# Table of Contents

1 Foreword                                                                                           ................................................................. 5
1.1 Prerequisites                                                                                      ................................................................. 5
1.2 Terminology                                                                                       ................................................................. 5
  Runtime Library                                                                                     ................................................................. 5
  Runtime Library Manager                                                                              ................................................................. 5
  Default Components                                                                                  ................................................................. 5
  Query API                                                                                           ................................................................. 6
1.3 UI Technologies                                                                                  ................................................................. 6

2 Overview of SAP-Delivered Patterns                                                                  ................................................................. 7

3 Patterns for Web Dynpro ABAP                                                                       ................................................................. 8
  3.1 CheckForMessages                                                                                 ................................................................. 8
    Pattern Signature                                                                                  ................................................................. 9
    Pattern Usage                                                                                       ................................................................. 10
    Pattern Code Explanation                                                                            ................................................................. 13

4 Patterns for CRM Web Applications                                                                  ................................................................. 17
  4.1 Pattern – SelectMenuItemByText                                                                  ................................................................. 17
    Pattern Signature                                                                                  ................................................................. 18
    Pattern Usage                                                                                       ................................................................. 18
    Pattern Code Explanation                                                                            ................................................................. 18
    Pattern Limitations                                                                                ................................................................. 22
  4.2 Pattern – SelectTransactionType                                                                  ................................................................. 23
    Pattern Signature                                                                                  ................................................................. 24
    Pattern Usage                                                                                       ................................................................. 24
    Pattern Code Explanation                                                                            ................................................................. 24

5 Patterns for SAP GUI                                                                                ................................................................. 29
  5.1 Pattern – ME21N_ExpandAllSections                                                               ................................................................. 29
    Pattern Signature                                                                                  ................................................................. 30
    Pattern Usage                                                                                       ................................................................. 30
    Pattern Code Explanation                                                                            ................................................................. 32
  5.2 Pattern – ME51N_ExpandAllSections                                                               ................................................................. 34
  5.3 Pattern – WaitForCondition                                                                      ................................................................. 34
1 Foreword

CBTA comes with a set of patterns that the test engineer can reuse when creating custom functions.

This document describes the patterns and explains how to adapt them to your needs.

In the Runtime Library Manager, you can write additional custom functions that the test scripts may need when automating the test of some business scenarios for which the common approach (based on default components) is not sufficient.

Note

The Runtime Library Manager is outside the scope of this document. For more information about how to use it, see CBTA – Runtime Library Manager – CBASE Customization.

1.1 Prerequisites

You have configured CBTA and installed the CBTA test tool on the client.

The following SAP Notes are prerequisites of the activities described in this document.

<table>
<thead>
<tr>
<th>SAP Note</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1778899</td>
<td>CBTA - Collective Note</td>
</tr>
</tbody>
</table>

1.2 Terminology

**Runtime Library**

The VB script coding for a test is sent to the client computer and executed by the VB script interpreter. The *Runtime Library* is a set of VB scripts providing helper classes, functions and procedures to simulate actions that are normally performed by a user.

**Runtime Library Manager**

The Runtime Library Manager is a client-side tool that allows the test engineer to customize the VB script libraries that are used to execute CBTA test scripts.

The tool writes additional custom functions that the test scripts may need when automating the test of some business scenarios for which the common approach (based on default components) is not sufficient.

**Default Components**

Default components perform atomic operations against UI elements. The runtime library comes with default components for all the UI technologies that CBTA supports.
Query API

When writing custom functions for web applications, the test engineer needs access to the HTML content that the application displays in Internet Explorer. This can be done using the query API.

For more information, see the CBTA – Test Automation - Query API.

1.3 UI Technologies

CBTA supports the following UI technologies.

<table>
<thead>
<tr>
<th>UI Technology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAP GUI</td>
<td>UI framework used by SAP applications</td>
</tr>
<tr>
<td>WebCUIF</td>
<td>Layer which generates the content displayed by SAP CRM applications.</td>
</tr>
<tr>
<td>UI5 &amp; FIORI</td>
<td>UI Framework used by SAP UI5 and Fiori apps</td>
</tr>
<tr>
<td>Unified Rendering Light Speed (LS)</td>
<td>Layer which generates the content displayed by SAP applications, such as:</td>
</tr>
<tr>
<td></td>
<td>- Web Dynpro application (ABAP)</td>
</tr>
<tr>
<td></td>
<td>- Web Dynpro applications (JAVA)</td>
</tr>
<tr>
<td></td>
<td>- Web GUI – SAP GUI content displayed in Internet Explorer</td>
</tr>
<tr>
<td>BSP</td>
<td>Applications based on Business Server Pages</td>
</tr>
<tr>
<td>Java Web Dynpro</td>
<td>Layer previously used by Java Web Dynpro applications.</td>
</tr>
<tr>
<td>Plain HTML pages</td>
<td>Content generated using regular HTML tags (with no, or few, scripting capabilities).</td>
</tr>
</tbody>
</table>

For test automation, each UI technology poses specific challenges, most of which the test recorder and test player can handle. However, for use cases in which the business scenario is highly dynamic, the default behavior is not always sufficient, so the test engineer may need to write custom functions.
2 Overview of SAP-Delivered Patterns

The following list shows the use of the patterns that are delivered by default with CBTA. SAP may deliver additional patterns via SAP Notes, depending on customer needs.

<table>
<thead>
<tr>
<th>UI Technology / Pattern Name</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WEB Dynpro ABAP</strong></td>
<td></td>
</tr>
<tr>
<td>CheckForMessages</td>
<td>Searches for application messages in a message container. This function verifies whether the applications report feedback, and checks for errors, warnings, etc.</td>
</tr>
<tr>
<td><strong>CRM Web UI (WebCUIF)</strong></td>
<td></td>
</tr>
<tr>
<td>SelectMenuItemByText</td>
<td>Searches for menu items by their text. This function is necessary in situations in which the ID of the item is not stable because it takes a new value each session, when the Item must be found using its text.</td>
</tr>
<tr>
<td>SelectTransactionType</td>
<td>Searches and selects a transaction type in a modal popup window. Some SAP CRM applications start by prompting the end user for a transaction type. This function performs this action by searching for the transaction type by its text, wherever it is in the list.</td>
</tr>
<tr>
<td><strong>SAP GUI</strong></td>
<td></td>
</tr>
<tr>
<td>ME21N_ExpandAllSections</td>
<td>Restores the initial state of the transaction ME21N/ME51N main screen. The screen consists of collapsible panels. Each panel can be expanded or collapsed, depending on how the transaction ended during the previous session.</td>
</tr>
<tr>
<td>ME51N_ExpandAllSections</td>
<td></td>
</tr>
<tr>
<td>WaitForCondition</td>
<td>Checks for a condition, and thus stops the execution flow temporarily. Some SAP GUI applications start a background job to compute business data. The test script may need to pause and wait for the background job to complete, before continuing.</td>
</tr>
</tbody>
</table>
3 Patterns for Web Dynpro ABAP

3.1 CheckForMessages

Use the CheckForMessage pattern to search for messages that Web Dynpro ABAP applications display in a message area.

