Extreme Makeover, Part 2: Wrap Your Scripts Up in a GUI Interface

HTML Applications (HTAs) provide a way for you to wrap your scripts up in a graphical user interface, an interface replete with check boxes, radio buttons, drop-down lists and all those other Windows elements you've grown to love. In this article, the second of a multi-part series, we'll show you how to add standard user interface controls to an HTA.

HTAs for Beginners: Part 2

Hello everyone, and welcome back to *HTAs for Beginners*. (Thank you, but, please, hold your applause until the end.) In our last episode we introduced you to HTAs (short for HTML Applications), a way to wrap your scripts in a graphical user interface. In that first article we showed you how to create a very simple HTA, one that – when you clicked a button – went out, grabbed some information about, say, the services installed on a computer, and then displayed that information right in the HTA itself. For many of you, that was a life-changing experience.

Well, OK. But it *was* an experience.

Our first article gave you enough information to create a rudimentary HTA, and no doubt some of you thought, "Well, now that I've learned how to add a button to an HTA I've probably learned all there is to know." On the contrary. Buttons are nice, but they are hardly the end-all and the be-all of HTA development: there are scores of other user interface elements that you can incorporate into an HTA, including list boxes, check boxes, and everyone's favorite, the radio button. And by astonishing coincidence, in this month's article (Part 2 in the series) we're going to show you how to add many of those controls to an HTA.

Please, no need to thank us; we're just doing our job.

If you're looking ahead, this month we'll go through the basics of adding various HTML controls to an HTA. Having mastered the use of the various controls, next month we'll start working on practical implementations, and begin building an HTA that does something cool and something useful. (No, we don't exactly know what that will be. But, hey, we still have a month to figure that out!) After that, well—we'll just have to see what happens after that.

At any rate, thank you all for coming today. We hope you enjoy the show, and please drive home safely.

A Note About This Month's Try-It-Yourself Exercises

Throughout this month's tutorial we'll periodically stop and give you a chance to test your new-found knowledge by asking you to write an HTA incorporating the current topic (for example, adding a drop-down list box to an HTA). Just to prove that we're not *totally* heartless, we've provided this basic template that can be used in all of the exercises:

```
<html>
<head>
<title>HTA Test</title>
<HTA:APPLICATION
     ID="objTest"
     APPLICATIONNAME="HTA Test"
     SCROLL="yes"
     SINGLEINSTANCE="yes"
>
</head>

<SCRIPT LANGUAGE="VBScript">

' PUT YOUR SUBROUTINES HERE

</SCRIPT>
```

```
<body>

' PUT THE HTML TAGS HERE

</body>
```

To use the template, copy the preceding code and paste it into Notepad (or some other text editor). Replace the line ` PLACE YOUR SUBROUTINES HERE with the script code and the line ` PUT THE HTML TAGS HERE with your HTML code. Save the whole thing with a .hta file extension, and you'll have yourself a (keep your fingers crossed) fully-functional HTA.

Top of page

Adding a Drop-down List Box

There's definitely a coolness factor to HTAs: after all, an HTA looks *way* cooler than a script that runs in a command window. On the other hand, looks aren't everything (fortunately for the Scripting Guys). In fact, there's no point in creating an HTA unless the HTA *does* something a regular script can't do.

So what *can* an HTA do that a plain-old .vbs file can't? Well, suppose you have a number of file servers and you periodically need to run a script to retrieve information from one of those computers. How can you target a single script to just one of your many file servers? Well, one tried-and-true way is to use command-line arguments; thus:

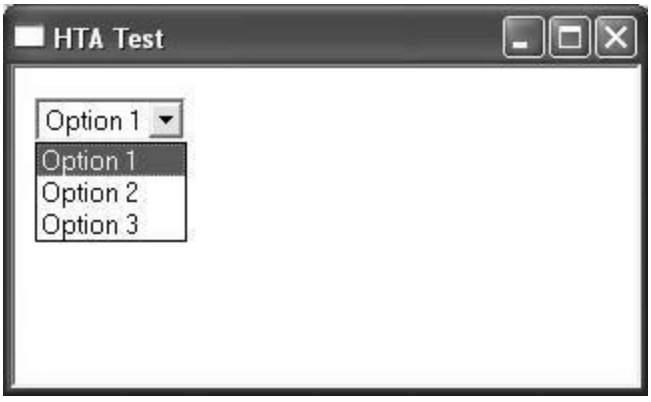
```
cscript myscript.vbs /server:atl-fs-01
```

What's wrong with using command-line arguments? Nothing, absolutely nothing.

