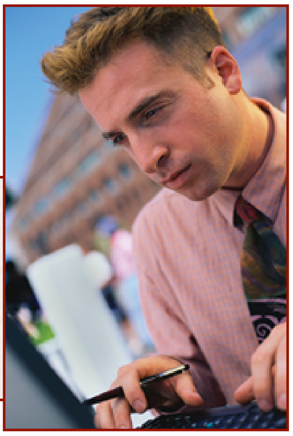
Javascript in the Corvid Servlet Runtime Templates



A system demonstrating this “How To” can be run under the “Javascript in the Corvid Servlet Runtime Templates” section of:

http://www.exsys.com/support/howto

The code for the sample system can be downloaded from the same page.

Javascript in the templates makes the web pages more active. Javascript can be used to detect or prevent user errors or automatically do things to help the user. This demo shows how to use Javascript to:

- Require the user to answer all questions and highlight the unanswered question.
- Automatically place the cursor in the first edit box, if any.
- Values such as 'None of the Above' can perform actions.
- Pop-up helper web pages that explain the question or results.

Javascript is a programming language. Javascript source code can be put inside web pages, causing the browser to run the Javascript program. Javascript is not Java (or applets or servlets) and is unrelated so don't confuse the two.

This How To explains and illustrates how to use Javascript for a few commonly needed operations in Corvid question screens. It assumes some knowledge of Javascript and how to use it in web pages.

Positioning the Cursor in an Edit Box

The 'QuestionTemplate.html' file for the demo system uses Javascript to position the cursor in the first text edit box on a page, if any, so the user can immediately type in an answer without having to remember to click inside the edit box first. This makes answering questions more intuitive for the end user:

Here is the first few lines of that template:

```
<html>
<head>
       <title>Corvid Runtime Question</title>
       <base href="CORVID_LINK_BASE">
       <script language="JavaScript"
        src="PutCursorInEditBox.js"></script>
        <!--The .js file must be located in CORVID_LINK_BASE-->
</head>
<body onload="javascript:SetFocusOnFirstEditBox(document.CorvidForm)">
```

The **<script>** tag causes the browser to get the file 'PutCursorInEditBox.js' and use it when needed.

The **<body>** tag has an attribute:

```
onload="javascript:SetFocusOnFirstEditBox(document.CorvidForm)"
```

which tells the browser to run the function named 'SetFocusOnFirstEditBox' when the web page is loaded. 'SetFocusOnFirstEditBox' is in the file 'PutCursorInEditBox.js'. The contents of ' PutCursorInEditBox.js' is:

```
function SetFocusOnFirstEditBox(form) {
   //Find the first edit box and put the cursor there.
   //for each element in the FORM
   for (i=0; i < form.elements.length; i++) {
      var elem = form.elements[i];
      var type = elem.type.toLowerCase();
      if (type == "text" || type == "password" || type =="textarea") {
          elem.focus();
          break;
       }
   }
}
```

The algorithm is: for each element inside the <form> tag, if the element is a TEXT, PASSWORD, or TEXTAREA edit box, then set focus to it (put the cursor in it) and stop (so it does not set focus to others, only the first).

The elements that would be selected in a template, would look something like:

```
<input type="text" name="[VARIABLE_NAME]" size="37">
<input type="password" name="[VARIABLE_NAME]" size="37">
<textarea name="[VARIABLE_NAME]" rows=3 cols=80></textarea>
```

This way of looping through each element is frequently used so you will see it in many Javascript functions.

The above example has all the key components necessary to use Javascript in a template:

1. The template contains the Javascript function or tells the browser to load a '.js' file that contains it. This is between <script> tags in the template.
2. The web page calls a Javascript function at the appropriate time. The example above calls the function when the web page is loaded.

If the Last Value is Selected, Clear All Others

For some questions, it is desirable to let the user select more than one value from a list, but one of the values is special and should not be selected with any of the other values. For example, a question might ask:

What is in your briefcase?

1. Report
2. Computer
3. Magazine
4. Candy
5. Pen
6. None of the above

The user can select any combination of the first 5 items, but the 6th has a special meaning, and should not be selected with any of the first 5. Radio buttons would force a single item to be selected and there may be more than one item to select. Checkboxes would allow any combination of values to be selected, but would not treat the last in a special way. Javascript allows using checkboxes, but makes the last one special.