Note

The CBTA_WEB_A_GETMESSAGEPARAM default component retrieves message parameters. It is generic; supports all Web UI technologies, but it cannot retrieve Web Dynpro-specific information, such as the message type (error, warning, info, etc.).

Figure 1 illustrates the type of messages that this pattern supports. It shows two messages that are displayed during the creation of a shopping cart.

CBTA cannot determine whether these messages are normal in the current business context, so the test script at runtime ignores them and continues.

These messages may be unexpected, and there are situations in which it you should verify whether the test script can continue. As the test engineer, you can create the custom function to check the content of the message area, from the pattern, in the runtime library manager.
Pattern Signature

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckForMessages</td>
<td>Searches for application messages in a message container. This function verifies whether the applications report feedback, and checks for errors, warnings, etc.</td>
</tr>
</tbody>
</table>

**INPUT PARAMETERS:**
- MessageContainerUri ➔ URI of the message container (optional)
- MessageType ➔ The type of the messages to search for
- MessageRule ➔ The rule to apply when checking the content of the message container
- MessagePattern ➔ Not used – reserved for future use
- Options ➔ Options for capturing screenshots

**OUTPUT PARAMETER:**
- Output ➔ Number of messages matching the criteria

**MessageContainerUri**

- URI of the HTML container displaying application messages. If empty, the function searches for messages in the message area displayed by the main document of the main window.

**DEFAULT VALUE:**

```
ls.rid=WDR_MESSAGE_AREA.MESSAGE_AREA.ROOTUIELEMENTCONTAINER
```

**Message Type**

The MessageType parameter can take the following values:

- E ➔ Errors
- W ➔ Warnings
- I ➔ Information

**Message Rule**

The MessageRule parameter can take the following values:

- F ➔ Forbidden (default)
- R ➔ Required
- A ➔ Allowed

**Options**

The option parameter can take the following values:

- /c ➔ Capture a screenshot
- /e ➔ Capture a screenshot of the message area only
**Pattern Usage**

Use the CheckForMessages pattern to check whether application errors are being displayed, and report any test execution failure.

With our shopping card scenario, the input parameters could be as below:

<table>
<thead>
<tr>
<th>Input Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageContainerURI</td>
<td>ls.rid=WDR_MESSAGE_AREA.MESSAGE_AREA.ROOTUIELEMENTCONTAINER</td>
</tr>
<tr>
<td>MessageType</td>
<td>E</td>
</tr>
<tr>
<td>MessageRule</td>
<td>F</td>
</tr>
<tr>
<td>MessagePattern</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td>/c /e</td>
</tr>
</tbody>
</table>

**Custom Function Creation**

Assume that the custom function has been created by the runtime library manager, and the corresponding library has been generated:

- Custom\CheckForMessages.vbs

Figure 2 shows what the runtime library manager then displays.

![Customization](image1)

**Custom Function Usage**

The custom function can be used by a CBTA test script, but it is not visible by default. Call it by using one of the following components:

- CBTA_WEB_A_INVOKE_FUNCTION
- CBTA_WEB_A_EXECUTE_STATEMENT

In the following example (Figure 3), the CBTA_WEB_A_INVOKE_FUNCTION calls the custom code and checks for application errors.
When using CBTA_WEB_A_INVOKE_FUNCTION, the test engineer must specify the name of the custom library, the name of the custom function and the values of its input parameters.

In this example (Figure 4) we have:

- **LibraryName** ➔ Custom\CheckForMessages.vbs
- **FunctionName** ➔ CheckForMessages
- **Parameter1** ➔ Empty because the `MessageContainerUri` parameter is optional
- **Parameter2** ➔ Message Type ➔ E for Errors
- **Parameter3** ➔ Message Rule ➔ F for Forbidden
- **Parameter4** ➔ N/A
Execution Report Example

By setting Message\textit{Type} to “E” (for error) and Message\textit{Rule} to “F” (for forbidden) the test will fail as soon the message container contains one or more applications errors (Figure 5).

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure5}
\caption{Execution Report Example}
\end{figure}
Pattern Code Explanation

Let’s now have a look at the pattern code, to clarify how the implementation checks for messages.

Overall Structure

The implementation consists of two functions.

- The first function (PATTERN_NAME) is a façade of the implementation; it includes exception handling which provides human-readable feedback in case of scripting errors.
- The second function (PATTERN_NAME_Impl) is the implementation.

Note

Do not confuse PATTERN_NAME and PATTERN_NAME_Impl words. They are both replaced by the function name that the test engineer specifies when creating the custom function.

Function PATTERN_NAME (MessageContainerUri, MessageType, MessageRule, MessagePattern, Options)

On Error Resume Next
EventComponentBegin()

If ConditionsManager().CheckConditions() Then
    PATTERN_NAME = PATTERN_NAME_Impl (MessageContainerUri, _
                                          MessageType, _
                                          MessageRule, _
                                          MessagePattern, _
                                          Options)
End If

EventComponentEnd()
End Function

Function PATTERN_NAME_Impl (MessageContainerUri, MessageType, MessageRule, MessagePattern, Options)

...  

End Function

Note

This structure is recommended for all custom code. It includes the exception handling, and calls two functions required to make the custom code compliant with CBTA requirements.

All SAP-delivered default components are built using this approach.

Initialization Phase

The first statements of the implementation are only used to import the library the implementation depends on.

