

## HowTo 7 - using a Custom Extensions in GSE

A guide to create your own Custom Extension for using in a GSE project

Product:	Guiliani Streaming Editor (GSE)
Release version:	2.2
Release date:	August 31, 2018

## Disclaimer

The information in this document is subject to change without notice and does not represent a commitment on the part of TES Electronic Solutions GmbH. The software described by this document is subject to a License Agreement and may not be copied to other media. No part of this publication may be copied, reproduced or otherwise transmitted or recorded, for any purpose, without prior written permission by TES Electronic Solutions GmbH.

“Guiliani.de”, “Guiliani”, “Guiliani Streaming Editor”, “GSE” and associated logos are (registered) trademarks of TES Electronic Solutions GmbH.

Windows, Visual Studio and Visual C++ are registered trademarks of Microsoft Corporation in the United States and/or other countries.

All other trademarks are the property of their respective owners and use of them does not imply any affiliation with or endorsement by them.

## **Guiliani, guiliani.de and GSE are products of**

TES Electronic Solutions GmbH  
Hanauer Landstrasse 328-330  
60314 Frankfurt am Main  
Germany

Email: [support@guiliani.de](mailto:support@guiliani.de)  
Website: <http://www.guiliani.de>  
Forum: <https://www.guiliani.de/forum/>  
HelpDesk: <https://guiliani.on.spiceworks.com/portal>

## **Table of contents**

1.	Introduction .....	5
1.1.	Assumed knowledge .....	5
1.2.	Prerequisites .....	5
1.3.	Documentation conventions .....	5
1.4.	Shortcuts.....	6
1.5.	This document's goal .....	7
2.	Using a Custom Extension with programming the GSE.....	8
2.1.	Step 1: Load project "step by step" .....	8
2.2.	Step 2: Adding a CustomExtension in GSE.....	8
2.2.1.	Adding images to burner 3 .....	8
2.2.2.	Adding a switch.....	10
2.2.3.	Adding a Custom Extension command .....	11
2.2.3.1.	Create a Custom Extension .....	12
2.2.3.2.	Copy files from the StreamRuntime .....	13
2.2.3.3.	Re-run CMake.....	16
2.2.3.4.	Recompile GSE in your IDE.....	17
2.2.3.5.	Look for our Custom Extension command in the GSE .....	17
2.2.3.6.	Add a parameter to our Custom Extension command .....	18
2.2.3.6.1.	ReadFromStream .....	19
2.2.3.6.2.	WriteToStream .....	20
2.2.3.6.3.	Do()-method .....	20
2.2.3.6.4.	Change CustomExtensionFuncs.cpp .....	22
2.2.3.6.5.	Add our Custom Extension command into the project.....	22
2.3.	Step 3: How to continue? .....	24
2.3.1.	Sample solution .....	24
2.3.2.	Continuing HowTos .....	24
3.	Index.....	25

## **Table of figures**

Fig. 1 This “How To” result.....	7
Fig. 2 Adding image for AID_IMAGE_7 .....	9
Fig. 3 Attributes AID_IMAGE_7 .....	9
Fig. 4 Attributes AID_IMAGE_8 .....	10
Fig. 5 Switch added.....	11
Fig. 6 Choose create Custom Extension .....	12
Fig. 7 Enter name for Custom Extension .....	12
Fig. 8 Custom Extension created .....	12
Fig. 9 Select .h files from the project .....	13
Fig. 10 .h files copied to StreamRuntime.....	14
Fig. 11 Our command is available .....	17
Fig. 12 Constructor added .....	18
Fig. 13 Define the parameter.....	19
Fig. 13 ReadFromStream .....	19
Fig. 15 WriteToStream.....	20
Fig. 16 Do()-method.....	20
Fig. 17 Needed includes .....	21
Fig. 18 CustomExtensionFuncs.cpp before changing .....	22
Fig. 19 CustomExtensionFuncs.cpp after changing.....	22
Fig. 19 Attribute EnlargeBurner.....	23

## 1. Introduction

This document explains step by step how to use a Custom Extension command in a GSE project.

### 1.1. Assumed knowledge

- Basic handling of GSE
- It is recommended to read “HowTo 1 - build a project step by step” and “HowTo 5 - using of CMake and an IDE for GSE”, first.
- This document is based on the project which was built during the HowTos 1 to 6. Therefore we recommend you to read these documents, too.

### 1.2. Prerequisites

- Unpacked Guiliani SDK including GSE
- Your project “step by step” created in “HowTo 6” (or the “step by step” project inside the folder called “*HowTo 6 - sample solution*”)

### 1.3. Documentation conventions

Whenever you can use keys from your computer’s keyboard, these will be displayed in square brackets (e.g.,” To run your project press [Ctrl] + [r].”).

Menu commands or file path used in this document will be shown in *italic*.

Text that appears in the software on controls will be printed in **bold and blue**.

- ▶ Whenever the reader of this document has to do something in his project, the text will start with this triangle.
- ➡ Results will be shown using this arrow.

In this document, we use icons whenever we will warn the user or will give him additional or important information.



The speech bubble icon will show additional helpful information.



Whenever a text begins with an exclamation mark icon, it contains important information that is essential for the current chapter.



A warning sign icon signals serious issues and potential risks that require your full attention.

