

## Creating a Desklet with AveScripter

*A tutorial (a desklet is what you make it)*

```
function highlight(mouseIsOn)

var layer = this.GetActiveLayer()

layer.lockUpdates();
if (mouseIsOn)
{
    layer
    layer
}
else
{
    layer
}
layer

<!-- A dummy url i
http://avedesk.ph
<xml src="http://
</xml>
RSS
.d="0" w
.d="1" w
fail
</version>1</version>
</skininfo>
<skindata>
<desklet>
<alpha>255</alpha>
```

### New Desklets

- Weather for AveScripter
- Calculator for AveScripter
- iTunesRemote for AveScripter
- FTP Desklet Update
- iTunes Lyrics
- Clock for AveScripter

Author: Andreas Verhoeven  
Grammar, proofreading: nightcrawler1089

## Contents

Introduction.....	page 3
Step 1: Creating the fundament for the desklet.....	page 4
Step 2: Using the RSS feed.....	page 10
Step 3: Scripts for interactivity.....	page 14
Step 4: Flipping Out! .....	page 17
Step 5: User Interface Controls.....	page 19
Step 6: Parameters and Settings.....	page 23
Step 7: Polishing our widget; the details.....	page 26

## Introduction

Creating desklets for AveDesk was always a pretty steep step. One had to have a C-compiler and needed to be at home in Windows Programming and C++.

**AveScripter** is here to change all of that. This scriptable desklet makes creating **desklets/widgets for AveDesk** a piece of cake.

This tutorial will explain how desklets can be created with AveScripter by creating a brand new “RSS New Desklets” – widget from scratch. Step-by-step, the desklet will be updated with new features. Each new feature will focus on a specific part of AveScripter.

## Our Widget

The widget we want to create should get a list of all the latest desklets added to <http://avedesk.philc.ca/> and display them in a nice and fancy way.

Here’s a mock up of our soon-to-be widget:



## Stuff you will need

- A good text-editor, like EditPlus2 [ <http://www.editplus.com> ]
- A graphics editor or pre-made graphics. In this tutorial, pre-made graphics are supplied.
- AveScripter and AveDesk 1.3, of course.

## Let’s Start

The first part of this series will focus on setting up the environment and the needed files for our “New Desklets” widget.

## Step 1: Creating the fundament for the desklet

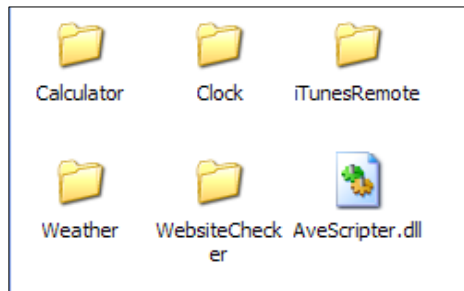
### Overview

In this step the basis for our “New Desklets” widget will be laid out. Each aspect will be discussed to get insights into the working of AveScripter.

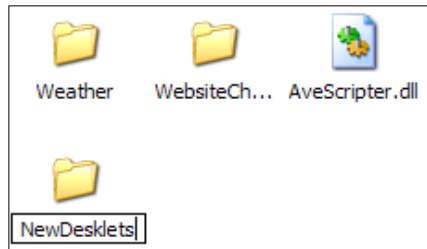
All finished files for this step can be found under *Final Files/Step1*.

### Setting up a directory

Each desklet made for AveScripter needs to be placed inside its own directory in the AveScripter folder (*AveDesk/Desklets/AveScripter*). If you open the *AveScripter* folder, you will see the folders for the desklets that came by default with AveScripter.



In the image on the left side, the *Calculator*, *Clock*, *iTunesRemote*, *Weather* and *WebsiteChecker* widgets for AveScripter are installed. Each widget, as said before, has its own directory.



A first good would be to create a directory for our widget, named “***NewDesklets***”. All files used by the *NewDesklets* widget will be placed inside this folder.

### Images

A desklet is defined as “a graphically mini-application that does something”. So, we need graphics! For this tutorial, all images needed are already created to save time. You can find the images we need in the *Images/Step1* folder.

For this step of the tutorial, there are two images: *background.png*, which makes up the background of our desklet and *rssknob.png*, which is a little RSS image.

We copy those two images from *Images/Step1* into the desklets *NewDesklets* folder.

Each image used by a desklet should be placed inside the particular desklets directory (*NewDesklets* in our case).

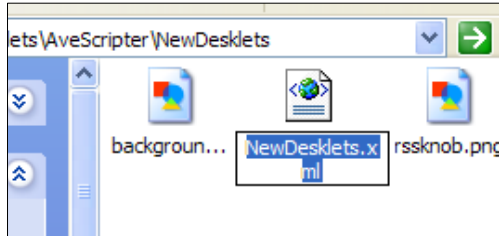
### XML

We now have a folder that holds the images used by our desklet. So, how does AveScripter knows how to create a desklet from that? The answer is simply, it doesn't. It's our task to glue

the images together into a desklet by describing how the images are composed together. This is done by creating a specially crafted **XML-file**.

(If you are not familiar with XML, you can learn about it here:

<http://www.w3schools.com/xml/default.asp> .)



So, the first thing we now need to do is to create an XML-file. We call this file *NewDesklets.xml* and place it inside our *NewDesklets* folder. Since this file is an XML-file, it needs to conform to the XML standards. One of the requirements is that every XML-document should have an `<?xml...>` node and one root-node. Of course, this is also true

for AveScripter XML-files.

So, the contents of this *NewDesklets.xml* file should be the following, initially:

```
<?xml version="1.0"?>
<!--
  Copyrights The AveScripter Tutorial, 2005.
-->
<root>
</root>
```

*Listing 1.*

Because we are proud of our work, we have also added a copyright notice under the `<?xml...>`-opening element. Here we have used XML-comments (everything between `<!--` and `-->` is seen as a comment) to add extra ‘meta’-information to the document. Those comments will be ignored by AveScripter completely, so you can add anything you want in the document when using comments.