ImportLibrary "WEB_WebController.vbs"
ImportLibrary "WEB_WebControls.vbs"
The custom function may return a value, which is made available to the subsequent steps of the test scripts via the output parameter of the CBTA_WEB_A_INVoke_FUNCTION component. The code below initializes the returned value to an empty string.

```plaintext
PATTERN_NAME_Impl = ""
```

The next statements validate the input parameters, and set them to their default values if they are not defined by the test script.

```plaintext
If IsNull(MessageContainerUri) Then
    MessageContainerUri="ls.rid=WDR_MESSAGE_AREA.MESSAGE_AREA.ROOTUIELEMENTCONTAINER"
End If

If IsNull(MessageType) Then
    MessageType = ""
End If

If IsNull(MessageRule) Then
    MessageRule = "F"
End If
```

The function writes information to the technical traces (associated with the execution report), using the `ReportDebugLog` method. This might help you to troubleshoot execution problems.

```plaintext
ReportDebugLog "CUSTOM Function - PATTERN NAME" & _
    vbCrLf & "- Uri: " & MessageContainerUri & _
    vbCrLf & "- MessageType: " & MessageType & _
    vbCrLf & "- MessageRule: " & MessageRule
```

**Implementation**

The implementation starts now. The code shown below relies on the Query API to access the Internet Explorer window and search the HTML content for UI elements.

```vbnet
' Preparation of the Query
Dim query, filter
Set query = LsController().CreateQuery()
query.ParentControlUri = MessageContainerUri

' Specifying the criteria (using filters and conditions)
Set filter = query.SetFilter()
filter.AddCondition "tag", "=", "IMG"
filter.AddCondition "ls.type", "=", "LN"
filter.AddCondition "ls.application", "=", "WDR MESSAGE_AREA"
filter.AddCondition "ls.view", "=", "MESSAGE_AREA"
filter.AddCondition "ls.field", "=", "MSG_LIST_ICON"
```
The query defined here searches for images (<img> HTML Elements) that are children of the message container.

The criteria also define conditions to only select the images that represent the message types. This is why Light Speed attributes are used here:

- `ls.type` - type of the Light Speed control - LN for link
- `ls.application` - Web Dynpro Application name
- `ls.view` - Name of the view used to display application messages
- `ls.field` - Name of the field in the view.

Additional conditions are set, depending on the message type being looked for. The conditions below check the name of a GIF file representing the type of the message.

**Note**

*Why do we rely on this information to determine the type of the message? Because this is the only information that the Light Speed framework provides. This is why this code has not been delivered as a default component.*

```vbnet
If MessageType="E" Then
    filter.AddCondition "src", "{endsWith}", "ErrorMessage.gif"
ElseIf MessageType="W" Then
    filter.AddCondition "src", "{endsWith}", "WarningMessage.gif"
ElseIf MessageType="I" Then
    filter.AddCondition "src", "{endsWith}", "SuccessMessage.gif"
Else
    ' No filter - This will show all messages (and count them)
End If
```

The query is now defined. The next statement resolves the query and retrieves a collection of Light Speed controls that match the criteria.

```vbnet
Dim controlCollection
Set controlCollection = query.Select()
If controlCollection Is Nothing Then
    ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - Message container not found"
Else
    ...
End If
```

If the collection is not empty, the implementation performs the verifications according to the `MessageRule` specified.

The `ReportLog` method writes human-readable feedback in the execution report.

```vbnet
If MessageRule="R" Then 'Required
    If controlCollection.Count = 0 Then
        ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - No message of type: " & MessageType
    Else
        ReportLog "PASSED", "PATTERN_NAME", "Message Type: " & MessageType & vbCrLf & "Number of Messages : " & controlCollection.Count
    End If
ElseIf MessageRule="F" Then 'Forbidden
    If controlCollection.Count = 0 Then
```

This following section iterates through the collection of controls, to retrieve the text of the message associated with the IMG HTML element. The GetProperty method retrieves the innerText HTML property containing the text being displayed.

```
Dim childControl, associatedControl
For i=0 To controlCollection.Count-1
    Set childControl = controlCollection.ControlAt(i)
    If Not childControl Is Nothing Then
        Dim imgId, spanId
        imgId = childControl.GetRelevantControl().GetLsId()
        spanId = Replace( imgId, "MSG_LIST_ICON", "MSG_LIST_TEXTVIEW" )
        Set associatedControl = childControl.FindElementById( spanId )
        If Not associatedControl Is Nothing Then
            ReportLog "INFO", "PATTERN_NAME", "Message #" & (i+1) & " - " & associatedControl.GetProperty( "innerText" )
        End If
    End If
Next
```

The last part checks the options parameter and captures a screenshot accordingly, by calling the WEB_CaptureScreen method (associated to the WEB_CAPTURESCREEN Default Component).

```
If InStr(Options, "/c") Then
    If InStr(Options, "/e") Then
        WEB_CaptureScreen MessageContainerUri, "/e"  ' Captures the Message Container only
    Else
        WEB_CaptureScreen MessageContainerUri, ""  ' Capture the whole browser window
    End If
Else
    WEB_CaptureScreen MessageContainerUri, ""  ' Capture the whole browser window
End If
```

The function finally sets the PATTERN_NAME_Impl to return the number of messages matching the MessageType.

```
PATTERN_NAME_Impl = "" & controlCollection.Count
```
Patterns for CRM Web Applications

4.1 Pattern – SelectMenuItemByText

Some CRM applications associate a context menu that lets the user perform additional actions, to some controls. The test recorder can normally record such actions and perform them at runtime, by opening the context menu and simulating a mouse click on the child item.

Unfortunately, for some scenarios the default behavior does not work, because the ID associated to the child item is not stable: the ID is different each time the CRM application runs.

The solution is to search for the child item by text, using the SelectMenuItemByText pattern.

Example

Figure 6 shows an example of such a menu item. Use the CRM application, for example, to create an incident:

- Business Role: SOLMANPRO
- Logical Link: SM-IM-CR

![Figure 6](image)

The Object Spy or the Microsoft Developer Tool (F12 in Internet Explorer) indicate that the ID includes a server-generated GUID, so you cannot search for the item by ID here.

![Figure 7](image)
Pattern Signature

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelectMenuItemByText</td>
<td>Searches for application messages in a messages container. This function verifies whether the applications reports feedback, and checks for errors, warnings, etc.</td>
</tr>
</tbody>
</table>

**INPUT PARAMETERS:**
- Uri ➔ URI of the control with which the menu is associated
- Text ➔ Text of the child item that is to be selected
- Parameter3 ➔ Not used
- Parameter4 ➔ Not used
- Options ➔ Several options influencing the behavior are available

**OUTPUT PARAMETER:**
- Output ➔ Set to DONE when the action has been performed

<table>
<thead>
<tr>
<th>Uri</th>
</tr>
</thead>
<tbody>
<tr>
<td>URI of the main control. i.e.: The control with which the context menu is associated.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>The text of the item. This text is case-sensitive except when the /u option is specified.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>The option parameter supports the following values:</td>
</tr>
</tbody>
</table>
- /u ➔ upper-case ➔ Item selection is then case-insensitive
- /click ➔ When this option is set, the implementation opens the context menu by simulating a mouse click on the main control, which is not done by default because a previous step of the test may have already done so.

