.
- string **GetFromVizEngine** (string command)
    - Sends *command* to the currently selected profile's Viz **editing Engine**. Returns the answer to the sent *command*.
- string **GetFromUnrealEngine** (string command)
    - Sends *command* to the currently selected profile's Unreal **editing Engine**. Returns the answer to the sent *command.*
- string **GetFromEngineAsync** (string command, string channelName, int timeout = -1)
    - Sends *command* to all the Engines in the specified channel *channelName*. Returns the answer to the sent *command*.
      If timeout is specified and larger than 0, the method times out after *timeout* milliseconds if it does not receive an answer within that time.
- string **GetFromEngineAsync** (string command, int timeout = -1)
    - Sends *command* to all the Engines in the specified selected channel of the template. Returns the answer to the sent *command*.
      If timeout is specified and larger than 0, the method times out after *timeout* milliseconds if it does not receive an answer within that time.
- string **GetFromVizEngineAsync** (string command)
    - Sends *command* to the currently selected profile's Viz **editing Engine**. Returns the answer to the sent *command*.
- string **GetFromUnrealEngineAsync** (string command)
    - Sends *command* to the currently selected profile's Unreal **editing Engine**. Returns the answer to the sent *command*.

Viz Engine Communication

```
// Get scene from parameter and set it to Viz Engine main layer
SendSingleCommand(GetParameterValue("MainSceneSelector"), "Main")

// Clear Main, Back and Front layers on channel
var CleanCommands = ["RENDERER*MAIN_LAYER SET_OBJECT", "RENDERER*BACK_LAYER
SET_OBJECT", "RENDERER*FRONT_LAYER SET_OBJECT"]
SendMultipleCommands(CleanCommands, "Viz")
```

```
// Query Viz channel and Viz editing engine for the currently loaded scene
GetFromEngine("SCENE SCENE*SCENE GET", "Viz")
GetFromVizEngine("SCENE SCENE*SCENE GET")
```

The **async** variants can be used only when using the JavaScript language and have the advantage that they do not lock up the UI.

Async Samples

```
Global.OnButtonPressed = async function (id)
{
    if( id == "getVersionButton" ){
        const answer = await GetFromEngineAsync("VERSION", "localviz")
        Console.WriteLine("Viz Version is: " + answer)
    }
}
```

Note: If you use **await**, the enclosing function needs to be **async**. You can add this attribute manually in case you use it within a Viz Arc callback.

4.1.14 Tracking Hub Command

arc provides quick access functions for sending messages to the configured Tracking Hub.

- void **SendSingleTHCommand** (string command)
    - Sends *command* to the Tracking Hub (if configured and connected).
- string **GetFromTH** (string command)
    - Sends *command* to the Tracking Hub (if configured and connected) and returns the answer.
- string **GetFromTHAsync** (string command)
    - The asynchronous version of **GetFromTH**.

4.1.15 SMM Handling

- void **SendToSMM** (string key, string value, bool doEscape)
    - Sends key-value pair to Shared Memory to the first channel of the current profile. *doEscape* specifies whether the value string is escaped.
- void **SendToSMM** (string key, string value, bool doEscape, string channel)
    - Sends key-value pair to Shared Memory to all Engines contains in *channel*. *doEscape* specifies whether the value string is escaped.
- void **SendToSMM** (string key, string value, bool doEscape, string channel, string destination)
    - Sends key-value pair to Shared Memory to all Engines contains in *channel*. *doEscape* specifies whether the value string is escaped.
    - *destination* can be either SYSTEM, COMMUNICATION or DISTRIBUTED

The shared memory updates are sent to the UDP or TCP port configured on the target Viz Engine; if both are configured, it is sent to the UDP port. The Viz Communication Shared Memory map is therefore utilized. You can read more on Shared Memory configuration in the Profiles section in the *Viz Arc User Guide*.

SMM Example

```
// Send to Viz Channel SMM the variable "Target1" with the value from TargetState
SendToSMM("Target1", TargetState.Value, false, "Viz")

// Send to Viz Channel SMM the variable "Target2" with the value "Hello World!".
// The last parameter "DISTRIBUTED" indicates that the value will be propagated to
all engines connected to the same Graphic Hub
SendToSMM("Target2", "Hello World!", false, "Viz", "DISTRIBUTED")
```

4.1.16 GPI

The connected GPI state can be changed via the **arc** functionalities:

- void **SignalGpiChannel** (int channelIndex, bool signalHigh)
  - Signals set the GPI channel at *channelIndex* to either high or low.

The following snippet presents a function that loads a scene to the "Main" channel and signals the GPI:

GPI Example

```
function LoadScene()
{
    // Get scene from parameter and set it to Viz Engine main layer
    SendSingleCommand(GetParameterValue("MainSceneSelector"), "Main")
    // Set gpi channel 2 to High
    SignalGpiChannel(2, true)
}
```

Note: GPI must be enabled on the config.

4.1.17 Viz Pilot

Viz Pilot data elements can be created from scripting using the method *CreatePilotDataElement*. There are two ways to invoke this method:

- async bool **CreatePilotDataElement** (BaseAction action, string name)
  - Creates a Viz Pilot data element of *action* and name *name*. The function returns true on success.
- The BaseAction method:
  - async bool **CreatePilotDataElement** (string name)
    - Creates a Viz Pilot data element with name *name.* The function returns true on success.

```
Global.OnButtonPressed = async function (id)
{
    try{
      if( id == "createpilotButton" )
      {
          // use global method
          await CreatePilotDataElement(GetAction("loadSceneA"), "LoadSceneA")
          // use method defined on the BaseAction itself
          await
GetAction("loadSceneA").CreatePilotDataElement("LoadSceneA_fromAction")
      }
    }catch(ex)
    {
        // error handle
        Console.WriteLine("ex " + ex + " " + ex.stack)
    }
}
```

4.1.18 Timer

- void **CreateTimer** (string id)
    - Creates a timer that can be accessed via its unique *id*.
- void **CreateTimer** (string id, int ms)
    - Starts a timer that can be accessed via its unique *id* and has a tick interval of *ms*.
- void **StartTimer** (string id, int ms)
    - Gets the timer identified by *id*, sets the tick interval to *ms* and starts it.
- void **StopTimer** (string id)
    - Gets the timer identified by *id* and stops it.

The following example creates a timer on the OnInit callback, makes use of two buttons to start/stop the timer and writes to the console whenever the timer ticks:

Timer Example

```
// Timer id
var heartBeatTimerId = "HeartBeat"

Global.OnInit = function ()
{
  // Create timer with id heartBeatTimerId
  CreateTimer(heartBeatTimerId)
}

Global.OnButtonPressed = function (id)
{
  if(id == "TimerStart")
    StartTimer(heartBeatTimerId, 1000)
  else if(id == "TimerStop")
```

```
    StopTimer(heartBeatTimerId)
}

// Script callback for timer ticks
Global.OnTimer = function (id)
{
  Console.WriteLine("Timer Tick " + id)
}
```

Note: Whenever a timer ticks the global script's callback **OnTimer** is called.

JavaScript Timer Functions

Viz Arc provides browser-style timer functions that allow you to execute code after a delay or at regular intervals. These functions mimic the standard JavaScript timer API found in web browsers, making it familiar for developers with web development experience.

Overview

The timer functions allow you to:

- Execute code once after a specified delay (`setTimeout`).
- Execute code repeatedly at fixed intervals (`setInterval`).
- Cancel pending or repeating timers (`clearTimeout` , `clearInterval`).

All timer functions return a unique timer ID that can be used to cancel the timer before it executes.

---

setTimeout

Executes a function once after a specified delay.

Syntax:

```
int setTimeout(function callback, int delay)
```

Parameters:

- `callback` : The function to execute after the delay
- `delay` : Time in milliseconds to wait before executing the function

Returns:

- `int` : A unique timer ID that can be used with `clearTimeout()`

Example - Basic Usage:

```
Global.OnButtonPressed = function (id)
{
    if (id == "delayedAction")
```

```
    {
        Console.WriteLine("Action will execute in 3 seconds...")
        setTimeout(function() {
            Console.WriteLine("3 seconds have passed!")
            SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT Scene1", "Main")
        }, 3000)
    }
}
```

Example - Using Named Functions:

```
function showGraphic()
{
    Console.WriteLine("Displaying graphic")
    GetSelectedChannel().SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT 
SCENE*some/scene/l3rd")
    SetData("lastUpdate", Date().toString())
}

Global.OnButtonPressed = function (id)
{
    if (id == "delayShow")
    {
        setTimeout(showGraphic, 5000) // Show graphic after 5 seconds
    }
}
```

Example - Storing Timer ID for Cancellation:

```
var pendingTimerId = null

Global.OnButtonPressed = function (id)
{
    if (id == "scheduleUpdate")
    {
        // Schedule an update for 10 seconds from now
        pendingTimerId = setTimeout(function() {
            Console.WriteLine("Executing scheduled update")
            GetParameter("StatusText").Value = "Updated"
        }, 10000)

        Console.WriteLine("Update scheduled with timer ID: " + pendingTimerId)
    }
    else if (id == "cancelUpdate")
    {
        if (pendingTimerId != null)
        {
            clearTimeout(pendingTimerId)
            Console.WriteLine("Scheduled update cancelled")
            pendingTimerId = null
        }
```

```
    }
}
```

clearTimeout

Cancels a timer created with `setTimeout` before it executes.

Syntax:

```
void clearTimeout(int timerId)
```

Parameters:

- `timerId` : The timer ID returned by `setTimeout()`

Example - Cancelling a Scheduled Action:

```
var countdownTimer = null

function startCountdown()
{
    Console.WriteLine("Graphics will load in 10 seconds...")
    countdownTimer = setTimeout(function() {
        GetSelectedChannel().SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT 
SCENE*Countdown")
        Console.WriteLine("Graphics set!")
    }, 10000)
}

function cancelCountdown()
{
    if (countdownTimer != null)
    {
        clearTimeout(countdownTimer)
        Console.WriteLine("Countdown cancelled")
        countdownTimer = null
    }
}

Global.OnButtonPressed = function (id)
{
    if (id == "btnStart")
        startCountdown()
    else if (id == "btnCancel")
        cancelCountdown()
}
```

setInterval

Executes a function repeatedly at a fixed time interval until cancelled.

Syntax:

```
int setInterval(function callback, int interval)
```

Parameters:

- `callback` : The function to execute repeatedly.
- `interval` : Time in milliseconds between each execution.

Returns:

- `int` : A unique interval ID that can be used with `clearInterval()` .

Example - Updating a Clock:

```
var clockIntervalId = null

function updateClock()
{
    var currentTime = new Date().toLocaleTimeString()
    SetParameterValue("ClockDisplay", currentTime)
    SetData("currentTime", currentTime)
}

Global.OnButtonPressed = function (id)
{
    if (id == "startClock")
    {
        // Update clock every second
        clockIntervalId = setInterval(updateClock, 1000)
        Console.WriteLine("Clock started")
    }
    else if (id == "stopClock")
    {
        if (clockIntervalId != null)
        {
            clearInterval(clockIntervalId)
            Console.WriteLine("Clock stopped")
            clockIntervalId = null
        }
    }
}
```

Example - Periodic Status Check:

```
var statusCheckId = null
```

```
function checkEngineStatus()
{
    var version = GetFromEngine("VERSION", "Main")
    Console.WriteLine("Engine Status Check: " + version)

    // Update status display
    SetParameterValue("EngineStatus", "Online - " + version)
}

Global.OnCreated = function()
{
    // Check engine status every 3 seconds
    statusCheckId = setInterval(checkEngineStatus, 3000)
}
```

Example - Animation Loop:

```
var animationId = null
var counter = 0

function animateCounter()
{
    counter++
    SetParameterValue("CounterValue", counter)

    // Stop after reaching 100
    if (counter >= 100)
    {
        clearInterval(animationId)
        Console.WriteLine("Animation complete")
        animationId = null
    }
}

Global.OnButtonPressed = function (id)
{
    if (id == "startAnimation")
    {
        counter = 0
        animationId = setInterval(animateCounter, 50) // Update every 50ms
    }
}
```

clearInterval

Cancels a repeating timer created with `setInterval`.

Syntax:

```
void clearInterval(int intervalId)
```

Parameters:

- `intervalId` : The interval ID returned by `setInterval()`

Example - Starting and Stopping a Refresh Loop:

```
var refreshId = null

function refreshData()
{
    Console.WriteLine("Refreshing data from API...")
    // Fetch and update data here
    var timestamp = Date().toString()
    SetData("lastRefresh", timestamp)
}

Global.OnButtonPressed = function (id)
{
    if (id == "btnStartRefresh")
    {
        if (refreshId == null)
        {
            refreshData() // Execute immediately
            refreshId = setInterval(refreshData, 5000) // Then every 5 seconds
            Console.WriteLine("Auto-refresh enabled")
        }
    }
    else if (id == "btnStopRefresh")
    {
        if (refreshId != null)
        {
            clearInterval(refreshId)
            refreshId = null
            Console.WriteLine("Auto-refresh disabled")
        }
    }
}
```

Best Practices

1. **Always Store Timer IDs**

```
// Good - can be cancelled later
var timerId = setTimeout(myFunction, 1000)

// Avoid - cannot be cancelled
setTimeout(myFunction, 1000)
```

2. **Avoid Memory Leaks**

```
var updateInterval = null

Global.OnButtonPressed = function (id)
{
    if (id == "startUpdates")
    {
        // Clear existing interval before creating a new one
        if (updateInterval != null)
            clearInterval(updateInterval)

        updateInterval = setInterval(updateFunction, 1000)
    }
}
```

3. **Handle Long-Running Intervals**

```
var pollCount = 0
var pollId = null

function pollAPI()
{
    pollCount++

    // Implement a maximum number of attempts
    if (pollCount > 100)
    {
        clearInterval(pollId)
        Console.WriteLine("Polling stopped after 100 attempts")
        return
    }

    // Your polling logic here
    Console.WriteLine("Polling... attempt " + pollCount)
}

pollId = setInterval(pollAPI, 100)
```

Complete Example - Countdown Timer

```
var countdownValue = 10
var countdownId = null

function countdown()
{
    if (countdownValue > 0)
    {
```

```
        SetParameterValue("CountdownDisplay", countdownValue.toString())
        Console.WriteLine("Countdown: " + countdownValue)
        countdownValue--
    }
    else
    {
        // Countdown finished
        clearInterval(countdownId)
        countdownId = null

        SetParameterValue("CountdownDisplay", "GO!")
        Console.WriteLine("Countdown complete!")

        // Execute action after countdown
        SendSingleCommand("RENDERER*MAIN_LAYER SET_OBJECT LaunchGraphic", "Main")
    }
}

Global.OnButtonPressed = function (id)
{
    if (id == "btnStartCountdown")
    {
        countdownValue = 10
        countdown() // Show initial value immediately
        countdownId = setInterval(countdown, 1000) // Then update every second
    }
    else if (id == "btnStopCountdown")
    {
        if (countdownId != null)
        {
            clearInterval(countdownId)
            countdownId = null
            Console.WriteLine("Countdown stopped")
        }
    }
    else if (id == "btnResetCountdown")
    {
        if (countdownId != null)
        {
            clearInterval(countdownId)
            countdownId = null
        }
        countdownValue = 10
        SetParameterValue("CountdownDisplay", countdownValue.toString())
    }
}
```

Performance Considerations

Warning: Too many timers or short intervals can cause UI freezing and performance issues.

Recommended Intervals

```
// ❌ BAD
setInterval(heavyOperation, 10)

// ✅ GOOD
setInterval(lightUpdate, 100)    // Light: 100ms minimum
setInterval(apiCall, 1000)        // Heavy: 1 second minimum
setInterval(polling, 5000)        // Polling: 5+ seconds
```

Best Practices

Limit Active Timers (< 15 per template)

```
// ❌ Multiple timers
for (var i = 0; i < 10; i++) setInterval(update, 1000)

// ✅ Single timer
var i = 0
setInterval(function() { update(i++ % 10) }, 1000)
```

Use setTimeout for Slow Operations

```
// ❌ Overlapping calls
setInterval(slowAPICall, 2000) // Takes 3 seconds!

// ✅ Wait for completion
function poll() {
    slowAPICall()
    setTimeout(poll, 2000)
}
```

Quick Checklist

- ✅ Intervals ≥ 100ms (1000ms+ for heavy operations)
- ✅ Keep total timers under 15
- ✅ Callbacks faster than interval duration

Notes

- These functions mimic the standard JavaScript timer API found in web browsers, making them familiar to web developers
- All timers are automatically cleaned up when the script is stopped or reloaded
- Timer callbacks are executed on the UI thread for safe interaction with Viz Arc parameters and UI elements
- Delays and intervals are specified in milliseconds (1000ms = 1 second)

- If an error occurs in a `setInterval` callback, the interval is automatically stopped to prevent repeated errors
- Unlike in browsers, additional arguments after the delay are not forwarded to the callback, and the *delay* argument is required.

