Script View

Version 1.1

Disclaimer

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

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

2020/02/10

Contents

1 Action Properties 8
1.1 Alpha 9
1.2 Chroma 9
1.3 Command 10
1.4 ControlObject 10
1.5 Director 11
1.6 Group 11
1.7 Image 11
1.8 Light 12
1.9 Key 12
1.10 Material 12
1.11 MSE 13
1.12 Multizone Chroma Key 13
1.13 Omo 13
1.14 PBR 13
1.15 RenderGraph 13
1.16 Scene Loader 14
1.17 Shared Memory 14
1.18 Telemetrics 14
1.19 Text 15
1.20 Tracking Hub Command 15
1.21 Transformation 15
1.22 Utah Router 15
1.23 Unreal Animation 15
1.24 Unreal Blueprint 16
1.25 Unreal Dispatcher 16
1.26 Unreal Scene Loader 16
1.27 Unreal Sequencer 16
1.28 Unreal Text 16
1.29 Vinten 16
1.30 Virtual Studio 17
1.31 Visibility 17
1.32 Viz Camera 17

1.33 Viz Clip ... 17
2 Control Object Classes ... 18
2.1 Control Container ... 18
2.2 Control Image ... 19
2.3 Control Material ... 19
2.4 Control Omo ... 19
2.5 Control Text ... 19
2.6 Control List ... 20
2.6.1 Single Cells Properties ... 20
2.7 Control Integer ... 21
2.8 Control Double ... 21
2.9 Control Boolean ... 21
3 Profiles Classes ... 22
3.1 Scripting Profile ... 22
3.2 Scripting Channel ... 22
3.3 Scripting Engine ... 22
4 Scripting Classes ... 24
4.1 General ... 25
4.2 Action ... 25
4.3 Control Object ... 26
4.4 Parameter ... 26
4.5 Channel ... 27
4.6 DataMap ... 28
4.7 Viz/Unreal Engine Communication ... 28
4.8 SMM Handling ... 29
4.9 GPI ... 29
4.10 Timer ... 30
4.11 StreamDeck ... 31
4.12 DataMap ... 32
4.13 File handling ... 33
4.14 Parameter Callbacks ... 33
4.15 Exposed Objects ... 34
4.15.1 Console ... 34
4.15.2 XmlDocument ... 34
4.15.3 FSO ... 35

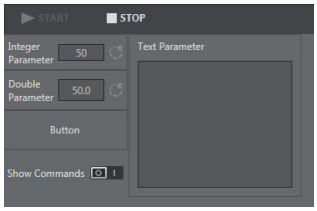
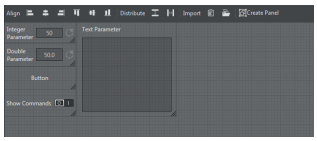
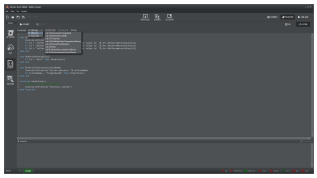
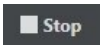
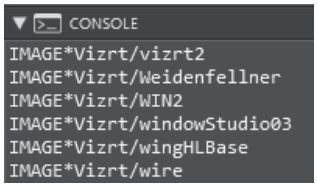
4.16 Main Script-only 35
4.16.1 Canvas Tabs Handling 35
4.16.2 Action Template Handling 36
4.16.3 Callbacks 36
4.17 Template Script - only 36
4.17.1 Action/Designer handling 36
4.17.2 Control Object Handling 37
4.17.3 Template action configuration 37
4.17.4 Callbacks 37
4.18 Parameters 38
4.18.1 Base parameters functionality 38
4.18.2 Layout 38
4.18.3 Dialogs 39
4.18.4 Input 40

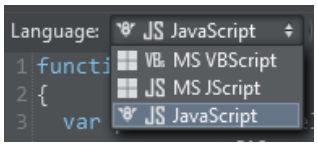
In Script View, you can write your own custom script in JavaScript language through Google's **V8** or Microsoft's **JScript** (ECMAScript3) or in **VBScript** language, as in the following example:

Language: JS JavaScript | Callbacks | Functions

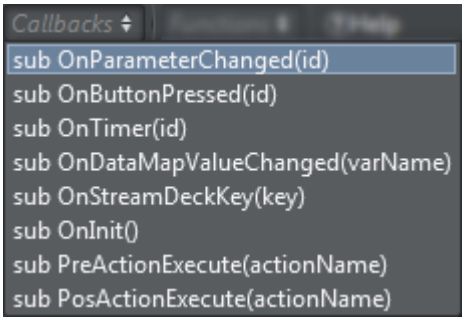
```
function OnInit()
{
  var profile = GetSelectedProfile();
  var eng = profile.VizEditingEngine;

  if( eng != null ){
    var images = eng.QueryEngine("IMAGE*/Vizrt GET");
    Console.WriteLine(images);

    var splitImgs = images.split(" " );

    for(var i = 0; i < splitImgs.length; ++i)
      Console.WriteLine(splitImgs[i]);
 }
}
```