## 1.4. Shortcuts

In the documents, we often select a command from the window. These can be selected by the following short cuts, too:

<i>File</i> → <i>New Project...</i>	[Ctrl] + [Shift] + [n]
<i>File</i> → <i>New Dialog...</i>	[Ctrl] + [n]
<i>File</i> → <i>Save Project...</i>	[Ctrl] + [s]
<i>File</i> → <i>Run Simulation...</i>	[Ctrl] + [r]

## 1.5. This document's goal

At the end of this HowTo you will have learned

- how to create a Custom Extension
- how to use a Custom Extension command
- and more

The main dialog will look like this:

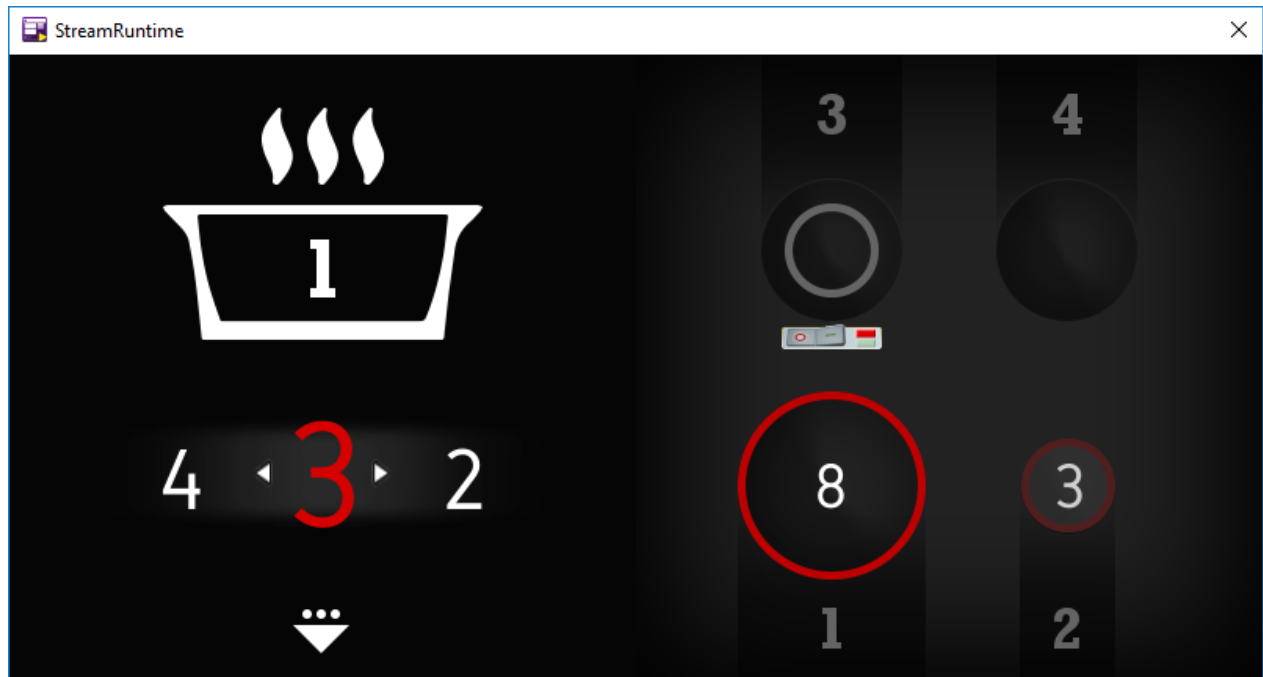


Fig. 1 This "How To" result

## 2. Using a Custom Extension with programming the GSE

### 2.1. Step 1: Load project “step by step”

- ▶ Start the GSE and load the created Step by Step project from HowTo 6 (step\_by\_step.gpr).

### 2.2. Step 2: Adding a CustomExtension in GSE

In the last HowTos (up to 6 - using CallApplication APIs in GSE) we added four UserCommands and one CallApplication API to one image (both [AID\\_IMAGE\\_5](#) and [AID\\_IMAGE\\_6](#)). This led to a lot of additional attributes, causing us to lose the overview of the attributes.

Would it not be better to put all these commands together in one command?

This is possible – using the Custom Extension command.

In this HowTo the Custom Extension command will be used to change the appearance of burner 3: it will have a small plate and when you click onto a switch, the plate will be enlarged. A second click will enable the small plate again.

#### 2.2.1. Adding images to burner 3

Before we can add the Custom Extension command, we have to add the images which will be shown when clicking onto the switch, and the switch itself.

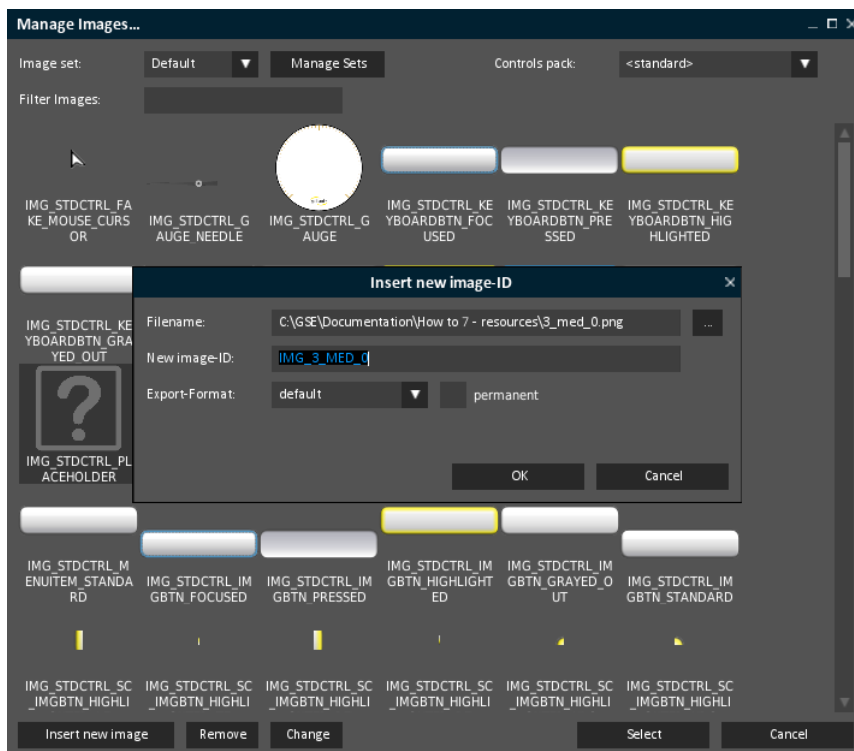
- ▶ Add two **Images** from the “**Controls**” window to our **Main** dialog.
- ▶ Set the following attributes:

Attribute	<a href="#">AID_IMAGE_7</a>	<a href="#">AID_IMAGE_8</a>
<a href="#">XPos</a>	480	495
<a href="#">YPos</a>	80	95
<a href="#">Width</a>	90	60
<a href="#">Hight</a>	90	60
<a href="#">Invisible</a>	unchecked	Checked
<a href="#">ImageID</a>	IMG_3_MED_0	IMG_3_SMALL_0



