Displaying message boxes in a Panther Web Application

When migrating Panther application code to Panther Web, a block of code that must request information from the user before continuing, such as JPL code that calls sm_message_box(), cannot be executed by the Jserver as-is. One migration technique is to split the block of code into separate pieces around the call to sm_message_box(), such that each piece can be executed by the Jserver in separate AJAX requests. Upon request, Prolifics provides sample code to facilitate use of this technique.

Prolifics also provides sample code for a WebSocket technique that may be used to solve the same sort of migration issue. An advantage of the AJAX technique is that the JServer does not wait for a response. It remains stateless and reusable for other requests. The response can be processed by any JServer that is free when the response comes back. However, it may require some work to split up JPL procedures, particularly when many message boxes appear within individual procedures. In those cases, the WebSocket technique may be preferable.

Using the sample code provided by Prolifics for the AJAX technique, calls to sm_message_box() should be replaced with calls to the sample JPL function, sm_web_message_box(). If the message to be used comes from the message file, an alternate form of this function, sm_web_msgno_box(), may be used instead.

Using a Message file is optional as it helps to have all messages in one place and making it easier when updates are required; especially for larger applications where messages are used repeatedly. You can pass the message string directly to the function. The function definition for proc sm_web_message_box() is like below.

```
proc  sm_web_message_box(text, title, options, icon)
{
        vars html_option ='<div hidden class="message_box_options">' ## options ## '</div>'
        vars html_ico = '<div hidden class="message_box_icon">' ## icon ## '</div>'
        vars msgId =  '<div hidden class="message_id">' ## ntext## '</div>'
        vars ttl =  '<div hidden class="message_box_title">' ## title## '</div>'
        title = html_option ## html_ico ## msgId ## ttl

        call sm_message_box(text, title, options, icon)
}
```

And can be called as

```
call sm_web_message_box(“Do you want to save changes?”, "Confirm",
SM_MB_YESNOCANCEL,SM_MB_ICONQUESTION)
```

In either case, a call to the sample function should be the last statement of a new JPL procedure that is split off from the original one, and the code that processes the users’ response should be at the start of another new JPl procedure.

The message produced by the first procedure is returned in HTML that is sent back to the browser,

where it is processed by JavaScript sample code into a JQuery dialog box. An AJAX request is then made to send the response (YES/NO, for example) from the dialog box to the JServer. The response is received in hidden fields in the web_enter() function, which, in turn, invokes the next JPL procedure that was split off from the original. Logic that handles the response will then execute there.

The instructions below illustrate how to handle sm_message_box type alerts in a Panther Application.

Create & Update Panther Message File

1. Create a custom message file to include all possible messages
   Example: you can create a message file named "appmsg" having messages as below
   APP_DUMMY = This is a dummy message.
   APP_NULLVAL = Please enter username and password.
   APP_INVALID = Invalid username or password. Try again!
   APP_WELCOME = Welcome to this application.
   APP_SAVE = Your updates are saved.
   APP_BEFORE_SAVE= Do you want to save changes?

2. Convert the asci version of your custom msgfile to binary using %SMBASE%\util\msg2bin.exe, run the below command
   msg2bin appmsg , This will generate appmsg.bin

3. Add global variables for the messages in your msgfile using %SMBASE%\util\msg2bin.exe, run command as below

   msg2hdr –jvpf appmsg , This will create appmsg.jpl, like below

   ```
   global APP_DUMMY(1) = 0    /* This is a dummy message.              */
   global APP_NULLVAL(1) = 1  /* Please enter username and password.   */
   global APP_INVALID(1) = 2  /* Invalid username or password. Try again! */
   global APP_WELCOME(1)      = 3  /* Welcome to GETFLIX.               */
   global APP_SAVE(1)     = 4  /* Your updates are saved.              */
   global APP_BEFORE_SAVE(1)  = 5  /* Do you want to save changes?     */
   ```

4. Load the custom msgfile and jpl file when your application initializes. Add the below to your web_startup()

   ```
   proc web_startup()
   {
           call sm_msgread("", 0, MSG_FILENAME, "appmsg.bin")
           public appmsg.jpl
   }
   ```

   Make sure these files are accessible to your application i.e. have them in Application directory or SMPATH

Invoke sm_web_msgno_box()

5. Call the below function where you want to show the message

   call
   sm_web_msgno_box(APP_SAVE,"Confirm",SM_MB_YESNOCANCEL,SM_MB_ICONQUESTION)

   Make sure this function call is the last command that is executed in your JPL procedure because commands following this function will not execute.