Pattern Usage

`SelectMenuItemByText` has 5 parameters to comply with the default component for calling custom functions:

- CBTA_CRM_A_INVOKE_FUNCTION

Leave Parameter3 and Parameter4 empty when calling this custom code.

Pattern Code Explanation

Like for all other patterns, the implementation consists of two functions. The first one is a façade to the implementation which includes execution error handling statements. The second method actually performs the job.

1. Note

The PATTERN_NAME fragment (see below) is automatically replaced by the final custom function name
when using the Code Assistant feature of the Runtime Library Manager.

```vbs
Function PATTERN_NAME ( Uri, Text, Parameter3, Parameter4, Options )
    On Error Resume Next  ' Important - Exception Handling - Do not change it!
    EventComponentBegin()
    If ConditionsManager().CheckConditions() Then
        PATTERN_NAME = PATTERN_NAME_Impl ( Uri, Text, Parameter3, Parameter4, Options )
    End If
    EventComponentEnd()
End Function

Function PATTERN_NAME_Impl ( Uri, Text, Parameter3, Parameter4, Options ) 'internal
    ImportLibrary "CRM_WebController.vbs"
    ImportLibrary "CRM_WebControls.vbs"
    PATTERN_NAME_Impl = "NOT DONE"
    ...
End Function
```

The first statements of the implementation initialize local variables, validate the input parameters and write information to the traces, for troubleshooting purposes.

```vbs
Dim bodyControl, bodyElement, collection, childElement, anchors, anchor, currentText
If IsNull(Uri) Then
    ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - Button Uri not specified"
End If
If IsNull(Options) Then
    Options = ""
End If
```

The next section checks the `Options` parameters, to determine whether it is necessary to open the context menu.

```vbs
If InStr(Options, "/click" ) Then
    CRM_WebControl_Click Uri
End If
```

Note

The implementation below invokes the CRM_WebControl_Click method, which corresponds to the default component:

- CBTA_CRM_CLICK

Once the menu item is opened, the implementation can continue and search for child items. The code below first accesses the body of the work area, using the GetControlByUri method that the CRM application controller exposes. This step waits for documents to be ready.

```vbs`
  Gets the body of the current workarea
```
When the body of the document is ready, the content of the menu should be visible and the implementation can proceed.

The statements below access the body HTML element using the GetHTMLElement method, and search for SPAN HTML elements using the MSHTML API (from Microsoft).

```vbscript
Set bodyControl = CrmController().GetCrmControlByUri("crm.area=WorkArea; tag=body")

' Gets the body of the current workarea
Set bodyControl = CrmController().GetCrmControlByUri("crm.area=WorkArea; tag=body")
If bodyControl is Nothing Then
    ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - Workarea not found"
Else
    Set bodyElement = bodyControl.GetWebControl().GetHTMLElement()
    If bodyElement is Nothing Then
        ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - Workarea not found"
    Else
        Set collection = bodyElement.all.tags("SPAN")
        If collection is Nothing Then
            ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - No SPAN HTML Elements"
        Else
            ReportLog "DONE", "PATTERN_NAME", "Count : " & collection.length
            ... End If
        End If
    End If
End If
```

The statements below do not rely on CBTA interfaces. The bodyElement implements some scripting interfaces (from the MSHTML API) and the `all` and `tags` properties return a set of objects that also implement scripting interfaces.

```vbscript
Set collection = bodyElement.all.tags("SPAN")
```

The `collection` variable contains a list of SPAN HTML elements in the work area of the CRM application. The implementation filters out the SPAN HTML elements that are not menu items by verifying whether the SPAN element has an ID including the `T_MENU_ITEM` fragment, and that it has at least one child element of type ANCHOR `<A>` HTML tag.

The visible text of the item is associated with the anchor element. Its value is retrieved by `innerText` property, and checked against the expected value.

```vbscript
For i=0 To collection.length-1
    Set childElement = collection.item(i)
    If Not IsNull(childElement.id) Then
        If InStr( childElement.id, ":T_MENU_ITEM:" ) Then
            Set anchors = childElement.all.tags("A")
            If anchors is Nothing Then
                ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - No Anchor HTML Elements"
            Else
                Set anchor = anchors.item(0)

                CBTA.Report CBTA.INFO, "PATTERN_NAME", "Anchor Element - id=" & anchor.id, ""

                currentText = anchor.innerText
                If InStr(Options, "/u" ) Then
                    currentText = UCase(currentText)
```
If the text matches the one being looked for, the implementation simulates a mouse click on the item, calling several methods exposed by the MSHTML API.

```
If InStr( currentText, Text ) Then
    anchor.FireEvent("onmousedown")
    anchor.setActive()
    anchor.focus()
    anchor.FireEvent("onmouseup")
    anchor.click()
    PATTERN_NAME_Impl = "DONE"
    Exit For
End If
```

For more information about this API, refer to:

- [Scripting Objects Interfaces from Microsoft](#)

Instead of calling methods exposed by the MSHTML API, the implementation could reuse the `CRM_WebControl_Click` method to select items, for example as follows:

```
Dim anchorUri
anchorUri = "label=" & anchor.innerText & "; crm.area=WorkArea; tag=A; id=" & anchor.id
CRM_WebControl_Click anchorUri
```
Pattern Limitations

This SelectMenuItemByText pattern searches for an HTML element in the body of an HTML document. The current implementation has the following limitations:

- It does not support the selection of sub items (children of another child item).
- It only searches for items in the main work area of the CRM application. The code must be adapted if the item is shown in the HTML document of a modal popup.
- The context menu associated to the main control is opened by mouse click. Some controls may expect a right-click instead.
- The code shown here assumes that the application can only display a single context menu, so it does not verify whether the item is associated with the main control.
4.2 Pattern – SelectTransactionType

The SelectTransactionType pattern searches for and selects a transaction type in the popup window that appears when starting some CRM applications.

Note

This pattern exists only to illustrate how to solve this problem and to explain to the Query API that CBTA can solve such issues. The equivalent component is CBTA_CRM_SelectTransactionType, which you can use without customizing the Runtime Library.

Use Case

The creation of a sales order is a typical example of business scenario that starts with the selection of the transaction type.