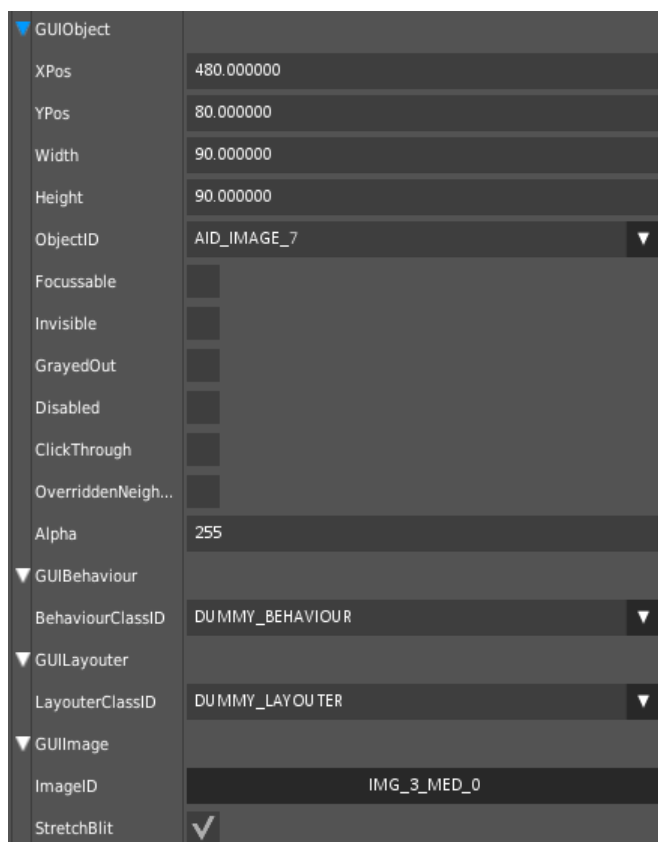
To add the images you have to click onto the button behind “[ImageID](#)”. Add the images “3\_med\_0.png” for [AID\\_IMAGE\\_7](#) and “3\_small\_0.png” for [AID\\_IMAGE\\_8](#) from the folder “*How to 7 – resources*” inside the data folder for this How To.





**Fig. 2 Adding image for AID\_IMAGE\_7**

➔ The attributes should be set like this:



**Fig. 3 Attributes AID\_IMAGE\_7**

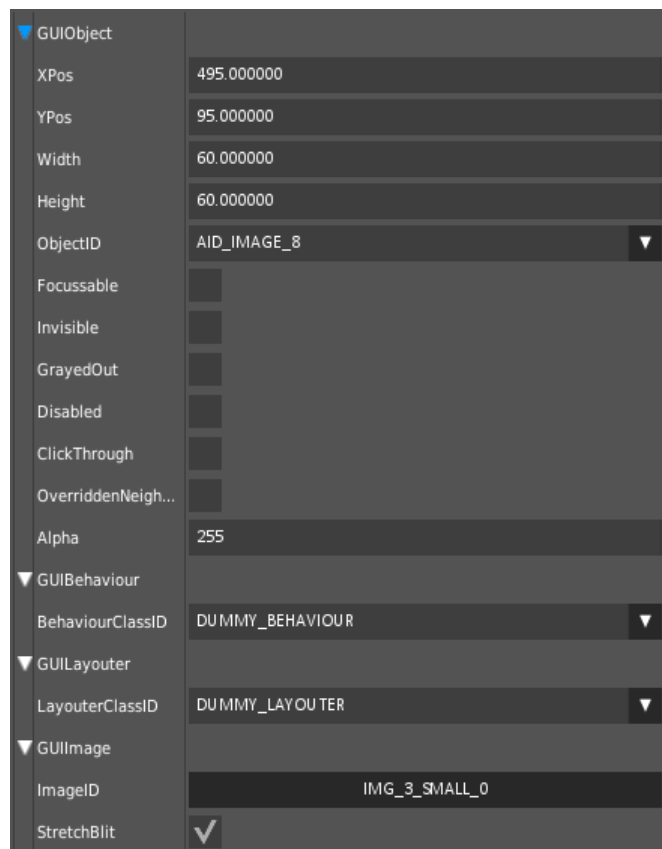


Fig. 4 Attributes AID\_IMAGE\_8

## 2.2.2. Adding a switch

The switch will have the functionality of a check box.

- ▶ Add a **Checkbox** from the “**Controls**” window to our **Main** dialog and set the following attributes:

Attribute	AID_CHECKBOX_1
<b>XPos</b>	493
<b>YPos</b>	172
<b>Width</b>	64
<b>Hight</b>	16
<b>GUIText / Width</b>	64
<b>GUIText / Height</b>	16
<b>Selected</b>	checked
<b>CheckBoxLayout</b>	ICON_FILL_OBJECT
<b>SelectedImageID...</b>	IMG_SWITCH_ON
<b>NotSelectedImageID...</b>	IMG_SWITCH_OFF



The images can be found inside the folder “*How to 7 – resources*” of your SDK’s documentation folder (“Switch\_on.png” and “Switch\_off.png”).

▶ Save your project and run it.