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

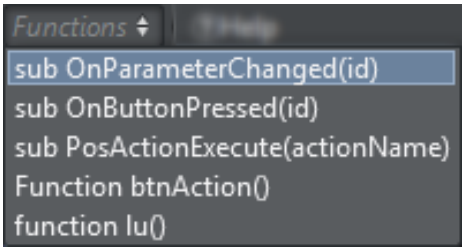
	To run the script, select the **Start** button  on the top left of the window.
	Form Design can be edited by selecting the **UI** button  . Every element can be selected and moved, aligned and distributed on the main form. To go back to code editing, select the **CODE** button.
	To edit a script, press the **Stop** button  on the top of the script main window.
	Console logs and debugs are displayed in the **CONSOLE** pane.

	Import code internally from the clipboard or an external text file in the script pane.
	Use the **Language** menu to select a scripting language.

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

1 Action Properties

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

- BaseAction **GetAction**(string actionName)

Note that you can have multiple actions in your project with the same name. In this case this function will return the first Action created with that name. Make sure to use unique names when accessing the actions through scripting.

Every Action type has these generic properties:

- **Name**
- **Description**

Every Action type has these generic methods:

- void **Execute()**

A sample on how to set the alpha value to 75% of an Alpha Action called *AlphaText* and executing the action from scripting:

Sample

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

There are specific properties/functions for each action type:

- Alpha
- Chroma
- Command
- ControlObject
- Director
- Group
- Image
- Light
- Key
- Material
- MSE
- Multizone Chroma Key
- Omo
- PBR
- RenderGraph
- Scene Loader
- Shared Memory

- Telemetrics
- Text
- Tracking Hub Command
- Transformation
- Utah Router
- Unreal Animation
- Unreal Blueprint
- Unreal Dispatcher
- Unreal Scene Loader
- Unreal Sequencer
- Unreal Text
- Vinten
- Virtual Studio
- Visibility
- Viz Camera
- Viz Clip

1.1 Alpha

Properties:

- **double** Alpha

1.2 Chroma

Properties:

- ChromaClassicContent **Classic**
  - double **U**
  - double **V**
  - double **InnerUVDiameter**
  - double **OuterUVDiameter**

- ChromaFusionContent **Fusion**
  - 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**

Sample

```
var action = GetAction("Chroma");
// sample for setting classic chroma keyer settings
action.Classic.U = 0.5;
action.Classic.V = 0.5;
action.Classic.InnerUVDiameter = 0.1;
action.Classic.OuterUVDiameter = 0.3;

// sample for setting some fusion keyer settings
action.Fusion.hueAdjust = -1140;
action.Fusion.saturationAdjust = 2.0;
```

1.3 Command

Properties:

- string **Command**

1.4 ControlObject

Refer to Control Object Classes

1.5 Director

Properties:

- string **DirectorType**
  possible values: START, STOP, CONTINUE, CONTINUE_REVERSE, PLAY_FROM, PLAY_FROM_REVERSE, FROM_TO, GO_TO, PAUSE

1.6 Group

This action has no additional public properties

1.7 Image

Properties:

- string **Image**
- double **PosX**
- double **PosY**
- double **RotX**
- double **RotY**
- double **RotZ**
- double **ScaX**
- double **ScaY**

The Image parameter can be assigned to as a GraphicHub path when the string starts with "IMAGE*", when it starts with "http" it will be assumed to be a web link (or a MediaService link), else it will be interpreted as a local file path, samples:

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";
```

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**

1.9 Key

Properties:

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

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)

1.11 MSE

Properties:

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

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**

1.13 Omo

Properties:

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

1.14 PBR

This action has no additional public properties

1.15 RenderGraph

Properties:

- string **GraphName**
- bool **DoToneMappping**
- double **ToneExposure**
- double **ToneSaturation**
- double **ToneContrast**
- bool **DoDOF**
- double **DofFocalLength**

1.16 Scene Loader

Properties:

- string **FrontUUID**
- string **MainUUID**
- string **BackUUID**
- string **GfxUUID**
- bool **FrontClear**
- bool **MainClear**
- bool **BackClear**
- bool **GfxClear**
- bool **FrontResetStage**
- bool **MainResetStage**
- bool **BackResetStage**
- bool **GfxResetStage**
- int **GfxLayerNumber** [0,...,17]

1.17 Shared Memory

Functions:

- String[] **GetKeys**()
- String[] **GetValues**()
- Void **AddKeyValue** (string key, string value)
- Void **InsertKeyValue** (int index, string key, string value)
- Bool **Remove** (string key)
- Void **RemoveAt** (int index)

1.18 Telemetrics

Properties:

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

1.19 Text

Properties:

- string **Text**

1.20 Tracking Hub Command

Properties:

- string **Command**

1.21 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**

1.22 Utah Router

Properties:

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

1.23 Unreal Animation

Properties:

- string **AnimationMode**
  possible values: LOAD, CONTINUE, PAUSE
- bool **IsLooping**

- double **PlayRate**
- double **BlendTime**
- string **SelectedAnimation**

1.24 Unreal Blueprint

Refer to Control Object Classes

1.25 Unreal Dispatcher

This action has no additional public properties

1.26 Unreal Scene Loader

This action has no additional public properties

1.27 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**

1.28 Unreal Text

Properties:

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

1.29 Vinten

This action has no additional public properties

1.30 Virtual Studio

Properties:

- int **SelectedSceneIndex**

1.31 Visibility

Properties:

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

1.32 Viz Camera

Properties:

- int **SelectedCamera**

1.33 Viz Clip

Properties:

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

2 Control Object Classes

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

2.1 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.

2.2 Control Image

Properties:

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

2.3 Control Material

Properties:

- Path (string)

2.4 Control Omo

Properties:

- Value (integer)

2.5 Control Text

Properties:

- Value (string)

2.6 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

2.6.1 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`
Double Cell	Value (double)		`table(0,5).x= 12.8`

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

2.7 Control Integer

Properties:

- Value (integer)

2.8 Control Double

Properties:

- Value (double)

2.9 Control Boolean

Properties:

- Value (boolean)

3 Profiles Classes

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

- Scripting Profile
- Scripting Channel
- Scripting Engine

3.1 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 **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*

3.2 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
- ScriptingEngine **GetEngineByName** (string name)
  - Return the first Engine found with name

3.3 Scripting Engine

- void **SendSingleCommand** (string command)

    - Sends the command to the Engine
- string **QueryEngine** (string command)
    - Queries the Engine with command

4 Scripting Classes

This section covers the following topics:

- General
- Action
- Control Object
- Parameter
- Channel
- DataMap
- Viz/Unreal Engine Communication
- SMM Handling
- GPI
- Timer
- StreamDeck
- DataMap
- File handling
- Parameter Callbacks
- Exposed Objects
    - Console
    - XmlDocument
    - FSO
- Main Script-only
    - Canvas Tabs Handling
    - Action Template Handling
    - Callbacks
- Template Script - only
    - Action/Designer handling
    - Control Object Handling
    - Template action configuration
    - Callbacks
- Parameters
    - Base parameters functionality
    - Layout
        - Panel
        - Tabs
        - Info
        - Label
    - Dialogs
        - Color
        - Directory
        - File

- Asset
- Input
  - Bool
  - Button
  - Double / Double Slider
  - Dropdown / Radio
  - Int / Int Slider
  - MultiText / Text
  - Triplet
  - Table
    - Properties
    - Methods
      - Cell handling
      - Columns handling
      - Rows handling

4.1 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 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.

4.2 Action

All actions in the current project can be accessed using the **GetAction** method and its content can be manipulated. Refer to Action Properties for more details.

- BaseAction**GetAction** (string actionName)
  - Returns the first action found with the name provided.

GetAction example

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

4.3 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.

Getting a specific ControlObject from a ControlObjectAction

```
// Get the ControlObject Action
var MatchDayAction = GetAction("MatchdayTable");
// Get Title ControlObject (ControlText) and change its value
GetControlObject(Co, "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";
```

4.4 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.

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");
```

4.5 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.

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");
```

4.6 DataMap

An internal Hash Map where data can be stored and exchanged between scripts. The Hash Map is not stored anywhere, so be aware that the Data Map will be empty when starting Viz Arc.

- void **SetData** (string key, dynamic value)
  - Sets the variable key with value
- dynamic **GetData** (string key)
  - Returns the value of key or null if the key does not exist

---

4.7 Viz/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 **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*

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");
```

4.8 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.