The type is not always in the same position (depending on the number of transaction types in the database), so you must adapt the test script to make it search for the transaction type using the text being displayed, instead of the row number.

Note

By default, the line is selected by row number. This information is collected while recording the scenario, and is used in the generated test script.

Figure 8 illustrates the automation of this step. The first column shows the transaction type, the second a short description. Both columns can identify the transaction type to be selected.

![Figure 8: Transaction Type Selection Popup](image)

The bottom of the screen includes a "pager" control to navigate to the next pages if the transaction type is not on the current page.
Pattern Signature

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>SelectTransactionType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Searches for and selects a transaction type in a popup window displayed when starting some CRM applications.</td>
</tr>
</tbody>
</table>

**INPUT PARAMETERS:**
- TransactionType ➔ Text in the Transaction Type column
- Description ➔ (optional) Text in the Description column
- Operator ➔ (optional) The operator to use when defining the filter (default is equals “=”)
- MaxPage ➔ (optional) The number of pages to search for the transaction type
- Options ➔ Several options influencing the behavior are available

**OUTPUT PARAMETER:**
- Output ➔ Report status (OK or FAILED)

**TransactionType**
- Text in the Transaction Type column (first column)

**Description**
- Text in the Transaction Type Description column (second column)

**Operator**
- Operator to compare the TransactionType and Description parameter values with the actual values of the respective columns. Supported operators are those documented in the Query API.

**MaxPage**
- MaxPage is 10 by default. The implementation navigates to the next page when the transaction type is not in the current one.

**Options**
- The option parameter can take the following values:
  - /t ➔ for trimming ➔ Ignores leading and trailing spaces when comparing the TransactionType and Description parameter values with the actual values of the respective columns.
  - /u ➔ upper-case ➔ for a case-insensitive comparison

Pattern Usage

SelectTransactionType has 5 parameters to comply with the default component CBTA_CRM_A_INVOKE_FUNCTION for calling custom functions.

Pattern Code Explanation

Like most of the patterns, the implementation consists of two functions. The first one is a façade to the implementation, which includes statements for handling execution errors. The second method performs the job.
Note

The PATTERN_NAME fragment (see below) is automatically replaced by the final custom function name when using the Code Assistant feature of the Runtime Library Manager.

```
Function PATTERN_NAME ( TransactionType, Description, Operator, MaxPage, Options )
  On Error Resume Next ' Important - Exception Handling - Do not change it!
  EventComponentBegin()
  If ConditionsManager().CheckConditions() Then
    PATTERN_NAME = PATTERN_NAME_Impl ( TransactionType, Description, Operator, MaxPage, Options)
  End If
  EventComponentEnd()
End Function

Function PATTERN_NAME_Impl ( TransactionType, Description, Operator, MaxPage, Options ) 'internal
  ImportLibrary "CRM_WebController.vbs"
  ImportLibrary "CRM_WebControls.vbs"
  PATTERN_NAME_Impl = "NOT DONE"
  ...
End Function
```

Initialization Phase

The first statements of the implementation initialize local variables and validate the input parameters.

```
Dim query, filter, subFilter, selectorInPopup, maxPageAsInt, i, uriNextPage, nextPageControl
maxPageAsInt = ToInt(MaxPage)
If IsNull(maxPageAsInt) Or maxPageAsInt = 0 Then
  maxPageAsInt = 10
End If
If IsNull(Operator) Then
  Operator = "=
End If
If IsNull(Options) Then
  Options = ""
End If
```

ReportDebugLog writes information to the traces for troubleshooting.

```
ReportDebugLog "CUSTOM Function - PATTERN_NAME" &
  " - Parameters: TransactionType " & TransactionType & " - Description: " & Description
ReportDebugLog "CUSTOM Function - PATTERN_NAME" &
```
Concrete Implementation

The implementation consists of a loop (using the FOR/NEXT statements) to search each page for the transaction type.

Within this loop, the Query API defines the row match criteria. The query object is created by the CRM controller.

<table>
<thead>
<tr>
<th>For i=1 To maxPageAsInt</th>
</tr>
</thead>
<tbody>
<tr>
<td>' Preparation of the CRM Query - Identifying the parent Table container (in the Popup)</td>
</tr>
<tr>
<td>Set query = CrmController().CreateQuery()</td>
</tr>
<tr>
<td>query.ParentControlUri = &quot;popupId=1; crm.id=A_btfollowup:V_ProcType:T_cellerator:C_proctype:I_&quot;</td>
</tr>
<tr>
<td>' Specifying the element type to search for - ROW_SELECTOR</td>
</tr>
<tr>
<td>Set filter = query.SetFilter()</td>
</tr>
<tr>
<td>filter.AddCondition &quot;crm.type&quot;, &quot;=&quot;, &quot;ROW_SELECTOR&quot;</td>
</tr>
<tr>
<td>filter.AddCondition &quot;crm.context&quot;, &quot;=&quot;, &quot;proctype&quot;</td>
</tr>
<tr>
<td>If Not IsNull(TransactionType) Then</td>
</tr>
<tr>
<td>' filtering on the Transaction Type</td>
</tr>
<tr>
<td>filter.AddCondition &quot;crm.column:proctype.process_type&quot;, Operator, TransactionType, Options</td>
</tr>
<tr>
<td>End If</td>
</tr>
<tr>
<td>If Not IsNull(Description) Then</td>
</tr>
<tr>
<td>' filtering on the Transaction Type Description</td>
</tr>
<tr>
<td>filter.AddCondition &quot;crm.column:proctype.proc_type_descr_20&quot;, Operator, Description, Options</td>
</tr>
<tr>
<td>End If</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>Next</td>
</tr>
</tbody>
</table>

The ParentControlUri property restricts the scope of the query. In this example, the URI of the table (shown in the popup window) is specified.

A filter is defined to specify what to search for.

- The first condition checks the crm.type attributes. It guarantees that only ROW_SELECTOR matches the criteria.
- The second condition checks for the context (C_ fragment of the ID that the WebCUIF framework generates)

Two additional conditions are defined, depending on the input parameters:

- When the TransactionType parameter is set, an additional condition is created to check for the content of the column with the technical name proctype.process_type
- When the Description parameter is set, another condition is created to check the content of the column with the technical name: proctype.proc_type_descr_20

Note

This Query API is documented separately. The API handles situations that are specific to the underlying UI technology.

