|  |
| --- |
| **DesktopX Scripting Tutorials** |
| Script Components |
|  |
| This series of tutorial will teach you everything there is to know about using Script Components with DesktopX: what they are, how to create and consume them, how to listen and fire events and more… |
| Author  **Julien TEMPLIER (Littleboy)**  Editor  **TBA**  Reviewers  **TBA** |
|  |
| **Current version:** |
|  |

1. Introduction to Script Components

Traditionally, scripting DesktopX widgets is done by writing your scripts directly into the editor. If you want to rationalize things a bit, you can use a controller object and have only one script (preferably external) to control the whole widget.

Still, there are several downsides with this approach:

* **Almost no code reuse**: you need to cut & paste portions of scripts every time you create a new widget
* **No support for COM events**: You can't listen to events from COM objects (especially useful if you want to do, say MSN Messenger-related widgets)
* **No debug support**: You can’t easily debug your scripts: the DesktopX editor will only give you the error info and location) and if you are using external scripts, it’s even worse!

Enter Script Components!

Script Components provide an easy way to create COM components in VBScript or JScript. This means you can write clean, self contained code and reuse it in each of your widgets. No more cut&paste. And you get runtime debug support!

This first tutorial will show you how to setup your machine for creating and debugging script components and the structure of a script component.

# Setup

By default, script debugging is turned off>

# Our first Script Component

A script component is basically an XML file containing some information about the type of component, a list of properties, methods and events, and the code itself.

Here is the basic structure of our script component:

* A <package> element: encloses one or more <component> elements. You can have several components in the same file
* A <component> element: encloses the component definition with its properties, methods and events declarations and the code itself
* A <registration> element: it will be used to register your script as a COM component. This element might not be necessary depending of the way you want to consume your components (see Tutorial 2)
* A <public> element: encloses the definitions for properties, methods and events
* A <script> element: contains the logic for your components, ie the code for each method and property. You can either use VBScript or JScript. You can have several <script> elements, each using a different script language.

<?XML version="1.0"?>

<package>

<?component error="true" debug="true"?>

<comment>

…LICENSE BLOCK…

</comment>

<component>

<comment>

Script Component tutorial for DesktopX

</comment>

<registration description="My first Script Component"

progid=" MyScriptComponent.Tutorial1"

version="1"

classid="{7D211F9F-E932-4091-922A-B2F0F0478BDA}"/>

<public>

<method name="Hello" internalName=”getData”>

<property name="value" internalname=”m\_value” />

</public>

<script language="VBScript">

<![CDATA[

Dim m\_value

m\_value = “Hello World”

Function DoSomething()

DoSomething = value

End Function

]]>

</script>

</component>

</package>

You can find this example file in the Source/Tutorial 1 folder. Script components files have the .wsc extension.

## The registration section

The <registration> section contains all the information needed to register the script. As said before, it is optional and you can skip it altogether. As we’re going to make use of it in later tutorial, let’s go over the different parts.

*description*: a text description of the component

*progid*: This is the name used to reference the component when creating a new instance (see Using your Script Components)

*version*: a numeric version number (used when requesting a specific version of a component). Use only numbers (no decimal points).

*classid*: a GUID generated by a class ID generation program (such as guidgen.exe)

As we will see in a later tutorial, you can also have a script section with pre and post-registration scripts.

## The public section

The <public> section contains the list of exposed properties, methods and events

### Methods

The <method> element declares a new method. The *name* value is the name of the method. You can have an optional *internalName* value if you want to implement the method under a different name in the script. You can also declare *parameter* values for the method (although this is not required and the component will work fine without it).

### Properties

The <property> element declares a new property. Like the <method> element, it has *name* and *internalName* values. You can also have read-only or write-only properties by adding a <get> or <put> element (each with an optional *internalName*). By default, the method used to read or write the property will be the name of the property plus the get\_ or put\_ prefix.

1. Using your Script Components

In the first tutorial, we learned how to setup our machine to develop and debug Script Component and we wrote our first script. It’s now time to use it in a widget.

We’ll see the different ways to load our components, each with their advantages and disadvantages. And we’ll start adding a little bit more code to our component.