4.1.19 StreamDeck

Any connected StreamDeck that is configured to be used exclusively with **arc**, can have its buttons customized using one of the following methods:

- void **SetStreamdeckKey** (int key, string label, int fontSize)
  - Baseline version, sets streamdeck key at *key index* image to a black square with *label* text of *fontSize* size.
- void **SetStreamdeckKey** (int key, string label, int fontSize, string imageFullPath)
  - Same as the baseline version but instead of a black block it sets a local image (at *imageFullPath*) as background. *imageFullPath* can be either a local file system path or a Graphic Hub path.
- void **SetStreamdeckKey** (int key, string label, int fontSize, int r, int g, int b)
  - Same as the baseline version but instead of black it uses an RGB color as background.
- void **SetStreamdeckKey** (int key, string label, int fontSize, int r, int g, int b, string imageName)
  - Same as baseline version using background color *r, g, b* and *imageName* on top of the background color (in case the image contains an alpha channel).
- void **SetStreamdeckKey** (int key, string label, int fontSize, string horAlignment, string vertAlignment, string textAlignment, int r, int g, int b, string imageName)
  - Same as the previous version, where text is *horAlignment* aligned horizontally, vertically by *vertAlignment* and the text itself is centerd through *textAlignment.*
    - hor*Alignment* can be either "Left", "Center" or "Right"
    - vert*Alignment* can be either "Top", "Center" or "Bottom"
    - *textAlignment* can be either "Left", "Center" or "Right"

Any key can have its contents cleared with the following method:

- void **ClearStreamdeckKey** (int key)
  - Clears the content of the Streamdeck key at key *index*

StreamDeck Key Configuration Example

```
function SetupStreamDeck()
{
    // Key 0: Black background, size 20 "Clear" text
    SetStreamdeckKey(0, "Clear", 20)
    // Key 1: Image background, size 20 "Load AR" text
    SetStreamdeckKey(1, "Load AR", 20, "D:/Soccer/Images/ARThumbnail.png")
    // Key 2: Blue background, size 20 "Continue" text
    SetStreamdeckKey(2, "Continue", 20, 0, 0, 255)
    // Key 3: Gray background, using headshot from Graphic Hub (image may contain an
alpha channel)
```

```
    SetStreamdeckKey(3, " ", 20, 100, 100, 100, "IMAGE*/Default/MasterImages/
headshot_0123")
   // Key 4: Gray background, using headshot from Graphic Hub (image may contain an
alpha channel), Text "John Doe" is top left aligned
    SetStreamdeckKey(4, "John Doe", 20, "Left", "Top", "Left", 100, 100, 100,
"IMAGE*/Default/MasterImages/headshot_0123")
}

Global.OnInit = function () {
    // Clean first 3 keys
    ClearStreamdeckKey(0)
    ClearStreamdeckKey(1)
    ClearStreamdeckKey(2)

    SetupStreamDeck()
}
```

Another sample that prints some useful information using the **external StreamDeck plugin**.

```
function printSDEvenyInfo(sdEvent)
{
    // print all the available information for a StreamDeck event
    Console.WriteLine("StreamDeck Event Info:")
    Console.WriteLine("Event Type:    " + sdEvent.EventType)
    Console.WriteLine("Device Index: " + sdEvent.DeviceIndex)
    Console.WriteLine("Device ID:     " + sdEvent.Id )
    Console.WriteLine("Column:        " + sdEvent.XKey )
    Console.WriteLine("Row:           " + sdEvent.YKey )
    Console.WriteLine("Payload:       " + sdEvent.Payload )

    // if it's a dial or touch event print additional info
    // for StreamDeck + devices only
    if( sdEvent.hasOwnProperty("Ticks") )
        Console.WriteLine("Ticks:         " + sdEvent.Ticks )
    if( sdEvent.hasOwnProperty("TapPosX") )
        Console.WriteLine("TapPosX:       " + sdEvent.TapPosX )
    if( sdEvent.hasOwnProperty("TapPosY") )
        Console.WriteLine("TapPosY:       " + sdEvent.TapPosY )
}

Global.OnStreamDeckTouchTap = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
Global.OnStreamDeckDialRotate = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
Global.OnStreamDeckDialDown = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
```

```
Global.OnStreamDeckDialUp = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
Global.OnStreamDeckKeyDown = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
Global.OnStreamDeckKeyUp = function (sdEvent)
{
    printSDEvenyInfo(sdEvent)
}
```

4.1.20 Graphic Hub REST

arc provides some methods that allow to retrieve information about the current Graphic Hub REST server in use. It is meant to help using the Graphic Hub REST interface directly.

- string **GetGHHost** ( )
  - Returns the Graphic Hub REST host name (for example, *localhost or 10.81.44.71*).
- string **GetGHPort** ( )
  - Returns the Graphic Hub REST port (for example, `19398`).
- string **GetGHConnectionString** ( )
  - Returns the complete connection string based on the configured Host and Port (for example, *http://localhost:* `19398`).
- string **GetGHUser** ( )
  - Returns the Graphic Hub REST user name (for example, *Guest* or *Admin*).
- string **GetGHAuthenticationValue** ( )
  - Returns the base64 authentication string which is a combination of the user name and password (for example, *QWRtaW46VmI6RGI=*).
- bool **ImportArchive** (string path)
  - Import a via archive through the REST service. Beware that all assets in the via override the content's of the Graphic Hub. Returns true on success and false on failure.
- async Task<bool> **ImportArchiveAsync** (string path)
  - Async version of the above method.

Below is a code sample that fetches all the image names of a given Graphic Hub path using the **GetGHConnectionString** and **GetGHAuthenticationValue** functions.

Sample

```
function getFolderId( path )
{
    let folderId = ""
    let request = new XMLHttpRequest()

    request.onreadystatechange = function() {
```

```
        if (request.readyState == 4 && request.status == 200 ) {
            //Console.WriteLine("respones: " + request.responseText)

            xmlDoc = new XmlDocument()
            xmlDoc.LoadXml(request.responseText)
            //Console.WriteLine("nodes " + xmlDoc.ChildNodes.Count)

            // create namespace manager
            nsmgr = new XmlNamespaceManager(xmlDoc.NameTable)
            // add namespace
            nsmgr.AddNamespace("x", "http://www.w3.org/2005/Atom")

            // search for x:model
            root = xmlDoc.DocumentElement
            folderId=root.SelectSingleNode("/x:feed/x:entry/x:id",
nsmgr).InnerXml.split(':')[2]
            Console.WriteLine("folder id " + folderId)
        }
    }
    request.open("GET", GetGHConnectionString()+"/translator/?path="+path, true)
    request.setRequestHeader("Authorization", "Basic " + GetGHAuthenticationValue())
    request.send();

    // fetch the images using the folder uuid
    GetImagesOfFolder(folderId)
}

function GetImagesOfFolder(folderId)
{
    let request = new XMLHttpRequest()

    request.onreadystatechange = function() {
        if (request.readyState == 4 && request.status == 200 ) {
            //Console.WriteLine("respones: " + request.responseText)

            xmlDoc = new XmlDocument()
            xmlDoc.LoadXml(request.responseText)
            //Console.WriteLine("nodes " + xmlDoc.ChildNodes.Count)

            // create namespace manager
            nsmgr = new XmlNamespaceManager(xmlDoc.NameTable)
            // add namespace
            nsmgr.AddNamespace("x", "http://www.w3.org/2005/Atom")

            // search for nodes 'entry'
            root = xmlDoc.DocumentElement
            imageNodes=root.SelectNodes("/x:feed/x:entry", nsmgr)
            //Console.WriteLine("nodes " + imageNodes.Count)

            let imageList = []

            for( node of imageNodes )
            {
```

```
                Console.WriteLine("image " + node.SelectSingleNode("./x:title",
nsmgr).InnerXml)
                imageList.push(node.SelectSingleNode("./x:title", nsmgr).InnerXml)
            }
            // set the dropdown
            // imagesDD.SetItems(imageList)
            }
    }
    request.open("GET", GetGHConnectionString()+"/files/" + folderId + "/?
term=IMAGE", true)
    request.setRequestHeader("Authorization", "Basic " + GetGHAuthenticationValue())
    request.send()
}
```

Other functions relative to the Graphic Hub:

- async Task<string[]> **GetImages**(path)
  - Returns a string list of images in the Graphic Hub.

```
async function fetchImages(path)
{
    Console.WriteLine("fetching images in " + path)

    const results =  await GetImages(path).then(results =>
    {
        dropdown_0.SetItems(results)
    })
}

Global.OnButtonPressed = function (id)
{
    if( id == "button_0" )
    {
        dropdown_0.Clear()
        fetchImages("sports/soccer/headshots")
    }
}
```

The above code snippet populates a dropdown with the image names contained in the Graphic Hub path *sport/soccer/headshots*.

4.1.21 DataMap

arc provides an interface (get and set) for interacting with Viz Arc's DataMap:

- dynamic **GetData** (string varName)
  - Returns the value belonging to the variable named *varName.* [dynamic] Returned value depends on what was set to *varName*.
- void **SetData** (string varName, dynamic value)
  - Inserts (or overwrites if varName already exists) *the key:value* pair into Viz Arc's DataMap. [dynamic] Input *value* can be of any type.

- void **SetDataAsync** (string varName, dynamic value)
  - Same as **SetData**, but returns immediately and applies the change asynchronously. Useful when setting many keys from time-critical code.
- bool **HasData** (string varName)
  - Returns true if varName exists in the DataMap.
- bool **DeleteDataMapKey**(string varName)
  - Removes a variable from the DataMap. Returns true if the key existed and was deleted.
- void **SubscribeDataMap** (string variableName)
  - Subscribes to a specific key (Empty string subscribes to all changes). The subbed variables feedback triggers the script callback "OnDataMapValueChanged".
- void **UnsubscribeDataMap** (string variableName)
  - Unsubscribes from a specific key (Empty string unsubscribes to all changes).
- string[] **GetDataKeys** ( )
  - Returns a complete list of all DataMap key entries.

DataMap Example

```
Global.OnInit = function ()
{
  // make sure OnDataMapValueChanged is called when "someData" changes
  SubscribeDataMap("someData")

  // use blank string to subscrive to all DataMap changes
  //SubscribeDataMap("")

  // create a timer that triggers every second
  CreateTimer("aTimer")
  StartTimer("aTimer", 1000)
}

// Callback for DataMap changes
Global.OnDataMapValueChanged = function (varName)
{
  if(varName == "someData")
    UpdateSomeData(GetData(varName))
}

function UpdateSomeData( theData )
{
  // do something here
  Console.WriteLine( "new data " + theData )
}

Global.OnTimer = function (id)
{
  // generate some fresh data using the current time for testing
  // such that OnDataMapValueChanged gets called
  if( id == "aTimer" )
```

```
    SetData("someData", Date.now())
}

function printDataMap()
{
  var keys = GetDataKeys()

  for( k of keys )
    Console.WriteLine(k + " = " + GetData(k) )
}
```

Note: Whenever a DataMap variable changes the global script's callback **OnDataMapValueChanged** is called.

4.1.22 NDI

arc provides an interface to handle metadata feedback from NDI sources.

- string[] **GetNDISourceList** ( )
  - Returns an array with all the names of the available NDI sources.
- string[] **GetNDIPTZSourceList** ( )
  - Returns an array with all the names of the available NDI sources with PTZ control capabilities.
- bool **SubscribeNdiSourceMetadata** (string source)
  - Subscribes to the metadata feedback on the NDI source identified by the provided source input. The feedback is sent to the datamap with key equal to the source name. Returns true on success, false otherwise.
- bool **UnsubscribeNdiSourceMetadata** (string source)
  - Unsubscribes the NDI feedback. Returns true on success, false otherwise.
- bool **SendNDIMetadata** (string name, string XMLString)
  - Sends a **XMLString** to a source identified by **name**. Returns true on success, false otherwise.

DataMap Example

```
Global.OnInit = function ()
{
  // Get a list of available NDI sources (can take some time to update)
  var sources = GetNDISourceList()

  // subscribe to metadata changes on a ndi stream
  SubscribeNdiSourceMetadata("NEWTEKPTZ (Channel 1)")

  // metadata will be written into the DataMap, so register to the DataMap changes
also
  SubscribeDataMap("NEWTEKPTZ (Channel 1)")
}

Global.OnDataMapValueChanged = function (varName)
```

```
{
  Console.WriteLine(varName + " changed")
  // NDI metadata is typically in xml format
  Console.WriteLine(GetData(varName).ToString())
}

Global.OnParameterChanged = function (id)
{
  if( id == "sendMetadata")
  {
    SendNDIMetadata("NEWTEKPTZ (Channel 1)", "<?xml version='1.0' encoding='UTF-8'?
><camera_control><command group_id='0' parameter_id='3' value='0.43'></
camera_control>")
  }
}
```

4.1.23 File Handling

- string **ReadTextFile** (string filename, string encoding = "UTF8")
    - Returns a *encoding* encoded string containing the whole content of the text file.
- bool **WriteTextFile** (string FullPath, string data, string encoding = "UTF8")
    - Writes a file at *FullPath* with its content equal to the encoded input *data*.
- string **GetApplicationDirectory** ( )
    - Returns the directory the Viz Arc executable runs from.

Valid encodings: "UTF8", "ASCII", "BigEndianUnicode", "Default" [System defined encoding], "UTF32", "UTF7"

File Handling Example

```
// Get the StartList file content from the directory defined by the Directory
parameter "WorkingDir"
ReadTextFile( WorkingDir.Value + "\\StartList.json")

// Write the results to the directory defined by the Directory parameter "WorkingDir"
WriteTextFile( WorkingDir.Value + "\\RaceResults.json", results)
```

4.1.24 Logging

- bool **AddLog** (string fileName, string content)
    - Appends *content* to *fileName*, and returns true on success, false otherwise. If the file *fileName* does not exist, it is created.

```
// register data map changes somewhere (e.g. SubscribeDataMap(""))
Global.OnDataMapValueChanged = function (varName)
```

```
{
    AddLog("c:/tmp/templateLog.txt", "getting data " + varName + " = " +
GetData(varName))
}
```

The produced log file content contains entries as the example below:

`2025/07/08 19:20:34.1389|getting data TIMECODE = 02:55:59:43`

4.1.25 JSON

- dynamic **ParseJson** (string data)
  - Deserializes the input *data* and returns a JSON object if successful.

On the returned JSON object you can access the members directly using their name. Use the "ToString()" method on any of the objects to convert them to strings.

```
var json = ParseJson("{time: '1994-11-05T13:15:30Z', title: 'Viz Arc', subtitle:
'Vizrt', messageId: 1}")
Console.WriteLine("the whole json " + json.ToString())
Console.WriteLine("the title is " + json.title.ToString())
```

When using the V8 Scripting Engine the built-in **JSON.parse** and **JSON.stringify** methods can be used

```
var json = JSON.parse('{"time": "1994-11-05T13:15:30Z", "title": "Viz Arc",
"subtitle": "Vizrt", "messageId": 1}')
Console.WriteLine("the whole json " + JSON.stringify(json))
Console.WriteLine("the title is " + json.title)
```

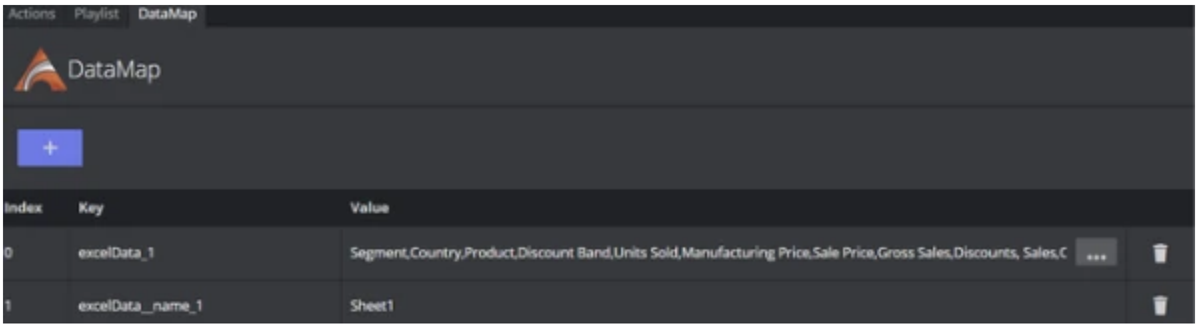
4.1.26 Excel

- bool **convertXLSToCSV** (string excelFilePath, string csvOutputFile, string separator = "\t", int worksheetNumber = 1)
  - Converts an existing *.xls* file *excelFilePath* to a comma separated CSV file *csvOutputFile* usign *separator* (default tab separator) and using worksheet number *worksheetNumber* (1 default being the first worksheet in the Excel file).
  - Returns true on successful conversion, false otherwise.
- string **convertXLSToCSVString** (string excelFilePath, string separator = "\t", int worksheetNumber = 1)
  - Converts an existing *.xls* file *excelFilePath* to a comma separated CSV string using *separator* (default tab separator) and using worksheet number *worksheetNumber* (1 default being the first worksheet in the Excel file).
  - On successful conversion the function returns the CSV output as a string.
- void **convertXLSToCSVDataMap** (string excelFilePath, string dataMapPrefix, string separator = "\t", int fromSheet = 1, int toSheet = -1)
  - Converts the worksheets of the Excel file *excelFilePath* and writes each worksheet's content as CSV to the DataMap key `dataMapPrefix<index>` (1-based worksheet index). The worksheet's name is

written to the key `dataMapPrefix_name_<index>` . Use *fromSheet*/*toSheet* to limit the range (-1 = all worksheets).

```
let separator = ";"
// convert excel to CSV file, use ; as separator and read the second sheet
convertXLSToCSV("c:/tmp/ExcelData.xlsx", "c:/tmp/ExcelData.csv", separator, 2)
// read the whole csv file into a string
var fileContent = ReadTextFile("c:/tmp/ExcelData.csv")
var EntryArr = fileContent.split("\n")

// First line is for the headers, ignore it
for(i = 0; i < EntryArr.length; i++)
{
    // split the row
    var spl = EntryArr[i].split(separator)
    if( spl.length <= 1 )
        continue;

    Console.WriteLine("row " + i + ":")

    // print colums one by one separated by a whitespace
    for( entry of spl )
        Console.Write( entry.trim() + " " )

    Console.WriteLine("")
}

// another writing all worksheets to the DataMap, use separator ","
convertXLSToCSVDataMap("c:/tmp/ExcelData.xlsx", "excelData_", ",")
```

How the resulting **DataMap** might look like after calling *convertXLSToCSVDataMap* when the excel file contains just one worksheet.