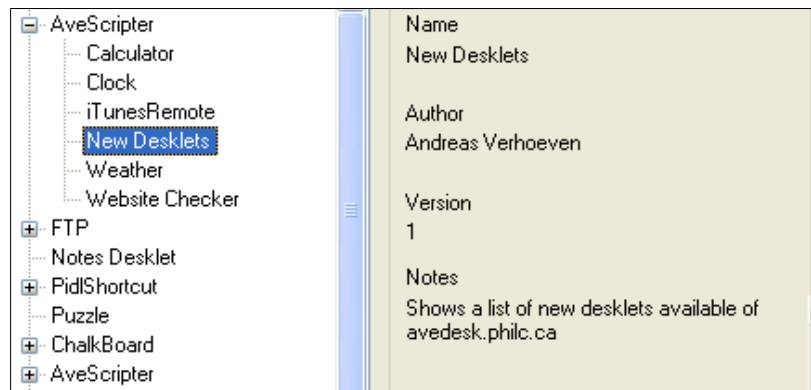
Next, to let AveDesk know what the name of our desklet is, we need to include information about its name, its author, etc... This is done by adding a *SkinInfo*-node under the root-node (the red text denotes what has been added):

```
<?xml version="1.0"?>
<!--
  Copyrights The AveScripter Tutorial, 2005.
-->
<root>
  <skininfo>
    <author>Andreas Verhoeven</author>
    <name>New Desklets</name>
    <notes>Shows a list of new desklets available of
avedesk.philc.ca</notes>
    <version>1</version>
  </skininfo>
</root>
```

All elements under the *skininfo*-node are pretty self-describing.

Once you have made those changes, you can open the *Add Desklet-dialog* of AveDesk, and, under AveScripter, there should be an item that says “*New Desklets*”.

*Isn't that cool?!*



## Adding the images

Of course, you tried to add our brand new desklet to your desktop – if you didn't, give it a try – but, only a question mark icon showed up. The reason is that we still haven't told AveScripter what images to use in the desklet. So, we now will add those images to the desklet.

Information about the desklets data goes under the *skindata*-node, which is of course also placed under the *root*-node.

```
<?xml version="1.0"?>
<!--
  Copyrights The AveScripter Tutorial, 2005.
-->
<root>
  <skininfo>
    <author>Andreas Verhoeven</author>
    <name>New Desklets</name>
    <notes>Shows a list of new desklets available of
avedesk.philc.ca</notes>
    <version>1</version>
  </skininfo>

  <skindata>

    <desklet>
      <alpha>255</alpha>
      <resources></resources>
    </desklet>

  </skindata>
</root>
```

As you may have noticed, under the *skindata*-node, there's a new *desklet*-node. This node will hold global information about the working of the desklet.

For now, we have defined the *alpha* level and the *resources* elements under the *desklet*-node.

The *alpha-level* defines the overall transparency of the desklet. 0 would mean totally see-thru and 255 means totally visible.

The *resources-element* names the folder that holds the image resources for our desklet, relative to the desklets folder. Because we place all our images in the root of our *NewDesklets*

folder, we leave this empty. But, if for example, you want to be tidy and clean and place all images under *NewDesklets/Images*, we should have set the *resources* to “Images”.

## Layers

In AveDesk and AveScripter, every graphically element (image, text) is a layer. If you are familiar with Adobe’s PhotoShop you will feel at home with this approach.

A layer can be seen as an alpha-blended canvas upon which can be drawn. By adding multiple of those canvasses (layers) on top of each other, a widget is composed of multiple images. The advantage of this approach is that, when you decide that a certain layer should be moved a couple of pixels, you don’t need to edit the whole image, but rather just move the layer. Also, layers have individual properties like an *alpha-level-value*.

So, to use our 2 images to make up a widget, we need to define two layer: one for the background and one for the little rssknob.

But – yes another one, we also need to tell the dimensions of our widget to AveScripter. This is done by defining a side and setting the width and height for that side:

```
<?xml version="1.0"?>
<!--
  Copyrights The AveScripter Tutorial, 2005.
-->
<root>
  <skininfo>
    <author>Andreas Verhoeven</author>
    <name>New Desklets</name>
    <notes>Shows a list of new desklets available of
avedesk.philc.ca</notes>
    <version>1</version>
  </skininfo>

  <skindata>

    <desklet>
      <alpha>255</alpha>
      <resources></resources>
    </desklet>

    <sides>
      <side id="0" width="300" height="220" />
    </sides>

  </skindata>
</root>
```

As you can see, the side-element is under the *sides*-node. AveScripter can actually use multiple sides for one widget. This, however will be handled in Step 4.

The side-node contains three properties: *id*, *width* and *height*. Width and height are the dimensions of our widget. Id=”0” just tells AveScripter that that particular side is the first side (we only have one!).

## Adding the layers

Finally, we are ready to add the layers to make the desklet look like something other than a question mark.

The layers are defined under a *layers*-node that is placed under the *skindata*-node. For each layer, we add a *layer*-element to our *NewDesklets.xml*:

```
<?xml version="1.0"?>
<!--
  Copyrights The AveScripter Tutorial, 2005.
-->
<root>
  <skininfo>
    <author>Andreas Verhoeven</author>
    <name>New Desklets</name>
    <notes>Shows a list of new desklets available of
avedesk.philc.ca</notes>
    <version>1</version>
  </skininfo>

  <skindata>
    <desklet>
      <alpha>255</alpha>
      <resources></resources>
    </desklet>
    <sides>
      <side id="0" width="300" height="220" />
    </sides>

    <layers>
      <layer src="background.png" x="0" y="0" />
      <layer src="rssknob.png" x="210" y="30" />
    </layers>

  </skindata>
</root>
```

As said, for each layer we have added a *layer*-element under *layers*. Each *layer*-element has three properties here: *src*, *x* and *y*.