➔ The project should look like this:

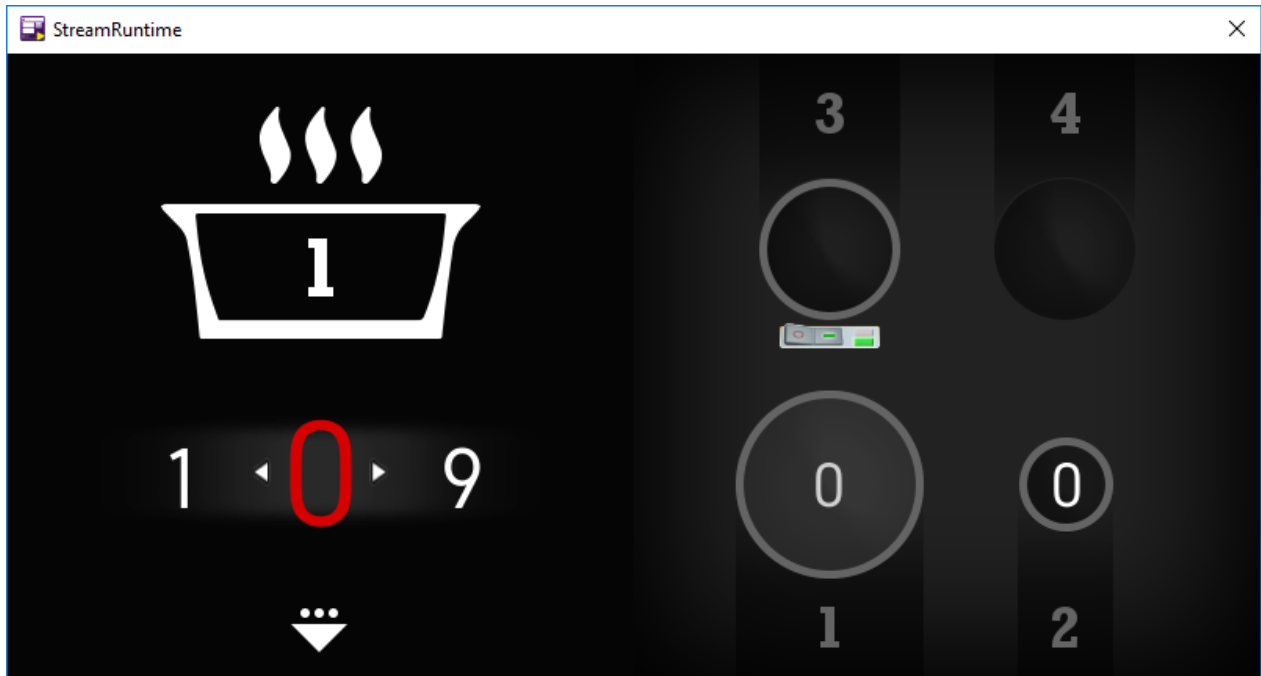


Fig. 5 Switch added

You can click onto the switch, but nothing happens.  
We have to add the Custom Extension command.

### 2.2.3. Adding a Custom Extension command

Before we add the command, just think about what the command should do:  
It should toggle the size of burner 3 – this is: changing the visibility of our images [AID\\_IMAGE\\_7](#) and [AID\\_IMAGE\\_8](#).

Now let us add the Custom Extension command.

## 2.2.3.1. Create a Custom Extension

- ▶ Choose “*Create Custom Extension...*” from the menu “*Custom Extensions*”.

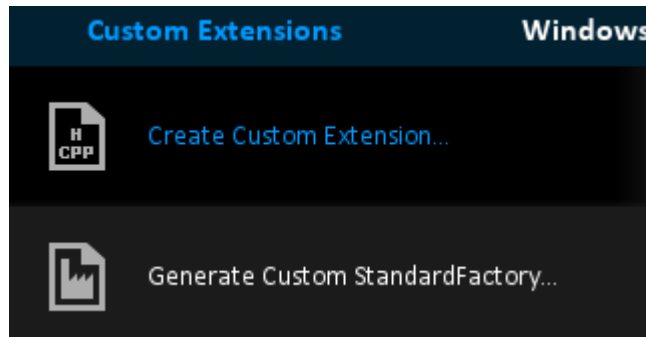


Fig. 6 Choose create Custom Extension

In the new opened window, “AddCommand” is pre-selected. So we do not have to choose a type for our Custom Extension.

- ▶ Enter “EnlargeBurner” as the “**Command class name**”.

The Command-ID will be created from the name we chose.

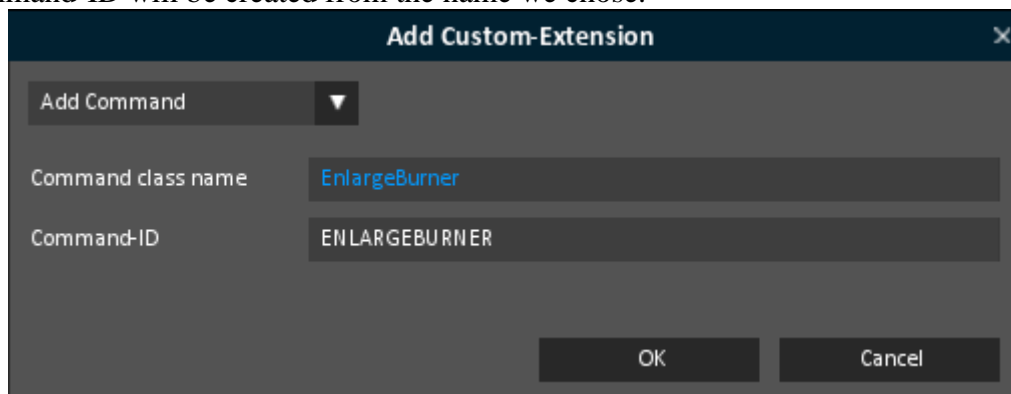


Fig. 7 Enter name for Custom Extension

- ▶ Click onto “**OK**”.
- ➔ The following message appears, telling us, that the Custom Extension has been created successfully and that you have to recompile the GSE to add the Custom Extension command to your project:

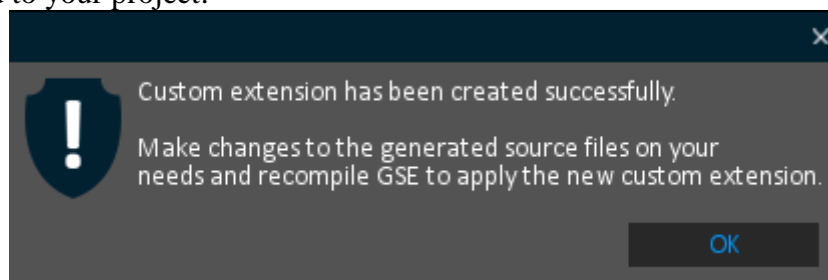


Fig. 8 Custom Extension created

- ▶ Just click onto “OK”.

The next thing we have to do is to add our command to the GSE.