4.1.27 Callbacks

- **OnParameterChanged** (string parameterID)

  - Called whenever a parameter (except button and table) changes. *parameterID* is the ID of the parameter that triggered the callback.
- **OnButtonPressed** (string buttonName)
  - Called when a parameter button is pressed. *buttonName* is the ID of the button that triggered the callback.
- **OnMiddleButtonPressed** (string buttonName)
  - Called when a button is pressed with the middle mouse button. *buttonName* is the ID of the button that triggered the callback.
- **OnRightButtonPressed** (string buttonName)
  - Called when a button is pressed with the right mouse button. *buttonName* is the ID of the button that triggered the callback.
- **OnButtonToggled** (id, checked)
  - Called whenever a Toggle Button parameter changes state. *id* is the parameter name, *checked* the new state.
- **OnTimer** (string timerID)
  - Called when a timer ticks (completes a cycle). *timerID* is the ID of the timer that triggered the callback.
- **OnDataMapValueChanged** (string varName)
  - Called whenever a DataMap variable changes. *varName* is the ID of the variable that was changed.
- **OnStreamDeckKey** (int key)
  - Called whenever a StreamDeck button is pressed. *key* indicates the index of the pressed button.
  - This callback is used in conjunction with the internal Stream Deck integration
- **OnCompanionKey** (companionEvent)
  - Called whenever a key event is received from Bitfocus Companion. *companionEvent* exposes Payload and DeviceName.
- **OnMonogramEvent** (jsonEvent)
  - Called whenever an event is received from a Monogram controller. *jsonEvent* contains the event as a JSON string.

The following StreamDeck callbacks are used with the external Stream Deck integration:

- **OnStreamDeckKeyUp** (sdEvent)
  - Called whenever a Stream Deck button has been released.
  - sdEvent contains the following fields:
    - string **EventType** (the event type, for example, "keyUp")
    - string **Id** (the unique ID of the device)
    - string **DeviceIndex** (the index of the Stream Deck device that generated the event)
    - string **DeviceName** (the name of the Stream Deck device that generated the event)
    - int **XKey** (the column index of the key)
    - int **YKey** (the row index of the key)
    - string **Payload** (the user defined payload)
- **OnStreamDeckKeyDown** (sdEvent)
  - Called whenever a Stream Deck dial has been pressed.
  - sdEvent contains the same fields as **OnStreamDeckKeyUp**
- **OnStreamDeckDialUp** (sdEvent)
  - Called whenever a Stream Deck dial has been released.
  - sdEvent contains the same fields as **OnStreamDeckKeyUp**

- **OnStreamDeckDialDown** (sdEvent)
  - Called whenever a Stream Deck dial has been pressed.
  - sdEvent contains the same fields as **OnStreamDeckKeyUp**
- **OnStreamDeckDialRotate** (sdEvent)
  - Called whenever a Stream Deck dial has been rotated; `Ticks` contains the amount and direction of the rotation.
  - sdEvent contains the same fields as **OnStreamDeckKeyUp** and additionally:
    - int **Ticks** positive number for clockwise rotation and negative for anticlockwise rotation. Minimum value is 1 and increases when doing fast movementes.
- **OnStreamDeckTouchTap** (sdEvent)
  - Called whenever the Stream Deck touch strip has been tapped; `TapPosX` / `TapPosY` contain the tap position.
  - sdEvent contains the same fields as **OnStreamDeckKeyUp** and additionally:
    - int **TapPosX** number from 0 to 200 (on Stream Deck + device) representing the horizontal touch position
    - int **TapPosY** number from 0 to 100 (on Stream Deck + device) representing the vertical touch position
- **OnMIDIEvent** (midiEvent)
  - Called whenever a midi event is registered on one of the attached and configured midi devices.
  - midiEvent contains the following fields:
    - string **DeviceName** (the name of the device triggering the midi event.
    - string **EventType** (either "ControlChange", "NoteOn" or "NoteOff").
    - int **Channel** (the control channel of the event).
    - int **Number** (the control number of the event).
    - int **Value** (the value of the event, in the range [0..127]).
    - int **Note** (the note of the event in case EventType is NoteOn or NoteOff).
    - int **Velocity** (the velocity of note event in case EventType is NoteOn or NoteOff).
- **OnDMXEvent** (dmxEvent)
  - Called whenever a dmx lighting value changes
  - dmxEvent contians the followinf fields:
    - short **Universe** (the Univers ehtat changed)
    - byte[] **DMXData** (the entire 512 byte long data array)
    - byte[] **change** (a 512 byte long array containing information about channel changes)
    - int **firstDiff** (the index of the first channel that changed)
  - bool HasChanged(int index)
    Call this function to check whether a certain channel has changed
- **Table Callbacks**
  - **OnTableColumnsChanged** (string tableID)
    - Called whenever a table parameter's columns change in number. *tableID* is the ID of the table that triggered the callback.
  - **OnTableRowsChanged** (string tableID)
    - Called whenever a table parameter's rows change in number. *tableID* is the ID of the table that triggered the callback.
  - **OnTableCellValueChanged** (string tableID, int row, int column, BaseBlock cell)

- Called whenever a table parameter's cell changes value. *tableID* is the ID of the table that triggered the callback. *row* and *column* indicate the position of the cell within the caller table parameter. *cell* is the cell object that was changed. Users can interact directly with it. When **Trigger on live changes** is enabled on the table property, this callback is called also while editing the cell, if not the callback is called when keyboard focus is lost on the edited cell.

Video Output Callbacks

- **OnVideoMouseLeftButtonDown** (Point point)
  - Invoked when the left mouse button is pressed on the preview output. *point.X* and *point.Y* are normalized coordinates in the range of [-0.5…0.5] where [0,0] is the center of the screen.
- **OnVideoMouseRightButtonDown** (Point point)
  - Invoked when the right mouse button is pressed on the preview output. *point.X* and *point.Y* are normalized coordinates in the range of [-0.5…0.5] where [0,0] is the center of the screen.
- **OnVideoMouseMove** (Point point)
  - Invoked when the right mouse moves on the preview output. *point.X* and *point.Y* are normalized coordinates in the range of [-0.5…0.5] where [0,0] is the center of the screen.
- **OnVideoMouseWheel** (double delta)
  - Invoked when the mouse wheel is turned on the preview output. delta is a floating point typically being -120/120 depending on the direction of the wheel turn and the hardware connected.
- **OnVideoKeyDown** (char charKey, uint rawKey, KeyEventArgs eventArgs)
  - Invoked when a keyboard down event has taken place on the preview output. *charKey* contains the actual character of the pressed key, *rawKey* is the numeric representation of the pressed key and *eventArgs is a* System.Windows.Input.KeyEventArgs instance from the operating system representing the raw event information.

4.1.28 Exposed Objects

Console

- void **Write** (string message)
  - Writes the message to the scripting console.
- void **WriteLine** (string message)
  - Writes the message to the scripting console followed by a new line.

Info: When Viz Arc's log level is set to **TRACE**, the strings sent to *Write* and *WriteLine* are also logged in the global log file.

MessageBox

- void **Show** (string message)
  - Shows a message box with its content equal to *message*.
- void **Show** (string message, string title)
  - Shows a message box with titled *title* and with its content equal to *message*.

File Handling Example

```
// Log an error and show a message to the user
Console.WriteLine("Unable to load data")
MessageBox.Show("Unable to load data", "Load Error")
```

XmlDocument

XmlDocument allows you to read XML files or strings and aggregate data using XPath. Read more about XMLDocument and other classes here.

```
// create XmlDocument and load a xml from disc
xmlDoc = new XmlDocument()
xmlDoc.Load("C:/tmp/TestData.xml")
Console.WriteLine("nodes " + xmlDoc.ChildNodes.Count)


// create namespace manager
nsmgr = new XmlNamespaceManager(xmlDoc.NameTable)
// add namespace
nsmgr.AddNamespace("x", "http://www.contoso.com/books")

// search for book nodes under the books node
root = xmlDoc.DocumentElement
nodeList=root.SelectNodes("/x:books/x:book", nsmgr)
Console.WriteLine("books " + nodeList.Count)

for( var book of nodeList )
    Console.WriteLine("ISBN: " + book.GetAttribute("ISBN") + " title: " +
book.SelectSingleNode("./x:title", nsmgr).InnerXml)
```

The content of the sample test file *C:/tmp/TestData.xml* might look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<books xmlns="http://www.contoso.com/books">
  <book genre="novel" ISBN="1-861001-57-8" publicationdate="1823-01-28">
    <title>Pride And Prejudice</title>
    <price>24.95</price>
  </book>
  <book genre="novel" ISBN="1-861002-30-1" publicationdate="1985-01-01">
    <title>The Handmaid's Tale</title>
    <price>29.95</price>
  </book>
  <book genre="novel" ISBN="1-861001-45-3" publicationdate="1811-01-01">
    <title>Sense and Sensibility</title>
    <price>19.95</price>
  </book>
```

```
</books>
```

XMLHttpRequest

XMLHttpRequest implements the standard XMLHttpRequest interface and is the recommended way of performing HTTP requests from scripts.

Methods

- void **open** (string method, string url, bool async)
    - Initializes a request. *method* is the HTTP verb ("**GET**", "**POST**", "**PUT**", "**DELETE**", ...). With *async* = true (recommended) send returns immediately and completion is reported through onreadystatechange; with *async* = false the script blocks until the response has arrived.
- void **open** (string method, string url, bool async, string user, string password)
    - Same as above, additionally passing credentials for HTTP authentication.
- void **setRequestHeader** (string header, string value)
    - Adds an HTTP request *header* with the respective *value*, e.g. **Content-Type** or **Authorization**. Must be called after open and before send.
- void **send** ()
    - Sends the request without a body (typical for GET).
- void **send** (body)
    - Sends the request with a request *body*, e.g. a JSON string for POST/PUT requests (set the Content-Type header accordingly).
- void **abort** ()
    - Cancels the current request.
- string **getResponseHeader** (string header)
    - Returns the value of a single response *header*.
- string **getAllResponseHeaders** ()
    - Returns all response headers as a single CRLF-separated string.

Properties

- **onreadystatechange** [Get, Set]
    - Assign a function that is called every time readyState changes; check for readyState === 4 to process the finished response.
- int **readyState** [Get]
    - The state of the request: 0 = uninitialized (before open), 1 = opened, 2 = headers received, 3 = loading (receiving), 4 = done.
- int **status** [Get]
    - The HTTP status code of the response (e.g. 200, 404), available once the response headers have been received.
- string **statusText** [Get]
    - The HTTP status text of the response (e.g. "OK").
- string **responseText** [Get]
    - The response body as a string. For JSON APIs, parse it with JSON.parse(request.responseText).
- **responseXML** [Get]
    - The response parsed as an XML DOM document (when the response is XML); supports DOM methods such as selectNodes and selectSingleNode, e.g. request.responseXML.selectNodes("//item").

- **responseBody** [Get]
  - The raw response body as a byte array (useful for binary responses).

The exposed XMLHttpRequest implements the standard members:

- **open** (method, url, async) / **open** (method, url, async, user, password)
- **send** ( ) / send (body) — Sends the request, optionally with a body.
- **abort** ( ) — Aborts a running request.
- **setRequestHeader** (header, value), **getResponseHeader** (header), **getAllResponseHeaders** ( )
- onreadystatechange, readyState, status, statusText
- responseText, responseXML, responseBody

POST Example

```
var request = new XMLHttpRequest()

request.onreadystatechange = function () {
  if (request.readyState === 4) {
    if (request.status === 200 || request.status === 201) {
      var result = JSON.parse(request.responseText)
      Console.WriteLine("Created with id " + result.id)
    } else {
      Console.WriteLine("Request failed: " + request.status + " " +
request.statusText)
    }
  }
}

request.open("POST", "https://jsonplaceholder.typicode.com/posts", true)
request.setRequestHeader("Content-Type", "application/json")
request.send(JSON.stringify({ title: "Hello", body: "From Viz Arc", userId: 1 }))
```

Prevent Caching

It is possible that requests through **XMLHttpRequest** get cached and the results of the queries might seem outdated. In order to prevent caching you can add a random number as parameter of the request. `request.open("GET", "https://jsonplaceholder.typicode.com/users?dummy="+Date.now(), true)`
In this case a dummy parameter is assigned with the current EPOCH date in milliseconds.

xlAppType

This type allows you to read Excel sheets directly.

Note: This object only works if there is a local Excel installation on the same machine where Viz Arc is running.

FSO

The FSO object allows you to read, create and write files.

- **OpenTextFile** (*filename*, [*iomode*, [*create*, [*format*]]])
  - *iomode* can be one of the following: IOMode.ForReading, IOMode.ForWriting, IOMode.ForAppending
  - *format* can be one of: `Tristate.TristateUseDefault` (system default encoding, UTF-8), `Tristate.TristateTrue` (Unicode/UTF-16), `Tristate.TristateFalse` (ASCII, the default).
- **CreateTextFile** (string fileName, bool overwrite = true, bool unicode = false) — Creates a new text file and returns a text stream for writing.
- bool **FileExists** (string path) — Returns true when the file exists.
- **DeleteFile** (string path) — Deletes the file.

The text stream returned by OpenTextFile/CreateTextFileexposes:

- string **ReadAll** ()
  - Reads the entire file content.
- **Write** (string text)
  - Writes text to the stream.
- **Close** ()
  - Closes the stream.

Reading a UTF8 Encoded Text File

```
var file = new FSO()
var stream = file.OpenTextFile("d:/testexport.txt")
// or
var stream = file.OpenTextFile("d:/testexport.txt", IOMode.ForReading, false,
Tristate.TristateTrue)

Console.WriteLine(stream.ReadAll())
```

4.1.29 xHost

The xHost object gives you access to virtually any .NET resource.

V8 Script Sample

```
var List = xHost.type('System.Collections.Generic.List')
var DayOfWeek = xHost.type('System.DayOfWeek')
var week = xHost.newObj(List(DayOfWeek), 7)
week.Add(DayOfWeek.Sunday)
```

You can even import entire assemblies:

V8 Enumerate Files in Directory

```
var clr = xHost.lib('mscorlib', 'System', 'System.Core', 'System.IO')
dropdown_0.Clear()
var dir = clr.System.IO.Directory
dropdown_0.SetItems(dir.GetFiles('c:\\tmp'))

// another sample that starts an external process "calc.exe"
var proc = xHost.lib('System.Diagnostics.Process')
proc.System.Diagnostics.Process.Start("calc.exe")
```

In the example above, a UI dropdown element named *dropdown_0* is populated with a file list contained in *c:\tmp* using .NET **System.IO.Directory** class instance.

Another example is how to convert a XML into JSON using the Newtonsoft library:

```
// get the library handle
var NewtonsoftLib = xHost.lib('Newtonsoft.Json')
// get a reference to the JsonConvert class
var JsonConvert = NewtonsoftLib.Newtonsoft.Json.JsonConvert

function convertXMLToJSON(fileName)
{
    // error checking omitted for clarity
    // read xml document from disc:
    let xmlDoc = new XmlDocument()
    xmlDoc.Load(fileName)

    // set the raw XML data into the DataMap
    SetData("infoXML", xmlDoc.InnerXml)

    // use C# Newtonsoft XML serializer
    let jsonOut = JsonConvert.SerializeXmlNode(xmlDoc)

    // set the converted json into the DataMap
    SetData("infoJson", jsonOut)
}
```

4.1.30 Time

`Time` (also `time`) exposes the .NET `DateTime` type, e.g. `Time.Now`, `Time.UtcNow`, `Time.Parse(...)`.

4.1.31 Lib

`Lib` (also `lib`) exposes the .NET *mscorlib* and *System.Core* type collections, e.g. `new Lib.System.Text.StringBuilder()` — a built-in alternative to `xHost.lib(...)` for the most common types.

4.1.32 host

`host` (also `Host`) exposes the ClearScript HostFunctions helpers, e.g. `host.newArr(...)`, `host.cast(...)`.

4.1.33 EnumToInt / EnumToString

`EnumToInt(enumValue)` and `EnumToString(enumValue)` convert .NET enum values to an int or string.

4.1.34 XmlNamespaceManager

`XmlNamespaceManager` is exposed alongside `XmlDocument` for namespace-aware XPath queries (see the XmlDocument example above, which already uses it).

4.1.35 Performance

The performance object provides access to performance-related information.

- **now** ()
  - Returns a high-resolution floating point value of the milliseconds elapsed since the script engine was started (see `timeOrigin`). Ideal for measuring durations; for an EPOCH timestamp use `Date.now()` instead.
- **timeOrigin** [Get] — The script engine's start time as milliseconds since the EPOCH (January 1, 1970 UTC); `timeOrigin + now()` yields the current EPOCH time in high resolution.
- **sleep** (milliseconds, precise)
  - Sleep for a certain amount of milliseconds. Set precise to **true** when a high precision timer is required.