The *src*-property, which stands for source, tells AveScripter which image to use for the particular layer. For our widget we have two layers, one using background.png and one using rssknob.png.

The *x*-property tells at what point (pixel) the layer (thus, the image) should drawn on the widget in the horizontal direction.

This is the same for the *y*-property, but in the vertical direction.



### **Running the desklet**

At last our desklet is ready to be added. AveScripter now knows enough information to draw our widget with the images we want, so let's add it to the desktop.

Our desklet behaves like every other desklet: you can move it around, change its z-ordering (style), duplicate and close it.

**The only problem is....** It doesn't do anything yet! In the next step of this tutorial, we will modify the desklet in such a way that it will actually show the new desklets on <http://avedesk.philc.ca>.



### **Conclusion**

We saw how to set up a widgets XML-file and directory for AveScripter. While this may have been a long sit, the next step will be much shorter and easier.

## Step 2: Using the RSS feed

### Overview

In this step we will enhance our desklet to actually show the New Desklets from <http://avedesk.philc.ca>'s RSS-feed. The concepts discussed in this step will be *XML-sources* and *Text-layers*.

All finished files for this step can be found under *Final Files/Step2*.

### Text layers

In the previous step we saw how to create layers, particularly layers that hold images. There's another type of layer also: *text-layers*.

A *text-layer* is simply a layer that shows text instead of an image.

Let's define a text-layer in our *NewDesklets.xml*:

```
...
    <layers>
        <layer src="background.png" x="0" y="0" />
        <layer src="rssknob.png" x="210" y="30" />
        <layer text="item one" x="20" y="260" height="15"
width="100" />
    </layers>
...
```

Besides the *x-* and *y-properties*, this *layer-element* also has *text-*, *height-* and *width-properties*.

The *text-property* defines the text-string the layer should display.

The *height-property* defines the height the text-string should maximally use.

The *width-property* defines the width the text-string should maximally use.



When you reload our widget now – duplicating is an easy way –, you will see that it now displays the text we assigned it. Only, it is very small. This is because the default text properties are used.

If we want the text to look different, we'll need to define some extra properties in the layer's description:

```
...
<layer text="item one" x="20" y="60" height="20" width="260"
fontname="Lucida Sans Unicode" fontsize="14" fontcolor="clWhite"
fontstyle="B" fontalign="Left+Top" />
...
```

As you see, several *font-properties* has been added, with most of them being very clear in what they do. Only some have special values.

The *fontcolor-property* has been defined with a name, *clWhite*. Every property that is a colour can have a named-value (a list is available in AveScripter's documentation), or HTML-colour codes, such as #FFFFFF.

The *fontstyle-property* has been set to bold by only using the character **B**, but we could also have written out Bold entirely (the capital is important, though).

### Adding more text-layers

We now know how to create a text-layer, so the next step is to add 5 extra text-layers; each separated 20 pixels from each other. This way, our widget will be able to display the latest uploaded desklets on the site (we provided spaces for six).



```
...
    <layers>
        <layer src="background.png" x="0" y="0" />
        <layer src="rssknob.png" x="210" y="30" />

        <layer text="item one" x="20" y="60" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="item two" x="20" y="80" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="item three" x="20" y="100" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="item four" x="20" y="120" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="item five" x="20" y="140" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="item six" x="20" y="160" height="20"
width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />

    </layers>
...
```

## The RSS feed

The information about the new desklets on <http://avedesk.philc.ca> is provided by an RSS feed, which has the URL <http://avedesk.philc.ca/modules/PDdownloads/latestrss.php?cid=1>. You can find more information about RSS here [http://www.w3schools.com/rss/rss\\_intro.asp](http://www.w3schools.com/rss/rss_intro.asp). Since RSS is just XML, we are going to use AveScripter's built-in XML feature to load the XML and display the actual items from the feed in the layers we have just added.

AveScripter's XML features allows us to easily load the data from the RSS feed. Loading, refreshing and caching of the XML data is done by AveScripter, so we don't have to write code to do that for us.

XML sources are defined under the *skindata*-node, in the **xmles**-node. For each source we need to add an **xml**-element under this node:

```
...
  <skindata>
    <desklet>
      <alpha>255</alpha>
      <resources></resources>
    </desklet>

    <xmles>
      <xml
src="http://avedesk.philc.ca/modules/PDdownloads/latestrss.php?cid=1"
interval="60" usewintmp="yes" />
      </xml>
    </xmles>
  </skindata>
...
```

Each *xml*-element has a *src*-property. This property is the URL that describes the place our XML-document is located at.

The *interval*-property defines after how many *minutes* the data should be refreshed. We used 60 here, to refresh the data every hour.

The *usewintmp*-property is set to *yes* (yes is the same as true or 1) to let AveScripter cache the data in Windows' Temp-folder. If there is no internet connection, the latest cached version of the document will be used.

## Linking XML and text-layers

We now have AveScripter loading the RSS feed for use every hour, but still nothing is displayed. To do this, we have to let each *text-layer* know which part of the XML data it should use. This is actually done very easily and without writing special code.

AveScripter supports a primitive form of data-binding. That is, we can link the text of a certain layer to a value in one of the XML-sources. Using the string `"!XMLTXT:0[expression]"`, the text for the layer will actually be taken from the first XML source. The value to take from the XML source is determined by what is entered for the *expression* part. This expression is a standard *XPath*-expression. More on XPath can be found here <http://www.w3schools.com/xpath/default.asp>.

The `!XMLTXT:` expression between the RSS feed and the layer is part of a larger set of

expressions, which are called “prefix-tags”.

The AveScripter documentation has a list of all possibilities and some others will be handled in the following steps of this tutorial as well. Those *prefix-tags* perform the binding between the XML feeds and the layers data.

Using the string “!XMLTXT:0[rss//channel//item[1]//title]”, we get the title of the first item node from our RSS feed. The second item would be “!XMLTXT:0[rss//channel//item[2]//title]”, and so on.