The 'None of the Above' answer must be the last answer for the question. The Javascript in the sample system will:

- Uncheckmark any other checks if the "None of the above" is checked
- Uncheck "None of the above" if it is checked and any other checkbox is selected

The template (.html) has this call to the Javascript for the checkbox:

```
<input type="checkbox" value="VARIABLE_VALUE_NUM" name="[VARIABLE_NAME]"
onClick="LastClearsOthers(this)">
```

When the user checkmarks any checkbox, it calls the function 'LastClearsOthers' and passes the <input> element that the user checkmarked. If **input_user_checkmarked** was the last one, it clears the others. If **input_user_checkmarked** was not the last one, it clears the last one. The Javascript is:

```
function LastClearsOthers(input_user_checkmarked) {
   var last_is_checked = false;
   var last_checkbox_index = -1;
       //Is overwritten so is -1 only when     processing the last checkbox.
   var form = document.CorvidForm;
   var uncheckmark_others = false;
   var return_variable = input_user_checkmarked.name;

   //for each element in the FORM, scanning in reverse order
   for (i=form.elements.length - 1; i >= 0; i--) {
      var elem = form.elements[i];
      var type = elem.type.toLowerCase();

      //if element is a checkbox that sets the same return variable
      if (type == "checkbox" && elem.name == return_variable) {
          //if this is the last checkbox
          if (last_checkbox_index == -1) {
             last_is_checked = elem.checked;
             last_checkbox_index = i;
             if (!last_is_checked)
                break;
               // if the checkbox that was just clicked on is the last checkbox,
               // set flag to uncheckmark others.
             if (elem == input_user_checkmarked)
                //if same object (not just equal object)
                uncheckmark_others = true;
                //else some other was clicked on so uncheckmark last and done
             else {
                elem.checked = false;
                break;
              }
          } else
            //else not last checkbox and last was clicked on so
            // uncheckmark this one
                elem.checked = false;
          }
      }
}
```

This has a similar loop as the previous example except it loops through the elements in reverse order, starting with the last. It ignores all elements except <input> tags that are of the "checkbox" type and those that return a value for the same variable as **input_user_checkmarked**. The first <input> tag to be processed is the last in the form so if it is **input_user_checkmarked**, then the user clicked on the last so set the flag 'uncheckmark_others', else uncheckmark it. The only thing left to do is loop through all the others and uncheckmark them if the flag was set.

Verify that the User has Answered All Questions

If a user does not answer one of the questions from Corvid that provides information needed in the system, Corvid will reask the question. However, this means sending the form data back to Corvid to rebuild the question screen and sending it back to the user. Also, if there are multiple questions on a screen, Corvid may not need to reask the unanswered question immediately, and may have to build a special screen for it. Javascript provides a way to force the user to answer all the questions on the screen before the data is sent back to Corvid, and to remind them which questions have not been answered. This is all done in the system user's browser, so there is less of a load on the server, and faster processing of input.

The template demo system verifies all questions were answered when the user clicks on the OK button. The <input> tag has:

```
      onClick="return verifyFormFilled(this);"
```

which calls the function 'verifyFormFilled' passing it the <input> object. If the function returns True, the submit occurs but if the function returns False, the submit is aborted. This example examines each element in the form to determine if it was answered. If not, it displays a message informing the user that they did not answer a question. It marks the unanswered question by making it yellow.

```
    function verifyFormFilled(button) {
        var form = button.form;
           // For each element (INPUT tag) in the FORM,
           // report an error if it is TEXT and
           // empty or radio/checkbox and none checkmarked.
           // for each element in the FORM
        for (i=0; i < form.elements.length; i++) {
           var elem = form.elements[i];
           var type = elem.type.toLowerCase();
           if ((type == "text" || type == "password" || type == "textarea")
               && (elem.value == "" || elem.value == " ")) {
                     //set focus to (put the cursor in) the edit box
                 elem.focus();
                     //set background of edit box to yellow
                 elem.style.backgroundColor = "#feff6f";
                     //report the error
```

```
                alert("You must type an answer in the yellow edit box
                containing the cursor.");
                    //abort the submit process
                return false;
         } else if ((type == "radio" || type == "checkbox" ) &&
              elem.checked == false) {
                //search for any input with same name that is checked
                count = 0;
                for (j=0; j < form.elements.length; j++) {
                     if (form.elements[j].name == elem.name &&
                        form.elements[j].checked)
                            count++;
                }
                if (count == 0) {
                     alert("You must select an answer for " + elem.name);
                     for (j=0; j < form.elements.length; j++) {
                        if (form.elements[j].name == elem.name)
                            form.elements[j].style.backgroundColor = "yellow";
                      }
                     //set focus to (put a dotted box around) the radio button
                     elem.focus();
                     return false;
               }
       } else if (type.indexOf("select") == 0) { //if starts with "select"
              var indx = elem.selectedIndex;
              if (indx < 0) { //if none selected
                 elem.style.backgroundColor = "#feff6f";
                 alert("You must select from the answers marked in yellow.");
                 return false;
              }
              //if the option text of the selected option is empty
              // or space, report error.
              var option_text = elem.options[indx].text;
              //the text between the <option> tags
              if (option_text == "" || option_text == " ") {
                 elem.style.backgroundColor = "#feff6f";
                 alert("You must select from the answers marked in yellow.");
                 return false;
              }
       }
   }
      //If got here, then form data is ok so submit the data by returning TRUE.
   return true;
}
```