   Example
   ```
   proc view()
   {
           dbms sql select  cust_id , state  from customer  where state =  :+st
           call sm_web_msgno_box(APP_SAVE, "Confirm",  \
           SM_MB_YESNOCANCEL,SM_MB_ICONQUESTION)
   }
   ```

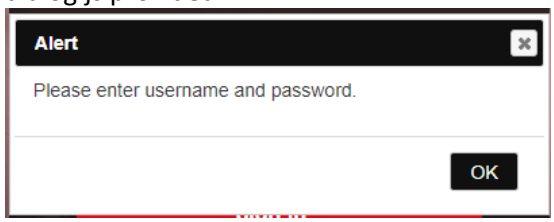
   Add the procedure to your jpl module
   ```
   proc  sm_web_msgno_box(ntext, title, options, icon)
   {
           vars html_option =   '<div hidden class="message_box_options">' ## options ## '</div>'
           vars html_ico = '<div hidden class="message_box_icon">' ## icon ## '</div>'
           vars msgId =  '<div hidden class="message_id">' ## ntext## '</div>'
           vars ttl =  '<div hidden class="message_box_title">' ## title## '</div>'
           title = html_option ## html_ico ## msgId ## ttl
           vars text = @app()->message[ntext]
           call sm_message_box(text, title, options, icon)
   }
   ```

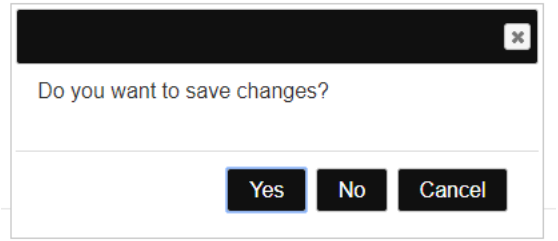
The above function call (sm_message_box()) will generate a <h2 >&<h3> in html source for that screen

Example:
```
<div class="sm_message_text">
    <h2>
        <b>
            <div hidden class="message_box_options">3</div>
            <div hidden class="message_box_icon"></div>
            <div hidden class="message_id">4</div>
            <div hidden class="message_box_title"> Confirm </div>:
        </b>
    </h2>
    <h3>Your updates are saved. <br/></h3>
    <hr>
</div>
```

Invoke a JQuery Dialog box

6. Now use a JQuery Dialog box to display this message as below. Have a look to the sample dialog.js provided.






```
//SM_MB_YESNOCANCEL
$("#msg-box-3").dialog({
   autoOpen  : false,
   modal: true,
   draggable: false,
   resizable: false,
   show: 'blind',
   hide: 'blind',
   width: 400,
   dialogClass: 'ui-dialog-osx',
   classes: {
      "ui-dialog": "custom"
   },
   buttons: {
         "Yes": function() {
           doSomething("yes")
               $(this).dialog("close");
         } ,
         "No": function() {
           doSomething("no")
            $(this).dialog("close");
      } ,
      "Cancel": function() {
              $(this).dialog("close");
   }
   }
});
var opt = $(".message_box_options").text();
var icon = $(".message_box_icon").text();
```

```
var title = $(".message_box_title").text() ;
if(opt==3){
        var msg = $(".sm_message_text h3:first").text();

        $('#msg-box-3').html('<p>'+msg+'</p>');
        $("#msg-box-3").dialog('option', 'title', title);
        $("#msg-box-3").dialog("open");
}
```

This can be done for different types of message boxes based on OPTION value i.e. SM_MB_OK, SM_MB_YESNO, SM_MB_YESNOCANCLE etc.

