User Guide
CBTA 3.0 SP2

PUBLIC

2013-10-09

Test Automation of Web Applications

Query API for Testing Dynamic Scenarios

Table of Contents

1 Foreword

The purpose of this document is to explain how to automate the testing of scenarios that cannot be automated using the regular approach based on components.

Like most of the modern applications, the web applications that SAP delivers display their content in Internet Explorer as HTML pages. The content of the HTML Pages can be very complex and dynamic. It is generated server-side by the presentation layer that may differ depending on the UI technology used for developing the application.

1.1 UI Technologies

At SAP, the main UI technologies are the following:

UI Technology	Description
WebCUIF	Layer that generates the HTML content for SAP CRM applications.
Unified Rendering Light Speed (LS)	Layer that generates the HTML content for SAP Applications such as: • Web Dynpro Application (ABAP) • Web Dynpro Applications (JAVA) • Web GUI – SAP GUI Content displayed in Internet Explorer
BSP	Applications based on Business Server Pages
Java Web Dynpro	Former layer used by Java Web Dynpro Applications.

Non-SAP UI Technologies

SAP competitors may use other UI technologies. Some are not supported and documented as known limitations. Some are generic enough and can be automated as plain HTML pages.

Non-SAP UI Technology		Description
Plain HTML Pages	*Supported*	Content generated using regular HTML tags (with no or a few scripting capabilities).
Adobe Flash	*Not Supported*	Layer used by Adobe Flash Applications and generated using the Adobe Flex software development kit.
Java Applets	*Not Supported*	Application written in Java. Java applets can appear in a frame of the web page or in a new application window

Each UI technology has its own logic and generates an HTML content that might be completely different than the other ones. From a test automation perspective this is quite challenging and may require writing custom functions for addressing the specific-issues.

By writing custom functions the test engineer can search the HTML content displayed by the application for specific UI elements. This can be done using some native COM interfaces (from Microsoft) that the Internet Explorer object

exposes. However, this approach is too complex and error-prone. The alternative to the native COM interfaces (from Microsoft) is to use queries via an API that is described hereafter.

1.2 Terminology

Runtime Library

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

Default Components

Default Components are components performing atomic operations against UI elements. The *Runtime Library* comes with default components for all the UI technologies that CBTA supports.

URI – Uniform Resource Identifier

The *Default Components* targeting a UI element do have an input parameter specifying the *URI*. The *URI* is a string identifying the UI element. Its syntax is quite flexible; it consists of a list of name/value pairs providing the information required for searching the HTML content.

Web Control

A *Web Control* is the entity displayed in web page that the end-user can identify and can interact with. Web Applications are heavily using controls such as:

- Input Fields and Checkboxes
- Dropdown list boxes (combobox)
- Trees
- Lists
- ...

The internal structure of a control is specific to the UI technology and can be a composition of several UI elements.

Meta-Information

Web controls may include properties and attributes providing information about the nature of the control. CBTA leverages this meta-information to generate comprehensive Test Scripts.

2 Query API

The purpose of the query API is to hide the HTML complexity and provide the ability to search for UI elements using filters and conditions.

2.1 Query Types

The query approach is similar whatever the underlying technology is. However some additional capabilities might be available for a certain UI technology. In other words, the queries are specialized for a certain type of applications. The table below shows the different query types and the corresponding UI technology.

Query Type	UI Technology	Applications
CRM Queries	WebCUIF	SAP CRM applications.
LS Queries	Unified Rendering Light Speed (LS)	Web Dynpro Application (ABAP) Web Dynpro Applications (JAVA) Web GUI – SAP GUI Content displayed in Internet Explorer
Web Queries	BSP and Plain HTML	Web Applications
JWDP Queries	Java Web Dynpro (Old releases)	Java Web Dynpro Applications.

Example of query:

- Search for all UI elements in a HTML Table where *Partner Function* is “Sales employee” and *Partner ID* is “244”.

This documentation explains how to build such query and how to manipulate the UI elements that are matching the criteria.

2.2 Application Controllers

Application controllers are the main entry point. They act as a handle to the application being tested and provide access to the HTML content that the current session displays.

Application Controller Types

CBTA 3.0 SP2 comes with several applications controllers. Each of them is specialized to handle specific use-cases that are depending on the underlying UI technology.

The table below explains the purpose of each of them.

Controller Name	Description
Light Speed Controller	Controller for applications that are built on top of the Unified Rendering Light Speed framework (such as Web Dynpro ABAP applications)
CRM Controller	Controller for SAP CRM Applications – i.e. Applications that rely on the WebCUIF UI technology.
Web Controller	Controller for web applications where the underlying UI technology is unknown or web applications that do not have a specific controller (like BSP Applications).
Java Web Dynpro Controller	Controller for application based on former releases of Java Web Dynpro.

Creating the Query Objects

The controller interfaces are entry points to the query interfaces. Each controller has (at least) a method to create a query objects.

Creating a Web Query Object

The script below creates a query object to search for web controls.

```
Dim controller, query
Set controller = WebController()
Set query = controller.CreateQuery()
```

Creating a WebCUIF Query Object

The script below creates a query object for searching WebCUIF Controls (for SAP CRM Applications). This time the *CrmController* is used.

```
Dim controller, query
Set controller = CrmController()
Set query = controller.CreateQuery()
```

Creating a Light Speed Query Object

The script below creates a query object to search for Light Speed controls. The code is similar; the only difference is that the web controller is not used anymore. The Light Speed controller is used instead.

```
Dim controller, query
Set controller = LsController()
Set query = controller.CreateQuery()
```

Creating a Web Query Object from any Controller

In the example below we explicitly ask for a Web Query (instead of a Light Speed query) by calling the *CreateWebQuery* method.

```
Dim controller, query
Set controller = LsController()
Set query = controller.CreateWebQuery()
```

Query Creations with Options

The query objects can be created using some options that are specific to the underlying UI technology. This additional parameter is optional.

The example below shows the creation of a Light Speed query using the option */ls (this option activates some optimizations. More details in the following sections)*.

```
Dim query
Set query = LsController().CreateQuery("/ls")
```

2.3 Web Controls

Light Speed Web Controls

Applications that are built on top of the *Light Speed* framework generate controls where at least one of the HTML elements provides meta-information via additional HTML attributes.

Attribute Name	Description
`ct`	Control type
`lsdata`	Light Speed data A collection of key/value pairs describing the control and its current state.

Example of *Light Speed* attributes for a control of type *checkbox:*

Light Speed Data	Description
`Checked`	State of the check box
`Enabled`	False when the checkbox is disabled (i.e. its state is not relevant)
`Readonly`	False when the state cannot be changed
`Text`	The text displayed next to the checkbox
`Tooltip`	The text shown in the tooltip

The LS Query interfaces have been specialized to benefit from this meta-information. It is therefore possible to search for a control using the Light Speed data.

CRM Web Controls

As already mentioned, CRM applications generate their content using the WebCUIF UI technology. With this technology each control has a hidden SPAN HTML element that provides meta-information via its ID attribute.

> Example of ID
>
> *A_btpartner:V_Partner:T_ROW_SELECTOR:C_btpartner:I_partner_fct:R_1*

The ID is made of several fragments that are described below:

Fragment Prefix	Description
A_	Application name
V_	View name
T_	Type of the control
C_	Context

Fragment Prefix	Description
I_	Interface – i.e.: a field name or an attribute name in scope of the C_ context
R_	Row number (e.g.: when embedded in a Table Container)
K_	Key (e.g.: item of a drop down list box can be selected by key)

The CRM Query interfaces have been specialized to benefit from this meta-information. It is therefore possible to search for a control using these fragments.

CRM Web Control Example

The screenshot below shows a kind of toolbar that a CRM application may display. Note that in this example the links are decorated with an image.

Figure 1: Screenshot Corporate Account Buttons

One may use the developer tool (F12 in Internet Explorer) to visualize the internal structure of button named “Save”. The result will show the following composition of HTML elements.