Well, OK, you *do* have to type in the computer name each time. And you do have to type that name in *correctly* each time. And you do have to memorize all the names of all your file servers. None of that is particularly inviting, but what choice do you have?

Well, how about this for a choice: you take an HTA, add a drop-down list box pre-populated with the names of all your file servers, and then simply select a name from the list any time you want to run a script. No typing (or typing errors) and no memorization of server names required.

In other words, a drop-down list box (like the one shown below) is not only aesthetically-pleasing, but can be a time- and frustration-saver as well.



Yes, we *do* make a pretty convincing case, don't we? Of course, we did leave out one minor point: how exactly do you add a drop-down list box to an HTA? Relax; we wouldn't build your hopes up like that and then let you down. As it turns out, the code for creating a drop-down list box (just like the code for creating all the other controls we'll show you in this article) is pretty simple and straightforward:

```
<select size="1" name="OptionChooser" onChange="TestSub">
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```

See, that's not so scary, is it? This standard HTML code starts off – as all drop-down lists and list boxes do – with a <select> tag. We simply type in a <select> tag along with the following parameters:

- **size.** This is the number of options that appear onscreen at any one time. Setting the size to 1 gives us a drop-down list. What if we set the size to something other than 1? Well, we don't want to spoil the surprise, but we'll talk about that when we talk about list boxes. For now, just remember this: if you want a drop-down list, set the size to 1.
- **name.** This is the name given to the control, and the name we'll use in our scripts to refer to that control. For example, to echo the value of our drop-down list (that is, to determine which option was selected) we use this single line of code:

```
Msgbox OptionChooser.Value
```

- **onChange.** This is the event handler for the control. What we're saying here is pretty simple: any time the value of this control changes (that is, any time someone selects a different option) run this subroutine (in this example, the subroutine named TestSub).

We should point out that including an event handler (onChange) as part of the control is purely optional; we don't *have* to run a script any time someone makes a selection in the drop-down list. Alternatively, we could have the user make a selection and then click a button in order to run a script; it's up to you to decide which approach you like the best. We should note, however, that there's actually a problem with the control as currently configured, something we'll discuss in more detail – and resolve – at the end of this section.

After the <select> tag, we stuff in three more lines of code, then add a closing </select> tag to indicate that we're done configuring this particular control. There you go: your very first drop-down list box.

Oh, good point: what *about* that stuff between <select> and </select>?

```
<option value="1">Option 1</option>
<option value="2">Option 2</option>
<option value="3">Option 3</option>
```

As you probably figured out (the tag name <option> *is* a dead giveaway, isn't it?) these are the list options, the actual items that appear when you click the drop-down list. Each individual item in the list consists of the following pieces:

- The <option> tag, with a unique **value** parameter. The value parameter can be anything you want, though the values should be unique. And while the values can exactly match the displayed text, they don't have to; in our preceding example, the first option has a value of **1** but the displayed text is **Option 1**. When we use a subroutine to determine which option was selected, we will typically work with the value of the option.
- The displayed text. Anything that lies between the <option> tag and the </option> tag will be displayed as-is in the drop-down list. In the example our first option shows up as **Option 1** in the drop-down list. What if we wanted it to show up as **Ping remote computer**? Then we'd configure the option like this:

```
<option value="1">Ping remote computer</option>
```

  Note that we don't need to change the value; as we said earlier, values can be anything we want them to be.
- The closing </option> tag. Simply so we know where one option leaves off and the next one begins.

You might have noticed that we don't give the individual options their own names. Why? Because we have no reason to name individual options. Instead, when you select an option the value of the selected option gets passed to the <select> object. This allows us to determine which option was selected by examining the value of the <select> object. (In case you're wondering, that's why option values need to be unique.) For example, here's a simple little subroutine that reports back the value of the selected option:

```
Sub TestSub
```

```
    Msgbox "You selected Option " & OptionChooser.Value & "."
End Sub
```

See? All we do is report back the value of the drop-down list itself (OptionChooser), which will just happen to be exactly the same as the value of the selected option. (Keep in mind that the value is going to be 1, 2, or 3, because those are the values we assigned the individual options.)

So What's Wrong with the Control as Configured?

We showed you a simplistic version of the drop-down list because we wanted you to see how easy it is to implement HTML controls in an HTA; consequently, we wanted to keep the code as short and concise as we could. However, there *is* a problem with the control and the onChange event. When you load the HTA, Option 1 will automatically be selected. That's fine, but suppose you want to run the script against Option 1; how do you do that? Believe it or not, it's tougher than you think. After all, you can't select the option; it already *is* selected. Because of that, clicking the option won't trigger the onChange event; onChange won't get triggered because nothing has changed. In fact, as configured, the only way to select Option 1 is to first select, say, Option 2 (which causes the script to run against *that* option) and *then* select Option 1.

Eep.

But don't worry, there's an easy way to fix this. All we need to do is add a blank option at the top, using code similar to this:

```
<select size="1" name="OptionChooser" onChange="TestSub">
    <option value="0"></option>
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```

Notice that our first option – with the value 0 – has no displayed text; it will appear blank in the drop-down list. We then need to modify our subroutine to ensure that the script runs only if the value of the selected option does not equal 0. In other words, the script will not run if you select the blank option:

```
Sub TestSub
    If OptionChooser.Value <> 0 Then
        Msgbox "You selected Option " & OptionChooser.Value & "."
    End If
End Sub
```

Make sense? If not, try this complete HTA and you'll see exactly what we mean:

```
<html>
<head>
<title>HTA Test</title>
<HTA:APPLICATION
     ID="objTest"
     APPLICATIONNAME="HTA Test"
     SCROLL="no"
     SINGLEINSTANCE="yes"
>
</head>

<SCRIPT LANGUAGE="VBScript">

    Sub TestSub
        If OptionChooser.Value <> 0 Then
            Msgbox "You selected Option " & OptionChooser.Value & "."
        End If
    End Sub

</SCRIPT>

<body>
```

```
<select size="1" name="OptionChooser" onChange="TestSub">
    <option value="0"></option>
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>

</body>
```

Click Option 1, 2, or 3 and you'll see a message box reporting the option you selected. Click the blank option and nothing will happen. Problem solved.

Try It Yourself

OK, let's see what you've learned so far. For our first try-it-yourself exercise, see if you can create an HTA that includes a drop-down list box with the following computer names as options:

- **atl-dc-01**
- **atl-dc-02**
- **atl-dc-03**

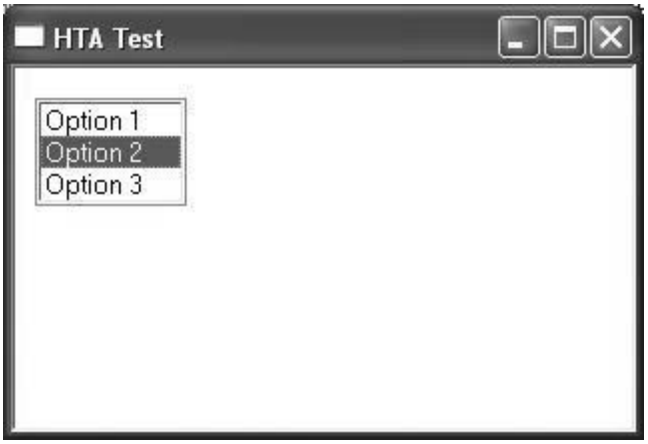
When you select one of the options, a subroutine should report back the name of the selected computer. (Hint: To do that, you might want to make the value of each option identical to the displayed text.) If you have problems getting this to work, click here to see the answer.

Note. Some of you might be wondering, "Is it OK to cheat? Is it OK to copy code out of the article, paste it into my exercise, make a minor modification or two, and then pass that off as a finished assignment?" You bet it is; in fact, that's the kind of thing we encourage you to do. After all, we just want you to be able to create HTAs; if you find that "cheating" is the easiest/fastest way to create an HTA, then cheat away.

Top of page

Adding a List Box

By definition a drop-down list box shows only one item at a time; if you want to see additional items you need to click the control and drop down a list of additional items. There's nothing wrong with that; however, there might be times when you'd find it easier/more effective to see multiple options at the same time. In other words, there might be times when you want to use a list box rather than a drop-down list:



Hey, why not; after all, variety *is* the spice of life. So let's talk about adding a list box to an HTA.