By replacing our dummy “item one”, “item two”, etc... texts of our *text-layers*, we link the data from the RSS feed to the output of the *text-layers*:

```
...
    <layers>
        <layer src="background.png" x="0" y="0" />
        <layer src="rssknob.png" x="210" y="30" />

        <layer text="!XMLTXT:0[rss//channel//item[1]//title]"
x="20" y="60" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="!XMLTXT:0[rss//channel//item[2]//title]"
x="20" y="80" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="!XMLTXT:0[rss//channel//item[3]//title]"
x="20" y="100" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="!XMLTXT:0[rss//channel//item[4]//title]"
x="20" y="120" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="!XMLTXT:0[rss//channel//item[5]//title]"
x="20" y="140" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />

        <layer text="!XMLTXT:0[rss//channel//item[6]//title]"
x="20" y="160" height="20" width="260" fontname="Lucida Sans Unicode"
fontsize="14" fontcolor="clWhite" fontstyle="B" fontalign="LT" />
    </layers>
...
```

## Conclusion

Now, when we reload the widget, it should actually show the newly uploaded desklets, instead of some dummy texts. In this step we have seen how to make *text-layers*, how to let AveScripter handle XML- and RSS-feeds for us and, finally, how to make the link between the *text-layer* and the RSS-feed using *prefix-tags*.



The next step of this tutorial will show how to let the user visit the newly added desklets by clicking on the links. In other words, we are going to add some interactivity to our desklet!

## Step 3: Scripts for interactivity

### Overview

In this step the items in our desklet will behave as real links: hovering over one will make it change colour, and clicking one will take the user to the page. We achieve this by adding a little JScript code to our desklet.

All finished files for this step can be found under *Final Files/Step3*.

### Setting up the script

Besides showing image- and text-layers, AveScripter – the name already implies it – can use scripts to enhance the desklet. The script will make it possible to dynamically change properties of layers, invoke actions and much more. AveScripter can handle two scripting languages: *JScript* and *VBScript*.

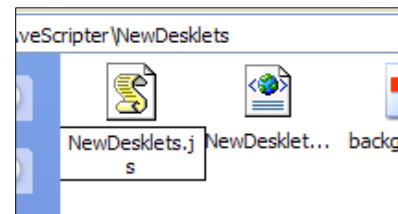
To prepare our desklet to use a script, we only need to add 2 properties to *NewDesklets.xml*: we need to tell which scripting-language we are going to use and which file holds the script. Those properties are added under the *desklet-node*:

```
...
<desklet>
  <alpha>255</alpha>
  <resources></resources>

  <language>JScript</language>
  <script>NewDesklets.js</script>
</desklet>
...
```

As you can see, we named our script-file *NewDesklets.js* and we will use JScript for this desklet. Thus, the next logical step would be to create a text-file called *NewDesklets.js*.

If you are not familiar with JScript (or programming at all), there are several JScript or JavaScript tutorials on the internet. It's a good idea to get a good understanding of how JavaScript works before continuing with this tutorial.



### Implementing highlighting of the links

The first thing we are going to do is to let our text-items look like links: when the user hovers the mouse over an item, it should change colour and get a line under it; when the mouse is moved off the item, the item should restore to its previous properties.

This behaviour is called highlighting and can be easily implemented in JScript using a function that changes the layers *font-properties*. We add the following code to *NewDesklets.js*:

```
function highlight(mouseIsOn)
{
    var layer = this.GetActiveLayer();
```

```

    if (mouseIsOn)
    {
        layer.FontColor = "clBlue";
        layer.Fontstyle = "Bold+Underlinded";
    }
    else
    {
        layer.FontColor = "clWhite";
        layer.FontStyle = "Bold";
    }
}

```

The working of this function is pretty simple. When called, it gets the active layer from AveScripter (`this.GetActiveLayer()`) - the active layer is the layer where the mouse is over – and then changes the style to blue and underlined if the `mouseIsOn` parameter is true, otherwise it changes the style to white.

Now, if we call this function as `highlight(true)` when the mouse enters one of our text-layers, the style will change to that of a link.. If we call `highlight(false)` when the mouse leaves the text-layer, the style of the item will change back to as it was. For each *text-layer*, we add the following properties in *NewDesklets.xml*:

```

<layer text="!XMLTXT:0[rss//channel//item[1]//title]" x="20" y="60"
height="20" width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT"
onmouseenter="highlight(true);" onmouseleave="highlight(false);"
fullhittest="yes" />

```

The value of the *onmouseenter*- and *onmouseleave*-properties will be executed as code. We simply call our highlight function with the correct parameter.

The *fullhittest*-property is set to *yes* to let the layer also hit-test on totally transparent areas. If we didn't do this, then, when the mouse is moved over a transparent part of the text, the highlight would be disabled again. Because this is frustrating to the user – he expects the text to work as a rectangle - , we have enabled full-hit-testing. To see what happens otherwise, disable the property.



### Changing the mouse cursor

We now have our items looking like hyperlinks, except the mouse cursor doesn't change to a hand like links do when you roll over them. This doesn't require code but can be changed with a simple property for the *text-layers*:

```

<layer text="!XMLTXT:0[rss//channel//item[1]//title]" x="20" y="60"
height="20" width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT"
onmouseenter="highlight(true);" onmouseleave="highlight(false);"
fullhittest="yes" mousecursor="32649" />

```

The *mousecursor*-property makes the mouse cursor change to another cursor when the mouse is over the layer. The value 32649 is a hand-cursor. A lot of other values are also available; you can look them up in the AveScripter documentation.

## Adding Clicking

The last thing we need to do is add a clicking feature: when the user clicks on a link, a browser window should open with the page. We can do this by extracting the correct link from the RSS feed in a JScript function and then “launch” that link.