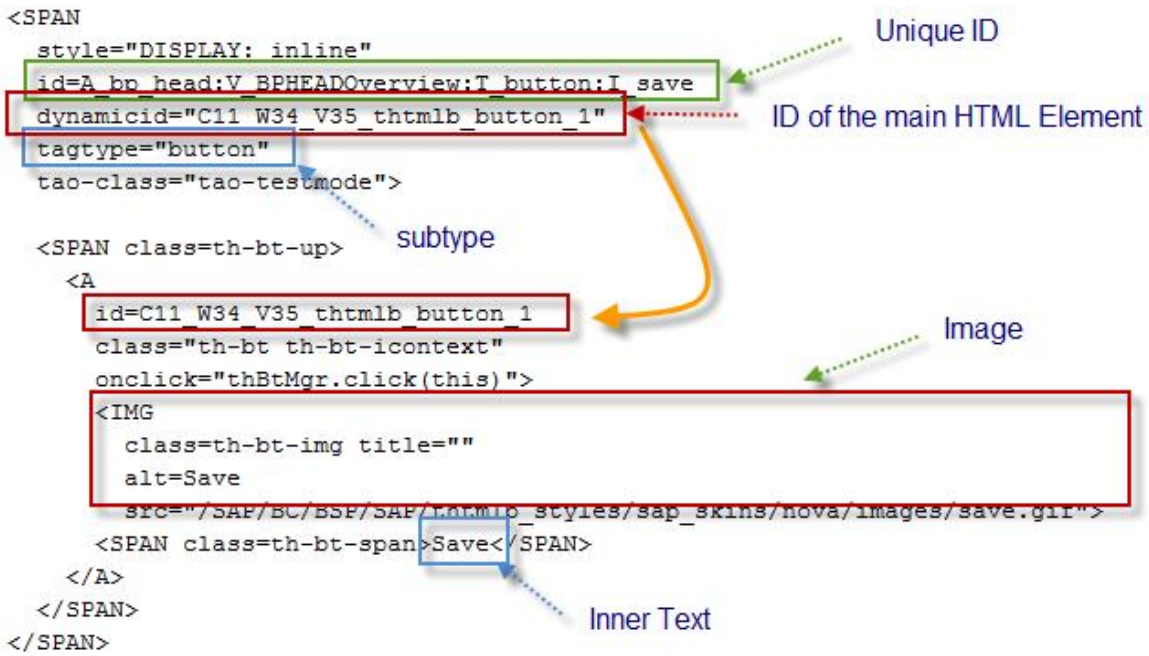


Figure 2: Button Internal Structure

As shown here the internal structure for this simple toolbar is already very complex. A lot of information is available and the Query API is there to hide this complexity. With the Query API the Test Engineers can, for instance, search the “Save” button using the text which is visible to end-user. This information is accessible via the *innerText* HTML property.

2.4 Queries, Filters and Conditions

The query API is meant for searching controls in HTML pages. The objects created with this API have the following characteristics.

- Queries are created using an application controller. Make sure to select the one matching the UI technology being tested.
- The query can have one or more filters.
- Each filter can have one or more conditions.
- Each Filter can have sub-filters.
- Each condition can check the values of the UI element attributes and properties

Query Class Diagram

The UML class diagram below illustrates the relationship between the application controller, the queries, the filters and the conditions.

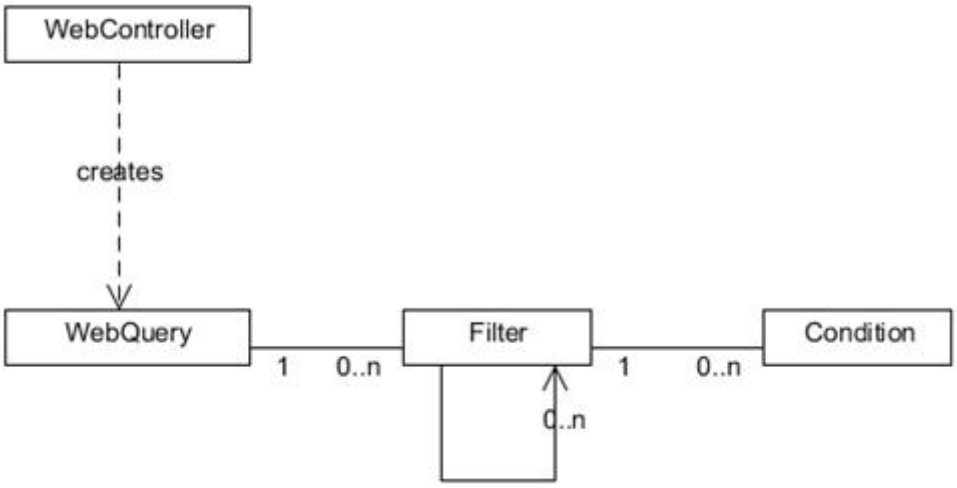


Figure 3: Web Query Class Diagram

Query Life-cycle

Queries have a restricted life-cycle. They are created in a certain context and can only be used in that context. In other words, a query can be used several times but for the current set of HTML documents only. This means that a new query must be created after navigating to another HTML page (i.e.: after each round-trip to the HTTP Server).

2.5 Interface Overview

Controller Interface

The controller is the entry point to the Query API; it lets you create Queries.

Method Name	Description
`CreateQuery`	Creates the object representing the query.

Keep in mind that there is a controller per UI technology. Make sure to select the one matching the element you are looking for.

For more details on controller interfaces refer to:

- Web Controller Interface
- Light Speed Controller Interface
- CRM Controller Interface

Query Interface

The main methods of the query interface are the following:

Method Name	Description
`SetFilter`	Removes all filters and defines the first one
`AddFilter`	Adds an additional filter
`Select`	Executes the query and returns a collection of controls
`SelectSingle`	Executes the query and returns the first control matching the criteria

Query Property	Description
`ParentControlUri`	Property defining the scope where to search for controls

As already mentioned the query interfaces can be specialized to leverage the meta-information that is specific to the underlying UI technology.

For more details on query interfaces refer to:

- Web Query Interface
- Light Speed Query Interface
- CRM Query Interface

Filter and Conditions Interface

The filter holds a list of conditions defining the criteria that the controls have to match.

Method Name	Description
`SetCondition`	Removes all conditions (if any) and defines the first one
`AddCondition`	Adds an additional condition
`SetFilter`	Removes all sub-filters and defines the first one
`AddFilter`	Adds an additional sub-filter

A condition contains information about the criteria that was defined beforehand when using the *SetCondition* method.

Property Name	Description
`AttributeName`	Name of the attribute (or the name of the property) used by this condition
`BooleanOperator`	Operator used when evaluating this condition
`Value`	Expected value of the attribute (or property)
`Option`	Options used when evaluating this condition

For more details on filter and conditions refer to:

- Web Filter Interface
- Web Condition Interface

For more details on attribute and property names refer to:

- HTML Attributes and Properties
- WebCUIF Attributes
- Light Speed Attributes

Web Control Interfaces

A query is meant for searching the HTML content for controls. The query resolution returns one or more controls. These controls implement interfaces providing access to the HTML attributes and properties of the underlying HTML elements.

For more details on control interfaces refer to:

- Web Control Interface
- CRM Control Interface
- Light Speed Control Interface

Keep in mind that a single control can be a composition of several HTML elements and that the API is specialized to provide access to the meta-information via additional interfaces such as:

- CRM Test Mode Control Interface
- Light Speed Relevant Control Interface

3 Concrete Examples

3.1 Example #1 - Contact Creation

The screenshot below shows the CRM Application UI for creating a Contact.