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

SMM example

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

4.9 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:

GetAction 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

4.10 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";

function OnInit()
{
  // Create timer with id heartBeatTimerId
  CreateTimer(heartBeatTimerId);
}

function OnButtonPressed(id)
{
  if(id == "TimerStart")
    StartTimer(heartBeatTimerId, 1000);
  else if(id == "TimerStop")
    StopTimer(heartBeatTimerId);
}

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

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

4.11 StreamDeck

Any connected StreamDeck can have its buttons customized via **arc** 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
- 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

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);
}

function OnInit()
{
    // Clean first 3 keys
    ClearStreamdeckKey(0);
    ClearStreamdeckKey(1);
    ClearStreamdeckKey(2);
}
```

4.12 DataMap

arc provides a simple 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) t*he key:value* pair into Arc's DataMap
  - [dynamic] Input *value* can be of any type

DataMap example

```
// Callback for DataMap changes
function OnDataMapValueChanged(varName)
{
        // Process results whenever "ShooterResults" is updated
        if(varName == "ShooterResults")
                UpdateResults(GetData(varName));
}

// Example of setting a DataMap variable
SetData("ShooterResults", "1 0 1 1 0");
```

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

4.13 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 inputed *data*
- dynamic **ParseJson** (string str)
  - Deserializes the input *data* and returns a json object if successful

⚠ **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);
```

4.14 Parameter 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
- **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** (string key)
  - Called whenever a StreamDeck button is pressed

    - *key* indicates the index of the pressed button
- **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

---

4.15 Exposed Objects

4.15.1 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

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");
```

4.15.2 XmlDocument

XmlDocument allows you to read xml files or strings and aggregate data using XPath.

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


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


// search for x:model
root = xmlDoc.DocumentElement;
nodeList=root.SelectNodes("//x:model");
Console.WriteLine("nodesList " + nodeList.Count);
```

4.15.3 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 of the following: Tristate.TristateUseDefault (system default), TriState.TristateTrue (Unicode), TriState.TristateFalse (ASCII)

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());
```

4.16 Main Script-Only

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

4.16.1 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
- string[] **GetActionsTabs** ( )
    - Returns a string array with all tab names

4.16.2 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

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

4.16.3 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)

4.17 Template Script – Only

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

4.17.1 Action/Designer handling

- void **ExecuteTemplate( )**

    - Execute 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 **PreviewTemplate( )**
    - Previews the owner template action or loaded template in the designer.

4.17.2 Control Object Handling

The template script allows the user to interact with the template's control objects.

- void **SetControlObject** (string objectID, dynamic value)
    - Sets the control object with id equal to *objectID's* value to *value*
    - The set object;s value will be sent on template execute/update

> **Note:** SetControlObject will only work on control objects that aren't already linked to parameters

4.17.3 Template action configuration

- bool **IsCommandHeaderVisible**
    - Indicates whether the CommandHeader should be visible on not on the template action
- bool **UpdateOnSelected**
    - When this flag is set, the script callbacks will only be triggered when the template action is selected on the action canvas (blue border)

4.17.4 Callbacks

- **OnCreated** ( )
    - Called when the template script is executed, i.e. when the template action is created and when the template opened on the designer is started
- **OnShow** ( )
    - Called when the template is shown, i.e. 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
- **OnContinue** ( )
    - Called when the template is continued
- **OnUpdate** ( )
    - Called when the template is updated

- **OnOut** ( )
  - Called when the template is taken out
- **OnPreview** ( )
  - Called when the template is previewed

4.18 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.

4.18.1 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 can be interacted with
- bool **IsVisible** [Get , Set]
  - Whether the parameter is visible or not
  - Invisible parameters are visible (displayed as grayed out) only while editing, i.e. script 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`

4.18.2 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: Tries 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 or not the 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

The label parameter doesn't have any type-specific functionalities.

4.18.3 Dialogs

Color

- string **Value** [Get , Set]
  - Gets/Sets the parameter's selected color in Hex format, e.g. #FF0A0A8C (#RRGGBBAA)

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.

Asset

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

    - Gets/Sets the selected asset's fullpath.
    - Set: The input path needs to be valid.

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

4.18.4 Input

Bool

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

Button

- void **Click** ( )
    - Trigger a click event on the button parameter

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

Dropdown / Radio

- string **Value** [Get , Set]
    - Gets/Sets the selected entry on the dropdown
- int **SelectedIndex** [Get , Set]
    - Gets/Sets the selected index of the dropdown
- 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* at the 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

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

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
- 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 **ZEnabled** [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 **ColumnCount** [Get]
    - Gets the current number of columns on the table

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 **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, i.e. 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

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)
    - Add a column of type *columnType* named *name* if specified, otherwise the default will be used
- void **AddMultipleColumn** (int count, string columnType)
    - Add *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 speficied, otherwise the default will be used
- vodi **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

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");

        // 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];
    }
}
```