The following function has been added to *NewDesklets.js* that will do this for us:

```
function onClickItem(number)
{
    var link = this.xmls(0).GetValue("XMLTXT:0[rss//channel//item[" +
number + "]]//link");
    this.IO.ShellExecutes(link);
}
```

The parameter number will be used to extract the link from the RSS-feed for the *n*th number item, where *n* is an integer between 1 and 6. For example, `onClickItem(2)` will launch the link to the second item.

The XML feed is also accessible in code. Again, we get the right value thru the usage of XPath. When the link is retrieved, we let Windows launch it by executing it.

The only thing left now is to add an onclick-event for each item in *NewDesklets.xml*:

```
<layer text="!XMLTXT:0[rss//channel//item[1]//title]" x="20" y="60"
height="20" width="260" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT"
onmouseenter="highlight(true);" onmouseleave="highlight(false);"
fullhittest="yes" mousecursor="32649" onclick="onClickItem(1)" />
```

The first item calls `onClickItem(1)`, the second item `onClickItem(2)`, and so on. Now the items will finally behave like real links!

## Conclusion

We have seen how we can use JScript in our desklet to make the text items behave like real links. Scripts, however, are for more powerful than this. In the next steps, we will further explore the usage of scripts.



## Step 4: Flipping Out!

### Overview

In this step we will make our desklet flip to a backside.

The concepts discussed in this step will be *Sides* and *Flipping*.

All finished files for this step can be found under *Final Files/Step4*.

### Adding the flippy

The first thing we are going to do is to add a so-called *flippy*. A flippy is a small image generally placed in the lower-right corner that, when clicked, will flip the widget to its backside.

First, copy the image *flipit.png* that is located in *Images/Step4* to our widget's directory.

Next, we add the following layer to the *layers-node*:

```
<layer src="flipit.png" x="274" y="194" alpha="0"
onmouseenter="!EFFECT:SHOW,SELF,255,10,10"
onmouseleave="!EFFECT:HIDE,SELF,0" mousecursor="32649" fullhittest="yes" />
```

We use full hit-testing here because the *flipit.png* is relatively small and allowing only clicking on the non-transparent parts would be a real pain. Also we use the hand mouse cursor again, to indicate to the user that this image will initiate an action when clicked.

More remarkable is that the flippy is totally invisible at start (its alpha is 0). The idea is to only let the flippy be visible when the mouse is moved over its position. To accomplish this, we use *prefix-tags* in the events, so we don't have to write a bunch of JScript code.

On mouse enter, the flippy will fade in totally; we do this by using the *Show-effect*. On mouse exit, we hide the flippy again by fading it out to zero alpha. More information about those effects can be found in the AveScripter documentation.



### Adding the backside

In step one of this tutorial we saw that we needed to add a *side* to be able to define layers: we just defined one side and used it. Now we are going to define more sides to be able to use a backside.

One of the big advantages of AveScripter is that multiple sides can be defined. Each side is associated with a set of layers and only one side is visible at the time. Switching between sides is easy.

Thus, when we want a front-side and a backside, we simply define two sides and associate layers with them. Normally, the front-side layers are shown, but when we flip to the backside, the backside layers will be shown.

Now, let's define an extra *side*:

```
<sides>
  <side id="0" width="300" height="220" />
```

```
<side id="1" width="300" height="220" />
</sides>
```

As you can see, defining a side is just adding an extra *side-element* under the *sides-node*. The only thing we need to make sure is that we give our new side an id that is different from the first side. We could only use a different width and height if we wanted, but for this tutorial we use the same width and height for the front- and back-side.

### Adding layers to the new side

The new side we've just created doesn't have any layers yet. Adding layers is a simple step: for each side, we add a new *layers-node* under the *skindata-node*, just like we did in step one for the front-side. The only thing different is that we now need to associate the *layers* with a *side*.

```
<layers side="1">
  <layer src="backside.png" x="0" y="0" />
</layers>
```

We associate this set of layers by adding a *side-property* to the *layers-node*. The value of this property is simply the *id* of the *side* that should hold the layers. If there is no *side-property* defined for a *layers-node*, the layers will be associated with the side with id 0.

As you can see, we already defined one layer for the backside, *backside.png*. You'll need to copy this file from *Images/Step4* to our widget's directory.

### Flipping

The only thing left now is to let clicking on the flippy switch from the front-side to the back-side. We do this by using a *prefix-tag* again – it could also be done in JScript code, but as good programmers, we are lazy and prefer the easy solution:

```
<layer src="flipit.png" x="274" y="194" alpha="0"
onmouseenter="!EFFECT:SHOW,SELF,255,10,10"
onmouseleave="!EFFECT:HIDE,SELF,0" mousecursor="32649" fullhitest="yes"
onclick="!FLIP:1" />
```

When the flippy is clicked, AveScripter will now flip the widget to the side with id 1, our backside. The “!FLIP:1” *prefix-tag* simply flips the widget to another side, in this case the side with id 1. In general, “!FLIP:x” (with x a number) will flip the widget to side x.

### Conclusion

In this small step we saw how to use multiple sides and how to flip between sides. We also used some more *prefix-tags* to add fading in and fading out effects for the flippy and to flip between sides.

When you try to flip our widget to the backside now, you will notice there is no way you can flip back to the front-side. In the next step, we will explore *controls* and make it possible to click a button to go back to the front-side again.



## Step 5: User Interface Controls

### Overview

In this step we will explore the usage of User Interface Controls. UI Controls (unfortunately also often referred to as widgets, so we will name them *controls*) are general-purpose UI elements like buttons, textfields, scrollbars, etc... Instead of creating the same controls over and over again, AveDesk (and therefore AveScripter) support a set of default UI controls that can be skinned, and will take full advantage of alpha-blending graphics.

The concepts discussed in this step will be *Controls*.

All finished files for this step can be found under *Final Files/Step5*.