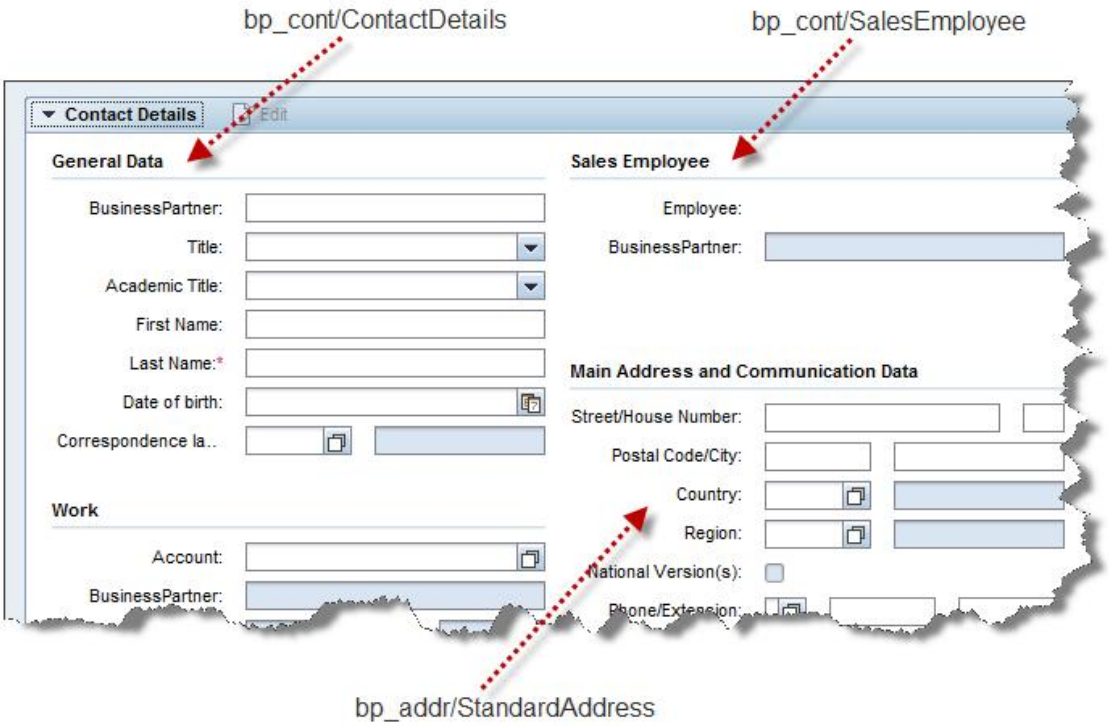


Figure 4: Contact Creation Screenshot

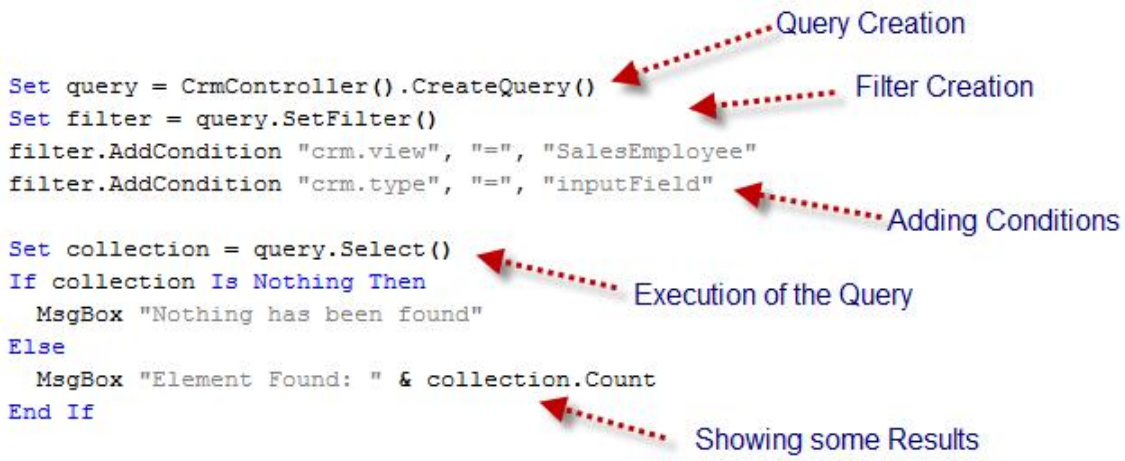
In this example three views are used for entering the contact information.

- bp_cont/ContactDetails
- bp_cont/SalesEmployee
- bp_addr/StandardAddress

The query API can be used to retrieve all the input fields that are part of the *SalesEmployee* view. This is what the following sections explain through concrete scripting examples.

Searching for Web Controls

The script below shows how to create and execute a query.




```
Set query = CrmController().CreateQuery()
Set filter = query.SetFilter()
filter.AddCondition "crm.view", "=", "SalesEmployee"
filter.AddCondition "crm.type", "=", "inputField"

Set collection = query.Select()
If collection Is Nothing Then
  MsgBox "Nothing has been found"
Else
  MsgBox "Element Found: " & collection.Count
End If
```

The VBScript gets access to the HTML document calling the *CrmController method* and creates a query using the *CreateQuery* method.

So far the query is empty. The first thing done here is to specify the criteria identifying the controls we are looking for. This is done using filters and conditions. In our example a single filter but with two conditions is specified. The filter is created using the *SetFilter* method and the two conditions are added using the *AddCondition* method.

Once the criteria are set, the query can be executed using the *Select* or the *SelectSingle* methods.

- The *Select* method returns a collection of CRM web controls matching the criteria.
- The *SelectSingle* method does not return a collection but the first CRM web control matching the criteria.

The next sections describe how to use the result of both the Select and SelectSingle methods though concrete examples.

Setting the Input Field Value

Our first example explained how to create and execute a query. Our second example will show how to use the result and manipulate the retrieved CRM web control object.

```
Dim controller, query, filter, crmControl, controlUri

Set controller = CrmController()
Set query = controller.CreateQuery()

Set filter = query.SetFilter()
filter.AddCondition "crm.view", "=", "SalesEmployee"
filter.AddCondition "crm.type", "=", "inputField"

Set crmControl = query.SelectSingle()
If Not crmControl Is Nothing Then
    controlUri = crmControl.GetControlUri()
    controller.SetElementValue controlUri, "1234"
End If
```

This script invokes the *SelectSingle* method to retrieve a single CRM web control (no need to iterate through a collection in that case). The script checks whether the *crmControl* variable is set before using it and then asks for the *URI* identifying the CRM web control calling the *GetControlUri* method. This call returns a *URI* which is compliant with the ones that the *Runtime Library* expects when using *Default Components*.

With this example, the *URI* identifying the CRM web control would be:

```
crm.area=WorkArea; tag=INPUT; ↵
crm.id=A_bp_cont:V_SalesEmployee:T_inputField:C_salesemployee:I_struct.salesemployee
```

With this *URI*, the script can invoke one of the methods exposed by the CRM web controller API to perform actions that are normally performed using *Default Components*. In our example the script simply sets the value of the Input Field to 1234. It does it by calling the *SetElementValue* method.

3.2 Example #2 - Appointment Creation

The scenarios that we have seen so far are not “dynamic”. Using the Query API is therefore not necessary because the *Default Components* are normally sufficient to automate them.

The next example explains how to select CRM web controls by filtering on the actual values of the INPUT HTML element that the browser displays.

The screenshot below shows the Table HTML element displayed by the CRM Application when creating an *Appointment.* One may notice that CRM UI elements have a different type (and eventually a sub-type) depending on their nature.

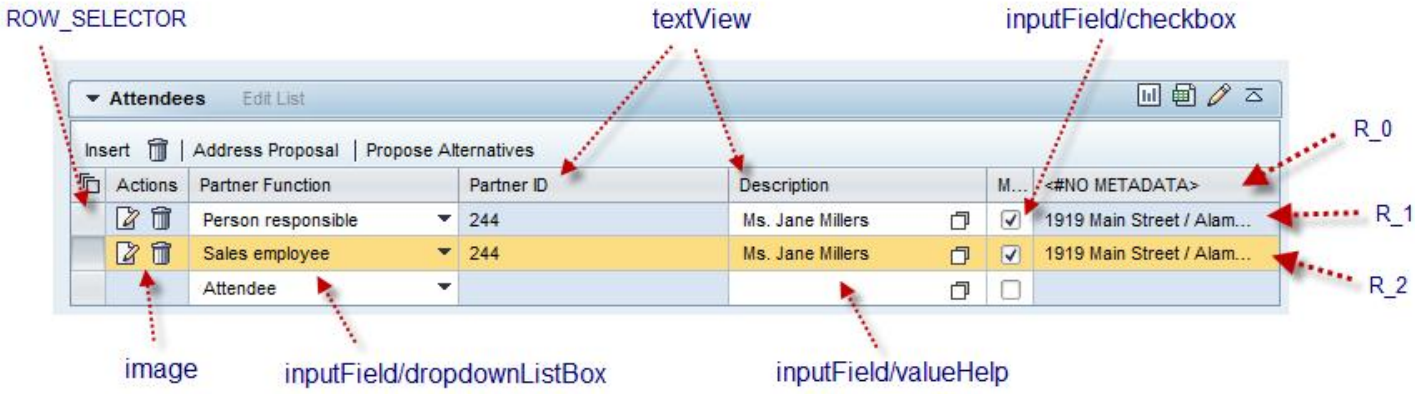


Figure 5: Appointment Creation Screenshot

Searching for Web Controls in a Table

Let's assume that we have to automate a dynamic scenario where the script must select the first row where the *Partner Function* is set to "Sales employee" and *Partner ID* is set to "244"

The difficulty here is that the conditions are to be checked not against the CRM web control being searched but against the CELL content that are in the same row but in different column. This king of query can be very complex when going through the internal hierarchy of HTML elements. To make it simple, the filters and the conditions have been specialized to hide the complexity and handle common use-cases like this one.