For each element (<input>, <textarea>, or <select>) of the <form>, if it was not answered, display a message and color it yellow and stop. The complication is each element type has to be handled differently. Unanswered edit boxes have the background color of the edit box changed to yellow and the cursor is placed inside the edit box. Unanswered radio buttons and checkboxes have a yellow box drawn around the radio button or checkbox. (Some browsers such as Firefox do not support this.) But to determine that the radio button is unanswered, it has to examine all radio buttons that set that same variable. Unanswered select lists have their background color changed to yellow. Yellow was chosen because most web pages do not use it so it will stand out. To change the color, change "#feff6f" to some other color.

The above Javascript has to either be in the web page between <script> ... </script> tags or in an external file that is referenced inside the <script> tags, like this:

```
<script language="JavaScript"

src="AnswersRequiredTemplate.js"></script>
```

The 'AnswersRequiredTemplate.html' file has that <script> tag at the bottom of the page to speed up the display of the web page.

Pop-up Helper Page in Frame

During the run, the user may need to be shown helper pages (pages that explain the question being asked or the results being displayed). These helper pages appear because the expert system displayed them, not because the user requested them (which can be accomplished using <A> tags). The key to having the help page displayed in the frame of a frameset is to use the 'onLoad' attribute of the body tag, like this:

```
<HTML>
<head>
      <title>Budget for the project</title>
</head>
<body onLoad='window.open("helper.html", "right_side")'>
```

The 'onLoad' executes when the web page is loaded, so it immediately displays the 'helper.html' page in the frameset named 'right_side'. This is good for a template that is unique to a single question. To make it work in a generic template, you want the file name and frame name to be stored with the variable and embedded as the parameters. Use the CORVID_REPLACE_# under the 'Servlet' tab in the 'Variables' window to store the names of the file and frame. Put the replacement keywords into the template, like this:

```
<HTML>
<head>
   <title> Budget for the project </title>
   <base href="CORVID_LINK_BASE">
</head>
<body onLoad='window.open("CORVID_REPLACE_1", "CORVID_REPLACE_2")'>
```

Whenever the variable uses this template, CORVID_REPLACE_1 must be the '.html' file name and CORVID_REPLACE_2 must be the frame name. If there is no frameset or you want it displayed in a new browser window, then set CORVID_REPLACE_2 to "_blank". Since IE does not honor the <base> in Javascript, the '.html' file must have a full path. You can easily and generically do so by using CORVID_LINK_BASE, like this:

```
CORVID_LINK_BASEfilename.html
```

Notice you do not put a "/" between them because the value of CORVID_LINK_BASE must already end with "/".

Since the function "open" is built into Javascript, it does not need to be loaded or in <script> tags. You do not have to use the same # in CORVID_REPLACE_# as 'QuestionWithPopupTemplate.html' used. You can use any # as long as you remain consistent.

Using the Demo System Files

- Edit the '.html' and '.js' files with a plain text editor, such as Notepad.
- Remember that the user can turn Javascript off in their browser. This could cause, for example, the 'None of Above' to not automatically uncheckmark the other values. If this will lead to incorrect results in the system, logic should be added in the Corvid system to detect this combination, even though the Javascript should block it.