### Images for the controls

The first thing we need to do - again – is to copy the image resources for the controls to our widget's directory. The image resources can be found in *Images/Controls*. You will need to copy the *Controls* folder to our widget's folder.

Once this is done, we need to add a property in *NewDesklets.xml* under the *desklet-node* to tell AveScripter where the controls resources can be found:

```
<desklet>
  <alpha>255</alpha>
  <resources></resources>
  <ctrlresources>Controls</ctrlresources>

  <language>JScript</language>
  <script>NewDesklets.js</script>
</desklet>
```

### Adding the controls to the backside

Now AveScripter knows where to get to find the resources for the controls. We can continue and create some controls. Like layers, controls also need to be defined in *NewDesklets.xml*. Also, controls are defined in a *controls-node*, with each control having a *control-element*. Another similarity with layers is that controls are also associated with a side.

We add the following section to *NewDesklets.xml* under the *skindata-node*:

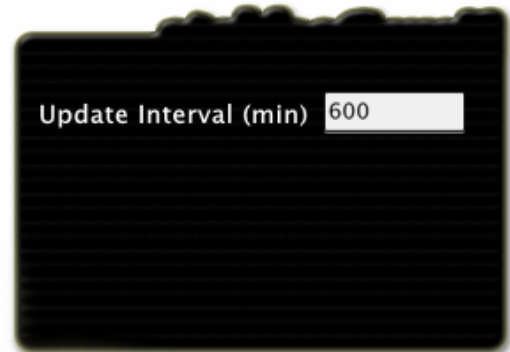
```
<controls side="1">
  <control name="editUpdateInterval" type="EDITBOX" x="184"
y="61" width="76" height="23" fontname="Lucida Sans Unicode" fontsize="9"
fontbg="clHighlight" fontalign="LT" value="600" />
</controls>

  <layers side="1">
    <layer src="backside.png" x="0" y="0" />
    <layer text="Update Interval (min)" x="20" y="60"
height="20" width="160" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />
    <layer src="controls/box.png" x="184" y="59" width="80"
height="24" />
  </layers>
```

As you can see, we have added a *controls-node* with one *control-element*. We have given the control the name “editUpdateInterval” and set the type to “EDITBOX”. The type of the control defines what it looks like and how it behaves. An *editbox-control* is a single line box where users can add and edit text. Other types include DROPDOWNLIST and VERTICAL\_SCROLLBAR. You can look up all possible types in the AveScripter documentation, since there will be new types available on a regular basis.

The other properties of the *control-element* are pretty straightforward. Only the *fontsize-property* behaves a little different here: because editboxes and textfields smooth their texts, this size is different from other sizes. For other types of controls, such as a *dropdownlist-control*, this is not the case.

Besides adding an *editbox-control*, we have also added 2 layers to the backside. One is a *text-layer* that goes in front of the editbox. The other layer is an *image-layer* that is used as the background for the editbox; the editbox does not have a background of its own, but is completely transparent aside from the text it holds.



If you reload our widget now and flip it to the backside, you will notice the *editbox-control* works like a regular editbox: you can select text, copy, paste, right-click, and so on.

### Adding an extra dropdownlist

We now know how to add controls, so we are going to add another one that allows the user to select the type of RSS feed he wants: “latest overall desklets” or “latest AveScripter desklets”. We will do this by providing a *dropdownlist-control* from which the user can select one of those options.

The following changes are made to *NewDesklets.xml*:

```
<controls side="1">
  <control name="editUpdateInterval" type="EDITBOX" x="184"
y="61" width="76" height="23" fontname="Lucida Sans Unicode" fontsize="9"
fontbg="clHighlight" fontalign="LT" value="600" />
  <control name="droplistType" type="DROPDOWNLIST" x="20"
y="120" width="150" height="23" fontsize="14" fontname="Lucida Sans
Unicode" fontstyle="B" visible="yes" fontcolor="clwhite" listvalues="All
Desklets=1;AveScripter Only=2;" value="1" />
</controls>

<layers side="1">
  <layer src="backside.png" x="0" y="0" />
  <layer text="Which Desklets To Show?" x="20" y="96"
height="20" width="200" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />
  <layer text="Update Interval (min)" x="20" y="60"
height="20" width="160" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />
  <layer src="controls/box.png" x="184" y="59" width="80"
height="24" />
</layers>
```

Again, we added a *text-layer* to the backside that describes what the control will do. This will make things more understandable for the user.

We have also added a *controls-element* with DROPDOWNLIST as its type. As you can see when you try the updated widget, our newly created *dropdownlist-control* is using the images from the *Controls/Combo box* folder; if you want the control to look different, you can change those images.



The *dropdownlist-control* uses almost the same properties as the *editbox-control* we added before. The biggest difference is that the *editbox-control* had a `value="600"` property, which just showed the string “600” in the control. The *dropdownlist-control* has a *listvalues-property*, which makes sense because the user can choose from a list of options. The values are separated by semi-colons and each value is of the form `DisplayedText=Key`. The `DisplayedText` will be shown in the control; the `Key` will reflect the option that is currently selected. For example, we set the value to “1” for our dropdownlist and it shows the text that goes with the entry 1: “All Desklets.”

### Adding a button

While a button is technically also a control, AveScripter supports button-like layers by default. Besides the *src-property* that defines what image to display, an *image-layer* can also have an optional *dsrc-property*. This *dsrc-property* defines the image that will be shown when the mouse is held down on the layer.

We add the following layer to *NewDesklets.xml* to create a done-button on the backside which will flip the widget back to the front-side:

```
<layers side="1">
  <layer src="backside.png" x="0" y="0" />
  <layer src="Controls/done.png"
dsrc="Controls/done_pressed.png" onclick="!FLIP:0" x="212" y="170" />
  <layer text="Which Desklets To Show?" x="20" y="96"
height="20" width="200" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />
  <layer text="Update Interval (min)" x="20" y="60"
height="20" width="160" fontname="Lucida Sans Unicode" fontsize="14"
fontcolor="clWhite" fontstyle="B" fontalign="LT" />
  <layer src="controls/box.png" x="184" y="59" width="80"
height="24" />
</layers>
```

As you can see, the *Controls* folder we copied to our widget’s directory also comes with two images for a done-button in the same style as the other controls.