Here is the VBScript coding doing the row selection:

```
Dim controller, query, filter, selector, controlUri

Set controller = CrmController()
Set query = controller.CreateQuery()
Set filter = query.SetFilter()
filter.AddCondition "crm.application", "=", "btpartner"
filter.AddCondition "crm.view", "=", "Partner"
filter.AddCondition "crm.context", "=", "btpartner"
filter.AddCondition "crm.type", "=", "ROW_SELECTOR"
filter.AddCondition "crm.column:btpartner.partner_no", "=", "244"
filter.AddCondition "crm.column~Partner Function", "=", "Sales employee"

Set selector = query.SelectSingle()
If Not selector Is Nothing Then
    controlUri = selector.GetControlUri()
    controller.Table_SelectRow controlUri
End If
```

This script shows the different type of conditions that the query supports. We have here some conditions checking the attributes of the CRM web controls. The API let you check the regular HTML Attributes and Properties but also some CRM-specific attributes that make it easy to check the fragments of the CRM IDs.

Using the Object Spy feature (or the developer tool from Internet Explorer) one may see that the CRM ID of the first row selector is:

```
A_btpartner:V_Partner:T_ROW_SELECTOR:C_btpartner:I_:R_1
```

Where the fragment provides meta-information about the CRM web control:

- The *A fragment* specifies the Application
- The *V fragment* specifies the View name
- The *T fragment* specifies the Type
- The *C fragment* specifies the Context Node
- The *I fragment* is the Interface – For Input Fields the Technical Name of field is put there
- The *R fragment* provides the row number

The condition can check the value of these fragments using CRM Attributes:

- The *crm.application* attribute checks the Application
- The *crm.view* attribute checks the View name

- The *crm.type* attribute checks for the UI element type
- etc.

The filtering done below guaranties that the query will only return the row selectors:

```
filter.AddCondition "crm.application", "=", "btpartner"
filter.AddCondition "crm.view", "=", "Partner"
filter.AddCondition "crm.context", "=", "btpartner"
filter.AddCondition "crm.type", "=", "ROW_SELECTOR"
```

We now need to select the second one only (the second visible line). If our scenario was not a dynamic one we could directly select the second row using a Default Component (e.g.: *SelectRow*) and specifying the URI:

```
A_btpartner:V_Partner:T_ROW_SELECTOR:C_btpartner:I_:R_2
```

Because our scenario is dynamic - the row we want to select may not always appear at the second position - we need to search for it by checking the value of the other columns. Two syntaxes are possible when filtering on column values depending on whether you pass the column title or the column technical name.

The syntax can be “crm.colum:<context>.<interface>” or “crm.colum~<Column Title>”.
In other words, the two ways of filtering below are equivalent:

```
filter.AddCondition "crm.column:btpartner.partner_no", "=", "244"
filter.AddCondition "crm.column:btpartner.partner_fct ", "=", "Sales employee"
```

...

```
filter.AddCondition "crm.column~Partner ID", "=", "244"
filter.AddCondition "crm.column~Partner Function", "=", "Sales employee"
```

In our example both syntaxes are being used:

```
filter.AddCondition "crm.column:btpartner.partner_no", "=", "244"
filter.AddCondition "crm.column~Partner Function", "=", "Sales employee"
```

3.4 Tips and Tricks

Iterating through a Collection of Web Controls

The code below shows how to iterate though the collection of CRM web controls retrieved by the execution of a query. The collection implements the Web Control Collection interface.

```
Dim collection
Set collection = query.Select()
If collection Is Nothing Then
  MsgBox "Nothing has been found"
Else
  MsgBox "Element Found: " & collection.Count
  Dim childControl
  for i=0 To collection.Count-1
    Set childControl = collection.ControlAt(i)
    MsgBox "Uri: " & childControl.GetControlUri()
  Next
End If
```

> **Note**
>
> *The VBScript "For Each" statement is not supported. This is a known limitation.*

Making Information available to Subsequent Components

Most of the time only part of the tested scenario is dynamic and requires using custom coding and queries. In such situation it might be useful to make the URI of the retrieved CRM web controls available to subsequent components and continue the remaining part of the scenario using *Default Components*. This is typically done using the *SetToken* method shown in the example below.

```
Dim crmControl, crmControlUri
Set crmControl = query.SelectSingle()
If crmControl Is Nothing Then
  CBTA.Log "Unexpected Situation - Control not found"
Else
  crmControlUri = crmControl.GetControlUri()

  CBTA.Log "Uri: " & crmControlUri
  ExecutionContext.SetToken "rowSelector", crmControlUri
End If
```

Subsequent components can reuse the value stored in the Execution Context via the *%token%* syntax. In this example the subsequent components may use %rowSelector% to perform operations on the *ROW_SELECTOR* that has been selected by the query.

Please refer to the Runtime Library documentation for further details about the *Execution Context*.

Writing Information to the Execution Report

The CBTA (Component Based Test Automation) Object provides convenience methods that are described in the Runtime Library documentation.

- The *CBTA.Log* method can be used for troubleshooting complex scenarios.

  ```
  Public Sub Log(Message)
  ```

- The *CBTA.Report* method is used to provide feedback to the tester via the Execution Report.

  ```
  Public Sub Report(Severity, Topic, Message, Options)
  ```

Please refer to the Runtime Library documentation for further details about the *CBTA API*.

Reducing the Scope of a Query

By default the CRM Queries are automatically searching for CRM web controls in the WorkArea FRAME of the main browser window. The consequence is that the query analyzes the whole content of body of the HTML Document and checks the filters and conditions for each of the CRM web controls that are found.

For performance reasons it might be interesting to reduce the scope of the query and specify where to start searching. For instance when searching for CRM web controls of type ROW_SELECTOR it is recommended to first specify the HTML table element being the parent container. This can be done using the *ParentControlUri* property of the query.

Here is an example:

```
'-------------------------------------------
' Query Example with Restricted Scope
'-------------------------------------------
Dim query, filter, subFilter

Set query = CrmController().CreateQuery()
query.ParentControlUri = "tag=DIV; crm.id=A_btpartner:V_Partner:T_cellerator:C_btpartner:I_"
```

> **Note**
>
> *The URI specified here should be determined using the Object Spy feature. Unfortunately (depending on the style being used) the Table HTML element cannot always be selected. This is a known limitation.*
>
> *The alternative to the Object Spy is to use the Developer Tool (integrated to Internet Explorer) and search for the CRM ID of the "cellerator" UI element.*

Searching for Web Controls in a Popup Window

As already mentioned, the queries are by default searching for CRM web controls in the *WorkArea* FRAME of the main browser window. Here again this default behavior can be change thanks to the *ParentControlUri* property.