```
let timeA = Performance.now()
Performance.sleep(500, true) // high perf sleep half second
// do something else
let timeB = Performance.now()

Console.WriteLine("timeA " + timeA)
Console.WriteLine("timeB " + timeB)
Console.WriteLine("time difference " + (timeB - timeA) + " ms")
```

```
Console.WriteLine("timeB EPOCH " + (timeB + Performance.timeOrigin))
```

4.1.36 Garbage Collection

Especially for long running scripts it might be useful to force calling the garbage collector from script.

- void **ForceGarbageCollection**(bool collectHostItems)
  - Calls the .NET garbage collector and frees unreferenced memory. Set *collectHostItems* to true for a more aggressive cleanup including the collection of script internal objects.

Garbage collection usually is triggered automatically. In some cases it might be useful to force garbage collection to control memory usage. Note that the garbage is always collected when stopping a script or when loading a new project.

4.1.37 SQL Sample

If you know the assembly name of a specific type, you can instantiate it using xHost.type(name, assemblyName) method.

```
var queryString = "SELECT * FROM someTable"
var connetionString = "Server=192.168.1.42,1433;UID=aUserID;PWD=SomePassword;"

// get types using xHost.type
var SqlConnection = xHost.type('System.Data.SqlClient.SqlConnection',
'System.Data.SqlClient')
var SqlCommand = xHost.type('System.Data.SqlClient.SqlCommand',
'System.Data.SqlClient')
var SqlDataReader = xHost.type('System.Data.SqlClient.SqlDataReader',
'System.Data.SqlClient')

connection = new SqlConnection(connetionString)
connection.Open()
var command = new SqlCommand(queryString, connection)
var reader = command.ExecuteReader()
// do something with the data
while (reader.Read())
{
    // iterate over the result and print the result on the console
    Console.WriteLine(reader.GetString(0))
}
connection.Close()
```

4.1.38 SQLite Sample

Another sample that uses the xHost.type function is the usage of a simple SQLite database. It is required that the SQLite libraries/DLLs are in the search path of the system or in the same directory as the Viz Arc executable.

```
var SQLiteConnection = xHost.type('System.Data.SQLite.SQLiteConnection',
'System.Data.SQLite')
// those two below are not needed for this sample as they dont get instantiated
explicityl
var SQLiteDataReader = xHost.type('System.Data.SQLite.SQLiteDataReader',
'System.Data.SQLite')
var SQLiteCommand = xHost.type('System.Data.SQLite.SQLiteCommand',
'System.Data.SQLite')

function ReadData(conn)
{
    // create q query
    sqlite_cmd = conn.CreateCommand()
    sqlite_cmd.CommandText = "SELECT Name FROM Artist LIMIT 10;" // read first 10
artists of the table

    // execute the query
    sqlite_datareader = sqlite_cmd.ExecuteReader();
    while (sqlite_datareader.Read())
    {
        // iterate over the result and print the result on the console
        Console.WriteLine(sqlite_datareader.GetString(0));
    }
}

function testSQLiteDB()
{
    // create a new connection specifying the database file name and the version
    // sample database can be found here https://github.com/lerocha/chinook-database
    conn = new SQLiteConnection("Data Source=C:\\tmp\\Chinook.db;Version=3;")
    try
    {
        // Open the connection:
        conn.Open()

        // read some data
        ReadData(conn)

        // close the connection
        conn.Close()
    }
    catch (ex)
    {
        Console.WriteLine("error in SQLite query " + ex)
    }
}

Global.OnInit = function ()
{
    testSQLiteDB()
}
```

4.1.39 TcpSend

The TcpSendAsync method, is used to send a short message to any host on any port of the network.

- Task<string> **TcpSendAsync** (string hostname, int port, string command, int timeoutInMs = 1000)
    - Send *command* to *hostname* on *port*. If the parameter *timeoutInMs* is not specified, a default timeout of 1 second is used.

```
Global.OnButtonPressed = async function (id)
{
    if( id == "sendTcp"){
        let response = await TcpSendAsync("192.168.1.22", 1234, "say something\0",
5000)
        Console.WriteLine("response from client " + response)
    }
}
```

4.1.40 HtmlAgility Example

The classes **HtmlDocument** and **HtmlWeb** are exposed by the HtmlAgility library and enable parsing and data extraction of html pages.

```
 //sample to use HtmlDocument class from HtmlAgility
  var doc = new HtmlDocument()
  doc.Load("c:/tmp/HtmlAgilityTest.html")

  for (var table of doc.DocumentNode.SelectNodes("//table")) {
    Console.WriteLine("Found: " + table.Id)

    for (var row of table.SelectNodes("tr")) {
      for (var cell of row.SelectNodes("th|td"))
        Console.Write(cell.InnerText + " ")
      Console.WriteLine("")
    }
  }

  // sample to use HtmlWeb class from HtmlAgility
  var html = "http://html-agility-pack.net/"
  var web = new HtmlWeb()
  var htmlDoc = web.Load(html)
  var node = htmlDoc.DocumentNode.SelectSingleNode("//head/title")
  Console.WriteLine("Node Name: " + node.Name + "\n" + node.OuterHtml)
```

The contents of the file from the sample above *c:/tmp/HtmlAgilityTest.html*:

HTML Sample with Table

```
<!DOCTYPE html>
<html>
<style>
    table, th, td {
        border: 1px solid black;
    }
</style>
<body>
    <h2>A HTML table to test HTML Agility</h2>
    <table style="width:100%" id="dataTable">
        <tr>
            <th>Name</th>
            <th>Number</th>
            <th>Country</th>
        </tr>
        <tr>
            <td>Athlete A</td>
            <td>42</td>
            <td>Italy</td>
        </tr>
        <tr>
            <td>Athlete B</td>
            <td>34</td>
            <td>Japan</td>
        </tr>
    </table>
    <p>Here is some more text</p>
</body>
</html>
```

Read more about HtmlAgility here.

4.1.41 Main Script-only

There are functionalities that are specific to Viz Arc's main script:

Canvas Tabs Handling

- void **SetActionsSelectedTab** (string tabName)
    - Looks for a tab named *tabName* and sets it as active.
- void **SetActionsSelectedTab** (int tabIndex)
    - Sets the Action selected tab to the tab at *tabIndex* index.
- string **GetActionsSelectedTabName** ( )
    - Returns the currently selected tab's name.
- int **GetActionsSelectedTabIndex** ( )
    - Returns the index of the currently selected action canvas tab (-1 when no canvas is available).

- string[] **GetActionsTabs** ( )
  - Returns a string array with all tab names.

Action Template Handling

Arc's main scripts allows the user to interact with template actions on the action canvas.

- void **PreviewSelectedTemplate** ( )
  - Previews the currently selected template action.
- void **ExecuteSelectedTemplate** ( )
  - Executes the currently selected template action.
- void **UpdateSelectedTemplate** ( )
  - Updates the currently selected template action.
- void **ContinueSelectedTemplate** ( )
  - Continues the currently selected template action.
- void **StillPreviewSelectedTemplate** ( )
  - Generates a still preview of the currently selected template action.

> **Note:** The method presented only works when one and only one action (template action) is selected on the action canvas.

Callbacks

- **PreActionExecute** (string actionName)
  - Called whenever an is executed and before the actual execution occurs.
  - actionName is the name of the action that is being executed.
- **PosActionExecute** (string actionName)
  - Called whenever an is executed and after the actual execution occurs.
  - actionName is the name of the action that is being executed.
- **OnInit** ( )
  - Called when the main script is started (User clicks on the **Start** button).

4.1.42 Template Script-only

The template script is a specific version that is used on the template designer and on the template action.

Action/Designer Handling

- void **ExecuteTemplate** ( )
  - Executes the owner template action or loaded template in the designer.
- void **ContinueTemplate** ( )
  - Continues the owner template action or loaded template in the designer.
- void **OutTemplate** ( )
  - Takes out the owner template action or loaded template in the designer.
- void **UpdateTemplate** ( )
  - Updates the owner template action or loaded template in the designer.

- void **UpdateTemplate** (string COs = null)
    - Updates the owner template action or loaded template in the designer. The parameter COs is a space separated list of Control Object ID's that shall be updated. For large templates containing a big amount of ControlObjects this is a very efficient alternative whenever only a small part in the scene needs to be updated. For example, UpdateTemplate("currentScore totalScore") updates only the two ControlObjects with id "currentScore" and "totalScore".
- void **UpdateTemplate** ("BP_Name.VariableName")
    - Updates the owner template action or loaded template in the designer. This method can be used if the scene is in Unreal Engine. For large templates containing a large amount of variables, this is a very efficient alternative when only a variable in the blueprint needs to be updated.
- void **UpdateTemplate** ("BP_Name", ["VariableName1", "VariableName2", ecc...])
    - Updates the owner template action or loaded template in the designer. This method can be used if the scene is in Unreal Engine. For large templates containing a large amount of variables, this is a very efficient alternative when only a few variables in the blueprint need to be updated.
- void **PreviewTemplate** ( )
    - Previews the owner template action or loaded template in the designer.
- void **PreviewExecuteTemplate** ( )
    - Executes the owner template action or loaded template in the designer to the preview channel.
- void **PreviewContinueTemplate** ( )
    - Continues the template's animation on the preview channel.
- void **PreviewOutTemplate** ( )
    - Takes the template out on the preview channel.
- void **PreviewUpdateTemplate** (string COs = null)
    - Updates the owner template action or loaded template in the designer to the preview channel. The parameter COs is a space separated list of Control Object ID's that shall be updated. For large templates containing a big amount of ControlObjects this is a very efficient alternative whenever only a small part in the scene needs to be updated. For example, UpdateTemplate("currentScore totalScore") updates only the two ControlObjects with id "currentScore" and "totalScore".
- bool **IsActionTemplateSelected** ( )
    - Returns true when this template is currently selected on the action canvas.
- string **GetSavedBy**( ), string **GetSavedOn**( ), string **GetVersion**( ), string **GetSavedAt**( )
    - Return the template's save metadata (author, host, Viz Arc version and save time).

Properties

- BaseAction **ThisAction**
    - Returns the script accessor of this template. **ThisAction** might be null when using the template editor.

```
if( ThisAction ){
  // set the action's name
  ThisAction.Name = "Hello"
  // set the action's tooltip description
  ThisAction.Description = "Hello Description"
}
```

4.1.43 Payloads

A **payload** is a snapshot of a template's data — its linked-parameter / control-object values — stored as a string. A payload can be dragged onto a **playlist** as a payload item, so one template action can produce many playlist entries that each replay a different saved state. Payloads are also how the MSE (Media Sequencer) and the REST API push values into a template.

Two payload string formats are *auto-applied* by Arc, detected by the first character: **JSON** (starts with `{`) is a flat map of `controlObjectId → value` and is the normal Arc format; **VDF XML** (starts with `<`) is the Vizrt Data Format used for MSE integration. A payload is not limited to these, however: a template that implements its own **OnSetPayload** can store and interpret a payload in any format it likes (see **Custom Payloads**).

```
{ "imageLeft": "VIZ/Images/logo.png", "title": "Hello World", "score": 3 }
```

```
<payload xmlns="http://www.vizrt.com/types" model="modelUri">
  <field name="title"><value>Hello World</value></field>
</payload>
```

Capturing a Payload

- string **GetTemplatePayload** ( )
    - Captures the current values of all of the template's linked control objects and returns them as a JSON payload string. The keys are control object IDs (the linked control object's `objectID`), not parameter names, and it recurses into panel children.
- void **SetPayload** (string payload)
    - Stores *payload* (a JSON string) on the template's drag badge, so it becomes the payload of the playlist item created when the badge is dragged onto a playlist.
- void **SetPayloadLabel** (string label)
    - Sets the human-readable *label* shown on the drag badge and on the resulting playlist item.
- void **SetPayloadColor** (string hexColor)
    - Sets the color used for the payload's playlist row, e.g. `"#FF4400"`.
- string **GetPayloadColor** ( )
    - Returns the payload's row color.
- string **GetLabelFromControlObjects** (string ids)
    - Builds a label automatically from the current values of the named control objects, joined with slashes. *ids* is a space-separated list of control object IDs — e.g. `GetLabelFromControlObjects("title score")` returns `"Hello World/3"`.

Capturing Example

```
// Capture the current values of all linked control objects as a JSON payload string.
var payload = GetTemplatePayload()          // -> '{"imageLeft":"...","title":"..."}'
```

```
// Store the payload and a label on the template's drag badge; these become the
// payload and label of the playlist item created when the badge is dragged out.
SetPayload(payload)
SetPayloadLabel("Slide 1 - Hello World")

// Or build the label automatically from selected control object values (slash-
joined).
SetPayloadLabel(GetLabelFromControlObjects("title score"))   // -> "Hello World/3"
```

Applying a Payload

- void **SetTemplatePayload** (string payload)
    - Applies a JSON *payload* to the template's linked parameters. Unlike the auto-apply path used when a playlist row fires, this *does* let **OnParameterChanged** callbacks fire (it is the inverse of **GetTemplatePayload**).
- bool SavePayloadToBoundRow ( )
    - Saves the current editor values back to the playlist row the template editor was opened from. Returns true on success.

Applying Example

```
// Apply a JSON payload to the linked parameters. Unlike the auto-apply path,
// this lets OnParameterChanged callbacks fire.
SetTemplatePayload('{"title":"Updated"}')
```

The OnSetPayload Callback

When a playlist payload item is previewed or executed, Arc calls the template’s **OnSetPayload** callback (a **Global** callback assigned in the template script):

```
Global.OnSetPayload = function(payload, isPreview) {
    // payload   : the stored payload string (JSON or VDF XML)
    // isPreview : true  -> the row was selected/previewed (do NOT go on air)
    //             false -> the row is being executed (execute follows immediately)
    var data = JSON.parse(payload)
    SetParameterValue("titleParam", data.title)   // apply however you like
}
```

Resolution order when a payload item fires:

1. If the script declares **OnSetPayload**, it is called and is fully responsible for applying the payload.
2. Otherwise, if the action installed a custom (native) payload handler, that handles it.
3. Otherwise Arc auto-applies the payload to the linked parameters (JSON or VDF), with **OnParameterChanged** callbacks suppressed.

So if you only need the standard “restore linked parameter values” behaviour you do **not** need to implement **OnSetPayload** at all — leave it out and Arc restores the state automatically.

Two methods allow you to use the built-in default handler within the **OnSetPayload** callback:

- void **BaseSetPayload** ( )
  - Applies Viz Arc's default payload-restore behavior to the payload currently being applied. Call it from within **OnSetPayload** when you want the default handling in addition to (or after) your custom logic.
- void **BaseSetPayload** (string payload)
  - Same as above, applying the given payload string instead.

Custom Payloads

A payload is just an opaque string as far as the playlist is concerned — it does not have to be a control-object JSON map or VDF XML. When a template implements its own **OnSetPayload**, that handler receives the stored payload string verbatim and is fully responsible for interpreting it, so the payload can be in any format you choose (a custom token, CSV, a URL, your own JSON shape, and so on). The built-in JSON / VDF auto-detection only governs Arc's auto-apply path, which is used when no **OnSetPayload** is declared.

Because the payload content is entirely under script control, the mechanism is not tied to Viz templates or VDF. It works the same way for **Flowics** templates (and any other template type): store whatever string the template needs with **SetPayload**, and parse it in **OnSetPayload** when the row is previewed or executed.

Custom Payload Example

```
// Store any custom string you like as the payload - here a simple "key=value" line.
SetPayload("matchId=42;home=GSW;away=NYK")
SetPayloadLabel("GSW vs NYK")

// Interpret it yourself when the playlist row is previewed/executed. The format is
// entirely your own; Arc does not try to auto-apply it because OnSetPayload is
declared.
Global.OnSetPayload = function(payload, isPreview) {
    var parts = {}
    payload.split(";").forEach(function(kv) {
        var p = kv.split("=")
        parts[p[0]] = p[1]
    })
    SetParameterValue("homeTeam", parts.home)
    SetParameterValue("awayTeam", parts.away)
}
```

MSE / VDF Payloads

- string **GetMSEVdfPayload** (string modelUri)
  - Builds an MSE-compatible VDF XML payload from the current control object values. *modelUri* is the MSE model URI. Returns the `<payload>` XML string.
- string **GetCurrentPayloadVdf** ( )
  - Builds and returns a VDF payload document from the template's current parameter values.

- bool **SetVdfPayload** (string vdfXml)
  - Applies a VDF XML payload to the template's linked parameters. Returns true on success, false otherwise.

MSE / VDF Example

```
// Build MSE-compatible VDF XML from the current control object values.
var vdf = GetMSEVdfPayload("modelUri")

