Session 5: HTML Dialogs

At the end of this session participants will be able to:

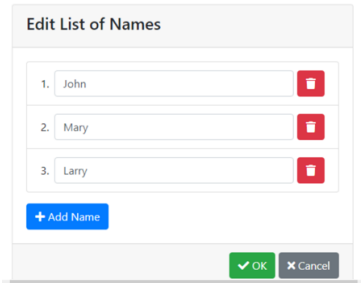
- Launch an HTML dialog from CSPro
- Pass data to an HTML dialog from CSPro logic
- Return data to CSPro logic from an HTML dialog
- Call CSPro functions from Javascript using ActionInvoker

HTML Dialogs

There are situations when the user interface options that CSPro provides are not sufficient. In these cases, we can create custom user interactions using standard web technologies: HTML and Javascript. Your CSPro program can then display the user interfaces you build in HTML/Javascript to replace or supplement the standard CSPro user interface.

As an example, lets redo the way the interviewer lists the household members. Currently, the interviewer fills out the entire row of the person roster for the first household member before recording the name of the second household member. It would be better to list just the names of all the household members first, and then fill in the rest of the roster for each member in turn. We could do that by splitting the roster into two with a first roster that has just the names. However, that would still mean entering one name at a time, each one on a separate screen on mobile devices. It would be easier for the interviewer to see the entire list of names on the screen at the same time.

We can do this using a custom interface we build in HTML/Javascript. We can use the web page name_list.html included in the materials for this session on the website http://teleyah.com/cspro/SriLankaNov2024/name_list.html.[1]



[1] This page was created using ChatGPT with the following prompt: *Create web page to let the user edit a list of names. The page should show the names with line numbers like: 1. John Smith 2. George Brown ... The names should be editable but the line numbers are not. The initial list of names should be configurable by providing a config object in the Javascript. If the initial list of names in the config is empty, the page should have five empty names. There should be a button to add a new empty name to the end of the list. There should be a delete button next to each name that will remove it from the list. Add an "OK" button that prints the list of names to the console filtering out the blank ones. Add a cancel button. Use bootstrap for styling and make it look slick. Use icons instead of text for all the buttons.*

To use this file in our CSPro application we need to copy it into the application directory. We can create a new Dialogs directory and put it there. Note that we need to make sure to deploy the html file along with the rest of the application to make sure it gets copied onto the mobile device.

The HTML for the name list editor contains link and script references that are hosted online. These will work correctly when we are connected to the internet but if we are offline, the page will not render correctly because it will not be able to load the necessary the styles and or Javascript libraries. Fortunately, CSPro includes offline versions of several popular libraries including Bootstrap CSS and jQuery. We just need to modify the references to point to the files in the CSPro installation on the tablet or computer.

```
<link rel="stylesheet" href="/external/bootstrap/bootstrap.min.css" />
<link rel="stylesheet" href="/external/bootstrap/bootstrap-icons.css">
<script src="/external/bootstrap/bootstrap.bundle.min.js"></script>
```

Now we need to show this web page in our CSPro application. We should display it just before the household roster. Create a new dictionary item that we can add to the form to use a placeholder for showing the HTML page. Let's call it HOUSEHOLD_MEMBER_LIST. We can add it to the INTERVIEW_RECORD and drop it onto the form just before the roster on the section B form. We can use this field to both display the web page and to confirm the list. For the confirmation, create a value set with two options: 1) Edit and 2) Continue. If the user chooses Edit we will display the web page to allow them edit the list, otherwise we move on to the next field. This avoids having the web page pop up every time the user enters the field.

We can display a simple read-only version of the list in the question text for this field, so the interviewer knows what they are confirming. Define a function to create the read-only view of the list that we can use as a fill in the question text.

```
function string householdMembersList()
    string listHtml= "<ol>";
    do numeric i = 1 while i <= totocc(PERSON_REC)
        listHtml = listHtml +
            maketext("<li>%s</li>", NAME(i));
    enddo;
    listHtml = listHtml + "</ol>";
    householdMembersList = listHtml;
end;
```

The logic for the field will be:

```
PROC HOUSEHOLD_MEMBER_LIST
onfocus

if $ = 1 then
    // Edit
    editHouseholdMembers();

    // Set to continue now that we have done the initial
    // listing.
    $ = 2;
else
    // Continue to next field
endif;
```