The screenshot below shows a concrete example of a CRM Application string with a popup where the Transaction Type must be selected:

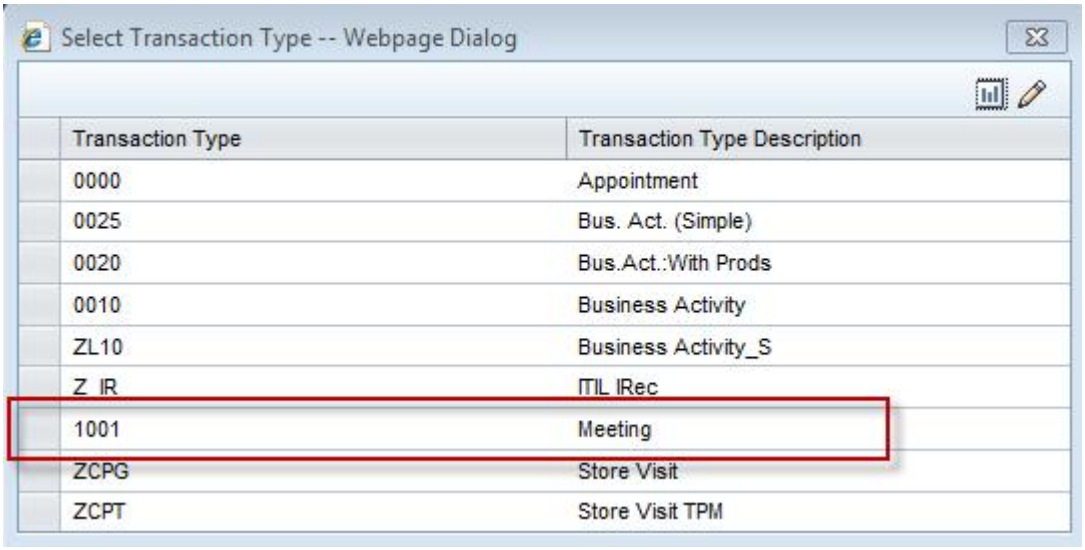


Figure 6: Select Transaction Type Screenshot

Here is the VBScript Example searching for a control in a modal popup window.

```
Dim controller, query, filter

Set controller = CrmController()
Set query = controller.CreateQuery()
query.ParentControlUri = "popupId=1; crm.area=WorkArea; tag=body"

Set filter = query.SetFilter()
filter.AddCondition "crm.type", "=", "ROW_SELECTOR"
filter.AddCondition "crm.context", "=", "proctype"
filter.AddCondition "crm.column:proctype.process_type", "=", "1001", "/i"
filter.AddCondition "crm.column:proctype.proc_type_descr_20", "=", "Meeting"

Dim selectorInPopup, selectorUri
Set selectorInPopup = query.SelectSingle()

selectorInPopup.HighLight "red"
selectorUri = selectorInPopup.GetControlUri()
controller.SearchResult_SelectRow selectorUri
```

In this example there is nothing new regarding the selection of the ROW_SELECTOR. The script uses some conditions to select the Transaction with 1001 and Meeting as *Type* and *Type Description*.

The difference here is that it now specifies that we want to search in the *Work Area* FRAME of the Popup by setting the *ParentControlUri* property:

```
query.ParentControlUri = "popupId=1; crm.area=WorkArea; tag=body"
```

In this example, it would have been possible to restrict the scope of the query even more by specifying the URI of the TABLE HTML element like shown below:

```
query.ParentControlUri = "popupId=1; crm.id=A_btfollowup:V_ProcType:T_cellerator:C_proctype:I_"
```

Combining Filters for Complex Queries

All examples so far were using only one filter with several conditions. One may have noticed that conditions are evaluated one by one and that the CRM Web UI elements are excluded as soon as a condition is not met. In other words, a "*Logical AND*" Operator is being used between conditions.

Our previous example was selecting the first row where:

Partner Function equals "Sales employee"

AND

Partner ID equals "244"

Let's now assume that instead of searching for the ROW_SELECTOR we want to retrieve a collection of controls that are either check-boxes OR input fields associated to an *F4 Help* popup. This can be done by combining several filters like in the example below:

```
Dim query, filter, subFilter, collection, childControl

Set query = CrmController().CreateQuery()
query.ParentControlUri = "tag=DIV; crm.id=A_btpartner:V_Partner:T_cellerator:C_btpartner:I_"

Set filter = query.SetFilter()
filter.AddCondition "crm.application", "=", "btpartner"
filter.AddCondition "crm.view", "=", "Partner"
filter.AddCondition "crm.context", "=", "btpartner"
filter.AddCondition "crm.column:btpartner.partner_no", "=", "244"
filter.AddCondition "crm.column~Partner Function", "=", "Sales employee"

Set subFilter = filter.SetFilter()
subfilter.AddCondition "crm.tagType", "=", "valueHelp"

Set subFilter = filter.AddFilter()
subfilter.AddCondition "crm.tagType", "=", "checkbox"

Set collection = query.Select()

If collection Is Nothing Then
  CBTA.Report CBTA.FAILED, "Demo Query", "No controls have been found", ""
Else
  CBTA.Report CBTA.INFO, "Demo Query", "Nb. Controls: " & collection.Count, ""

  for i=0 To collection.Count-1
    Set childControl = collection.ControlAt(i)
    CBTA.Report CBTA.INFO, "Demo Query",  "Uri: " & childControl.GetControlUri(), ""
  Next
End If
```

In this example the main filter does not check for the ROW_SELECTOR type anymore; as a consequence all cells within the row match its criteria. The difference is now that we have defined two sub-filters where the exact type of the UI elements we are looking for is specified.

The first sub-filter checks for elements that are associated with an *F4 Help* popup using the valueHelp *tagType*.

```
Set subFilter = filter.SetFilter()
subfilter.AddCondition "crm.tagType", "=", "valueHelp"
```

The second one checks for checkbox UI elements.

```
Set subFilter = filter.AddFilter()
subfilter.AddCondition "crm.tagType", "=", "checkbox"
```

It is important to understand that we have here two sub-filters; each of them having a single condition checking for the `crm.tagType` attribute.

- the first sub-filter is created by calling the *SetFilter* method on the main filter object,
- the second one is added thanks to the *AddFilter* method.

By defining several filters we somehow specify that we are looking for cells matching both CRM Tag Types. This is possible because, unlike conditions, the "*Logical OR*" Operator is being used between filters.

Here is the Result in the Execution Report:

Components\SAP CRM\Controls\SearchResult\SelectRow	Demo Query	Element Found: 2
Components\SAP CRM\Controls\SearchResult\SelectRow	Demo Query	Uri: tag=INPUT; crm.id=A_btpartner:V_Partner:T_inputField:C_btpartner:I_partner_no_descr:R_2
Components\SAP CRM\Controls\SearchResult\SelectRow	Demo Query	Uri: tag=INPUT; crm.id=A_btpartner:V_Partner:T_inputField:C_btpartner:I_mainpartner:R_2
		Execution Traces Debug Log Log Folder

Reusing Default Component Implementations

The controller interfaces also expose methods for interacting with the UI of the application. There is (at least) a method per *Default Component* that CBTA delivers.

The two examples below are somehow equivalent:

```
CRM_Table_SelectRow crmControl.GetControlUri()

If ReportStatus = "FAILED" Then
  ' Nothing to do
End If
```

...

```
Set operationResult = CrmController().Table_SelectRow crmControl.GetControlUri()

If operationResult.Status = "FAILED" Then
  CBTA.Report CBTA.FAILED, "Demo Query", "Row selection failed", ""
Else
  CBTA.Report CBTA.INFO, "Demo Query", "Row has been selected", ""
End If
```

The first example invokes the component implementation while the second one invokes the corresponding method via the interface of the controller. The advantage of the first approach is that it benefits from a default exception handling (writing information in the execution report when an error occurs). In the second example the script is responsible for checking whether the operation succeeded or not.

> **Note**
>
> *In general, the public methods of the controller interfaces do return an Operation Result Object implementing the Operation Result Interface. The information available there can be used to provide a human-readable feedback in the execution report.*
>
> RELATED LINK:
>
> - *Writing Information to the Execution Report*

Mapping between Public Methods and Component Implementations

The list below shows a few examples of methods (exposed by the controller interfaces) and the corresponding component implementation.

Controller	Method Name	Component Implementation (VB Script Function)
CrmController()	CaptureScreen	CRM_CaptureScreen
CrmController()	WebControl_Click	CRM_WebControl_Click
CrmController()	SetElementValue	N/A
CrmController()	Table_SelectRow	CRM_Table_SelectRow
CrmController()	SearchResult_SelectRow	CRM_SearchResult_SelectRow
WebController()	CaptureScreen	WEB_CaptureScreen
WebController()	WebControl_SetValue	WEB_WebControl_SetValue
WebController()	WebControl_CheckProperty	WEB_WebControl_CheckProperty

3.5 Example #3 – Finding a Row by Content

CBTA comes with Default Components that allow searching tables for a row by the content of one of the cells.