// Apply a VDF XML payload to the linked parameters.
SetVdfPayload(vdf)
```

Notes

- Payloads operate on the template's linked parameters / control objects.
- **GetTemplatePayload** keys are control object IDs (the linked control object's `objectID`), not parameter names; it recurses into panel children.
- **Preview vs execute:** `isPreview === true` means selected/cued — apply values but put nothing on air; `isPreview === false` is followed immediately by the action's execute.
- Auto-apply suppresses callbacks (**OnParameterChanged** does not fire), whereas **SetTemplatePayload** does fire them — choose accordingly.
- Format auto-detection applies to Arc's auto-apply path only: a payload starting with `{` is treated as JSON, `<` as VDF XML, and anything else is ignored. A template that implements **OnSetPayload** receives the raw payload string and may use any format (see Custom Payloads).
- Payloads are not Viz- or VDF-specific; custom payloads work in **Flowics** templates and any other template type.
- The REST API can also carry payloads (for example a `vdfPayload` argument when opening a template, and a playlist `payloadRow` endpoint) — see the REST API reference, which is outside script scope.

Control Object Handling

The template script allows the user to interact with the template's control objects, only supported for Viz and Flowics templates.

- void **SetControlObject** (string objectID, dynamic value)
  - Sets the control object with id equal to *objectID's* value to *value*. The set object's value is sent on template execute/update.
  - ControlObject ID's set by this methods which have not been present in the payload during template creation, are added dynamically.

> **Note:** SetControlObject only works on control objects that aren't already linked to UI parameters.

- bool **HasControlObject** (string objectID)

- Returns true when the *objectId* has been assigned wither through the UI or through code using the SetControlObject method, otherwise it returns false.
- string **GetLinkedControlObject** (string parameterName)
    - Returns the ID of the control object the given parameter is linked to (empty when unlinked).
- void **SetLayerAlternative** (string layerName, string value)
    - Sets the alternative to be used for the Transition Logic layer *layerName* when building combo payloads.
- string **GetLayerAlternative** (string layerName)
    - Returns the alternative currently set for the given layer.
- void **ClearLayerAlternative** (string layerName)
    - Clears the alternative for the given layer.

Template Channels Handling

The template script allows the user to change the program output channel.

- void **SetSelectedChannel** (string name)
    - Sets the selected program output channel to *name*.
- void **SetSelectedChannel** (ScriptingChannel channel)
    - Sets the template's selected channel from a channel object (e.g. obtained through GetSelectedProfile().GetChannel(...)).
- ScriptingChannel **GetSelectedChannel** ( )
    - Returns the currently selected program channel of the template action.

ScriptingChannel

The ScriptingChannel class is used for finer control on the Engines contained in the channel.

Properties

- **Name**
    - The channel's name.
- **Count**
    - The number of Engines in the channel.

Methods

- void **SendSingleCommand** (string command)
    - Sends *command* to all the Engines in the channel.
- void **SendMultipleCommands** (string[] commands)
    - Sends all the input *commands* to all the Engines in the channel.
- void **SendToSMM** (string key, string value, bool doEscape)
    - Sends key-value pair to Shared Memory to all Engines contained in the channel. doEscape specifies whether the value string is escaped.
- void **SendToSMM** (string key, string value, bool doEscape, string destination)
    - Sends key-value pair to Shared Memory to all Engines contained in the channel. doEscape specifies whether the value string is escaped.

    - *destination* can be either SYSTEM, COMMUNICATION or DISTRIBUTED
- void **SendToSMM** (string key, string value)
    - Two-argument form of **SendToSMM**, the value is escaped by default.
- ScriptingEngine **GetEngine** (int index) / ScriptingEngine **GetEngine** (string name)
    - Returns an individual Engine of the channel by index or name (also available through the indexer, e.g. `channel[0]`). See the **Engine** section for the members of the returned object.

Template Scene Handling

The template script allows the user to set the scene that should be loaded when executing.

- void **SetSceneFullpath** (string fullpath = null)
    - The input fullpath is the value that is sent to the Engine when executing the template. When no full path is provided the user config value is removed and the original attached scene is used.
- string **GetBaseContainerPath** ( )
    - Returns the current base container path where the root control object is located.
- bool **SetBaseContainerPath** (string basePath)
    - Sets the base container path where the root control object is located. This might be useful to redirect the destination of the ContolObjects to a different container in the same scene.
- string **GetDirector** ( )
    - Gets the current director executed on **Execute** or on **Continue**.
- bool **SetDirector** (string dir)
    - Sets the director executed on **Execute** or on **Continue**.

Template Action Configuration

- bool **IsCommandHeaderVisible**
    - Indicates whether the CommandHeader should be visible on the template action.
- bool **UpdateOnSelected**
    - When this flag is set, the script callbacks are only triggered when the template action is selected on the action canvas (blue border).

Callbacks

- **OnCreated** ( )
    - Called when the template script is executed (when the template action is created and when the template opened on the designer is started).
- **OnDestroyed** ( )
    - Called when template is being stopped/destroyed. Can be used for cleanup, save state, stopping/canceling timers.
- **OnShow** ( )
    - Called when the template is shown (when the template action's pop-up is opened, when the action becomes embedded and when the template opened on the designer is started).
- **OnExecute** ( )
    - Called when the template is executed.

- **OnPreviewExecute** ( )
  - Called when the template is executed to the preview channel.
- **OnContinue** ( )
  - Called when the template is continued.
- **OnPreviewContinue** ( )
  - Called when the template is continued to the preview channel.
- **OnUpdate** ( )
  - Called when the template is updated.
- **OnPreviewUpdate** ( )
  - Called when the template is updated to the preview channel.
- **OnOut** ( )
  - Called when the template is taken out.
- **OnPreviewOut** ( )
  - Called when the template is taken out on the preview channel.
- **OnPreview** ( )
  - Called when the template is previewed.

Note: The **OnDestroyed** callback has a maximum execution time of five seconds. If it exceeds this, the template is forcefully stopped.

4.1.44 Common Callbacks

Callbacks that can be used in the global script and template scripts

- **OnVideoMouseLeftButtonDown** (point)
  - Called when the user pressed the left mouse button on the video output.
    - **point.X** normalized value in the range [-0.5, 0.5]
    - **point.Y** normalized value in the range [-0.5, 0.5]
- **OnVideoMouseRightButtonDown** (point)
  - Called when the user pressed the right mouse button on the video output.
    - **point.X** normalized value in the range [-0.5, 0.5]
    - **point.Y** normalized value in the range [-0.5, 0.5]
- **OnVideoMouseMove** (point)
  - Called when the user moved the mouse on the video output.
    - **point.X** normalized value in the range [-0.5, 0.5]
    - **point.Y** normalized value in the range [-0.5, 0.5]
- **OnVideoMouseWheel** (delta)
  - Called when the user rolled the mouse wheel on the video output.
    - **delta** is a floating point value, typical values are -120.0 for mouse wheel down and 120.0 for mouse wheel up.
- **OnTrackerAction** (string action)
  - Called on certain Object Tracker events. the *action* parameter determines the type of event:
    - **take:** Triggered when Object Tracker is taken On Air.
    - **takeout:** Triggered when Object Tracker is taken Off Air.
    - **preview:** Triggered when Object Tracker preview is taken.

- **previewout:** Triggered when Object Tracker preview is taken out.
- **newTracker <index>:** Triggered whenever a new object has been selected for tracking. *index* is 1 based.
- **lostTracker <index>:** Triggered whenever a tracked object has lost tracking. *index* is 1 based.

Sample Usage of Object Tracker Script API

```
Global.OnTrackerAction = function (action)
{
    Console.WriteLine("tracker action " + action )

    if( action == "take" )
        GetAction("DATA").Execute()
    else if( action.startsWith("newTracker" ) ){
        // we want to take off air whatever is On Air when we select a new tracked
object
        TakeOutTracker()
        Console.WriteLine("OFF AIR" )
    }
}
```

- **OnArenaPosition** (double screenX, double screenY, double worldX, double worldY, double worldZ)
    - Called when the user clicks on the Viz Arena view with the positioning tool.
        - **screenX** and **screenY** are the screen coordinates of the mouse click. The lower left corner of the arena screen is the origin (0,0).
        - **wolrdX**, **worldY** and **worldZ** are Viz Engine world coordinates, the units (default meters) are the same as for the selected **Viz Arena** project.

4.1.45 Parameters

Parameters are the base components of Viz Arc's scripting. A list of all existing parameters types and their associated properties is presented below.

Base Parameters Functionality

The following properties and methods are shared among all parameters

- string **Label** [Get, Set]
    - Gets/sets the label that is displayed on the UI.
- bool **IsEnabled** [Get, Set]
    - Gets/sets the enabled status of the parameter. Disabled parameters cannot be interacted with and do not trigger events.
- bool **IsVisible** [Get, Set]
    - Sets whether the parameter is visible. Invisible parameters are visible (displayed as grayed out) only while editing (and scripts are not running).
- double **X** [Get, Set]
    - Gets/sets the horizontal position of the parameter on the canvas.

- double **Y** [Get, Set]
  - Gets/sets the vertical position of the parameter on the canvas.
- double **Width** [Get, Set]
  - Gets/sets the width of the parameter.
- double **Height** [Get, Set]
  - Gets/sets the height of the parameter.
- void **SetColor** (byte r, byte g, byte b, byte a = 255)
  - Sets the parameter's color to the input RGBA color.
- string **Color**
  - Gets/sets the parameter's selected color in Hex format, for example, #FF0A0A8C (#RRGGBBAA).
- int **ColorR**
  - Gets/sets the parameter's selected red color value in the range **[0, 255]**.
- int **ColorG**
  - Gets/sets the parameter's selected green color value in the range **[0, 255]**.
- int **ColorB**
  - Gets/sets the parameter's selected blue color value in the range **[0, 255]**.
- int **ColorA**
  - Gets/sets the parameter's selected alpha value in the range **[0, 255]**.
- string **Tooltip** [Get, Set]
  - Gets/sets the tooltip of the UI element.
- string **LinkedCO** [Get]
  - The name of the associated ControlObject (Template Scripting only).
- void **UpdateDataLink**()
  - Forces an explicit evaluation of the DataLink expression associated with this parameter.
- int **ZIndex** [Get, Set]
  - Gets/sets the parameter's z-order on the canvas.
- string **FontColor** [Get, Set]
  - Gets/sets the parameter's font color as hex string (setting it enables the custom font color).
- bool **UseCustomFontColor** [Get, Set]
  - Enables/disables the custom font color.
- bool **Bold** / bool **Italic** / bool **Underline** [Get, Set]
  - Gets/sets the font style flags.
- string **FontStyle** [Get, Set]
  - Gets/sets the font style as a string of flags.
- int **FontSize** [Get, Set]
  - Gets/sets the font size (clamped to 1–100).
- string **LinkedDataKey** [Get, Set]
  - Gets/sets the DataMap key the parameter is linked to (see UpdateDataLink).
- string **LinkedDataQuery** [Get, Set]
  - Gets/sets the query/filter applied to the linked data.

A sample use of the **LinkedCO** property:

```
Global.OnParameterChanged = function (id)
{
```

```
    // get linked ControlObject id associated to parameter 'id'. It is null if it's
not linked to any CO.
    let linkedCO = GetParameter(id).LinkedCO

    if( linkedCO )
    {
        Console.WriteLine("Changed: " + id + ", Linked Control Object ID: " +
linkedCO)

        // live update the template
        UpdateTemplate(linkedCO)
    }
}
```

Layout

The layout parameters allow the user to organize and improve the usability of a script/template.

Panel

- BaseParameter [ ] **Children** [Get]
    - Returns an array with all of the panel's children.
- BaseParameter **GetParameter** (string parameterID)
    - Tries to find a child with id equal to *parameterID*. Returns it if successful.

Tabs

- string **Value** [Get, Set]
    - Set: Attempts to find a tab with its name equal to the input. If found, sets it as selected tab.
    - Get: Returns the name of the selected tab.
- BaseParameter [ ] **Children** [Get]
    - Returns an array with all of the panel's children.
- BaseParameter **GetParameter** (string parameterID)
    - Tries to find a child with id equal to *parameterID*. Returns it if successful.
- bool **AllowReordering**
    - Whether a user can reorder the tabs.
- int **SelectedIndex** [Get, Set]
    - Gets/sets the index of the selected tab.

Info

- string **Value** [Get, Set]
    - Gets/sets the info text that displays on the parameter.

Label

TextColor

- string **TextColor** [Get, Set]
    - Gets/sets the labels text color in Hex format, for example, #FF0A0A8C (#RRGGBBAA).

Dialogs

Color

- string **Value** [Get, Set]
    - Gets/sets the parameter's selected color in Hex format, for example, #FF0A0A8C (#RRGGBBAA).
- int **R** [Get]
  Gets the value of the red component in the range **[0, 255]**.
- int **G** [Get]
  Gets the value of the green component in the range **[0, 255]**.
- int **B** [Get]
  Gets the value of the blue component in the range **[0, 255]**.
- int **A** [Get]
  Gets the value of the alpha component in the range **[0, 255]**.
- double **RPercent** [Get]
  Gets the value of the red component in the range **[0, 1]**.
- double **GPercent** [Get]
  Gets the value of the green component in the range **[0, 1]**.
- double **BPercent** [Get]
  Gets the value of the blue component in the range **[0, 1]**.
- double **APercent** [Get]
  Gets the value of the alpha component in the range **[0, 1]**.
- void **SetR** (int R)
  Sets the red component in the range **[0, 255]**.
- void **SetG** (int G)
  Sets the green component in the range **[0, 255]**.
- void **SetB** (int B)
  Sets the blue component in the range **[0, 255]**.
- void **SetA** (int A)
  Sets the alpha component in the range **[0, 255]**.
- void **SetRGB** (int R, int G, int B)
  Sets the red, green and blue components in the range **[0, 255]**.
- void **SetRGBA** (int R, int G, int B, int A)
  Sets the red, green, blue and alpha components in the range **[0, 255]**.
- void **SetRPercent** (double R)
  Sets the red component in the range **[0, 1]**.

- void **SetGPercent** (double G)
  Sets the green component in the range **[0, 1]**.
- void **SetBPercent** (double B)
  Sets the blue component in the range **[0, 1]**.
- void **SetAPercent** (double A)
  Sets the alpha component in the range **[0, 1]**.
- void **SetRGBPercent** (double R, double G, double B)
  Sets the red, green and blue components in the range **[0, 1]**.
- void **SetRGBAPercent** (double R, double G, double B, double A)
  Sets the red, green, blue and alpha components in the range **[0, 1]**.

DateTime

The DateTime parameter provides comprehensive date and time selection with extensive scripting capabilities including component accessors, formatting helpers, Unix timestamp support, and range constraints.

- string **Value** [Get, Set]
  - Gets/sets the date/time as ISO 8601 string (for example, "2025-01-15T14:30:00").
- string **MinDate** [Get, Set]
  - Gets/sets the minimum selectable date (ISO format).
- string **MaxDate** [Get, Set]
  - Gets/sets the maximum selectable date (ISO format).
- bool **EnableTime** [Get, Set]
  - Gets/sets whether time selection is enabled (not just date).
- bool **ShowSeconds** [Get, Set]
  - Gets/sets whether seconds are shown in the time picker.

Component Accessors (Read-Only)

- int **Year** [Get]
  - Gets the four-digit year (for example, 2025).
- int **Month** [Get]
  - Gets the month (1-12).
- int **Day** [Get]
  - Gets the day of month (1-31).
- int **Hour** [Get]
  - Gets the hour (0-23).
- int **Minute** [Get]
  - Gets the minute (0-59).
- int **Second** [Get]
  - Gets the second (0-59).

Methods

- DateTime **GetDateTime**()
  - Gets the value as a .NET DateTime object.
- void **SetDateTime**(DateTime dt)

  - Sets the value from a .NET DateTime object.
- void **SetDate**(int year, int month, int day)
  - Sets the date portion while preserving the time.
- void **SetTime**(int hour, int minute, int second = 0)
  - Sets the time portion while preserving the date.
- long **GetUnixTimestamp**()
  - Gets the value as Unix timestamp (seconds since January 1, 1970).
- void **SetFromUnixTimestamp**(long timestamp)
  - Sets the value from a Unix timestamp.
- string **GetDateString**()
  - Gets a formatted date string.
- string **GetTimeString**()
  - Gets a formatted time string.
- string **GetDateTimeString**()
  - Gets a formatted date and time string.

Examples

Basic Date/Time Access:

```
Global.OnCreated = function() {
    // Set initial value to now
    eventDateTime.SetDateTime(new Date())
}

Global.OnParameterChanged = function(id) {
    if (id === "eventDateTime") {
        // Access individual components
        Console.WriteLine("Year: " + eventDateTime.Year)
        Console.WriteLine("Month: " + eventDateTime.Month)
        Console.WriteLine("Day: " + eventDateTime.Day)
        Console.WriteLine("Hour: " + eventDateTime.Hour)
        Console.WriteLine("Minute: " + eventDateTime.Minute)
        Console.WriteLine("Second: " + eventDateTime.Second)
        // Get ISO string
        Console.WriteLine("ISO: " + eventDateTime.Value)
    }
}
```

Set Date and Time Separately:

```
Global.OnCreated = function() {
    // Set date to January 15, 2025 (preserves current time)
    eventDateTime.SetDate(2025, 1, 15)
    // Set time to 14:30:00 (preserves current date)
    eventDateTime.SetTime(14, 30, 0)
}
```

Unix Timestamp Conversion:

```
Global.OnCreated = function() {
    // Set from Unix timestamp
    let timestamp = 1704067200 // Jan 1, 2024 00:00:00 UTC
    eventDateTime.SetFromUnixTimestamp(timestamp)
    Console.WriteLine("Set to: " + eventDateTime.GetDateTimeString())
}

function getTimestamp() {
    // Get current value as Unix timestamp
    let ts = eventDateTime.GetUnixTimestamp()
    Console.WriteLine("Unix timestamp: " + ts)

    // Send to external system
    SetData("eventTimestamp", ts)
}

function syncWithServer() {
    // Receive timestamp from server
    let serverTimestamp = arc.GetData("serverTime")
    if (serverTimestamp)
      eventDateTime.SetFromUnixTimestamp(serverTimestamp)
}
```

Countdown Timer:

```
let countdownInterval = null;

Global.OnCreated = function() {
    // Set target date
    targetDateTime.SetDate(2025, 12, 31)
    targetDateTime.SetTime(23, 59, 59)
    // Start countdown
    countdownInterval = setInterval(function() {
        updateCountdown();
    }, 1000)
}
Global.OnDestroyed = function() {
    if (countdownInterval)
        clearInterval(countdownInterval);
}

function updateCountdown() {
    let now = new Date()
    let target = new Date(targetDateTime.Value)
    let diff = target - now // milliseconds

    if (diff <= 0) {
        countdownText.Value = "EVENT STARTED!"
        clearInterval(countdownInterval)
```