The editHouseholdMembers() function will show the HTML page as a modal dialog using the `htmldialog` function. It takes the path to the HTML file to display.

```
function editHouseholdMembers()
    htmldialog("../Dialogs/household_members_dialog.html");
end;
```

When we test it, we find that the dialog is displayed but there is no way to dismiss it, so we are stuck unless we use the task manager to kill CSEntry. We should close the dialog when the interviewer taps on either the OK or Cancel buttons. To do this, we need to modify the Javascript to call a method on the action invoker, a special object that CSPro exposes to Javascript. First, we need to include the action invoker script in our web page:

```
<script src="/action-invoker.js"></script>
```

Then we create an instance of the ActionInvoker:

```
const CS = new CSProActionInvoker();
```

Finally, inside the handler for the button we call the closeDialog method on the action invoker.

```
document.getElementById("okBtn").addEventListener("click", function () {
    CS.UI.closeDialog();
});

document.getElementById("cancelBtn").addEventListener("click", function () {
    CS.UI.closeDialog();
});
```

Now the dialog is dismissed when the user taps on the close or cancel buttons.

Currently our dialog is a bit small, and we must scroll to see the buttons. We can pass display options to the `htmldialog` function to make it bigger. The display options should be a JSON string with width and height parameters:

```
{"width": 640, "height": 480}
```

If we want to make the dialog fill the whole screen we can use the MaxDisplayWidth and MaxDisplayHeight properties for the width and height:

```
function editHouseholdMembers()
    string displayOptions = maketext('{ "width": %d, "height": %d}',
                                     tonumber(getproperty("MaxDisplayWidth")),
                                     tonumber(getproperty("MaxDisplayHeight")));

    htmldialog(
        "../Dialogs/household_members_dialog.html",
        displayOptions := displayOptions
    );
end;
```

Returning Data from HTML Dialogs to CSPro

Next, we need to get the list of names from the dialog and copy them into the household roster. We can send data back from Javascript to CSPro by adding it as a parameter to the closeDialog() method of the action invoker. In the Javascript, ee modify the handler of the OK button to pass the list of names to the closeDialog() call.

```
        document.getElementById("okBtn").addEventListener("click", function () {
            const finalNames = Array.from(nameList.querySelectorAll(".name-input"))
                .map(input => input.value.trim())
                .filter(name => name !== "");
            CS.UI.closeDialog({result: finalNames});
        });
```

Back in the CSPro application, the value of result (finalNames in this case) will be returned from the call to `htmldialog`. If we print out the return value, we can see that it is a string that contains a JSON list of the names from the HTML dialog. We need to parse this list in CSPro to copy the names into the name column of the household roster. We can use the method `updateValueFromJSON` to do this. It is a method on the list object (as well as many other CSPro logic objects like array and string). Declare a CSPro list called names and then call `updateValueFromJSON` and pass it the JSON returned from the call to `htmldialog.` That converts from a JSON string to a list of strings. Now we can iterate over the list and copy the list elements into the NAME column of the roster.

```
    string resultJson = htmldialog(
        "../Dialogs/household_members_dialog.html",
        displayOptions := displayOptions
    );
    if resultJson <> "" then
        list string names;
        names.updateValueFromJSON(resultJson);
        do numeric i = 1 while i <= names.length()
            NAME(i) = names(i);
        enddo;
    endif;
```

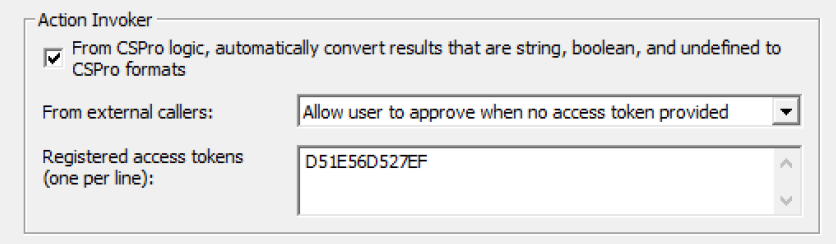
Once some of the household members have been entered, we should really do something smarter than just copying from the result into the household roster. We should check for names that were removed and delete the corresponding rows in the roster. That is left as an exercise for the reader.