With CBTA 3.0 SP2 the following components are available:

Component Name	Description
`CBTA_LS_T_FindRow`	This component is meant for searching a Light Speed Tables for a row by checking the value of a single cell. INPUT PARAMETERS: URI ➔ Specifies the Uniform Resource Identifier of the table container ColumnTitle ➔ The title of the column (the visible text) Operator ➔ Boolean operator used to compare the actual value with the expected one Content ➔ The expected value of the cell Options ➔ Some options can be used to convert the values before comparing them. OUTPUT PARAMETER: The number of the row matching the criteria.
`CBTA_CRM_T_FindRow`	Idem as CBTA_LS_T_FindRow but for SAP CRM Web Applications

It is important to understand that both components have been implemented using the Query API that is described in this document. This means that a Test Engineer can copy/paste the source code of the component implementation and adapt it to customer-specific needs.

> **Note**
>
> *The code of the* CBTA_LS_T_FindRow *is part of the Runtime Library; it is visible in the* LS_Functions.vbs *file.*
>
> *The Runtime Library Manager must be used for customizing the Runtime Library and write custom functions. When opening the Runtime Library for edition the* LS_Functions.vbs *file is made available locally (on the file system) at the location specified by the Test Engineer.*

VB Script Function - LsTable_FindRowByContent

```
Public Function LsTable_FindRowByContent ( Uri, ColumnTitle, Operator, Content, Options )
  On Error Resume Next
  EventWebComponentBegin()
  LsTable_FindRowByContent = LsTable_FindRowByContent_Impl( Uri, ColumnTitle, Operator, _
                                                            Content, Options )
  EventWebComponentEnd()
End Function

Public Function LsTable_FindRowByContent_Impl ( Uri, ColumnTitle, Operator, Content, Options )
  LsTable_FindRowByContent_Impl = ""

  If IsNull(Uri) Or IsNull(ColumnTitle) Or IsNull(Content) Then
    Exit Function
  End If
  If IsNull(Operator) Then
    Operator = "="
  End If
  If IsNull(Options) Then
    Options = ""
  End If

  ReportDebugLog "LsTable_FindRowByContent" & _
                   vbCrLf & "Uri: " & Uri & _
                   vbCrLf & "ColumnTitle: " & ColumnTitle & _
                   vbCrLf & "Operator: " & Operator & _
                   vbCrLf & "Content: " & Content & _
                   vbCrLf & "Options: " & Options

  Dim controller, query, filter
  Set controller = LsController()
  Set query = controller.CreateLsQuery()
  query.ParentControlUri = uri

  Set filter = query.SetFilter()
  filter.AddCondition "tag", "=", "TD"
  filter.AddCondition "ls.subtype", "=", "SC"
  filter.AddCondition "ls.column~" & ColumnTitle, Operator, Content, Options

  Set control = query.SelectSingle()
  If control Is Nothing Then
    ReportLog "FAILED", "LsTable_FindRowByContent", "Operation Failed - Row could not be found"
  Else
    LsTable_FindRowByContent_Impl = control.GetRelevantControl().GetLsRow()

    If InStr(Options, "/Select" ) Then
      ReportLog "INFO", "LsTable_FindRowByContent", _
        "Operation succeeded - Row has been selected" & vbCRLf & _
        "Row is: " & LsTable_FindRowByContent

      ReportDebugLog "LsTable_FindRowByContent - Row Selector Uri: " & control.GetControlUri()

      WEB_WebControl_Table_SelectRow control.GetControlUri()
    Else
      ReportLog "INFO", "LsTable_FindRowByContent", _
                "Operation succeeded - Row is: " & LsTable_FindRowByContent
    End If
  End If

End Function
```

LsTable_FindRowByContent Explained

The following sections are explaining in detail how this component has been implemented.

Exception Handling

The actual implementation consists of two functions. The first function is there only because the VB Script syntax is quite poor regarding the exception handling and the "*On Error Resume Next*" keywords (used here) are the only option available for us.

Note that all default components are built using the same concept; the first simply surrounds the invocation of the real implementation to make sure to properly intercept potential errors.

```
Public Function LsTable_FindRowByContent ( Uri, ColumnTitle, Operator, Content, Options )
  On Error Resume Next
  EventWebComponentBegin()
  LsTable_FindRowByContent = LsTable_FindRowByContent_Impl( Uri, ColumnTitle, Operator, _
                                                            Content, Options )
  EventWebComponentEnd()
End Function
```

Validating Input Parameters

The first statements of the implementation are there for validating the input parameters and thus avoid common issues.

```
Public Function LsTable_FindRowByContent_Impl ( Uri, ColumnTitle, Operator, Content, Options )
  LsTable_FindRowByContent_Impl = ""

  If IsNull(Uri) Or IsNull(ColumnTitle) Or IsNull(Content) Then
    Exit Function
  End If
  If IsNull(Operator) Then
    Operator = "="
  End If
  If IsNull(Options) Then
    Options = ""
  End If

...
End Function
```

In this example the URI, some parameters are mandatory, some have a default value. The script checks all of them and reacts accordingly.

Writing Information to the Execution Report

The purpose of the code below is just to provide some feedback to the Test Engineer. The value of each input parameter is written to the Execution Report.

```
ReportDebugLog "LsTable_FindRowByContent" & _
                    vbCrLf & "Uri: " & Uri & _
                    vbCrLf & "ColumnTitle: " & ColumnTitle & _
                    vbCrLf & "Operator: " & Operator & _
                    vbCrLf & "Content: " & Content & _
```

```
                    vbCrLf & "Options: " & Options
```

Creation of the Query restricted to a Table Container

When the input parameters are consistent, a query is created using the Light Speed Controller and the scope of the query is restricted to the table by setting the *ParentControlUri* property like shown below.

```
Dim controller, query, filter
Set controller = LsController()
Set query = controller.CreateLsQuery()
query.ParentControlUri = uri
```

Definition of the Criteria

As already mentioned, our goal here is to find a certain row by checking the content of one of the cells. This is done thanks to a single filter with 3 conditions.

```
Set filter = query.SetFilter()
filter.AddCondition "tag", "=", "TD"
filter.AddCondition "ls.subtype", "=", "SC"
filter.AddCondition "ls.column~" & ColumnTitle, Operator, Content, Options
```

The first condition specifies here the tag used by the Light Speed Framework to generate a cell. Note that the regular <TD> HTML tag is being used here.

> **Note**
>
> *This condition is not mandatory but filtering out the other tags as here a significant impact on performances since the number of UI elements matching this condition might be very low.*

The second condition relies on a Light Speed Data to determine the sub-type of control. The Light Speed Framework marks all cells with the "SC" sub-type whatever the actual type is (input field, checkbox, etc...)

For more information on data that are specific to the Light Speed Framework refer to:

- Light Speed Attributes

The third condition is the most important one; it uses the ColumnTitle parameter that has been specified to search for the cell displayed in the column we are interested in. This condition also uses the Operator, Content and Options parameters to define the value that we expect to get from the cells.

For more details on the possible values for the operator and the options refer to:

- Boolean Operators
- Options for Conditions

Resolving the Query

The criteria have been defined; the query is now ready for being executed. The code below triggers the query resolution.

```
Set control = query.SelectSingle()
If control Is Nothing Then
    ReportLog "FAILED", "LsTable_FindRowByContent", "Operation Failed - Row could not be found"
Else
    LsTable_FindRowByContent_Impl = control.GetRelevantControl().GetLsRow()
    ...

End If
```

Note that by calling the *SelectSingle* method the *CBTA_LS_T_FindRow* component only searches for a single row (i.e. the first row matching the criteria).

It would have been possible to retrieve a collection of cells matching the criteria by calling the *Select* method instead.

Returning the Row Number

The code below gets the row number by using some Light Speed Data that are made available via interfaces that are specific to the underlying UI technology.

For more details on Light Speed interfaces refer to:

- Light Speed Relevant Control Interface

```
    LsTable_FindRowByContent_Impl = control.GetRelevantControl().GetLsRow()
    ...
```

> **Note**
>
> *The CBTA_LS_T_FindRow returns the row number via its output parameter.*
>
> *The internal method used to set the output parameter is out of the scope of this document (and not shown here).*

Selecting the Row

The *CBTA_LS_T_FindRow* component does not only search for the row. It also provides the ability to select it. To achieve this, the script has to ask for the URI of the cell matching the criteria and then trigger the row selection.