CRM applications are built on top of the WebCUIF framework. The query created in the example below leverages the meta-information that this framework inserts in the HTML content, so that the test engineer can handle common use cases like this one.
Note

The query is created within the loop because the query object has a restricted life which is bound to the HTML document being displayed. When navigating to the next page, the current HTML document is discarded and a new one is displayed, so a new query is necessary.

Query Resolution

Once defined, the query can be resolved to get the controls that match the criteria. In this use case, we only expect a single control (a single row), so we use the SelectSingle method here.

The selectorInPopup variable receives the query result.

```vbnet
' Selecting the first element matching the criteria
ReportDebugLog "CUSTOM Function - PATTERN_NAME - now resolving query - Current Page: #" & i
Set selectorInPopup = query.SelectSingle()

If selectorInPopup Is Nothing Then
    uriNextPage = "label=Navigation to Page #" & (i+1) & "; " & 
    "popupId=1; tag=SPAN; crm.tag=TD; " & _
    "crm.id=A_btfollowup:V_ProcType:T_PAGER:C_proctype:I_:K_pag_pg:" & (i+1)

    Set nextPageControl = CrmController().GetCrmControlByUri(uriNextPage)
    If nextPageControl Is Nothing Then
        ReportLog "FAILED", "PATTERN_NAME", "Operation Failed - Last page has been reached"
        Exit For
    Else
        ' Control not found - Let's navigate to the next page (if any)
        CRM_WebControl_Click uriNextPage
        If ReportStatus <> "OK" Then
            Exit For
        End If
    End If
Else
    ... 
End If
```

This code builds the URI of the next page and simulates a mouse click to navigate to it.

Figure 9: Transaction Type Selection Pager

Note

When the last page is reached, the loop is interrupted and the test script may fail since no transaction type has been selected.
Row Selection

When the row selector UI element is found, the selectorInPopup variable is set and the implementation uses the GetControlUri method to determine the URI of the row selector, and calls the CRM_Table>SelectRow function to perform the selection.

```csharp
' Control found - let's click on the ROW_SELECTOR
CRM_Table_SelectRow selectorInPopup.GetControlUri()

If ReportStatus = "FAILED" Then
    ReportDebugLog " CUSTOM Function - PATTERN_NAME - Operation Failed"
Else
    ReportDebugLog " CUSTOM Function - PATTERN_NAME - Operation Succeeded"
End If

PATTERN_NAME_Impl = ReportStatus
Exit For
```

The PATTERN_NAME_Impl variable provides feedback about the status of the operation. It is the value received by the output parameter of the CBTA_CRM_A_INVOKE_FUNCTION component.

Result: The transaction type is selected. The Exit For statement interrupts the loop to avoid searching the next pages.
5 Patterns for SAP GUI

5.1 Pattern – ME21N_ExpandAllSections

The `ME21N_ExpandAllSections` pattern restores a recognized initial state if a transaction starts in a different state, depending, for example, on user preferences, to let the test script continue in the expected context.

**Example**

This problem typically occurs with transaction ME21N, where the main screen consists of several collapsible panels. The initial state of the panels is unknown at startup. Each panel can be expanded or collapsed, depending on how the transaction ended during the previous session.

Figure 10 shows the Item Overview panel in its collapsed state.

![Figure 10: Item Overview pane](image)

The test script needs to expand this panel before continuing. Figure 11 shows the same screen with the panel expanded.
The custom code of this pattern verifies the state of each panel and expands them if necessary.

Note

This situation can normally be resolved using default components such as CBTA_A_SETCONDITION, but not with transaction ME21N, which has a very specific handling of its panels. It uses several subscreens to display the content of each panel, so a custom function is necessary.

Pattern Signature

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>ME21N_ExpandAllSections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checks for the state of the header sections and expands them automatically.</td>
<td></td>
</tr>
</tbody>
</table>

**INPUT PARAMETERS:**
- None

**OUTPUT PARAMETER:**
- None.

Pattern Usage

Assume that the custom function has been created previously, using the Code Assistant feature of the Runtime Library Manager (Figure 12).

This custom function has no input parameters, so it can be called by the CBTA_GUI_A_EXECUTESTATEMENT default component.
Test Script Definition

Figure 13 example of how to call it.

![Image showing test script definition](image)

**Input Parameters**

The component for calling the custom function expects at least the following parameters:

- **Library** ➔ The name of the VB script file.
- **Statement** ➔ The statement to execute.

In this example, because the custom function does not expect input parameters, the statement is simple (Figure 14).

![Image showing execute statement input parameters](image)
Execution Report

The actions performed by the custom function are displayed in the execution report, as shown in the example execution report below (Figure 15).

<table>
<thead>
<tr>
<th>INFO</th>
<th>2 - CBTA GUI A EXECUTE STATEMENT</th>
<th>Expand</th>
<th>Operation has been performed Target: Expand Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>INFO</td>
<td>2 - CBTA GUI A EXECUTE STATEMENT</td>
<td>Expand</td>
<td>Operation has been performed Target: Expand Item Overview</td>
</tr>
<tr>
<td>INFO</td>
<td>2 - CBTA GUI A EXECUTE STATEMENT</td>
<td>Expand</td>
<td>Operation skipped - the section is already expanded</td>
</tr>
<tr>
<td>INFO</td>
<td>3 - CBTA GUI SETTEXT</td>
<td>SetPropertyValue</td>
<td>Tablecell_Material(0) - (Text = P-101)</td>
</tr>
<tr>
<td>INFO</td>
<td>4 - CBTA GUI SETFOCUS</td>
<td>GuiVConsetSetText</td>
<td>Target: Tablecell_Short Text(0)</td>
</tr>
</tbody>
</table>

Figure 15: ME21N Execution Report

Pattern Code Explanation

The implementation consists of sub-procedures. The first one is a façade of the implementation. The second method performs the job.

Note

The PATTERN_NAME fragment (see below) is automatically replaced by the final custom code name when using the Code Assistant feature of the Runtime Library Manager.