```
            return
        }

        // Convert to days, hours, minutes, seconds
        let days = Math.floor(diff / (1000 * 60 * 60 * 24))
        let hours = Math.floor((diff % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60))
        let minutes = Math.floor((diff % (1000 * 60 * 60)) / (1000 * 60))
        let seconds = Math.floor((diff % (1000 * 60)) / 1000)

        countdownText.Value = days + "d " + hours + "h " + minutes + "m " + seconds + "s"
    }
```

Date Arithmetic:

```
function addDays(days) {
    let current = new Date(eventDateTime.Value)
    current.setDate(current.getDate() + days)
    eventDateTime.SetDateTime(current)
}

function addHours(hours) {
    let current = new Date(eventDateTime.Value)
    current.setHours(current.getHours() + hours)
    eventDateTime.SetDateTime(current)
}

Global.OnButtonPressed = function(id) {
    if (id === "addOneDay")
        addDays(1)
    else if (id === "addOneHour")
        addHours(1)
    else if (id === "setToNow")
        eventDateTime.SetDateTime(new Date())
    else if (id === "setToMidnight")
        eventDateTime.SetTime(0, 0, 0)
}
```

Working with JavaScript Date Objects:

```
Global.OnCreated = function() {
    // DateTime parameter value is ISO string, compatible with JS Date
    let jsDate = new Date(eventDateTime.Value)
    Console.WriteLine("JavaScript Date: " + jsDate)

    // Set from JavaScript Date
    let tomorrow = new Date()
    tomorrow.setDate(tomorrow.getDate() + 1)
    eventDateTime.Value = tomorrow.toISOString()
}
```

> **Note:** The Value property always uses ISO 8601 format ("2025-01-15T14:30:00"), which is compatible with JavaScript's Date constructor and most APIs.

> **Note:** DateTime values are stored without explicit timezone information. If you need timezone-aware dates, handle timezone conversion in your script.

> **Note:** Year, Month, Day, Hour, Minute, and Second properties are read-only. Use SetDate() or SetTime() methods to modify them.

Directory

- string **Value** [Get, Set]
  - Gets/sets the selected directories fullpath.
  - Set: The input value must be a valid directory in the file system.

File

- string **Value** [Get, Set]
  - Gets/sets the selected file's fullpath.
- bool **WatchFile** [Get, Set]
  - Enable/disable the watchfile feature.
- bool **ReadRawContent** [Get, Set]
  - Set to true if the raw file content should be read into the DataMap. Set to false if it is an excel or csv file.
- string **Separator** [Get, Set]
  - The separator when reading a csv file. Default is “,”.
- string **Sheet** [Get, Set]
  - The name of the excel sheet to be read.
- string **StartCell** [Get, Set]
  - The name of the start cell to read the content from (for example, “B5”).
- string **EndCell** [Get, Set]
  - The name of the end cell to read the content from (for example, “H11”).
- bool **HasHeaders** [Get, Set]
  - Indicates whether or not the first row of the table content is a header row.
- bool **ConvertToJson** [Get, Set]
  - Whether to convert the content of the file into a json format.
- string **DataMapTarget** [Get, set]
  - The name of the key to be used in the DataMap to send the content of the file to.
- void **ReloadFile**()
  - Forces to read the content of the file. Might be used when **WatchFile** is disabled.

Asset

- string **Value** [Get, Set]
  - Get: Returns the selected asset's full path.

    - Set: The input path needs to be valid. The image is loaded asynchronously.
- string **Prefix** [Get, Set]
    - The prefix to be added to the **Value** when sent to the engine.
- string **Postfix** [Get, Set]
    - The postfix to be added to the **Value** when sent to the engine.
- async void **SetImage** (string name)
    - An awaitable method to set an image *name*.
- void **EraseImage**( )
    - Clears the current image (assigning null to `Value` has no effect — use this instead).

Note: Valid input values are: Graphic Hub items (Image, Geom, Material), Media service links (http://...) or local file system files.

WebView

This component lets you view a web page.

- string **Value** [Get, Set]
    - Gets/sets the URL of the web page to be visualized.
- void **Reload** ()
    - Reloads the current page.
- void **GoBack** ()
    - Navigates back in history.
- void **GoForward** ()
    - Navigates forward in history.
- void **ExecuteJavascript** (string method)
    - Invokes *method* on the currently loaded web page.
- void **ExecuteJavascript** (string method, params object[] args)
    - Invokes *method* with *args* as arguments on the currently loaded web page.

Note: The browser used for rendering is based on CEF. It might not play all video codecs.

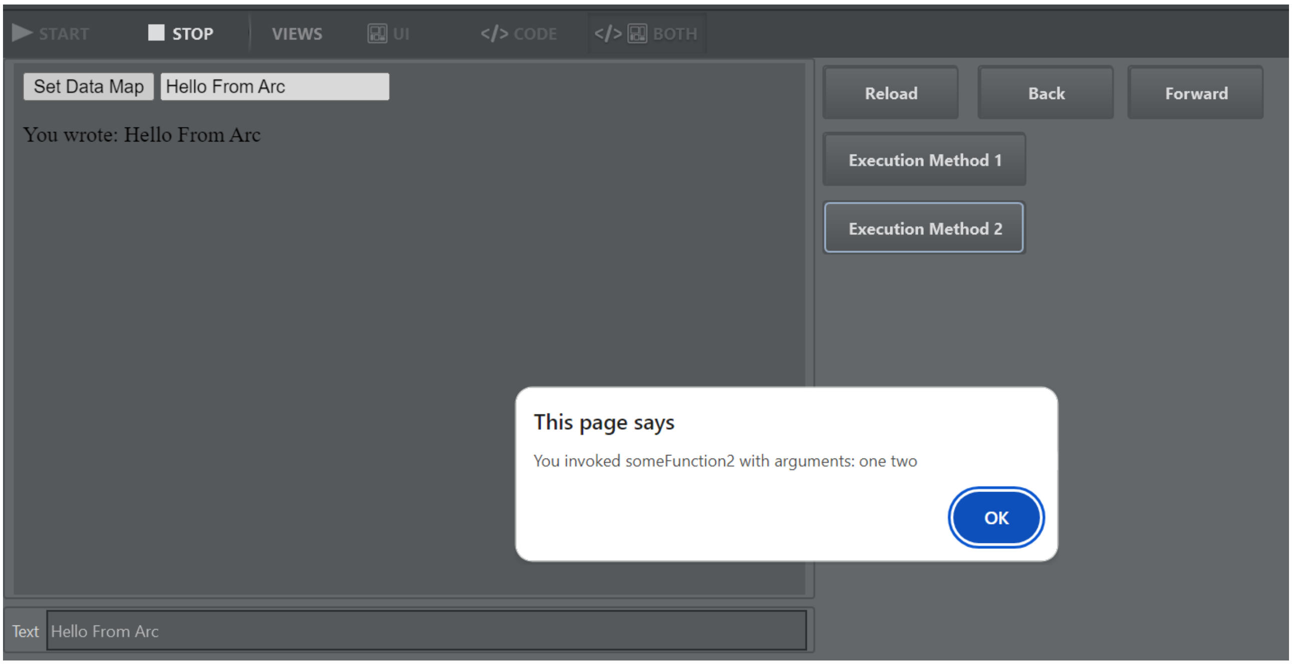
A sample html file that might be loaded in a WebView:

```
<!DOCTYPE html>
<html>
<body>

<button onclick="DataMapButton()">Set Data Map</button>
<input type="text" id="someText" oninput="wroteSomeText()">

<p id="demo"></p>

<script>
function DataMapButton() {
  let text = document.getElementById("someText").value;
  // set the data map variable 'fromWebPage' to the entered value
```

```
  arc.SetData("fromWebPage", text);
}
function wroteSomeText() {
  let text = document.getElementById("someText").value;
  document.getElementById("demo").innerHTML = "You wrote: " + text;
  // interact with arc and set the parameter "text_0" to the value just entered
  arc.SetParameterValue("text_0", text);
}

function someFunction1()
{
   alert("You invoked someFunction1!")
}

function someFunction2(para1, para2)
{
   alert("You invoked someFunction2 with arguments: " + para1 + " " + para2)
}
</script>
</body>
</html>
```

Note the **arc** object which gets injected and gives access to all scripting methods available in Viz Arc. For example, *arc.SetData("matchResult", "0:1")* sets a variable on the **DataMap**.

A Viz Arc template interacting with the html page above after clicking the *Execute Method 2* button:



The code of the Viz Arc template might look like this, note the **ExecuteJavascript** methods that allow you to interact with the web page.