- The URI is retrieved by calling the following *GetControlUri* method
- The selection is made by calling the WEB_WebControl_Table_SelectRow function which corresponds to the *CBTA_WEB_SelectRow* Default Component.

Note that in the example below the selection is only made when the Options parameter contains “/Select”

```
If InStr(Options, "/Select" ) Then

  ReportLog "INFO", "LsTable_FindRowByContent", _
    "Operation succeeded - Row has been selected" & vbCRLf & _
    "Row is: " & LsTable_FindRowByContent

  ReportDebugLog "LsTable_FindRowByContent - Row Selector Uri: " & control.GetControlUri()
  WEB_WebControl_Table_SelectRow control.GetControlUri()

Else
  ReportLog "INFO", "LsTable_FindRowByContent", _
            "Operation succeeded - Row is: " & LsTable_FindRowByContent
End If
```

4 Appendix

4.1 Attribute and Property Names

HTML Attributes and Properties

The condition can be used for checking the regular HTML Attributes and Properties. The following attribute names are supported:

Attribute Name	Description
`id`	Returns the id of the HTML element
`tag`	Returns the tag of the HTML element
`innerText (*)`	Returns the value of the *innerText* property of the HTML element
`value (*)`	Returns the value attribute of the HTML element
`readOnly (*)`	Returns the value of the *readOnly* attribute as a Boolean
`disabled (*)`	Returns the Boolean value of the *disabled* attribute as a Boolean
`class`	A list of styles that are applied to the HTML element.
`className`	Idem as class
`href`	Returns the "*href*" attribute of a link (i.e. of the anchor HTML element).

 Caution

(*) Note that depending on the UI technology, some HTML attributes may not reflect the visual aspect of the web controls.

For instance, the UI Technologies do not necessarily rely on disabled HTML attribute for enabling/disabling a control. This means that the disabled attribute is meaningless. In such situation another attribute (i.e.: an attribute specific to the underlying UI technology) may provide the information.

WebCUIF Attributes

The WebCUIF Attributes that can be used when testing CRM applications are all starting with the “crm.” prefix. The following attributes are supported:

Attribute Name	Description
`crm.value`	Returns the value of the CRM web control. The underlying HTML attribute used to retrieve the information may vary depending on the UI element type. For instance, the *value* HTML attribute is returned for input HTML elements but for texts and labels the *innerText* HTML attribute is used instead.
`crm.id`	Returns the CRM ID of the CRM web control. This ID is unique within the HTML content. It has a CRM Specific syntax and provides the following information: • The *A fragment* specifies the Application • The *V fragment* specifies the View name • The *T fragment* specifies the Type • The *C fragment* specifies the Context Node • The *I fragment* is the Interface (optional) – for Input Fields the Technical Name of the field is put there • The *R fragment* provides the row number (optional) • The K *fragment* provides the key (optional)
`crm.type`	Returns the type of the CRM web control. Note that this type corresponds to the T_ fragment of the CRM ID. The type is independent from the internal HTML Tag used to render the UI element. For example, the *inputField* type is used for checkboxes.
`crm.tagType`	Returns the tag type (a.k.a. as a subtype) depending on the UI element type. CRM web controls of type “inputField” may have their *tagType* set to: • *checkbox* – for checkbox controls • *dropdownListBox* – for combobox controls • *valueHelp* – for input fields decorated with a button opening the F4 help.
`crm.application`	Returns the A_ fragment of the CRM ID.
`crm.view`	Returns the V_ fragment of the CRM ID.
`crm.context`	Returns the C_ fragment of the CRM ID.
`crm.interface`	Returns the I_ fragment of the CRM ID.
`crm.row`	Returns the R_ fragment of the CRM ID (if any).
`crm.key`	Returns the K_ fragment of the CRM ID (if any).
`crm.column.title`	Returns the title of the column containing the CRM web control. This of course only makes sense when the R_ fragment exists.
`crm.wrongName`	Returns [wrongName] - this to highlight situations where an unexpected attribute name is passed as input parameter.
`crm.column:<field>`	Returns the value of the cell containing the field specified with the <field> fragment. The <field> fragment provides the technical name identifying a field being displayed by the cell. It must be of the form: • <context>.<interface>

Attribute Name	Description
`crm.column~<title>`	Returns the value of the cell displayed by the column having <title> as visible text in its column header.

Light Speed Attributes

Applications based on the Unified Rendering Light Speed framework generate controls that are including some meta-information. This information can be used by queries when searching for controls.

Attribute Name	Description
`ls.type`	Returns the type of the control
`ls.subtype`	Returns the subtype of the control (if any)
`ls.id`	Returns the id of the HTML element that represents the best the current control
`ls.value`	Returns the value of the HTML element that represents the best the current control
`ls.tag`	Returns the tag of the HTML element that represents the best the control

Web Dynpro ABAP Attributes

The attributes listed below are available only when testing applications developed using Web Dynpro ABAP. They provide further information about the fields such as the view they belongs to.

Attribute Name	Description
`ls.component`	Returns the name of the component the control belongs to
`ls.application`	(alias) Idem as ls.component
`ls.view`	Technical name of the view the control belongs to
`ls.field`	Technical name of the field in the corresponding view
`ls.row`	Row number (when the control is embedded within a table)
`ls.column.title`	Returns the title of the column containing the control. This only makes sense when the control is embedded within a table.
`ls.column:<field>`	Returns the value of the cell containing the field specified with the <field> fragment. The <field> fragment provides the technical name identifying a field being displayed by the cell.
`ls.column~<title>`	Returns the value of the cell displayed by the column having <title> as visible text in its column header.

Light Speed Data

Light Speed Controls have an HTML attribute providing some additional meta-information – the "*ls.data*" attribute.

This information is parsed and made available to the test framework as Light Speed attributes that can be used when defining conditions.

The exhaustive list of *Light Speed* attributes is not provided here because each control may have a different set of data (depending on its nature). One may use the *Object Spy* feature to spy for a control and see the data that is available.

Common Light Speed Attributes

Attribute Name	Description
`ls.data.value`	Returns the value of the control.
`ls.data.state`	Returns the state of the control. For instance, checked for a checkbox.
`ls.data.enabled`	Returns whether the control is enabled or not (possible values: true/false).
`ls.data.readonly`	Returns whether the control value can be modified or not (possible values: true/false).
`ls.data.text`	Returns the text of the control. Note that this is most of the time similar to the “innerText” HTML attribute. However, when the control is complex, the innerText attribute may return a multiline value (including some carriage returns characters).
`ls.data.tooltip`	The tooltip associated to the control.

4.2 Boolean Operators

The following operators are the ones that the Query API supports when defining conditions.

Boolean Operator	Description
=	Equals
==	Idem as =
<>	Not Equals
!=	Idem as <>
<	Is Lower Than
<=	Is Lower or Equal
>	Is Greater Than
>=	Is Greater Or Equals
{contains}	Contains
{startsWith}	Starts With
{endsWith}	Ends With
{matches}	Value matches a regular expression. The regular expressions are expressed using the C# syntax.

4.3 Options for Conditions

For most of the attributes, the conditions compare operand values as strings and the comparison is case-sensitive like in the example below.

```
filter.AddCondition "crm.column~Partner Function", "=", "Sales employee"
```

The script may need to do a case-insensitive comparison. In that case an additional parameter is used to specify options when declaring the condition.

Example for a case-insensitive comparison:

```
filter.AddCondition "crm.column~Partner Function", "=", "SaLeS EmPlOYee", "/u"
```

With the /u option both values are converted to uppercase before comparing them.

The options supported are listed below

Option	Description
`/u (for uppercase)`	Non case-sensitive comparison. Both operand values are converted to uppercase before comparing them.
`/t (for trimmed)`	Both operand values are trimmed to remove heading and trailing spaces before comparing them.
`/b (for Boolean)`	Both operand values are converted to a *Boolean* value (true/false) before comparing them. This option is implicit for attributes that are Boolean by nature such as: readOnly disabled
`/i (for integer)`	Both operand values are converted to an *Integer* before comparing them. The value is not truncated; the value 10.50 is for instance converted to 11.
`/f (for float)`	Both operand values are converted to a float before comparing them.