If you are looking for the commands available at the moment (e.g. look at “**CommandClassID**” for the check box), you will see, that our command has not been added, yet.

- ▶ Close the GSE and your IDE.

## 2.2.3.2. Copy files from the StreamRuntime

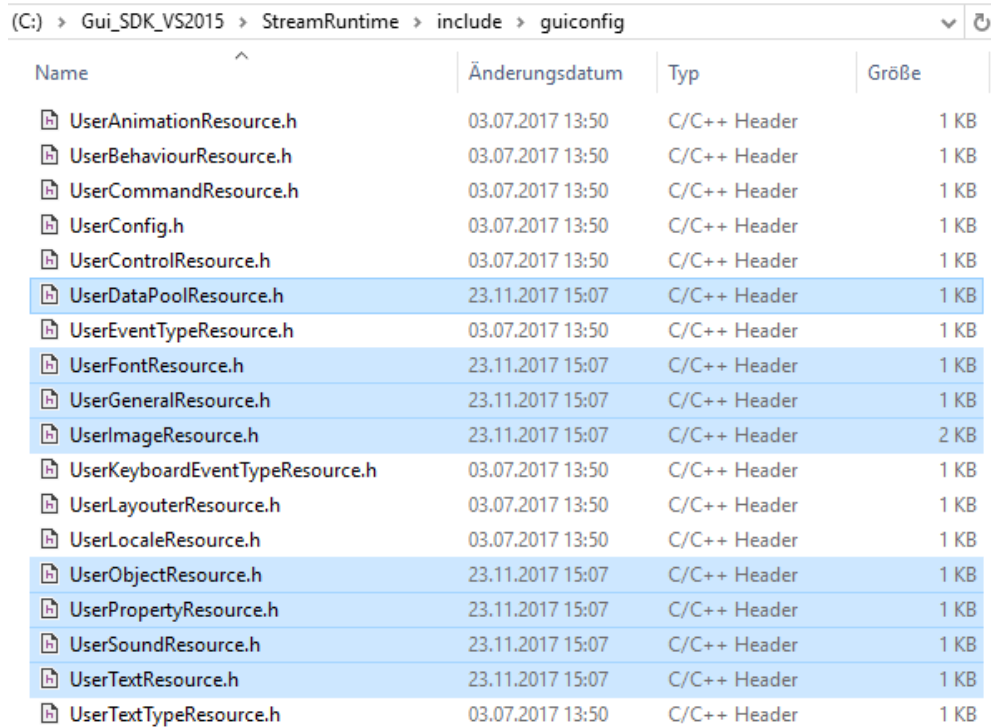
We want to change the size of burner 3 using different images. Our Custom Extension command needs to know the object IDs. Therefore we need to copy the needed Stream Runtime files from our project folder to the appropriate StreamRuntime folder.

- ▶ Open the folder “*C:\step\_by\_step\temp*” and select all “.h” files.

Name	Änderungsdatum	Typ	Größe
fonts	23.11.2017 15:07	Dateiordner	
images	23.11.2017 15:07	Dateiordner	
other_resources	23.11.2017 15:07	Dateiordner	
sounds	23.11.2017 15:07	Dateiordner	
DataPool.xml	23.11.2017 15:07	XML-Dokument	1 KB
Default.lng	23.11.2017 15:07	LNG-Datei	1 KB
fonts_Default.xml	23.11.2017 15:07	XML-Dokument	2 KB
images_Default.xml	23.11.2017 15:07	XML-Dokument	16 KB
Main.xml	23.11.2017 15:07	XML-Dokument	30 KB
other_resources_Default.xml	23.11.2017 15:07	XML-Dokument	1 KB
properties_Default.xml	23.11.2017 15:07	XML-Dokument	4 KB
Settings.xml	23.11.2017 15:07	XML-Dokument	4 KB
sounds_Default.xml	23.11.2017 15:07	XML-Dokument	1 KB
StreamRuntime.exe	23.11.2017 13:31	Anwendung	4.972 KB
StreamRuntime.log	23.11.2017 15:07	Textdokument	10 KB
StreamRuntimeConfig.xml	23.11.2017 15:07	XML-Dokument	1 KB
UserDataPoolResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserFontResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserGeneralResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserImageResource.h	23.11.2017 15:07	C/C++ Header	2 KB
UserObjectResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserPropertyResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserSoundResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserTextResource.h	23.11.2017 15:07	C/C++ Header	1 KB

Fig. 9 Select .h files from the project

- ▶ Copy the selected “.h” files to our GSE, into the folder “C:\Gui\_SDK\_VS2015\StreamRuntime\Include\GUIconfig”.
- ▶ Replace existing files.