```
Global.OnButtonPressed = function (id)
{
    switch( id )
    {
    case "reloadBrowser":
        webview_0.Reload()
        break
    case "backBrowser":
        webview_0.GoBack()
        break
    case "forwardBrowser":
        webview_0.GoFoward()
        break
    case "exe1":
        webview_0.ExecuteJavascript("someFunction1")
        break
    case "exe2":
        webview_0.ExecuteJavascript("someFunction2", "one", "two")
        break
    }
}
```

Bool

- bool **Value** [Get, Set]
  - Gets/sets the parameter's bool value.

Button

- void **Click** ( )
  - Triggers a click event on the button parameter.
- string **BackgroundImage**
  - Gets/sets the path to the background image. It can be either a local file path (for example, *c:/tmp/someimage.png*) a Graphic Hub path (for example, *IMAGE*/project/A/imageButton*) or a URL (for example, *http://storage.internal/image.jpg*).
- string **ImageMargin**
  - Gets/sets the margins in pixels of the background image. Specify either one, two or four comma separated margins. A value of `5` applies a margin of 5 pixels in all directions, a value of `5, 3` applies a margin of 5 pixels on the left and right and a margin of 3 at the top and bottom. A value of `1, 2, 3, 4` applies the respective margins in the order left, top, right and bottom.
- string **DirectorPath**
  - Gets/sets the Stage Director to be executed on click. For example, *$Director$SubDirector*
- string **DirectorExecute**
  - Gets/sets the type of action type to execute on click. Possible values are *<nothing>, START, CONTINUE, CONTINUE REVERSE, PAUSE or RESET*.
- string **ActionExecute**
  - Gets/sets the name or UUID of the action to be executed on click.

- bool **TakeEngineSnapshotSync** (width = -1, height = -1, withAlpha = true, timeoutMS = 5000)
    - Takes a snapshot from Viz Engine. Viz Engine is selected from the currently selected channel or in template designer it is taken from the Viz Editing Engine. This command works only with Viz Engine.
        - **width** of the snapshot in pixels. `-1` uses the button's current width. `0` captures the full frame width.
        - **height** of the snapshot in pixels. `-1` uses the button's current height. `0` captures the full frame height.
        - **withAlpha** `true` , captures with alpha channel. When `false` , captures without alpha.
        - **timeout** in milliseconds for the snapshot operation.
- async <bool> **TakeEngineSnapshot**(width = -1, height = -1, withAlpha = true, timeoutMS = 5000)
    - awaitable method of the **TakeEngineSnapshotSync.**
- void **ClearSnapshot**()
    - Clears the snapshot background image and resets all related properties to their default values.

Toggle Button

- bool **Value** [Get, Set]
    - Alias for **IsChecked**.
- void **SetCheckedColor** (byte r, byte g, byte b, byte a = 255)
    - Sets the toggle's color to the input RGBA color when the toggle is in it's checked state.
- string **CheckedColor**
    - Gets/sets the toggle's color in Hex format, for example, #FF0A0A8C (#RRGGBBAA) when the toggle is in it's checked state.
- int **CheckedColorR**
    - Gets/sets the toggle's red color value in the range **[0, 255]** when the toggle is in it's checked state.
- int **CheckedColorG**
    - Gets/sets the toggle's green color value in the range **[0, 255]** when the toggle is in it's checked state.
- int **CheckedColorB**
    - Gets/sets the toggle's blue color value in the range **[0, 255]** when the toggle is in it's checked state.
- int **CheckedColorA**
    - Gets/sets the toggle's alpha value in the range **[0, 255]** when the toggle is in it's checked state.
- string **CheckedLabel**
    - Gets/sets the label text displayed when the button is in the checked state.
- string **CheckedImage** [Get, Set]
    - Gets/sets the background image shown while the button is checked.

- string **BackgroundImage**
    - Gets/sets the path to the background image. It can be either a local file path (for example, *c:/tmp/someimage.png*) a Graphic Hub path (for example, *IMAGE*/project/A/imageButton*) or a URL (for example, *http://storage.internal/image.jpg*).
- string **ImageMargin**
    - Gets/sets the margins in pixels of the background image. Specify either one, two or four comma separated margins. A value of `5` applies a margin of 5 pixels in all directions, a value of `5, 3` applies a margin of 5 pixels on the left and right and a margin of 3 at the top and bottom. A value of `1, 2, 3, 4` applies the respective margins in the order left, top, right and bottom.

- bool **IsChecked**
    - Gets/sets the toggle button's state to checked or unchecked.
- string **ContainerPath**
    - Gets/sets the Viz scene container path (for example, *$object$ALL$left*) to be used when **VisibilityCheckd/VisibilityUncheckd** or **KeyChecked/KeyUnchecked** actions are set.
- string **DirectorPath**
    - Gets/sets the Stage Director path (for example, *$Director$SubDirector*)to be executed when **DirectorCheckd/DirectorUnchecked** actions are set.
- string **ActionChecked**
    - The action name or uuid to be executed when the toggle button gets **checked**.
- string **ActionUnchecked**
    - The action name or uuid to be executed when the toggle button gets **unchecked**.
- string **VisibilityChecked**
    - Gets/sets the visibility of the container specified in **ContainerPath** when the toggle button gets **checked**. Possible values are *<nothing>, ON or OFF*.
- string **VisibilityUnchecked**
    - Gets/sets the visibility of the container specified in **ContainerPath** when the toggle button gets **unchecked**. Possible values are *<nothing>, ON or OFF*
- string **DirectorChecked**
    - Gets/sets the action to be executed on the director specified in **DirectorPath** when the toggle gets **checked**. Possible values are *<nothing>, START, CONTINUE, CONTINUE REVERSE, PAUSE or RESET*.
- string **DirectorUnchecked**
    - Gets/sets the action to be executed on the director specified in **DirectorPath** when the toggle gets **unchecked**. Possible values are *<nothing>, START, CONTINUE, CONTINUE REVERSE, PAUSE or RESET*.
- string **KeyChecked**
    - Gets/sets the key action of the container specified in **ContainerPath** when the toggle button gets **checked**. Possible values are *<nothing>, ACTIVE, INACTIVE, COMBINE WITH BG ON, COMBINE WITH BG OFF*.
- string **KeyUnchecked**
    - Gets/sets the key action of the container specified in **ContainerPath** when the toggle button gets **un checked**. Possible values are *<nothing>, ACTIVE, INACTIVE, COMBINE WITH BG ON, COMBINE WITH BG OFF*.
- string **OverlayID**
  Gets/sets the Flowics overlay ID to control automatically when the button is toggled. Only works with Flowics templates.

Double / Double Slider

- double **Value** [Get, Set]
    - Gets/sets the parameter's double value.
- double **MinValue** [Get, Set]
    - Gets/sets the parameter's minimum double value. Input value needs to be lower than the current MaxValue.
- double **MaxValue** [Get, Set]

    - Gets/sets the parameter's maximum double value. Input value needs to be higher than the current MinValue.
- bool **ShowReset** [Get, Set]
    - Enable or disable the reset to the default button on the UI.

Double Slider only:

- double **MinRange** [Get, Set]
    - Gets/sets the parameter's minimum range double value.
- double **MaxRange** [Get, Set]
    - Gets/sets the parameter's maximum range double value.
- bool **RangeEnabled** [Get, Set]
    - If true, forces the slider to remain within the specified Min/Max range.

Dropdown / Radio

- object **Value** [Get, Set]
    - Gets/sets the selected entry on the dropdown. By default gets/sets the selected option's text. When the parameter's **Value As Index** option is enabled, gets/sets the zero-based index of the selected option instead (numbers assigned to it select by index).
- int **SelectedIndex** [Get, Set]
    - Gets/sets the selected index of the dropdown.
- string **SelectedText** [Get]
    - Always returns the selected option's text, regardless of the Value As Index mode.
- int **Count** [Get]
    - Gets the number of entries on the dropdown.
- int **IndexOf** (string option)
    - Looks for an entry equal to *option*. Returns its index if found, -1 otherwise.
- void **Insert** (int index, string option)
    - Inserts an entry with value *option* at *index* position. *index* needs to be between 0 and Count.
- void **Add** (string option)
    - Adds an entry with value *option* atthe end of the entry list.
- void **Remove** (string option)
    - Looks for an entry equal to *option*. Removes it if found.
- void **RemoveAt** (int index)
    - Removes the entry at position *index*. *index* needs to be between 0 and Count.
- void **SetItems** (string[] entries)
    - Sets the dropdown's entry list to the input *entries*.
- string **Get** (int index)
    - Returns the entry located at *index* position. *index* needs to be between 0 and Count.
- string parameter[int index]
    - Array accessor for entries. Returns the entry located at *index* position.
- void **Clear** ( )
    - Removes all entries from the dropdown.

Radio only:

- string **CheckedColor** [Get, Set]

    - Gets/sets the color of the checked radio option as hex string.
- void **SetCheckedColor** (int r, int g, int b, int a = 255)
    - Sets the checked color by components (0–255).

Int / Int Slider

- int **Value** [Get, Set]
    - Gets/sets the parameter's int value.
- int **MinValue** [Get, Set]
    - Gets/sets the parameter's minimum int value. Input value needs to be lower than the current MaxValue.
- int **MaxValue** [Get, Set]
    - Gets/sets the parameter's maximum int value. Input value needs to be higher than the current MinValue.
- bool **ShowReset** [Get, Set]
    - Enable or disable the reset to the default button on the UI.

MultiText / Text

- string **Value** [Get, Set]
    - Gets/sets the parameter's text value.

Triplet

- double **X** [Get, Set]
    - Gets/sets the parameter's X double value.
- double **Y** [Get, Set]
    - Gets/sets the parameter's Y double value.
- double **Z** [Get, Set]
    - Gets/sets the parameter's Z double value.
- double **DefaultX** [Get, Set]
    - Gets/sets the default parameter’s X value.
- double **DefaultY** [Get, Set]
    - Gets/sets the default parameter’s Y value.
- double **DefaultZ** [Get, Set]
    - Gets/sets the default parameter’s Z value.
- bool **XEnabled** [Get, Set]
    - Gets/sets the enabled status of the X value.
- bool **YEnabled** [Get, Set]
    - Gets/sets the enabled status of the Y value.
- bool **Z Enabled** [Get, Set]
    - Gets/sets the enabled status of the Z value.
- bool **AllowProportional** [Get, Set]
    - Gets/sets whether the user can toggle the proportional lock.
- bool **IsProportional** [Get, Set]

- Gets/sets the state of the proportional lock.

Table

Properties

- string **Value** [Get]
  - Gets an string containing the table content in a XML format (much like ControlList).
- int **MinimumRows** [Get, Set]
  - Gets/sets the parameter's minimum number of rows. Input value needs to be lower than the current MaximumRows.
- int **MaximumRows** [Get, Set]
  - Gets/sets the parameter's maximum number of rows. Input value needs to be higher than the current MinimumRows.
- int **MinimumColumns** [Get, Set]
  - Gets/sets the parameter's minimum number of columns. Input value needs to be lower than the current MaximumColumns.
- int **MaximumColumns** [Get, Set]
  - Gets/sets the parameter's maximum number of columns. Input value needs to be higher than the current MinimumColumns.
- int **RowCount** [Get]
  - Gets the current number of rows on the table.
- int **RowHeight** [Get, Set]
  - Gets/sets the row height in pixels.
- int **ColumnCount** [Get]
  - Gets the current number of columns on the table.
- int **SelectedRow** [Get]
  - Gets the currently selected row. In case of multi-selection it returns the first selected row. If no row is selected, `-1` is returned.
- int **SelectedIndex** [Get]
  - An alias for **SelectedRow**.
- bool **ShowRowNumbers** [Get, Set]
  - Shows/hides the row-number column.
- bool **MatchHeaders** [Get, Set]
  - Tries to match headers of the incoming data, with the names of the column header names of the table.
- bool **MatchDataColumns** [Get, Set]
  - Creates and matches columns of the table according to incoming data.
- bool **MatchDataRows** [Get, Set]
  - Creates and matches as many rows in the table as there are in the incoming data.
- bool **TransposeData** [Get, Set]
  - Swaps rows and columns, when set to true.
- bool **TriggerOnAllChanges** [Get, Set]
  - Triggers the **OnTableCellValueChanged** callback when set to true, for when a table cell value has been changed by user input.

- bool **CanUserChangeNumberRows** / **CanUserReorderRows** / **CanUserResizeRows** / **CanUserChangeNumberColumns** / **CanUserReorderColumns** / **CanUserResizeColumns** [Get, Set]
    - Enable/disable the corresponding user interactions in the operator UI.

Methods

Cell Handling

- BaseCell **Accessor** [int row, int column] [Get]
    - Gets the cell located at *row*-indexed row and *column*-indexed column.
- BaseCell **GetCell** (int row, int column)
    - Gets the cell located at *row*-indexed row and *column*-indexed column.
- void **SetCellValue** (int row, int col, dynamic value)
    - Sets the cell's value (located at [row, column]) to *value.* [dynamic] *value* can either be a string or have a type that is compatible with the target cell.
- string **GetCellValue** (int row, int col)
    - Gets the cell's (located at [row, column]) string value representation.
- void **SetFromArray** (string[] values)
    - Fills the table from a string array.
- void **ClearColumnValues** (int columnIndex)
    - Resets all the cell's values in *columnIndex* column.
- void **ClearRowValues** (int rowIndex)
    - Resets all the cell's values in *rowIndex* row.
- void **ClearAllValues** ( )
    - Resets all the cell's values.
- void **Clear** ( )
    - Removes all the content (all columns and rows are deleted).

Columns Handling

Inserting a column from code requires the user to specify the type of column that needs to be created, the valid column types are:

- **bool:** Column with BoolCell
- **string:** Column with StringCell
- **int:** Column with IntCell
- **ivec2:** Column with IntDupletCell
- **ivec3:** Column with IntTripletCell
- **double:** Column with DoubleCell
- **dvec2:** Column with DoubleDupletCell
- **dvec3:** Column with DoubleTripletCell
- **asset:** Column with AssetCell
- **dropdown:** Column with dropdown items

All column interactions take into consideration the maximum and minimum number of columns of the table

- void **AddColumn** (string columnType)

- void **AddColumn** (string columnType, string name)
    - Adds a column of type *columnType* named *name* if specified, otherwise the default is used.
- void **AddColumn**(string columnType, string controlObjectId, string displayName)
    - Adds a column of type *columnType* named *displayName*. The *controlObjectId* is the internal name used for ControlObject mapping.
- void **AddMultipleColumn** (int count, string columnType)
    - Adds *count* columns of *columnType* type.
- void **InsertColumn** (int index, string columnType)
- void **InsertColumn** (int index, string columnType, string name)
    - Inserts a column at *index* index of *columnType* type named *name* if specified, otherwise the default is used.
- void **InsertColumn**(int index, string columnType, string controlObjectId, string displayName)
    - Inserts a column at *index* index of *columnType* type named *displayName*. The *controlObjectId* is the internal name used for ControlObject mapping.
- void **InsertMultipleColumn** (int index, string columnType, int count)
    - Inserts *count* columns at *index* index of *columnType* type.
- void **RemoveColumnAt** (int index)
    - Removes column at *index* index.
- void **MoveColumn** (int targetIndex, int newPosition)
    - Moves column from *targetIndex* position to *newPosition*.
- void **ClearColumns** ( )
    - Removes all columns.
- double **GetColumnWidth** (int index)
    - Returns the column *width* in pixels of column.
- void **SetColumnWidth** (int index, double width)
    - Sets the column *width* in pixels of column with index *index*.
- string **GetColumnName** (int index)
    - Returns the column *label*.
- void **SetColumnName** (int index, string name)
    - Sets the column label to *name* of column with index *index*.
- void **SetColumnDisplayName** (int index, string name) / string **GetColumnDisplayName** (int index)
    - Sets/gets the column's display label (SetColumnName/GetColumnName are aliases).
- void **SetColumnControlObjectID**(int index, string controlObjectId)
    - Sets the column's internal ControlObject ID used for mapping in case it is bound to a Viz Engine's ControlList object.
- int **GetColumnIndexByName** (string name)
    - Returns the index of the first column matching *name*. Returns -1 otherwise.
- int **GetColumnIndexByControlObjectID**(string name)
    - Returns the index of the first column matching the control object *name*. Returns -1 otherwise. This can be used for ControlList generated tables in Viz Templates.
- string **GetColumnControlObjectID**(int index)
    - In case the table is bound to a Viz Engine's ControlList object, this function returns the id associated to the colum's ControlObject.
- void **SetColumnReadOnly** (int index, bool isReadOnly)
- void **SetColumnReadOnly** (string name, bool isReadOnly)

    - Sets a column to be read only or not.
- bool **GetColumnReadOnly** (int column)
- bool **GetColumnReadOnly** (string name)
    - Returns whether a column is read only.
- void **SetColumnEditorOnly** (int index, bool editorOnly)
- void **SetColumnEditorOnly** (string name, bool editorOnly)
- bool **GetColumnEditorOnly** (int column)

DropDown Handling

- void **SetDropdownOptions** (int column, string [] options)
    - Set dropdown items for an entire column (column must be of type 'dropdown')
- string[] **GetDropdownOptions** (int column)
    - Returns a list of dropdown options for *column*.
- void **AddDropdownOption** (int column, string option)
    - Appends *option* to all dropdowns in *column*.
- void **RemoveDropdownOption** (int column, string option)
    - Removed the first occurence of *option* from all dropdowns in *column*.
- void **ClearDropdownOptions** (int column)
    - Clears all dropdowns in *column*.
- void **SetCellDropdownOptions** (int row, int column, string [] options)
    - Set dropdown items in specific cell (column must be of type 'dropdown')
- string[] **GetCellDropdownOptions** (int row, int column)
    - Returns an array of the items of the dropdown in row/column.
- void **ClearCellDropdownOptions** (int row, int column)
    - Clears the items of the items of the dropdown in row/column

Rows Handling

All row interactions take into consideration the maximum and minimum number of rows of the table

- void **SetNumberRows** (int count)
    - Adds/removes rows until the table's RowCount is equal to *count*.
- void **AddRow** ( )
    - Adds a Row to the table.
- void **AddMultipleRow** (int count)
    - Adds *count* rows to the table.
- void **InsertRow** (int index)
    - Inserts a row at *index* position to the table.
- void **InsertMultipleRow** (int index, int count)
    - Inserts *count* rows at *index* position to the table.
- void **RemoveRowAt** (int index)
    - Removes row at *index* position.
- void **MoveRow** (int targetIndex, int newPosition)
    - Moves row from *targetIndex* position to *newPosition*.
- void **ClearRows** ( )
    - Removes all rows.

Table Parameter Example

```
// Open the cvs file with the starters, parse the content and add all riders to the
RaceTable (TableParameter)
function LoadRaceTable()
{
    // Setup columns from UI, Comment if already done manually
    //RaceTable.Clear()
    //RaceTable.AddColumn( "string", "Horse")
    //RaceTable.AddColumn( "string", "Trainer")
    //RaceTable.AddColumn( "string", "Jockey")
    //RaceTable.AddColumn( "string", "Owner")
    //RaceTable.AddColumn( "string", "Colors")
    //RaceTable.AddColumn( "string", "Horse CN")
    //RaceTable.AddColumn( "asset", "Silk")

    // Clear rows
    RaceTable.ClearRows()

    var i = 0
    var FileContent = arc.ReadTextFile("D:/Horses/Starter.csv")
    var EntryArr = FileContent.split("\n")

    // First line is for the headers, ignore it
    for(i = 1; i < EntryArr.length; i++)
    {
        // Split the rider content
        var splitContent = EntryArr[i].split(",")

        // CVS file has great amount of data but we only want to display certain
stuff
        RaceTable.AddRow()
        RaceTable.GetCell(i-1, 0).Value = splitContent [19]
        RaceTable.GetCell(i-1, 1).Value = splitContent [22]
        RaceTable.GetCell(i-1, 2).Value = splitContent [25]
        RaceTable.GetCell(i-1, 3).Value = splitContent [27]
        RaceTable.GetCell(i-1, 4).Value = splitContent [34]
        RaceTable.GetCell(i-1, 5).Value = splitContent [20]
        // image assets need to be assigned using the SetCellValue method
        RaceTable.SetCellValue(i-1, 6, splitContent[21])
    }
}
```

OCR

The OCR parameter reads text from regions of an **NDI** video source (only NDI sources are supported; the source and the regions are configured in the parameter's UI). Each recognized value is written to the DataMap under the key

`<shared memory prefix><region name>`, where scripts can consume it through the DataMap functions and the `OnDataMapValueChanged` callback.

- bool **Tracking** [Get, Set]
  - Starts/stops the OCR analysis.
- void **RefreshVideoSources** ( )
  - Refreshes the list of available video sources.

4.1.46 Unreal

These are helper functions to invoke BluePrint functions:

- void **InvokeBPFunction** (string blueprintName, string functionName, params object[] arg)
  - Invokes the function *functionName* on the Blueprint *blueprintName,* with a variable number of parameters (typically strings and/or numbers).
- void **InvokeBPFunction** (string blueprintName, string functionName)
  - Invokes the function *functionName* on the Blueprint *blueprintName*, without any arguments.

The methods are invoked on all Unreal Engines of the template's currently selected channel.

```
Global.OnExecute = function ()
{
    // before executing the template, update some data on the Blueprint invoking
updateData
    InvokeBPFunction("BP_main", "updateData", "Hello", "World", 42.0, 3)
    // change the appearance of the car
    InvokeBPFunction("BP_CarManager_Blue", "Change Car", 1, true, "This is the new
car model");
}
```

4.1.47 Flowics

Overview

Flowics templates can programmatically control individual overlays through scripting. These methods only work with Flowics-type templates.

Available Methods

- void **ShowOverlay**(string ids)
  - Show one or more overlays. Parameter is space-separated overlay IDs (for example, *n10717 n10718*).
- void **HideOverlay**(string ids)
  - Hide one or more overlays. Parameter is space-separated overlay IDs.
- void **SetOverlayState**(string id, string state)
  - Set overlay state for a single overlay. State can be `in`, `out`, or `idle`.
- void **SetOverlayStates**(string states)

    - Set multiple overlay states using prefix notation. Use + to show, - to hide (for example, *+n10717 -n10718*).
- void **ShowAllOverlays**()
    - Show all overlays in the template.
- void **HideAllOverlays**()
    - Hide all overlays in the template.
- void **GotoFirst**(string overlayId, string controlId)
    - Navigates to the first item in a Flowics list item.
        - overlayId - Overlay ID (for example, *n10717*)
        - controlId - Control ID within the overlay control (for example, *list-1*, *carousel-main*)
- void **GotoNext**(string overlayId, string controlId)
    - Navigates to the next item in a Flowics control.
        - overlayId - Overlay ID (for example, *n10717*)
        - controlId - Control ID within the overlay control (for example, *list-1*, *carousel-main*)
- void **GotoPrev**(string overlayId, string controlId)
    - Navigates to the previous item in a Flowics control.
        - overlayId - Overlay ID (for example, *n10717*)
        - controlId - Control ID within the overlay control (for example, *list-1*, *carousel-main*)
- void **GotoItem**(string overlayId, string controlId, string path, string value)
    - Navigates to a specific item in a Flowics list based on a field value match.
        - overlayId - Overlay ID (for example, *n10717*)
        - controlId - Control ID within the overlay control (for example, *list-1*, *carousel-main*)
        - path - Path to the field to match (for example, *id*, *name*)
        - value - Value to search for in the specified field
- void **FlowicsSetTimer**(string timerId, string timerValue)
    - Sets a Flowics timer/stopwatch to a specific value without starting it (paused state).
        - timerId - The global data provider ID for the timer
        - timerValue - The timer value to start from (for example, "00:00:00" or milliseconds)
- void **FlowicsSetTimerAndPlay**(string timerId, string timerValue)
    - Sets a Flowics timer/stopwatch to a specific value and starts playing.
        - timerId - The global data provider ID for the timer
        - timerValue - The timer value to start from (for example, "00:00:00" or milliseconds)
- void **FlowicsStartTimer**(string timerId, string timerValue)
    - An alias for above's **FlowicsSetTimerAndPlay**.
- void **FlowicsSetTimerAndPlayRange**(string timerId, string timerValue, string timerStart, string timerEnd)
    - Sets a Flowics timer/stopwatch with a range (start/stop values) and starts playing.
        - timerId - The global data provider ID for the timer
        - timerValue - The current clock value
        - timerStart - The start value for the range
        - timerEnd - The stop value for the range (timer stops when reached)
- void **FlowicsPauseTimer**(string timerId)
    - Pauses a Flowics timer/stopwatch.
        - timerId - The global data provider ID for the timer
- void **FlowicsPlayTimer**(string timerId)
    - Plays/resumes a Flowics timer/stopwatch.

        - timerId - The global data provider ID for the timer
- void **FlowicsSetTimerAndPause** (string timerId, string timerValue)
    - Sets a Flowics timer/stopwatch to a specific value and pauses it.
- void **FlowicsResumeTimer**(string timerId)
    - Alias for **FlowicsPlayTimer**.
- void **FlowicsResetTimer**(string timerId)
    - Resets a Flowics timer/stopwatch to its initial value.
        - timerId - The global data provider ID for the timer

Examples

Basic Overlay Control

```
function OnButtonPressed(id) {
    if (id=== "showLowerThird") {
        // Show single overlay
        ShowOverlay("n10717")
    }
    else if (id=== "hideLowerThird") {
        // Hide single overlay
        HideOverlay("n10717")
    }
}
```

Multiple Overlays

```
function showGraphicsPackage() {
    // Show multiple overlays at once (space-separated)
    ShowOverlay("n10717 n10718 n10719")
}

function hideGraphicsPackage() {
    HideOverlay("n10717 n10718 n10719")
}
```

Set Overlay State

```
function controlOverlay(overlayId, action) {
    if (action === "in") {
        SetOverlayState(overlayId, "in")
    }
    else if (action === "out") {
        SetOverlayState(overlayId, "out")
    }
```

```
    else if (action === "idle") {
        SetOverlayState(overlayId, "idle")
    }
}
```

Batch State Changes

```
function updateOverlayStates() {
    // Use prefix notation: + = show, - = hide
    // This shows n10717 and n10719, hides n10718
    SetOverlayStates("+n10717 -n10718 +n10719")
}
```

Show/Hide All

```
Global.OnExecute = function (){
    // Show all overlays when template goes on-air
    ShowAllOverlays()
}

Global.OnOut = function () {
    // Hide all overlays when template goes off-air
    HideAllOverlays()
}
```

ToggleButton Integration

Toggle buttons can automatically control Flowics overlays by setting the OverlayID property:

```
function ShowBreakingNews() {
    // Link toggle button to Flowics overlay
    myToggle.OverlayID = "n10717";
    // Now toggling the button automatically shows/hides the overlay
    myToggle.IsChecked = true;  // Shows overlay
}
```

⚠ **Note:** These methods only work with Flowics templates. Calling them on Viz or Unreal templates has no effect.

4.1.48 Video

Set the video and PTZ source.

- void **SetVideoSource** (string name)

    - Sets the name of the preview source window.
- void **SetNDIPTZVideoControl** (string ptzControl)
    - Sets the name of the NDI PTZ control overlay source.
- void **SetFlowicsOutput** (string URL)
    - Sets the global Flowics output URL.

4.1.49 Using async/await

Some methods are declared **async**, meaning the method might be time consuming (for example, while waiting for an answer from a server). To avoid locking up the UI, one can **await** an async method, such that Viz Arc can continue to process other events like user interaction. The async method might be processed on a different thread.

The following example shows how to retrieve the version of the first engine of the active channel on a template:

```
Global.OnButtonPressed = async function (id)
{
  if( id == "getFromEngineButton"){
    let answer = await GetFromEngineAsync("VERSION", 3000)
    Console.WriteLine("engine version is " + answer)
  }
}
```

Note that it is mandatory to declare the calling functions as **async,** when using **await**. The async keyword can be safely added to any Viz Arc callback, as shown with *OnButtonPressed*.

Using try/catch

It is highly recommended to use a try/catch block around awaited methods, otherwise, eventual errors are not detected

4.2 Profile

This section contains a list of properties and functions grouped by type that are useful for communicating with Profile, Channel, and Engine (Viz Engine and Unreal Engine).

- Scripting Profile
- Scripting Channel
- Scripting Engine

4.2.1 Scripting Profile

- string **Name** [Get]
  - Returns the profile's name.
- int **NumChannels** [Get]
  - Returns the number of channels.
- ScriptingChannel **VizEditingEngine** [Get]
  - Returns the configured Viz Editing Engine of the profile.
- ScriptingChannel **UnrealEditingEngine** [Get]
  - Returns the configured Unreal Editing Engine of the profile.
- ScriptingChannel **VizProgramChannel**[Get]
  - Returns the configured Viz program Engine of the profile.
- ScriptingChannel **UnrealProgramChannel**[Get]
  - Returns the configured Unreal program Engine of the profile.
- ScriptingChannel **VizPreviewChannel**[Get]
  - Returns the configured Viz preview Engine of the profile.
- ScriptingChannel **UnrealPreviewChannel**[Get]
  - Returns the configured Unreal preview Engine of the profile.
- ScriptingChannel **Accessor** [int index] [Get]
  - Returns the *index*-indexed Scripting Channel.
- ScriptingChannel **GetChannel** (int index)
  - Returns the *index*-indexed Scripting Channel.
- ScriptingChannel **GetChannel** (string channelName)
  - Returns the first channel found with name *channelName.*

4.2.2 Scripting Channel

- string **Name** [Get]
  - Returns the channel's name.
- int **NumChannels** [Get]
  - Returns numbers of Engines in the channel.
- ScriptingChannel **Accessor** [int index] [Get]
  - Returns the *index*-indexed Scripting Engine Class.
- void **SendSingleCommand** (string command)
  - Sends the command to all the Engines in the channel.
- void **SendCommands** (string[] commands)

    - Sends a list of commands to all the Engines in the channel.
- ScriptingEngine **GetEngineByName** (string name)
    - Returns the first Engine found with name.

4.2.3 Scripting Engine

- void **SendSingleCommand** (string command)
    - Sends the command to the Engine.
- void **SendCommands** (string[] commands)
    - Sends a list of commands to the Engine.
- string **QueryEngine** (string command, int timeout = -1)
    - Queries the Engine with command. If timeout is specified and larger than 0, the method times out after *timeout* milliseconds if it does not receive an answer within that time.
- Task<string> **QueryEngineAsync** (string command, int timeout = -1)
    - Queries the Engine with command asynchronously. If timeout is specified and larger than 0, the method times out after *timeout* milliseconds if it does not receive an answer within that time.
- string **GetFlowicsOutput**()
    - Returns the Flowics live output URL associated to the API token of this engine. Return null if it is not a Flowics output engine.

4.3 Control Object

After having accessed the action holding the list of ControlObjects thought the **GetAction** method, the single ControlObjects can be retreived using the global method

- BaseControlObject **GetControlObject**(BaseAction action, string ControlObjectID)

Most Control Object types have the following generic properties:

- **Text** (String)
  - This property adapts to all objects (execute string)
  - `IntControl.Text = "5"`
  - `ImageControl.Text = "IMAGE*FolderA/SubfolderB/ImageName"`
- **ID** (String)
  - Returns ObjectID
- **Description**
  - Returns the object description

Each Control Object type has specific properties:

- Control Container
- Control Image
- Control Material
- Control Omo
- Control Text
- Control List
  - Single Cells Properties
- Control Integer
- Control Double
- Control Boolean

4.3.1 Control Container

Properties:

- Visibility
- Position
  - posX (double)
  - posY (double)
  - posZ(double)
- Rotation
  - rotX (double)
  - rotY (double)
  - rotZ (double)
- Scaling
  - scaX (double)
  - scaY (double)

    - scaZ (double)

This type doesn't have the Text property.

4.3.2 Control Image

Properties:

- Path (string)
- Position
    - posX (double)
    - posY (double)
- Scaling
    - scaX (double)
    - scaY (double)

4.3.3 Control Material

Properties:

- Path (string)

4.3.4 Control Omo

Properties:

- Value (integer)

4.3.5 Control Text

Properties:

- Value (string)

4.3.6 Control List

Example:

```
sub OnInit()
    'declare variables
    dim objAction, table, cell
    dim output1, output2, output3
    'get table obj action
    objAction = arc.GetAction("object")
    table = arc.GetControlObject (objAction, "controlObj_ID")
    Console.WriteLine("Table name: " & table.Text)
    'set values in single cells inside the table
    table(0,0).value = false
```

```
    table(0,1).value = 5
    table(2,5).x = 12
    table(3,6).value = "IMAGE*/Default/GER"
    'assign values to a variable and show in debug console
    output1 = table(0,0).Text
    output2 = table(0,1).Text
    output3 = table(0,2).Text
    Console.WriteLine("cell - " & output1 & " | " & output2 & " | " & output3)
end sub
```

Properties:

- Accessor
  - **table[int row, int col]:** returns a cell
  - **nbcolumns (integer):** number of columns
  - **nbrows (integer):** numbers of rows

Single Cells Properties

Cell	Type	Additional Information	Example
BaseCell	Text (string)	Sets or gets value as string. Common to every cell type.	`table(0,5).x= 12 (intCell)` `table(0,5).text = "12" (intCell)`
BoolCell	Value (boolean)		`table(0,5).active= true`
DoubleCell	Value (double)		table(0,5).x= 12.8
DupletCell	X (double) Y (double)		table(0,5).text = "0.55 0.2"
GeomCell	Value (string)		table(0,5).value = "GEOM*/folder/geometry"
ImageCell	Value (string)		table(0,5).value = "IMAGE*/folder/image"
IntCell	Value (integer)		
MaterialCell	Value (string)		table(0,5).value = "MATERIAL*/folder/material"
TextCell	Value (string)		

Cell	Type	Additional Information	Example
TripletCell	X (double) Y (double) Z (double)		table(0,5).text = "0.55 0.23 1.23"

4.3.7 Control Integer

Properties:

- Value (integer)

4.3.8 Control Double

Properties:

- Value (double)

4.3.9 Control Boolean

Properties:

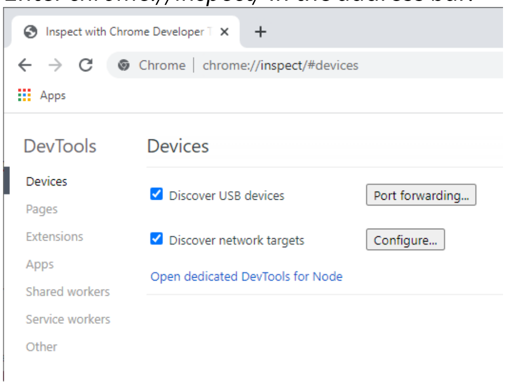
- Value (boolean)

5 Debugging Scripts

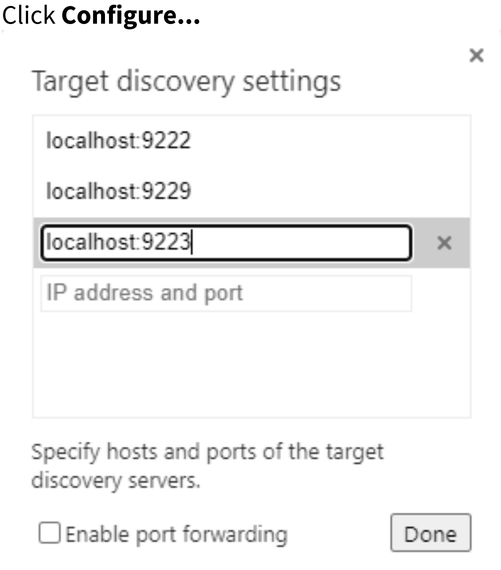
5.1 DevTools

You can use any chrome based web browser (for example, Google Chrome or Microsoft Edge) to step through Viz Arc scripts.

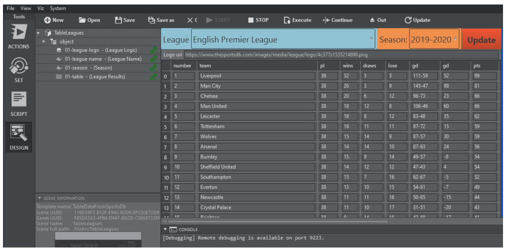
1. Open your browser.
2. Enter *chrome://inspect/* in the address bar.



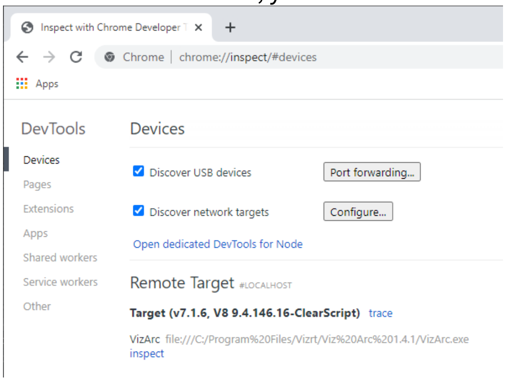
3. Click **Configure...**



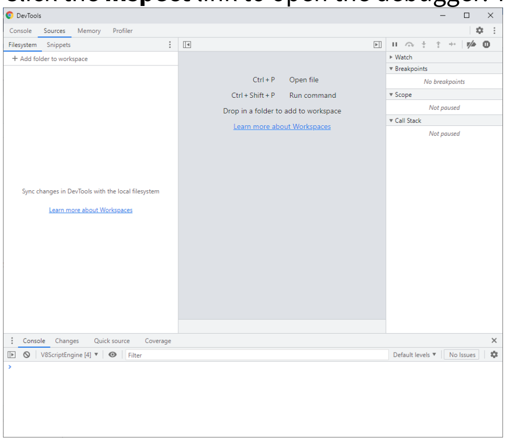
   Add the host name where Viz Arc is running and specify the debug script port (by default, port `9222` for the global script and port `9223` for the template builder script). Confirm by clicking the **Done** button.
4. Open a template in Viz Arc's template builder and start it.



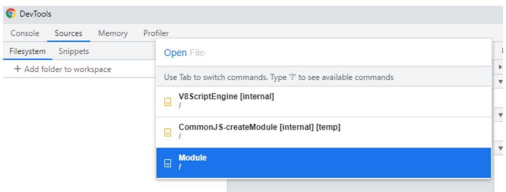
5. In the Chrome browser, you should now see the Viz Arc script available for debugging.



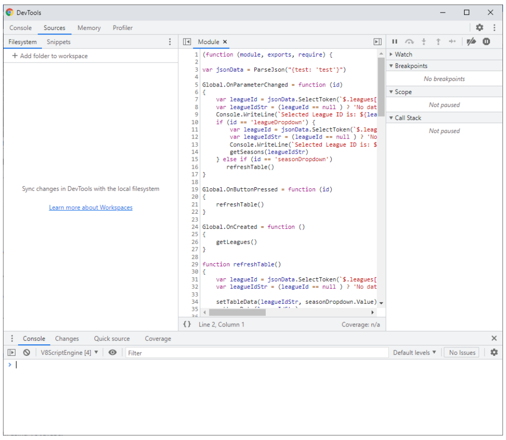
6. Click the **inspect** link to open the debugger. The first time you open the debugger it does not show any code.



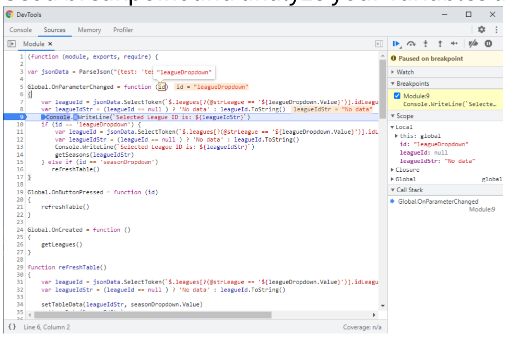
Hit **CTRL + P** and select *Module*.



You'll now be able to set breakpoints and see the code.



7. Set a breakpoint and analyze your variables and code.



5.2 Visual Studio Code

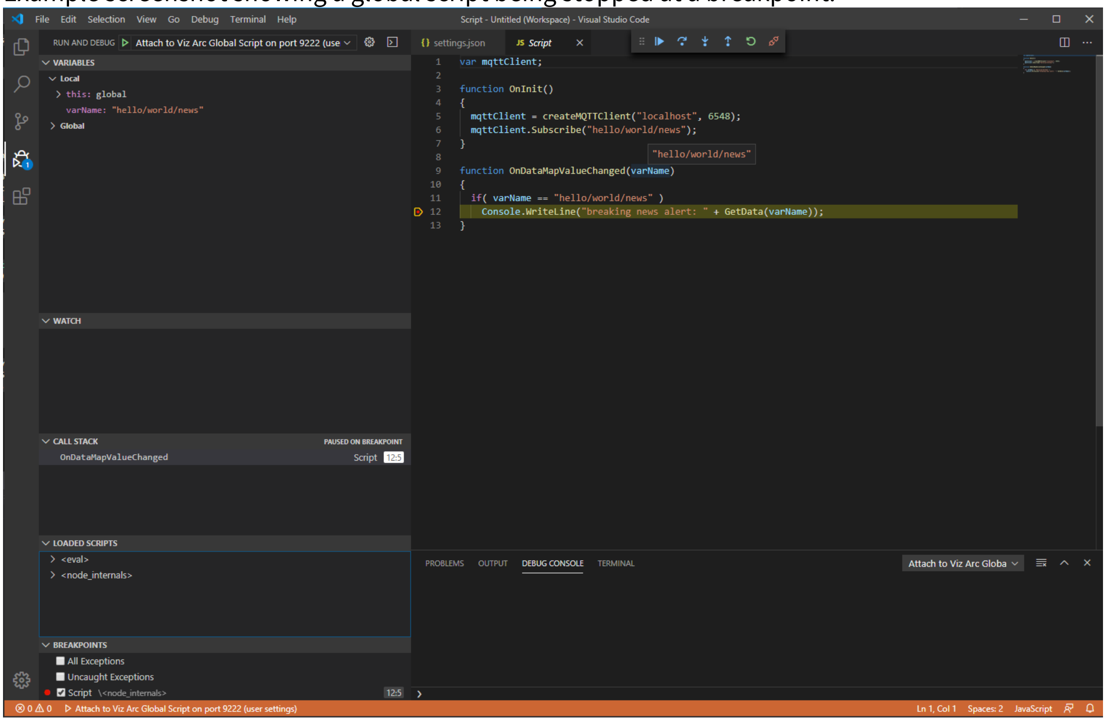
You can debug scripts with Visual Studio Code when using V8 JavaScript in the global script or in any scripted template.

1. Install and launch Visual Studio Code.
2. Set up one or more Viz Arc V8 debug configurations:
    a. Click **File** > **Preferences** > **Settings** to open your user settings.
    b. Locate or search for the **Launch** configuration and click **Edit in settings.json**.
    c. Add the following section to the file:

```
{
 "debug.javascript.usePreview": false,
 "launch": {
    "version": "1.2.0",
    "configurations": [
        {
            "name": "Attach to Viz Arc Global Script on port 9222",
            "type": "node",
            "request": "attach",
            "protocol": "inspector",
            "address": "localhost",
            "port": 9222
        },
```

```
            {
                "name": "Attach to Viz Arc Template Scrip on port 9223",
                "type": "node",
                "request": "attach",
                "protocol": "inspector",
                "address": "localhost",
                "port": 9223
            }
        ]
    }
}
```

   d. You can specify additional configurations for different hosts, port numbers, and other options. See Node.js debugging in VS Code for more information.
   e. Click **File > Save**.
3. If you'd like to debug your application remotely, you must also make sure that your firewall allows incoming connections to your TCP port.
4. Attach the Visual Studio Code debugger to your application:
   a. Click **View > Debug** to bring up the Debug view.
   b. Select the appropriate debug configuration at the top of the Debug Side Bar.
   c. Click **Debug > Start Debugging**.
   Example screenshot showing a global script being stopped at a breakpoint.



> **Note:** There are two different ports in use: One for the global script (default `9222`) and one for template scripts (default `9223`). Template scripts can be debugged only when running in the designer. The ports can be configured in the global configuration settings.

See Also

- Node.js debugging in VS Code