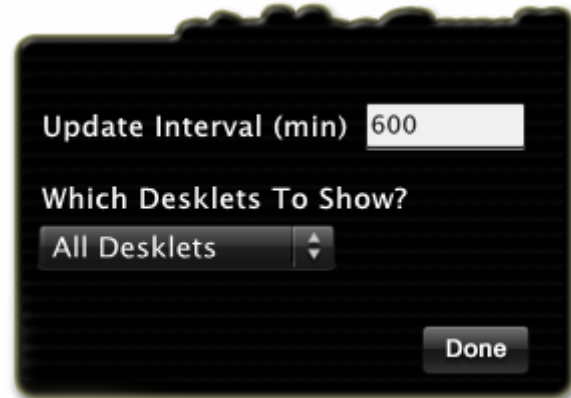
We also use the “!FLIP:” *prefix-tag* to flip back to the front-side (with the id 0).

When you now reload our widget, you will see it has a front-side where the actual data is and a backside with the settings. Also, we can now flip between the back- and the front-side.

## Conclusion

In this step, we have seen how AveScripter support UI controls and we have created two controls: an editbox and a dropdownlist. We have also seen how layers can be used as buttons by using the *dsrc-property*.

However, our pretty controls still don't serve any purpose because they don't do anything. In the next step we will let the user really use the controls to change the widget's settings by using AveScripter's *parameters*.



## Step 6: Parameters and Settings

### Overview

In this step we will use parameters to make our widget configurable for the end user. The concepts discussed will be *parameters*, *events for controls* and *using controls and parameters in JScript code*.

All finished files for this step can be found under *Final Files/Step6*.

### Settings

Our desklet is already working: it shows a list of recent uploaded desklets. But, different users have different needs. For example, one user wants our widget to update itself every 10 minutes, while others are more than happy with a daily update. Therefore, we shouldn't hardcode certain things, but let the user configure them. In short, we want settings.

### Parameters

And parameters are the way to achieve that with AveScripter. All parameters are defined in the *parameters-node* under the *skindata-node*. For each parameter, there is a *parameter-element*. Every parameter has a name and a value. This value of the parameter can change through scripting and can be accessed by its name. Because of this, parameters are the perfect tool to accomplish user-settings.

When we want to make something a setting, we introduce a parameter for it. Now, when we need the setting, we just request the parameters value. Updating settings is done by changing the value of the parameters.

Another important aspect of parameters is that they can be saved when AveDesk is closed. When AveDesk and our widget are reloaded, the parameter will still hold the value that it had before AveDesk closed. This way, user-settings will be saved also.

The best property of parameters is that can be easily added to the XML file. It doesn't require any JScript- or VBScript-code to use them. If a certain parameter has the name "ParamName", the parameters value can directly be used by typing %ParamName% anywhere in the XML file. %ParamName% will simply be replaced with the actual value of the parameter.

### Adding parameters

For our widget's backside, we added two controls: one for the update interval and one for which desklets to show. These will also be our settings, thus we need to create parameters for them. We add the following under the *skindata-node* in the *NewDesklets.xml* file:

```
<parameters>
  <param name="RssType" default="1" save="yes" />
  <param name="UpdateInterval" default="60" save="yes" />
</parameters>
```

This gives our 2 parameters. We tell AveScripter to save them both (*save="yes"*) when AveDesk and our widget are closed. Also, for each parameter a default value has been set. This default value will be used when our widget is instantiated for the first time.

## Changing the XML source to use the parameters

The parameters are now here, but the XML-source for the RSS-feed is still using hard-coded values. We will change the XML-source to use the values of the parameters instead. Because we can simply use `%RssType%` and `%UpdateInterval%` to use the parameters in *NewDesklets.xml*, this is an easy job.

We change the following in *NewDesklets.xml*:

```
<xmls>
    <xml
src="http://avedesk.philc.ca/modules/PDdownloads/latestrss.php?cid=%RssType
%" interval="%UpdateInterval%" usewintmp="yes" />
    </xmls>
```

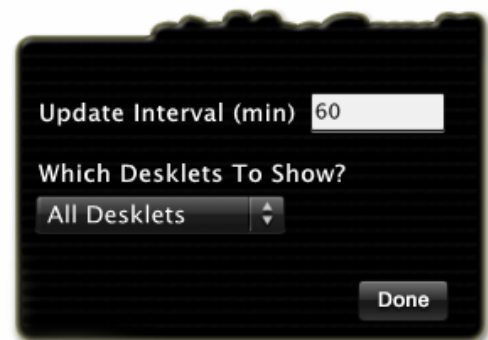
## Changing the controls to use the parameters

We now have two parameters for our settings, but the controls we created in the previous step are still using hard-coded values. It's time to change that too.

We change the following in *NewDesklets.xml*:

```
<controls side="1">
    <control name="editUpdateInterval" type="EDITBOX" x="184"
y="61" width="76" height="23" fontname="Lucida Sans Unicode" fontsize="9"
fontbg="clHighlight" fontalign="LT" value="%UpdateInterval%" />
    <control name="droplistType" type="DROPDOWNLIST" x="20"
y="120" width="150" height="23" fontsize="14" fontname="Lucida Sans
Unicode" fontstyle="B" visible="yes" fontcolor="clwhite" listvalues="All
Desklets=1;AveScripter Only=2;" value="%RssType%" />
</controls>
```

If you reload our widget and flip it to the backside, you will see that the *editbox-control* for the update interval no longer shows the hard-coded value "600", but, instead shows the default value for the *UpdateInterval-parameter*.