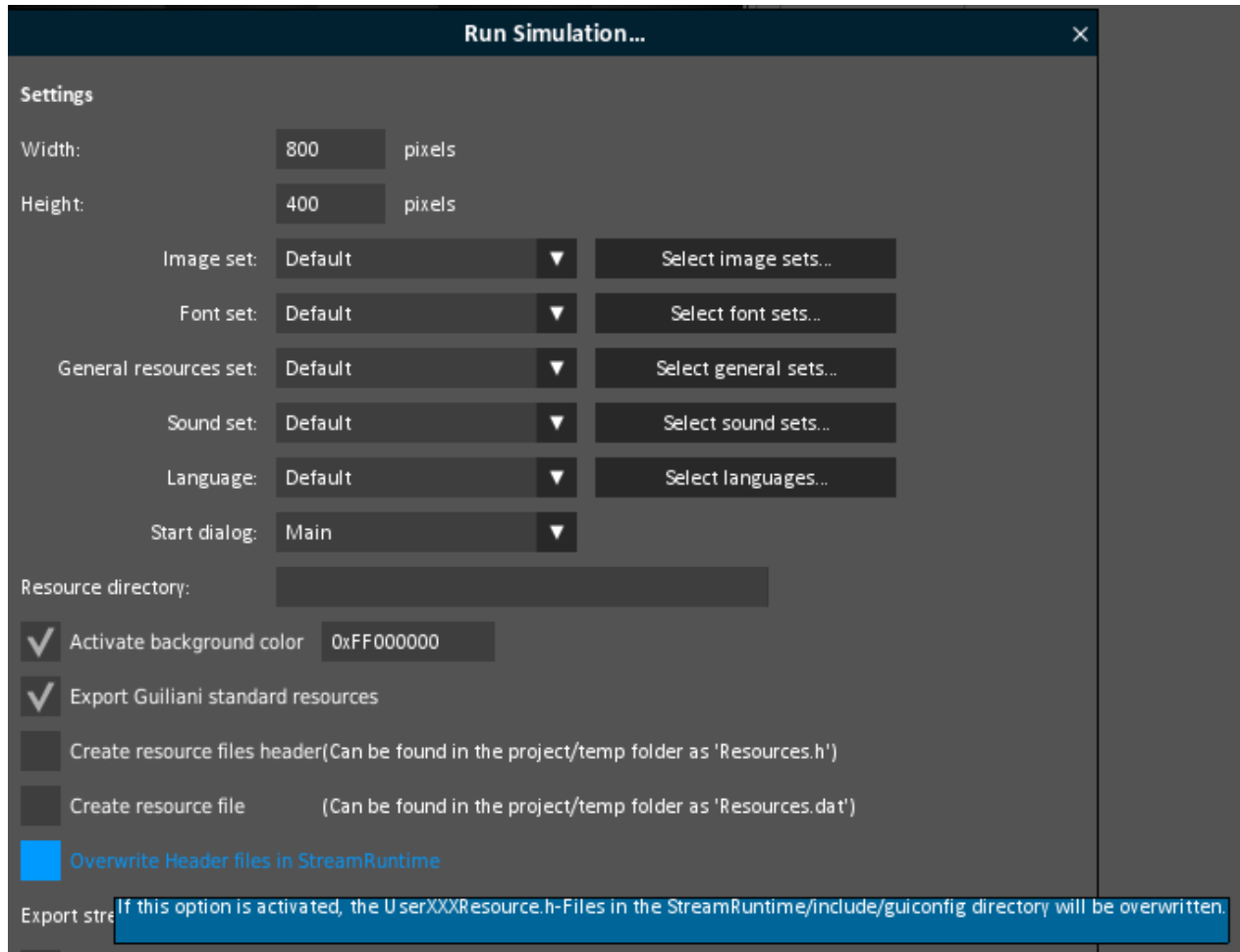


Name	Änderungsdatum	Typ	Größe
UserAnimationResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserBehaviourResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserCommandResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserConfig.h	03.07.2017 13:50	C/C++ Header	1 KB
UserControlResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserDataPoolResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserEventTypeResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserFontResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserGeneralResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserImageResource.h	23.11.2017 15:07	C/C++ Header	2 KB
UserKeyboardEventTypeResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserLayouterResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserLocaleResource.h	03.07.2017 13:50	C/C++ Header	1 KB
UserObjectResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserPropertyResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserSoundResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserTextResource.h	23.11.2017 15:07	C/C++ Header	1 KB
UserTextTypeResource.h	03.07.2017 13:50	C/C++ Header	1 KB

Fig. 10 .h files copied to StreamRuntime



If you use the StreamRuntime folder you have specified in CMAKE, you have not to copy the files. Just mark the checkbox “Overwrite Header files in StreamRuntime” when you run the simulation.



## 2.2.3.3. Re-run CMake

- ▶ Start CMake.  
If you have not used it since HowTo 5, all inputs should still be available. If not, just refer to “How to 5 - using of CMake and an IDE for GSE” on how to set up your project.
- ▶ Just click onto “**Configure**” and then onto “**Generate**”.

Your project will be re-generated. This means, the files for your Custom Extension command have been added.




You will find your files here:

“C:\Gui\_SDK\_VS2015\StreamRuntime\Include\CustomExtension”,  
“C:\Gui\_SDK\_VS2015\StreamRuntime\Source\CustomExtension”.



## 2.2.3.4. Recompile GSE in your IDE

As in the HowTos 5 and 6, we will use Microsoft's Visual Studio as an example for the IDE. If you have problems with starting MS Visual Studio, please refer to "How to 5 - using of CMake and an IDE for GSE" chapter 2.2.

 For this HowTo, we will use "Guiliani\_2.1\_SDK\_including\_GSE\_Windows-Desktop\_VS2015" and "Microsoft Visual Studio Express 2015 (Microsoft Visual Studio 14.0)".

- ▶ Open our project's build folder "Build\_GSE\_HowTo" we created in HowTo 5. Inside you will find the file "GSE.sln". Double click onto it.
- ▶ Set GSE as the StartUp Project.
- ▶ From the menu "Build" choose "Rebuild Solution" to make a new compilation of the GSE.
- ▶ This should be successful without any errors.
- ▶ Now select "Debug / Start Debugging" from the menu (or press F5) to start the GSE directly out of your development environment (Microsoft Visual Studio).

## 2.2.3.5. Look for our Custom Extension command in the GSE

- ▶ Select (click onto) "**AID\_CHECKBOX\_1**" in the "**Object Hierarchy**" window.
- ▶ Click onto the triangle in the field behind "**CommandClassID**".

You can see that our command has been included and is available for using:

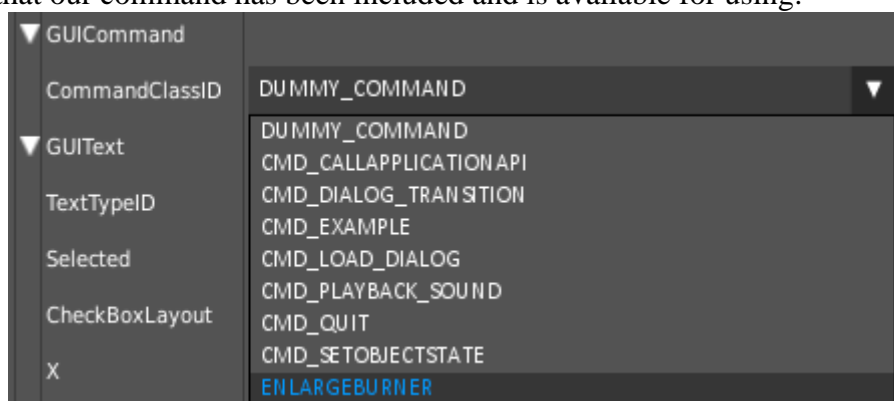


Fig. 11 Our command is available

If you select “**ENLARGEburner**” you can use our command.

