Server actions recipe

Some actions such as data merging or complex document manipulation are not possible to carry out on the client. Consequently these kind of actions require the document data to be manipulated on the server. Regardless of what functionality is required, the steps required to go from editing a document on the client, carrying out the action on the server and returning the result to the client are identical in every case and are as follows:

1. Save the data (client)

If the user has modified the document, the document must be saved before contacting the server.

2. Handle the save data async completion event (client)

Saving data is an asynchronous action and we must wait for the async completion event before we can call the server. The async completion event must be added to the OWrite's $event method.

3. Perform the actions (server)

Implement the server executed method which carries out the action and returns a result to a client executed *return* method.

GENERAL NOTE: Server methods can return results to a client's *return* method by executing the *Quit method* command passing it the result data.

4. Deal with the result (client)

Implement the client executed *return* method which deals with the results of the server action.

As an example, this technical note will demonstrate how to test if a user has entered valid data in a set of table cells. It is assumed that the previously prepared document contains a table with the cells named "Title", "FirstName", "LastName" and "EMail". This technical note will **not** show how to create a document with the specific cells. Please refer to technical note TN0027 for help regarding inserting tables on the client. Alternatively, the fat-client Document Manager example can be used to prepare such a document.

1. Save the data (client)

To save the data, we could add a client executed remote form class method called $clientCheckCells() which will save the data.

Saving data is an asynchronous action, thus the method cannot do anything further after the call to $savedata(). JS-OWrite will trigger an async completion event once all the data has been saved. The async completion event will pass through the async ID and async data that was provided when calling $savedata().

GENERAL NOTE: JS-OWrite implements a number of methods that are asynchronous. These methods all have the additional async parameters so further actions can be performed when the async completion event is received.

Sample Code 1: (for testing all cells)

```
# save data with async ID "CHECK_CELLS" and async data "ALL"
Do $cinst.$objs.jsOWrite.$savedata("CHECK_CELLS","ALL")
```

Sample Code 2: (for testing the currently selected cell or all if there is no current cell)

```
# save data with async ID "CHECK_CELLS" and async data set to the name
                                                    of the currently selected cell
If $cinst.$objs.jsOWrite.$curobjtype = kWriObjTypeTableCell
  Calculate objName as $cinst.$objs.jsOWrite.$curobjname
Else
  Calculate objName as "ALL"
End if
Do $cinst.$objs.jsOWrite.$savedata("CHECK_CELLS",objName)
```

2. Handle the save data async completion event (client)

In the JS-OWrite $event() method, implement the evAsyncDone event and handle the case for "CHECK_CELLS". When the event is received, all data has been saved and we can call the server method. Because server communications are also executed asynchronously, we must wait for the server response before we can handle the outcome. This is explained in section 4.

IMPORTANT NOTE: The OWrite's $event() method must be set to execute on the client, which we recommend doing in all implementations.

Sample Code:

```
On evAsyncDone
   Switch (pAsyncId)
     Case "CHECK_CELLS"
       # call server method with the additional async data
       Do $cinst.$serverCheckCells(pAsyncData)
   End switch
```

3. Perform the actions (server)

Implement the server method $serverCheckCells(pWhich), which will build and iterate a list of cell objects, extracting cell content from the document. If it finds a cell that is empty, the object's selection range and object name is added to a result error list which will be returned to the client.

Sample Code:

```
# prepare our result list
Do resultList.$define(objName,objSelStartString,objSelEndString)
# load the data into a non-visual OWrite server object
Do objOWrite.$loaddata(ivDocumentData)
# define and build list of table cells
#    objName is a local variable of type character and
#    objType,objIdent,objFstSel and objLstSel are 32bit integers
Do objList.$define(objType,objIdent,objFstSel,objLstSel,objName)
Do objOWrite.$getobjslist(objList,kWriObjTypeTableCell)
# iterate the list and check each cell's name against the specified
#    cell name or a list of cell names if the client specified "ALL"
Calculate cellsToTest as "#Title#FirstName#LastName#EMail#"
For objList.$line from 1 to objList.$linecount
  If (pWhich="ALL")
    # we are to test all cells in cellsToTest
    Calculate objName as con('#',objList.objName,'#')
    Calculate doTest as pos(objName,cellsToTest)>0
  Else
    # a specific cell was specified by client
    Calculate doTest as (pWhich=objList.objName)
  End if
  If (doTest)
    # select the content of the cell and save it to a local variable
    Do objOWrite.$setselection(objList.objIdent,kWriSelectStart,kWriSelectEnd)
    Do objOWrite.$savedata(objContent,kWriFmtText,kFalse,kWriSaveSelection,
                                                                    kTrue)
    # now we would test the validity of the cell content
    #    in our example we only test for empty
    If not(len(objContent))
      # if the cell is empty we add it to our result list
      #    first fetch a selection range suitable for the client
      Do objOWrite.$getselection(objSelStartString,objSelEndString,objIdent,
                                                          kWriSRJSDefault)
      Do resultList.$add(objName,objSelStartString,objSelEndString)
    End if
  End if
End for
#  return the result list to the client
Quit method resultList
```

4. Deal with the result (client)

Implement the *return* method $serverCheckCells_return(pResultList). A client *return* method, which must always be set to execute on the client, has the same name as the server method with the added postfix "_return". JS-Omnis will pass the data returned by the server method as the first parameter to the *return* method. What name you give the parameter does not matter.

Sample Code:

```
# see if our result list contains any errors ...
If pResultList.$linecount
  # ... if it does, build the message text,
  Calculate message as "The fields"
  For pResultList.$line from 1 to pResultList.$linecount
    Calculate message as con(message,' ',pResultList.objName)
  End for
  Calculate message as con(message," must be provided")
  # display the message,
  OK message [message]
  # and set the selection to the first empty cell
  Do $cinst.$objs.jsOWrite.$setselection
                    (pResultList.objSelStartString,pResultList.objSelEndString)
End if
```

Document History

02 July 2019: first publication