## Updating the parameters through the controls

You might have noticed that when you changed the value of the controls, the parameters still kept their old value. What we need to do is to change the values of our parameters when the controls are changed. We have two choices here: we can update the value of a parameter immediately if its corresponding control has changed or we can update all parameters when the *done-button* is pressed.

We choose for the last option, because it's less work (programmers are lazy, again!).

The idea is to let a small function in our JScript *NewDesklets.js* file update all parameters when the *done-button* is pressed. We first design this small piece of code and after that we update the *done-button* to call the code when it's pressed.



We add the following function to *NewDesklets.js*:

```
function onClickDone()
{
    var oldUpdateInterval = this.parameters("UpdateInterval").value;
    var oldRssType = this.parameters("RssType").value;
    var newUpdateInterval = this.controls("editUpdateInterval").value;
    var newRssType = this.controls("droplistType").value;

    if(oldUpdateInterval != newUpdateInterval || oldRssType !=
newRssType)
    {
        this.parameters("UpdateInterval").value =
parseInt(newUpdateInterval);
        this.parameters("RssType").value = newRssType;
        this.xmls(0).Reset();
    }

    this.Redraw(true);
    this.FlipTo(0);
}
```

This function will compare the old values of the parameters with the current value of the corresponding controls. If any of them is different, we update the parameters with the new settings and then reset the XML source. Finally we force the widget to redraw and then flip back to side 0 (the front-side).

This small piece of code also shows some interesting aspects of using JScript code with AveScripter.

For example, it's possible to access all controls and their properties by their name with `this.controls(name)`. Also, it is possible to access the parameters on the same way. Redrawing and flipping is also possible in JScript code.

### Updating the done-button

The final task for this step is to update the *done-button* to call our `onClickDone()` function instead of flipping directly. Recall that the done-button was actually a layer, so we change the following in *NewDesklets.xml*:

```
<controls side="1">
    <control name="editUpdateInterval" type="EDITBOX" x="184"
y="61" width="76" height="23" fontname="Lucida Sans Unicode" fontsize="9"
fontbg="clHighlight" fontalign="LT" value="%UpdateInterval%" />
    <control name="droplistType" type="DROPDOWNLIST" x="20"
y="120" width="150" height="23" fontsize="14" fontname="Lucida Sans
Unicode" fontstyle="B" visible="yes" fontcolor="clwhite" listvalues="All
Desklets=1;AveScripter Only=2" value="%RssType%" />
</controls>
```

### Conclusion

If you reload our widget, you will see that it will now update itself based on the settings made on the backside. Also, when you close AveDesk and load it again, the widget will still use the settings as you left them.

In this step we saw how to use *parameters* to make our desklet configurable for the user. *Parameters* can easily be used in the widget's XML file and in JScript code. In the next step, we finish our widget by paying attention to some small details.

## Step 7: Polishing our widget; the details

### Overview

Our widget is almost done. In this step we will pay attention to some small details and finally package it for distribution.

All finished files for this step can be found under *Final Files/Step7*.

### The Close Button

In AveDesk, if you tap the control key when a desklet is selected, a red close button should become visible. This button allows the user to close the widget with a nice animation.

If you select our widget and make the close button appear, you'll notice that it's pretty far away from the widget. This is because the button is at its default position (0, 0). We can easily change this.

The close button's coordinates are set per side (different sides can have different shapes of course). Here's how to do it:

```
<sides>
  <side id="0" width="300" height="220" closex="12" closey="12" />
  <side id="1" width="300" height="220" closex="14" closey="24" />
</sides>
```

The *closex-* and *closey-properties* simply define the coordinates where the close button is placed. If you update our widget, you'll see that the close button now shows up at the right spot.

A fun thing to do is to place the close button at different positions: at the right side, in the middle, etc... and watch how AveDesk's close operation looks.



### Packaging our widget for distribution

Files for AveDesk are distributed in the .aveinst-format. Files in this format can be automatically installed by AveDesk, thus making life easier for the user. It can also automatically download AveScripter, if it's not already installed.

An AveInst-file is simply a zip file with a special *avedesk\_installer.xml* file describing which files to copy and what actions to take when installing. A sample *avedesk\_installer.xml* file for AveScripter widgets can be found in this tutorials *Files/Installer* folder.

One thing to pay attention to is that every AveInst-package's *GUID-field* must be unique. We can generate GUIDs with a small script, provided under the *Files/Installer folder as gen\_guid.vbs*. To generate a new GUID for usage in our AveInst package, just run the file (your virus-scanner will probably scream out loud, just tell it to authorize the script). The script will popup a small box with a new GUID. Just copy this value into *avedesk\_installer.xml* where it says NEWGUIDHERE.

Another thing we can provide is a splash screen for when our widget is installing. For this tutorial, a splash screen is provided under the *Images* folder as *about.png*, but you are free to create your own.

When you are done, place our widget's directory inside the zip's file in the *Desklets/AveScripter* folder. Also copy the new *avedesk\_installer.xml* to the root of the zip-file. Finally, copy the *about.png* file also to the root of the zip-file and then rename the zip file to *NewDesklets.aveinst.*

You can now choose to upload your widget to <http://avedesk.philc.ca> if you are happy with it.

## Conclusion

In this final step, we have placed the close button at its right position and packaged the widget into an *aveinst*-file, ready for distribution.

Overall, we have seen how to create a desklet for AveScripter from scratch by making an XML-file and writing a small bit of JScript code. This tutorial, however, only covered the basics of AveScripter. Much more topics are worth exploring: *sliding*, *using XMLHttpRequest*, *scrolling*, *XSLT-transform*, and so on.

While a lot can be learned from studying other AveScripter widgets, also some of those topics will be discussed in upcoming tutorials, so stay tuned.

## Thanks

Big thanks go of course to Phil Caetano for creating AveScripter. Also, the other needs to thanks *Loofygun*, *Ron21*, *Septimus* and *Nightcrawler1089* for supporting and proof reading.