But first we should think about what is our command good for:

When we click onto the switch, the burner will be enlarged or will shrink.

But should this command only be available for burner 3?

No, we should give our command a parameter containing the burner number, so we can handle other burners later.

## 2.2.3.6. Add a parameter to our Custom Extension command

- ▶ Close the GSE without saving the project.
- ▶ In your IDE open the files “*EnlargeBurner.cpp*” (“C:\Gui\_SDK\_VS2015\StreamRuntime\Source\CustomExtension”) and “*EnlargeBurner.h*” (“C:\Gui\_SDK\_VS2015\StreamRuntime\Include\CustomExtension”).



For Microsoft VS 2015/2012 users: you will find the files in the „StreamRuntime“ project, inside the folders “Header Files\Custom Extension” and “Source Files\Custom Extension”.

We have to define the call for our command in the .h file.

- ▶ Enter the following code:

```
/** Constructor.  
@param kParam Optional parameter which represents the burner number*/  
EnlargeBurner(const eC_String& kParam) : m_kParam(kParam) {};
```

```
6 class EnlargeBurner : public CGUICommand  
7 {  
8 public:  
9 EnlargeBurner();  
10 /** Constructor.  
11 @param kParam Optional parameter which represents the burner number*/  
12 EnlargeBurner(const eC_String& kParam) : m_kParam(kParam) {};  
13 ;
```

Fig. 12 Constructor added

The next step is to define our parameter as an `eC_String`.

▶ Enter the following code:

```
eC_String    m_kParam;
```

```
protected:
    /// Implements the actual command functionality.
    void Do();

    eC_String    m_kParam;
```

Fig. 13 Define the parameter

In the `.cpp` file we first have to make sure that our parameter can be entered in the GSE and will be stored.

This will be done in the functions “ReadFromStream” and “WriteToStream”.

### 2.2.3.6.1. ReadFromStream

Reading will be done using `GETINPUTSTREAM.READ_STRING`. You have to give a string as a parameter that will represent the text that will be shown in the GSE.

▶ Enter the following code:

```
m_kParam = GETINPUTSTREAM.READ_STRING("BurnerNumber");
```

```
#if defined(GUILIANI_STREAM_GUI)
void EnlargeBurner::ReadFromStream()
{
    const eC_UInt cuiVersion = ReadStreamingHeader( ENLARGEBURNER_VERSION);

    switch (cuiVersion)
    {
    case 1:
        /// *****
        /// NOTE:    Insert custom attribute read calls here.
        /// *****
        m_kParam = GETINPUTSTREAM.READ_STRING("BurnerNumber");
        break;
    default:
        break;
    }

    /// base class of CGUICommand is read at the end due to additional commands
    CGUICommand::ReadFromStream();
}
#endif
```

Fig. 14 ReadFromStream

## 2.2.3.6.2. WriteToStream

Writing will be done using `GETOUTPUTSTREAM.WriteString`. You have to give the parameter itself and a string with the parameter's name.

▶ Enter the following code:

```
GETOUTPUTSTREAM.WriteString(m_kParam, "BurnerNumber");
```

```
#if defined(GUILIANI_WRITE_GUI)
void EnlargeBurner::WriteToStream(const eC_Bool bWriteClassID)
{
    WriteStreamingHeader( bWriteClassID, XMLTAG_COMMANDCLASSID, ENLARGEBURNER, ENLARGEBURNER_VERSION);

    // *****
    // NOTE:   Insert custom attribute write calls here.
    // *****
    GETOUTPUTSTREAM.WriteString(m_kParam, "BurnerNumber");

    // base class of CGUICommand is read at the end due to additional commands
    CGUICommand::WriteToStream();

    WriteStreamingFooter( bWriteClassID );
}
#endif
```

Fig. 15 WriteToStream

## 2.2.3.6.3. Do()-method

The code that will handle what our command will do has to be entered in the `Do()`-method of our `EnlargeBurner` command.

Here, you will see what has to be implemented:

```
19 void EnlargeBurner::Do()
20 {
21 #if defined(STREAMRUNTIME_APPLICATION)
22 // *****
23 // NOTE:   Implement your specific code here (between
24 //         the defined 'STREAMRUNTIME_APPLICATION')
25 // *****
26 CGUIImage* pkImageBig = static_cast<CGUIImage*>(GETGUI, GETGUI.GetAndCheckObjectByID(AID_IMAGE_7, "BurnerImage"));
27 CGUIImage* pkImageSmall = static_cast<CGUIImage*>(GETGUI, GETGUI.GetAndCheckObjectByID(AID_IMAGE_8, "BurnerImage"));
28
29 if (m_kParam == 3)
30 {
31     if ((pkImageBig) && (pkImageSmall))
32     {
33         pkImageBig->SetInvisible(!pkImageBig->IsInvisible());
34         pkImageSmall->SetInvisible(!pkImageSmall->IsInvisible());
35     }
36 }
37 #endif
38 }
```

Fig. 16 Do()-method

Let us go through the code line by line:

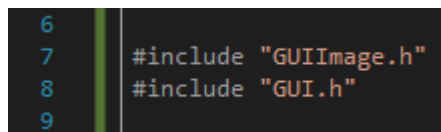
- 26: We need access to the image, which represent the large plate from burner 3.  
The last string is the name of the object that will be shown in a message in case an exception will be thrown.
- 27: The same as in line 26, but this time for our smaller plate.
- 29: We will implement what has to be done, if burner 3 sent the command.
- 31: Only work with the two images if we are sure, that the pointers show to them.
- 33: Toggle the state of invisibility for the big plate.
- 34: Toggle the state of invisibility for the small plate.