Sub PATTERN_NAME ()

On Error Resume Next ' Important - Exception Handling for SAPGUI - Do not change it!
EventComponentBegin()

If ConditionsManager().CheckConditions() Then
   PATTERN_NAME_Impl
End If

EventComponentEnd()

End Sub

Sub PATTERN_NAME_Impl () 'internal
   ReportDebugLog "CUSTOM Function - PATTERN_NAME"
   ...
End Sub
Implementation

The sub procedure calls the `Expand_Impl` function for each collapsible section in the ME21N transaction.

```vbscript
Sub Expand_Impl ( Uri ) 'internal
    Set GS_guiControl = Nothing
    On Error Resume Next ' Switching ON a local error handling
    Set GS_guiControl =GuiScripting().FindGuiControlByUri(Uri)
    On Error Goto 0 ' Switching OFF the local error handling
    If GS_guiControl Is Nothing Then ' Control not found
        ReportLog "INFO", "Expand", "Operation skipped - the button does not exists"
    Else
        If GS_guiControl.IconName = "DAARSO" Then ' Control has been found - let's click on it
            ReportLog "INFO", "Expand", "Operation has been performed"
            & vbCrlf & "Target: " & GetMeaningfulName(GS_guiControl)
        Else
            GS_guiControl.Press
        End If
    End If
End Sub
```

The `Expand_Impl` procedure is also delivered with the pattern code. It searches for the collapsible button by URI, calling the `FindGuiControlByUri` method. This call returns a reference to a GuiScripting object (a GuiButton in this case). The code then determines whether the panel is expanded or collapsed, using the `IconName` property.

Note

The logic is specific to the ME21N transaction.

Checking the `IconName` is not the regular way of determining the collapsible state, but it is the only information available in the ME21N transaction.
5.2 Pattern – ME51N_ExpandAllSections

ME51N_ExpandAllSections is analog to ME21N_ExpandAllSections. The only difference is that it expands ME51N collapsible panels. For more information, see: Pattern – ME21N_ExpandAllSections.

5.3 Pattern – WaitForCondition

The WaitForCondition pattern makes a test script wait for the background process to complete before continuing.

With SAP GUI, some transactions can start asynchronous background processes on the serve, so the UI cannot display the operation result automatically. The end user must perform actions (like pressing ENTER) to update the UI with the server status.

**Pattern Signature**

<table>
<thead>
<tr>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitForCondition</td>
<td>Performs an action several times until a text is displayed in the SAP GUI UI.</td>
</tr>
</tbody>
</table>

**INPUT PARAMETERS:**

- **Uri** ➔ URI identifying the SAP GUI control displaying the operation result
- **TextToCheck** ➔ Text the control displays when the background process is still running
- **WaitTime** ➔ Time to wait between attempts
- **MaxAttempts** ➔ Number of attempts (maximum)
- **Options** ➔ Not used

**OUTPUT PARAMETER:**

- **Output** ➔ Status (NOT_DONE or DONE)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uri</td>
<td>URI of the control displaying the text for which the wait condition checks. If empty, the URI of the status bar is used by default.</td>
</tr>
</tbody>
</table>

**DEFAULT VALUE:**

- label=Main Window; type=GuiMainWindow; id=/app/con[0]/ses[0]/wnd[0]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TextToCheck</td>
<td>Text (or part of the text) that is displayed when the background process is still running.</td>
</tr>
</tbody>
</table>

**DEFAULT VALUE:**

- Scheduled.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaitTime</td>
<td>Time to wait between attempts. Default value is 1000 (one second)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaxAttempts</td>
<td>Maximum number of attempts. When the max. number of attempts is reached, the implementation stops checking and exits with NOT_DONE as output parameter. The default value is 30.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Options</td>
<td></td>
</tr>
</tbody>
</table>
Pattern Name

No options are so far supported by this pattern.

Pattern Usage

The pattern has 5 parameters, so it can be called by the CBTA_GUI_A_INVOKE_FUNCTION default component.

Pattern Code Explanation

Like most of the patterns, the implementation consists of two functions. The first one is a façade of the implementation, which includes execution error handling statements. The second method performs the job.

Initialization Phase

The first statements validate the input parameters and set their default values, if necessary.

```vbscript
If IsNull(Uri) Then
    Uri = "label=Status Bar; type=GuiStatusbar; id=/app/con[0]/ses[0]/wnd[0]/sbar"
End If
If IsNull(TextToCheck) Then
    TextToCheck = "Scheduled"
End If
If IsNull(WaitTime) Then
    WaitTime = "1000" '1 second
End If
If IsNull(MaxAttempts) Then
    MaxAttempts = "30" '30 * 1000 ==> 30 seconds max!
End If
```

The code writes information to the traces, for troubleshooting, and initializes the result to “NOT DONE”.

```vbscript
ReportDebugLog "CUSTOM Function - PATTERN_NAME" & _
vbCrLf & "- Uri:" & Uri & _
vbCrLf & "- TextToCheck:" & TextToCheck & _
vbCrLf & "- WaitTime:" & WaitTime & _
vbCrLf & "- MaxAttempts:" & MaxAttempts

PATTERN_NAME_Impl = "NOT DONE"
```

Implementation

A loop (using the For/Next keywords) checks several times for the text being displayed, until the limit (defined though the maxAttempts input parameter) is reached.

- The GetPropertyValue method retrieves the text displayed by the control the URI refers to.
- It then checks if this text contains the value in the TextToCheck input parameter.
- If so, it waits for the time specified and presses ENTER to refresh the screen.
- If not, it assumes the background process is over and breaks the iteration.
Dim i, statusBarUri, mainWindowUri, statusBarText, max
mainWindowUri = "label=Main Window; type=GuiMainWindow; id=/app/con[0]/ses[0]/wnd[0]"
max = ToInt(MaxAttempts)

For i=1 To max

    ReportDebugLog "PATTERN_NAME - Now getting the text of the Control"
    statusBarText = GetPropertyValue ( Uri, "Text", Null )
    ReportDebugLog "PATTERN_NAME - Control Text: " & statusBarText

    If InStr(statusBarText, TextToCheck) Then
        ' Let's wait
        ReportLog "INFO", "PATTERN_NAME", "Waiting for background job to complete - attempt #" & i
        GS_Wait WaitTime, Null

        ' Let's Press Enter to refresh the UI
        GS_PressKey mainWindowUri, "Enter", Null
    Else
        ' Text not visible anymore
        ReportLog "DONE", "PATTERN_NAME", "Background job has completed"
        PATTERN_NAME_Impl = "DONE"

        ' Let's stop the loop
        Exit For
    End If
Next
6 Patterns for Miscellaneous

6.1 Search into Excel File

Function to search into Excel file some text and return the row number where the first occurrence has been found.