One of the best things about list boxes is this: after you know how to create and use a drop-down list you'll know how to create and use a list box as well. For example, here's the code that creates a list box (shown above) with three options:

```
<select size="3" name="OptionChooser" onChange="TestSub">
    <option value="1">Option 1</option>
```

```
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```

What's the difference between this code and the code that created the drop-down list? Just one thing: in this case we've set the value of the size parameter to 3. That means we want to show three options onscreen at the same time. If we set the size to 12 we'd show 12 options onscreen at the same time. And if we set the size to 1 we'd have a drop-down list. It's that simple.

Incidentally, you don't have to make a list box the same size as the number of options you have. What if you have 100 options but only have room onscreen to show 10 at a time? That's fine: just set the size to 10 and a scroll bar will automatically be added to the list box, enabling you to scroll up and down and access all 100 options. Play around with this a bit and see for yourself.

Try It Yourself

This is basically the same exercise as the first; this time, however, we want you to use a list box instead of a drop-down list. See if you can create an HTA that includes a list box (one that shows three items at a time) with the following computer names as options:

- **atl-dc-01**
- **atl-dc-02**
- **atl-dc-03**
- **atl-dc-04**
- **atl-dc-05**
- **atl-dc-06**

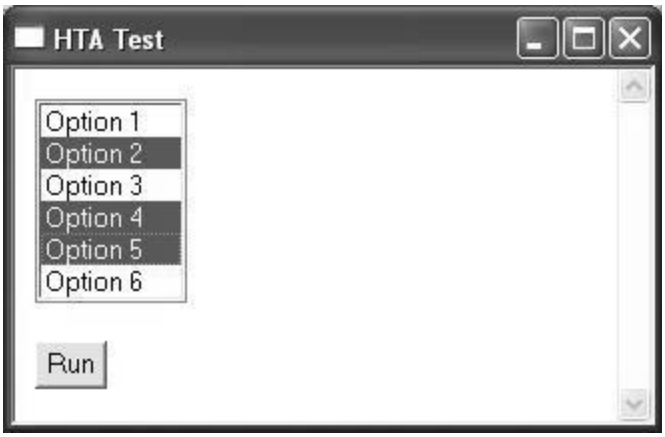
When you select one of the options, a subroutine should report back the name of the selected computer. If you have problems getting this to work, click here to see the answer. For extra credit, remove the onChange event handler and see if you can add a button that will report back the name of the selected computer. If you have problems with *that*, don't sweat it; we'll review the code for adding a button later on.

Handy hint. After adding the list box be sure and include a <BR> tag or a <P> tag; that will ensure that your controls don't line up horizontally across the screen.

Top of page

Adding a Multi-Select List Box

We have one more list box variation to show you. In HTML you can create something known as a multi-select list box. With a multi-select list box you can hold down the Ctrl key and select as many options as you want. For example, do you need to select options 2, 4, and 5, all at the same time? Then go ahead and do it:



We understand your concerns: no doubt creating a multi-select list box takes a lot of very sophisticated, highly-technical skill, the kind of skill only a Scripting Guy could have. Just how complicated *is* creating a multi-select list box? This complicated:

```
<select size="3" name="MultiListbox" multiple>
    <option value="1">Option 1</option>
    <option value="2">Option 2</option>
    <option value="3">Option 3</option>
</select>
```

No, we didn't paste the wrong code in by mistake. The truth is that creating a multi-select list box is pretty much like creating a regular old list box. In fact, there are only two differences:

- **We added the multiple attribute to the <select> tag. That's what turns a plain-old list box into a way-cool multi-select list box.**
- **We removed the onChange event from the <select> tag. Why? Well, we don't want to run a script each time the list box changes in some way; if that happened we'd select the first option and then the script would run, without waiting for us to select additional options. Instead we want the user to be able to select all the desired options and then click a separate button to kick off a script. Thus we need to remove the onChange event handler and then add a button that will run the TestSub subroutine.**

In fact, the only tricky part about adding a multi-select list box crops up when you need to determine which options – if any – were selected. You can't just echo back the Value when dealing with a multi-select list box; the singular notion "Value" is meaningless when you can have multiple items with multiple values. Instead, you'll have to loop through all the options and determine which ones have a Selected property equal to True. That's how you can identify which items have been selected and which ones haven't.

No, it's not too hard. In fact, all it takes are a few lines of code:

```
Sub TestSub
    For Each objItem in MultiListbox.Options
        If objItem.Selected Then
            strSelected = strSelected & objItem.Value & vbCrLf
        End If
    Next
    Msgbox "You selected the following items: " & vbCrLf & strSelected
End Sub
```

See? All we do is create a For Each loop to cycle through all the options (items) in the list box; that's what this line of code does:

```
For Each objItem in MultiListbox.Options
```

For each option (item) we check the value of the Selected property. If Selected is True that means the item was selected; we then store the value of that particular option in a variable named strSelected. If Selected is False we don't do anything except move on to the next item in the collection. After looping through the entire set of options we then echo back a message similar to this, detailing the options that were selected:



That's all there is to it.

Incidentally, if you never *did* figure out how to add a button to your HTA, here's a hint:

```
<input id=runbutton  type="button" value="Run" name="run_button"  onClick="TestSub">
```

Try It Yourself

Similar to the previous two exercises, see if you can create an HTA that includes a *multi-select* list with the following computer names as options:

- **atl-dc-01**
- **atl-dc-02**
- **atl-dc-03**
- **atl-dc-04**
- **atl-dc-05**
- **alt-dc-06**

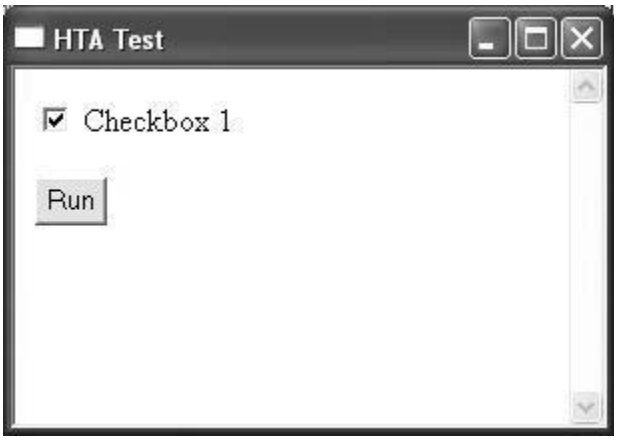
In addition, your HTA should include a button that, when clicked, calls a subroutine that reports back *all* the options that were selected. (Remember, with a multi-select list box you can hold down the Ctrl key and select more than one option at a time.) If you have problems getting this to work, click here to see the answer.

Top of page

Adding a Check Box

Picture an HTA that lists a number of different scripts you can run against a computer; maybe there's a script that performs a hardware inventory, one that performs a software inventory, another that retrieves performance statistics. (Yes, it's exactly the way medieval mystics pictured Paradise.) When you load that HTA, you'd like to be able to pick and choose which scripts to run and which scripts not to run. Maybe this time out you want to run the two inventory scripts but not the performance script; the next time out maybe you want to do the hardware inventory and get the performance statistics.

In other words, you want options that can be toggled on and off independently (that is, choosing the hardware script doesn't automatically cause you to run the software script). If you're thinking to yourself, "Hmmm, this sounds like a job for a check box," you're absolutely right:



The check box is just another variation of the <input> control (a control we learned about last month), this one with a type of, well, checkbox:

```
<input type="checkbox" name="Checkbox1"> Checkbox 1
```

Basically all you have to do is add an <input> tag along with the type and name parameters. (Note: No closing tag – that is, no </input> tag – is required.) After adding the tag simply type the label that appears next to the tag. In our preceding example, we gave the check box the label Checkbox 1. (Yes, they *do* pay us to think up stuff like that. Why do you ask?) If we wanted a check box with the label Run disk cleanup script we'd use this code:

```
<input type="checkbox" name="Checkbox1"> Run disk cleanup script
```

Scripting Guys Tip. What if you'd like to have the check box checked by default? In that case, just set the value of the checked parameter to True:

```
<input type="checkbox" name="Checkbox1" value="Checkbox1" checked=True> Checkbox 1
```

To determine whether or not a check box has been checked all we do is look at the value of the Checked property. If True then the box is checked; if False, then the box is *not* checked. Here's some sample code that does that checking for us:

```
Sub TestSub
    If Checkbox1.Checked Then
        Msgbox "You checked the check box."
    Else
        Msgbox "You did not check the check box."
    End If
End Sub
```

What if we have more than one check box in the HTA? In that case, you just need to write separate If-Then blocks that examine the Checked property of each of those check boxes. In other words:

```
Sub TestSub
    If Checkbox1.Checked Then
        Msgbox "You checked check box 1."
    Else
        Msgbox "You did not check check box 1."
    End If
    If Checkbox2.Checked Then
        Msgbox "You checked check box 2."
    Else
        Msgbox "You did not check check box 2."
    End If
End Sub
```

Note that we do two things in this subroutine: first we look at the value of the Checked property for Checkbox1, then we do the same thing for Checkbox2.

Try It Yourself

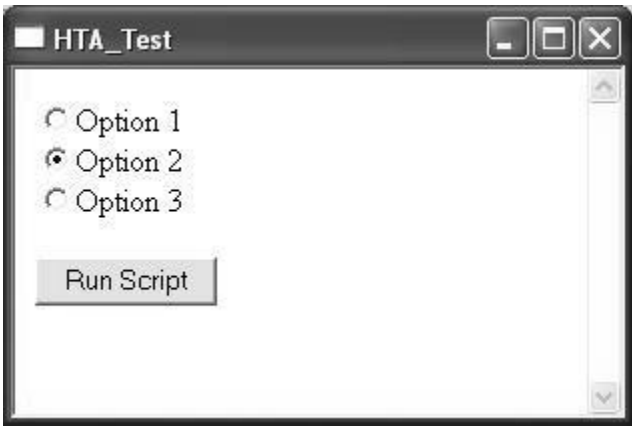
Create an HTA that includes two check boxes, one labeled atl-dc-01 and one labeled atl-dc-02. The HTA should also include a button that, when clicked, tells you whether either (or both) of the two check boxes has been checked. If you have problems getting this to work, click here to see the answer.

Top of page

Adding Radio Buttons

Radio buttons are designed for items that are mutually-exclusive: that is, you can be one of those things, but you can't be more than one of those things. For example, you can choose to save a file or you can choose not to save a file, but you can't choose to simultaneously save and not save a file. (That sounds like one of those clever human tricks Captain Kirk would use on the old Star Trek show to send a computer into an endless loop and once again defeat cold-blooded logic. We'd like to see Captain Picard do *that*.)

In other words, radio buttons are different from check boxes. Suppose you had 10 check boxes on screen. By definition you could select any 2 or any 5 or even all 10 of those check boxes. With radio buttons (shown below) things are very different: you can select one – and only one – radio button within a group:



Yes, hardly the control for the Me Generation; after all, nowadays we want it *all*. But with a radio button, you only get one.

We should mention from the outset that the radio button is not the most intuitive HTML control you'll find yourself using. Up to this point all the controls we've created have had unique names. That makes sense; how else would you distinguish one control from another? With radio buttons, however, things are different. Say we want to create a group of three radio buttons, like the group we just showed you. In that case, we need to create three separate radio buttons, each with the exact same name.

No, we're totally serious: all the radio buttons in a group must have the same name (although they should have different values). That's how our script knows that these buttons belong together. (In other words, that's how the script knows that these buttons are mutually-exclusive.) If that doesn't make sense, don't worry; we'll show you exactly how it works.

Here's some HTML code that creates radio buttons labeled Option 1, Option 2, and Option 3:

```
<input type="radio" name="RadioOption" value="1">Option 1<BR>
<input type="radio" name="RadioOption" value="2">Option 2<BR>
<input type="radio" name="RadioOption" value="3">Option 3<P>
```

As you can see, all three of these buttons have the same name: RadioOption. That leads to the obvious question: if all the buttons have the same name, how can you tell them apart? That is, how can you tell if a user selected Option 1 rather than Option 2 or Option 3?

Good question. When it comes to radio buttons, all the controls with the same name are put into an array that has the same name as the control (in our preceding example, that means an array named RadioOption). To determine which button was selected all we have to do is set up a For Each loop to loop through that array, taking a peek at each button and seeing whether or not the Checked property is True. If Checked is True, then that button was selected. Your code for looping through an array of radio buttons might look something like this:

```
Sub TestSub
    For Each objButton in RadioOption
        If objButton.Checked Then
            Msgbox "You selected Option " & objButton.Value & "."
        End If
    Next
End Sub
```

As you can see, all we're doing is looking to see whether or not the Checked property is True; if it is, we then echo back the button Value (this, by the way, is why button Values need to be unique). If the Checked property isn't True, we don't do anything at all. Like we said, not the most intuitive thing in the world, but it works.

Incidentally, it's possible to have more than one group of radio buttons within the same HTA; just make sure that each *group* is given a different name. If you have a set of radio buttons, all with the name RadioOption, then you'll want to give your next set of radio buttons a name like SecondRadioOption. As you've probably figured out by now, all the names you give to controls are arbitrary; name your controls whatever you want.

(Just make sure you avoid using blank spaces anywhere in the name.)

Try It Yourself

For this exercise, let's see if we can create an HTA that includes a group of radio buttons; the individual radio buttons should have the following computer names as labels:

- **atl-dc-01**
- **atl-dc-02**
- **atl-dc-03**

The HTA should also have a button that, when clicked, reports back the name of the selected computer. If you have problems getting this to work, click here to see the answer.

Top of page

Adding a Text Box

Drop-down lists, radio buttons and other fancy user interface controls can be a great timesaver for your users: it's much faster (and much less error-prone) to have users select from a predetermined set of computers than it is to have them type in the desired computer name. However, you can't always provide your users with predetermined options. For example, suppose you have an HTA for creating user accounts. There's no way to provide drop-down lists pre-populated with all the possible first names and all the possible last names for a new user; it just won't work. Instead, you have to provide a mechanism that allows people to enter *any* information. And the best mechanism for that is the humble – yet highly useful – text box:



Humble, useful, and very easy to implement: to add a text box to your HTA just use an <input> tag, specifying the type (text), a name, and, optionally, the size (that is, the number of characters wide):

```
<input type="text" name="BasicTextbox" size="30">
```

If you don't include a size, by default the text box displays 20 characters.

Handy hint. The size of a text box is just the width of the text box onscreen; in no way does the size determine the number of characters that can be typed into a text box. If you *do* want to limit the number of characters a user can type into a text box then add the maxLength parameter. For example, this code displays a text box 30 characters wide, but only allows you to type a maximum of 25 characters:

```
<input type="text" name="BasicTextbox" size="30" maxLength="25">
```

Little-known historical fact. The Scripting Guys once participated in a "post-mortem" for a project that took a couple years to complete and was riddled with problems all the way through. We were allowed to air our complaints using an online form...a form that accepted a maximum of 255 characters in the Complaints box. In case you're wondering, this note consists of 392 characters by itself.

So how can we retrieve the information that's been typed into a text box? That's easy: we just have to

retrieve the **Value** property. For example, this simple little subroutine echoes back whatever's been typed into a text box named BasicTextbox:

```
Sub TestSub
        Msgbox BasicTextbox.Value
End Sub
```

By the way, it's also very easy to programmatically "type" something into a text box. All you have to do is set the Value of the text box. For example, this subroutine "types" **Fabrikam, Inc.** into a text box named BasicTextbox:

```
Sub TestSub
    BasicTextbox.Value = "Fabrikam, Inc."
End Sub
```

Would you ever want to do something like this? Sure. Suppose you have a tool for creating a new user account. The person creating the account selects the department from a drop-down list. If, say, all members of the Accounting department work in the same building in the same city, then you could have a subroutine that automatically fills in the building and city. Yet another reason why HTAs are so handy.

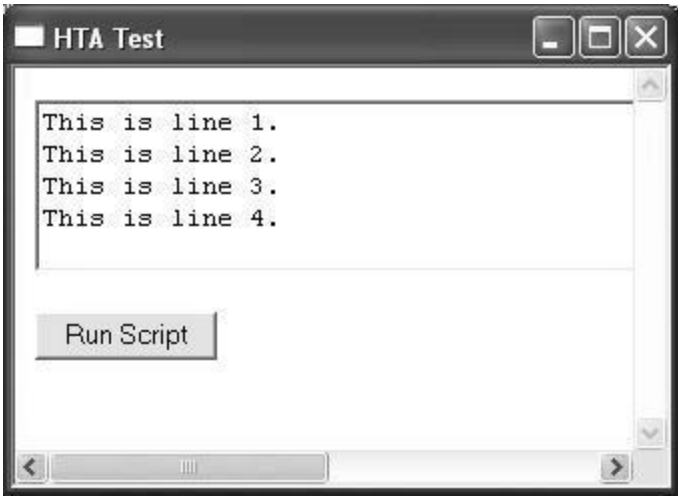
Try It Yourself

OK, this is an easy one: create an HTA that includes a text box and a button; when the button is clicked, it should report back whatever was typed into the text box. If you have problems getting this to work, click here to see the answer.

Top of page

Adding a Multi-Line Text Box

The standard text box is fine if you're entering nothing more than a first name or last name; because the text box is limited to a single line, however, it's not as useful for entering anything more substantial. If you need more room to enter (or to display) text, then you can use the <textarea> control, and get something that looks like this:



Now that's more like it. And don't worry; configuring a multi-line text box is very easy:

```
<textarea name="ScriptArea" rows=5 cols=70></textarea>
```

As you can see, all we need to do is toss in a <textarea> tag (and the accompanying </textarea> tag) and set the following parameters:

- **name.** The name of the control.

- **rows. The number of lines that appear onscreen. A scrollbar will automatically be added to the control to enable you to type additional lines.**
- **cols. The width of the control, in characters. If you set cols to 70 then you should be able to type 70 characters, using Courier 12-point, on a single line.**

You retrieve the value of a multi-line text box the same way you retrieve the value of a regular text box:

```
Sub TestSub
    Msgbox ScriptArea.Value
End Sub
```

That's true. But you don't have to *tell* anyone how easy that is. Let them think you're doing something only a programming genius could do!

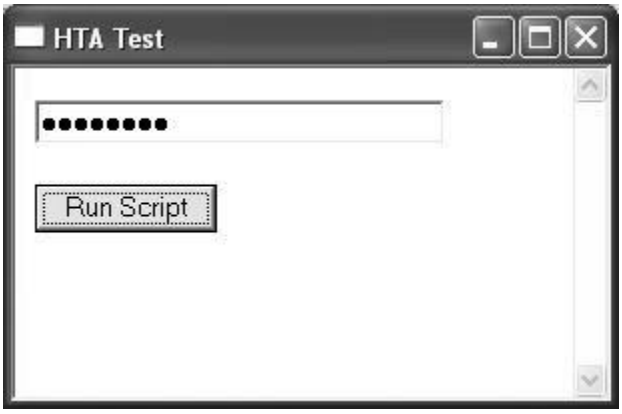
Try It Yourself

This is a variation on the previous exercise. Instead of creating an HTA that includes a text box and a button, create an HTA that includes a multi-line text box and a button. The multi-line text box (textarea) should be sized for 40 columns and 5 rows; the button, when clicked, should display whatever was typed into the text box. If you have problems getting this to work, click here to see the answer.

Top of page

Adding a Password Box

One final control we want to introduce is the password box. The password box is just like the text box, but with one important exception: everything you type into a password box is "masked" so that the characters are not visible onscreen. In other words:



The code for adding a password box is almost identical to the code for adding a text box. To add a password box you use the <input> tag and you specify a name and size; the only difference is that you set the type to "password" rather than "text":

```
<input type="password" name="PasswordArea" size="30">
```

And, yes, you're absolutely right: retrieving the information typed into a password box is as simple as getting back the value of that control:

```
Sub TestSub
    Msgbox PasswordArea.Value
End Sub
```

To tell you the truth, we're big fans of simple and easy.

Try It Yourself

For our final exercise we'll give you another easy one: create an HTA that includes a password box and a

button; when the button is clicked, it should report back whatever was typed into the password box. If you have problems getting this to work, click here to see the answer.

Top of page

In Our Next Exciting Episode....

We covered an awful lot in this one article, and you might find yourself thinking, "Wow, how can I possibly remember all that?" Here's a hint: don't even *try* to remember everything we covered here. Instead, use this month's article as a reference. If you decide to put a check box or two into your next HTA, then refer back to the section on check boxes. If you have no need to put a check box or two in your next HTA, well, then it doesn't really matter whether or not you've mastered the art of check box implementing now, does it?

On the bright side, you now know (or can easily find out) how to implement all the more-commonly-used HTML controls. Armed with that knowledge, next month we'll begin using those controls to create more practical (and far more valuable) HTAs.

If you have questions or comments about this article, drop us a line at scripter@microsoft.com (in English, if possible).

Top of page

Manage Your Profile

© 2009 Microsoft Corporation. All rights reserved. Contact Us | Terms of Use | Trademarks | Privacy Statement

Microsoft