To make the dialog box work, please add these headers to your HTML Template

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/jqueryui/1.12.1/jquery-ui.css">
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script src="https://code.jquery.com/ui/1.12.1/jquery-ui.js"></script>
<script src="/js/dialogs.js"></script>
```

Also put with the ids as below

```
<div id="msg-box-0"></div>
<div id="msg-box-3"></div>
<div id="msg-box-4"></div>
```

7. For YES/NO type of message boxes the response can be sent back to Jserver using AJAX calls by way of hidden fields such as **“event”** , **“msg_id”** in your screen

Example

```
function doSomething (event) {
        var msgId = $(".message_id").text();
        callService (event,msgId,serviceCallback);
}

function callService(reply,msgId,callback) {
var serviceScreen = 'screen1';// name of screen to be posted
var webid  = $('input[name="'+"__server_data__"+'"]').val();
var formdata = {
            __server_data__:webid,
            __posted_screen_name__ : serviceScreen,
            __end_of_jpl_vars__    : 0
        };
formdata[‘n_event’]=reply;
formdata[‘n_msg_id’]=msgId;

var saveData = $.ajax({
                                        type: 'POST',
                                        url: serviceScreen,
                                        data: formdata,
                                        success: callback
});
}

function serviceCallback(resultData) {
$('html').html(resultData); //replace the response in the current html document
}
```

Handle the response in the web_enter()

8. The response can be handled in proc web_enter of that screen as below

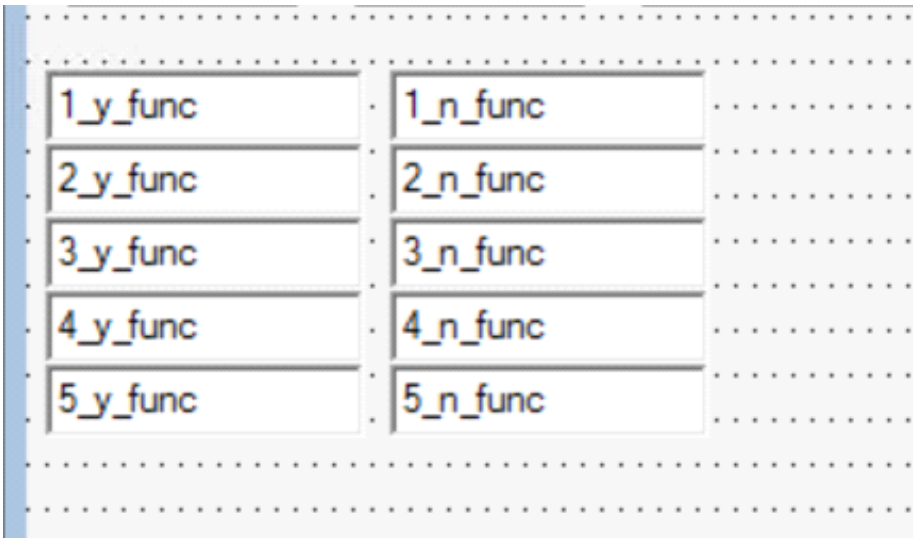
```
proc web_enter
{
if (event=="yes")
      call update()   //your logic
 if (event=="no")
     call noUpdate()//your logic
 }
```

If there are multiple YES/NO messages in one screen then an array of msg_id, and callback functions can be used to differentiate responses for different messages.
Example

```
proc web_enter
{
    if (event=='yes')
        call :+y_callback[msg_id]
   if (event=='no')
     call :+n_callback[msg_id]

}
```

Here **y_callback** and **n_callback** are arrays of field on your screen. These arrays have the name of proc to be called based on the msg_id when response is “Yes” or “No”.



For more information about sm_web_msgno_box() and its options, look into our Panther documentation.

https://docs.prolifics.com/panther/html/prg_html/libfu231.htm

For details on our WebSocket technique, email support@prolifics.com