Filename : absolute path on file system
TheValue: text to search into content

```vbscript
Function SearchExcel ( FileName, TheValue, Parameter3, Parameter4, Options )
    On Error Resume Next ' Important - Exception Handling - Do not change it!
    EventWebComponentBegin()
    If ConditionsManager().CheckConditions() Then
        SearchExcel = SearchExcel_Impl ( FileName, TheValue, Parameter3, Parameter4, Options )
    End If
    EventWebComponentEnd() ' Important - Exception Handling - Do not change it!
End Function

Function SearchExcel_Impl ( FileName, TheValue, param3, param4, Options ) 'internal
    SearchExcel_Impl = -1
    Dim objExcel, objWorkbook, objRange
    GS_Wait "1000", "" 'WAIT FOR THE FILE IS AVAILABLE
    Set objExcel = CreateObject("Excel.Application")
    Set objWorkbook = objExcel.Workbooks.Open(FileName)
    objExcel.DisplayAlerts = False
    ' Dim objFound
    For i = 1 To objWorkbook.Worksheets.Count
        Set objRange = objWorkbook.Worksheets(i).UsedRange
        Set objFound = objRange.Find(TheValue)
        If Not (objFound Is Nothing) And Not IsEmpty(objFound) Then
            SearchExcel_Impl = objFound.Row
            Exit For
        End If
    Next
    If SearchExcel_Impl <> -1 Then
        ReportLog "PASSED", "SearchInFile", "Value '" & TheValue & "' was found - Line #" & objFound.Row
    Else
        ReportLog "FAILED", "SearchInFile", "Value '" & TheValue & "' was not found in the file '" & FileName & "'
    End If
    SetOutputParameter "RowNumber", "" & SearchExcel_Impl
    objWorkbook.Close False
End Function
```

'==================================================================
If Not IsEmpty(Environment) Or CBASE_BOOTSTRAP Then
    'This line prevents from loading the library twice.
    ExecutionContext().DeclareLibrary "ExcelExtension.vbs"
End If
6.2 Download PDF from Internet Explorer

```vbs
Function FetchContent ( Uri, FileName, Parameter3, Parameter4, Options )
    ' Important - Exception Handling - Do not change it!
    On Error Resume Next
    EventWebComponentBegin()
    If ConditionsManager().CheckConditions() Then
        FetchContent = FetchContent_Impl ( Uri, FileName, Parameter3, Parameter4, Options )
    End If
    EventWebComponentEnd() ' Important - Exception Handling UI- Do not change it!
End Function

Function FetchContent_Impl ( Uri, FileName, param3, param4, Options ) 'internal
    ImportLibrary "WEB_WebController.vbs"
    ImportLibrary "WEB_WebControls.vbs"
    FetchContent_Impl = "NO_CONTENT"
    If IsNull(Uri) Then
        Exit Function
    End If
    If IsNull(Options) Then
        Options = "" ' No options by default - /Click - to click on the first Chart Data
    End If
    Uri = EvaluateUriValue( Uri )
    ReportDebugLog "CUSTOM Function - FetchContent - Parameters: Uri " & Uri
    Dim operationResult
    Set operationResult = WebController().WebControl_FetchContent(Uri, FileName, Options)
    FetchContent_Impl = WEB_HandleOperationResult( "WebControl_FetchContent", operationResult, Null )
End Function

'=================================================

If Not IsEmpty(Environment) Or CBASE_BOOTSTRAP Then
    'This line prevents from loading the library twice.
    ExecutionContext().DeclareLibrary "FetchContent.vbs"
End If
```

7 Appendix

7.1 Patterns Location

The SAP-delivered patterns are stored in the MIME repository of the SAP Solution Manager system, under SAP ➔ PUBLIC ➔ CBTA ➔ PATTERNS.

Patterns are stored in sub-folders, depending on their nature. In the example below (Figure 16), there are three sub-folders:
- ABAP Web Dynpro
- CRM Web UI
- SAP GUI

Figure 16: Patterns in the MIME Repository
7.2 Runtime Library Manager

The Runtime Library Manager customizes the runtime library. It can:

- Open the runtime library for editing
- Write custom code manually
- Write custom code using the patterns (via the code assistant)
- Test the custom code (before submitting it)
- Submit the changes to make them available to all testers

Code Assistant

When using the code assistant, a guided procedure helps you select a pattern and generate the custom code.

![Figure 17: Starting the Code Assistant](image)
Selecting a Pattern

The selection of a pattern is a prerequisite for creating a custom function. If you use the assistant instead of doing it manually, the generated code will be ready for use.

Select a pattern according to your needs or the problem that you are facing. The UI technology of the application being tested is the main criterion to consider when selecting a pattern.

Figure 18: Pattern Selection
Generating a Custom Function

The code assistant finishes by generating the custom code, using the function name that the test engineer specified.

![Runtime Library Manager](image1)

**Figure 19: Custom Code Generation**

The VB script containing the custom code is stored locally on the file system, at the location specified by the test engineer. Adapt the VB script to your needs, and test it, before submitting the changes.

```
Function XXX_SelectTransactionType (TransactionType, Description, Operator, MaxPage, Options )
On Error Resume Next
ImportLib "CRM_WebController.vba"
ImportLib "CRM_WebControls.vba"

Option Explicit

Function XXX_SelectTransactionType_Impl (TransactionType, Description, Operator, MaxPage, Options ) 'Impl

Function XXX_SelectTransactionType_Impl (TransactionType, Description, Operator, MaxPage, Options ) 'Impl
```

**Figure 20: Custom Code Sample**

For more information, see CBTA - Runtime Library Manager – CBASE Customization.
8 References

8.1 SAP Notes

The SAP Notes mentioned in this document.

<table>
<thead>
<tr>
<th>SAP Note</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>1778899</td>
<td>CBTA - Collective Note</td>
</tr>
<tr>
<td>1912801</td>
<td>CBTA – RTL Manager – Runtime Library Customization</td>
</tr>
</tbody>
</table>

8.2 Materials and Documentations

Some additional documents have to be considered to take benefit of the Query API.

<table>
<thead>
<tr>
<th>Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBTA – User Guide</td>
</tr>
<tr>
<td>CBTA – Runtime Library and Default Components</td>
</tr>
<tr>
<td>CBTA – Runtime Library Manager</td>
</tr>
<tr>
<td>CBTA – Query API</td>
</tr>
<tr>
<td>CBTA – Object Spy</td>
</tr>
<tr>
<td>CBTA – Test Recorder</td>
</tr>
</tbody>
</table>

8.3 Scripting Objects Interfaces from Microsoft

The MSHTML API is a set of Microsoft COM interfaces that provides access to the HTML content that Internet Explorer displays. The official documentation is online on MSDN at:


Scripting Objects Interfaces mentioned in this document:
<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Method Name</th>
<th>Link</th>
</tr>
</thead>
</table>