When we run this and click on the OK button, we get a security warning that "a web page or program is trying to access CSPro functionality". To avoid this, we need to use an access token when creating the action invoker. An access token can be any string as long you register it in your data entry application in the logic settings under application properties and you use the *same string* when creating the action invoker in your Javascript.

In our HTML page, when creating the action invoker, add the access token to the constructor call.

```
const CS = new CSProActionInvoker("D51E56D527EF");
```

And in the logic settings, add the access tokens in the "registered access tokens" box.



Having to rerun the data entry application every time we make changes to our HTML can be frustrating. A faster option is to edit our HTML in a new CSPro tool called CSCode. When you open your file in CSCode, set the language to "CSPro HTML Dialog". Now when you hit the run button on the toolbar, you will see an additional window on the bottom that lets you set the inputs to the dialog, the same arguments you would pass to the `htmldialog` function from your CSPro logic. On the left side there is an output window that shows the result that was passed to closeDialog when the dialog is closed.

Sending Data from CSPro to an HTML Dialog

If the interviewer has already entered some names and comes back to the dialog, we should show the existing names rather than starting with a blank dialog. We need to pass the current list of names from the roster to the Javascript code in our HTML page. We do this using the inputData parameter of the `htmldialog` function. The input data must a be string containing JSON. In our case, we can pass it a JSON list of names like:

```
["John", "Mary", "Larry"]
```

We could construct the list using `maketext` in a loop and build up a string but an easier way is to use `getValueJson` which generates the JSON representation of a CSPro object like a list, array or string. We can use a loop to build a list of names and then call `getValueJson` on the list.

```
list string names;
do numeric i = 1 while i <= count(PERSON_REC)
    names.add(NAME(i));
enddo;

string inputData = names.getValueJson();
string resultJson = htmldialog(
    "../Dialogs/household_members_dialog.html",
    displayOptions := displayOptions,
    inputData := inputData
);
```

Then in our HTML dialog we need to read the inputData and use it to populate the list of names. We do this with the method getInputData() on the action invoker. We then set the input data as the initial names list in the config which will be used to fill in the in the names in the dialog.

```
document.addEventListener("DOMContentLoaded", function () {
    const CS = new CSProActionInvoker("D51E56D527EF");
    const input = CS.UI.getInputData();
    const config = {
        initialNames: input
    };
```

Group Exercise

In section D04 of the household application we ask for the top three reasons for food insecurity in order of importance. Currently this is implemented using check boxes which doesn't convey the order well. A better input control for this would be one that explicitly shows the first second and third choices. In the course materials there is top 3 ranking HTML page that has the HTML and Javascript to do that. You can find it at http://teleyah.com/cspro/SriLankaNov2024/top_n_ranked.html. Use this instead of check boxes in question D04. This will require you to:

1) Copy the top_n_ranked.html into the Dialogs folder of the CSPro application
2) Update the Javascript code to include the action invoker and to use the local versions of bootstrap to support offline use.
3) Update the itemList in the config object in the Javascript code to use the causes from the questionnaire instead "item 1", "item 2"…
4) Update the topList in the config to use the previous choice. Read the previous choice from the inputData via the action invoker and set the topList in the config to the inputData. You can test this using CSCode.
5) Update the Javascript to call closeDialog via the action invoker when the user clicks the OK button. In the event listener for the OK button, the variable topItems will contain the list of items that were selected. Pass topItems to the call to closeDialog to return it back to CSPro. You can test this using CSCode.
6) In the CSPro application change the type of the variable for D04, CAUSES_OF_FOOD_INSECURITY from an alpha to a single digit number with three occurrences (one for each cause). We no longer need it to be alpha because we will not use checkboxes.
7) Update the CSPro application to use the HTML dialog. Like we did for the names list, add a new placeholder variable with options Edit and Continue in the value set and drop it on the form in place of CAUSES_OF_FOOD_INSECURITY and remove CAUSES_OF_FOOD_INSECURITY from the form. In the placeholder field, show the existing top 3 choices in the question text so the interviewer can see their previous choices. If they choose to Edit, show the HTML dialog and pass it the current top 3 choices in the inputData.
8) In the CSPro application get the result from the call to htmldialog and use it to set the choices in CAUSES_OF_FOOD_INSECURITY so that the result is saved.