▶ Enter the following code:

```
CGUIImage* pkImageBig = static_cast<CGUIImage*>(GETGUI, GETGUI.GetAndCheckObjectByID
(AID_IMAGE_7, "BurnerImage"));
CGUIImage* pkImageSmall = static_cast<CGUIImage*>(GETGUI, GETGUI.GetAndCheckObjectByID
(AID_IMAGE_8, "BurnerImage"));

if (m_kParam == 3)
{
    if ((pkImageBig) && (pkImageSmall))
    {
        pkImageBig->SetInvisible(!pkImageBig->IsInvisible());
        pkImageSmall->SetInvisible(!pkImageSmall->IsInvisible());
    }
}
```

We have to add two defines so the compiler knows how to handle “CGUIImage” and “GETGUI”.



```
6
7 #include "GUIImage.h"
8 #include "GUI.h"
9
```

Fig. 17 Needed includes

▶ Enter the following code:

```
#include "GUIImage.h"
#include "GUI.h"
```

One thing is left: we have to add the parameter to the attributes list in the GSE (our command is there, as we have seen before).

## 2.2.3.6.4. Change CustomExtensionFuncs.cpp

In this file our Custom Extension command “EnlargeBurner” is included already.

```
51  /*<CUSTOM_COMMANDS_BLK_BEGIN*/  
52      rkCommands.push_back(CommandDescriptor(ENLARGEBURNER, "ENLARGEBURNER", new EnlargeBurner()));  
53  /*<CUSTOM_COMMANDS_BLK_END*/
```

Fig. 18 CustomExtensionFuncs.cpp before changing

But we have to add the description for the attribute:

```
51  /*<CUSTOM_COMMANDS_BLK_BEGIN*/  
52      rkCommands.push_back(CommandDescriptor(ENLARGEBURNER, "ENLARGEBURNER", new EnlargeBurner("Enter burner number")));  
53  /*<CUSTOM_COMMANDS_BLK_END*/
```

Fig. 19 CustomExtensionFuncs.cpp after changing

▶ Change the following code from:

```
rkCommands.push_back(CommandDescriptor(ENLARGEBURNER, "ENLARGEBURNER", new  
EnlargeBurner()));
```

▶ to:

```
rkCommands.push_back(CommandDescriptor(ENLARGEBURNER, "ENLARGEBURNER", new  
EnlargeBurner("Enter burner number")));
```

## 2.2.3.6.5. Add our Custom Extension command into the project

- ▶ Build the GSE.
- ▶ Now start the GSE directly out of your development environment.
- ▶ Load the “step\_by\_step” project.
- ▶ Select (click onto) “**AID\_CHECKBOX\_1**” in the “**Object Hierarchy**” window.
- ▶ Click onto the triangle in the field behind “**CommandClassID**” and choose “**ENLARGEBURNER**”.

You can see, we have our own command attribute called “**EnlargeBurner**”:

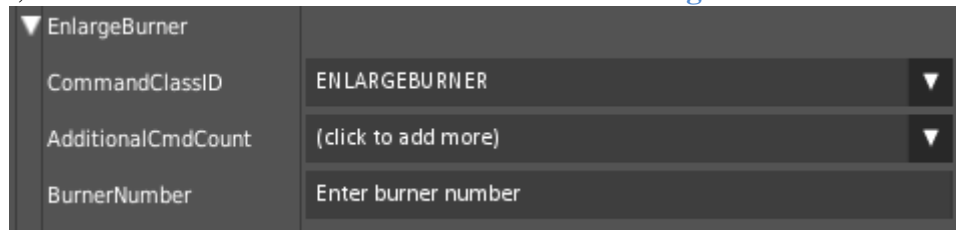


Fig. 20 Attribute EnlargeBurner

And our command has the attribute “**BurnerNumber**” (as we entered it in the code).

- ▶ Enter “**3**” as “**BurnerNumber**”.
- ▶ Save your project and run it.

Now the burner changes its size when clicking onto the switch.



Do not forget to copy the .h files and recompile the GSE if you want to use the stand alone StreamRuntime.

That was it! Now you have created your own Custom Extension successfully.

## 2.3. Step 3: How to continue?

### 2.3.1. Sample solution

Now it should be clear how to create and use your own Custom Extension command. If you encountered problems or wish to have the solution without creating the project on your own, we have added the sample solution into the folder called “*How to 7 - sample solution*” inside the documentation folder. Here you will find the GSE Project (step\_by\_step.gpr).

For this HowTo there are no additional resources.



For Windows user inside the folder “temp” there is an executable StreamRuntime.exe.

### 2.3.2. Continuing HowTos

You will find an overview of continuing HowTos in the document “How to 0 - an overview of building GSE projects”.

Don't forget to visit our homepage [www.guiliani.de](http://www.guiliani.de) to get more information, demos, help, videos and the latest news about guiliani and GSE.



## 3. Index

---

### A

Add a parameter to our Custom Extension command	21
Add our Custom Extension command into the project	27
Adding a Custom Extension command	8, 12
Adding a switch	11
Adding images to burner 3	8
Assumed knowledge	5

---

### C

Change CustomExtensionFuncs.cpp	27
CMake	
Re-run	19
Continuing How Tos	29
Copy files from the StreamRuntime	15
Create a Custom Extension	13
Custom Extension	
Create	13
Using	8
Custom Extension command	
Add a parameter	21
Add to the project	27
Adding	8, 12
Custom Extension command in the GSE	20
CustomExtensionFuncs.cpp	
Change	27

---

### D

Do()-method	24
Documentation conventions	5

---

### E

EnlargeBurner.cpp	
Do()-method	24
ReadFromStream	23
WriteToStream	24

---

### G

GSE	
Custom Extension command	20
guiconfig	17

---

### H

How to continue?	29
------------------	----

---

### I

IDE	
Recompile GSE	20
Images	
Adding	8

---

### L

Load project	8
--------------	---

---

### P

Prerequisites	5
Project	
Load	8

---

### R

ReadFromStream	23
Recompile GSE in your IDE	20
Re-run CMake	19

---

### S

Sample solution	29
Short cuts	6
StreamRuntime	
Copy files	15
Switch	
Adding	11

---

### T

temp folder	15
-------------	----

---

### U

Using a Custom Extension	8
--------------------------	---