5 Appendix – Technical Information

5.1 Web Query API

Interfaces

Web Controller Interface

The web controller interface is the main entry point for testing web applications.

It exposes the following methods:

Method Name	Description
`CreateQuery`	Creates a query object for searching the HTML content for Web controls
`CreateWebQuery`	Idem as CreateQuery method of the w**eb control**ler.
`GetWebControlByUri`	Resolves the URI parameter and returns the corresponding web control (if any)
`GetElementByUri`	Resolves the URI parameter and returns the corresponding HTML element (if any)

Web Query Interface

The Web Query Interface is the main interface of all query objects.

It exposes the following methods and properties:

Method Name	Description
`SetFilter`	Removes all filters and defines the first one INPUT PARAMETERS: No parameters ➔ Use the SetCondition and AddCondition methods to define the criteria.
`AddFilter`	Adds an additional filter. INPUT PARAMETERS: No parameters ➔ Use the SetCondition and AddCondition methods to define the criteria.
`SelectSingle`	Resolves the query and returns the first web control that matches the criteria.
`Select`	Resolves the query and returns a collection of web controls. INPUT PARAMETERS: maxHits ➔ (optional) Specifies the max number of control to return. RELATED LINKS: • Iterating through a Collection of Web Controls
`GetMainControl`	Resolves the *ParentControlUri* (if specified) and returns the corresponding web control.

It exposes also expose the following properties:

Property Name	Description
`ParentControlUri`	Property defining the scope where to search for controls. Its value is the URI of the parent container where to search. When left empty the query resolution searches the main document of the main page. RELATED LINK: • Reducing the Scope of a Query • Searching for Web Controls in a Popup Window

Web Filter Interface

A filter object has the following capabilities:

- It holds a list of conditions defining the criteria that the controls have to match.
- It can have sub-filters defining additional criteria.

Method Name	Description
`SetCondition`	Removes all conditions (if any) and defines the first one. INPUT PARAMETERS: AttributeName ➔ specifies the attribute name (or the property name) BooleanOperator ➔ specifies the operator to used when evaluation the condition Value ➔ specifies the expected value. Options ➔ specifies some conversion options RELATED LINKS: • Attribute and Property Names • Boolean Operators • Options for Conditions
`AddCondition`	Adds an additional condition. INPUT PARAMETERS: Same parameters than the SetCondition option.
`RemoveAllConditions`	Remove all conditions (if any)
`SetFilter`	Removes all sub-filters and defines the first one INPUT PARAMETERS: No parameters ➔ Use the SetCondition and AddCondition methods to define the criteria. RELATED LINK: • Combining Filters for Complex Queries
`AddFilter`	Adds an additional sub-filter INPUT PARAMETERS: No parameters ➔ Use the SetCondition and AddCondition methods to define the criteria.

Method Name	Description
`RemoveAllFilters`	Remove all filters (if any)

Web Condition Interface

Conditions are defined using the *SetCondition* and the *AddCondition* methods of the *Web* Filter interface. The condition object that is created exposes the following properties.

Property Name	Description
`AttributeName`	Name of the attribute (or the name of the property) used by this condition
`BooleanOperator`	Operator used when evaluating this condition
`Value`	Expected value of the attribute (or property)
`Option`	Options used when evaluating this condition

Web Control Interface

The Web Control Interface is meant for providing access to the HTML attributes and properties of the underlying HTML element.

It exposes the following methods:

Method Name	Description
`GetHtmlElement`	Returns the underlying HTML element. The object returned here implements Microsoft interfaces depending on its nature.
`GetControlUri`	Returns the URI identifying the control in the current HTML content. This URI can be used when calling a Default Component Implementation to simulate user actions such as a mouse click.
`GetAttribute`	Returns the value of an HTML attribute of the underlying HTML element. INPUT PARAMETER: AttributeName ➔ Name of the attribute (i.e.: value)
`GetProperty`	Returns the value of an HTML property of the underlying HTML element. INPUT PARAMETER: PropertyName ➔ Name of the property (e.g.: innerText)
`HighLight`	Used for highlighting the HTML element at runtime (for troubleshooting purposes) INPUT PARAMETER: ColorName➔ Name of the color used when highlighting the control (e.g. green, red, blue, etc…)

Method Name	Description
	property (e.g.: innerText)
`FindFirstChildElement`	Searches for a child control. INPUT PARAMETERS: TagName ➔ HTML Tag to search for (e.g.: SPAN) Recurse ➔ (optional) True to search recursively (in the HTML DOM hierarchy of HTML elements)
`FindLastChildElement`	Searches for the latest child control. INPUT PARAMETERS: TagName ➔ HTML Tag to search for (e.g.: SPAN) Recurse ➔ (optional) True to search recursively (in the HTML DOM Hierarchy of HTML elements)
`FindChildElementAtPosition`	Searches for a child control at a certain position (index). INPUT PARAMETERS: TagName ➔ HTML Tag to search for (e.g.: SPAN) Index ➔ The position number in the collection of children Recurse ➔ (optional) True to search recursively (in the HTML DOM hierarchy of HTML elements)
`FindChildElements`	Returns a collection of child controls. INPUT PARAMETERS: TagName ➔ HTML Tag to search for (e.g.: SPAN) Recurse ➔ (optional) True to search recursively (in the HTML DOM Hierarchy of HTML elements)

Web Control Collection Interface

The collection of web controls is the result what results the query resolution when calling the *Select* method.

The test script can iterate through the collection using the following methods:

Method Name	Description
`ControlAt`	Returns the web control at the position specified INPUT PARAMETERS: Index ➔ The position number in the collection of web controls RELATED LINKS: • Iterating through a Collection of Web Controls

Property Name		Description
`Count`	Read only	Number of controls available in the collection

Operation Result Interface

Interface providing results when calling default component implementation.

Property Name		Description
`Status`	Read only	`DONE, PASSED, WARNING, FAILED`
`Feedback`	Read only	Message providing feedback about the operation performed
`Value`	Read only	Output value of the operation
`Comment`	Read only	Additional information (optional)
`ImageTitle`	Read only	Title of the screenshot (if any)
`ImagePath`	Read only	Path to the screenshot (if any)
`OutputParameters`	Read only	Collection of output parameters

5.2 Interfaces Specialized for WebCUIF

WebCUIF Interfaces

CRM Controller Interface

The controller interface for CRM exposes the following methods.

Method Name	Description
`CreateCrmQuery`	Creates a query object for searching the HTML content for CRM controls
`CreateQuery`	Idem as *CreateCrmQuery* method when using this CRM Controller.
`CreateWebQuery`	Idem as CreateQuery method of the web controller.
`GetWebControlByUri`	Resolves the URI parameter and returns the corresponding CRM Control (if any)

CRM Query Interface

The CRM Query interface exposes the same methods then the Web Query interface that it extends. The only difference is that the query result returns WebCUIF controls (instead of regular web controls).

CRM Control Interface

The CRM Control interface extends the web control interface. In other words it inherits from the web control interface capabilities.

It exposes the additional methods listed below:

Method Name	Description
`GetTestModeControl`	Returns the control providing meta-information about the CRM Control. This control is in general a hidden SPAN element that the WebCUIF framework inserts in the HTML content when CRM applications are started in Test Mode.
`GetTestModeElement`	Returns the SPAN element providing meta-information about the CRM control.

CRM Test Mode Control Interface

This interface provides access to the meta information that the WebCUIF framework inserts into the HTML content when running CRM application in test mode.

Method Name	Description	Equivalent when filtering
`GetCrmId`	Returns the ID of the control	`crm.id`
`GetCrmType`	Returns the type of the control (T_ fragment of the ID)	`crm.type`
`GetCrmTagType`	Returns the subtype of the control. As defined by the tagType HTML attribute (if any)	`crm.tagType`
`GetApplication`	Returns the application the control belongs to (A_ fragment of the ID)	`crm.application`
`GetViewName`	Returns the view the control belongs to (V_ fragment of the ID)	`crm.view`
`GetCrmContext`	Returns the context the control belongs to (C_ fragment of the ID)	`crm.context`
`GetCrmInterface`	Returns the technical name of the control (I_ fragment of the ID)	`crm.interface`
`GetCrmRow`	Returns the row number (when embedded within a cell in a Table)	`crm.row`
`GetCrmKey`	Returns the key of the control (when child item of a dropdown list box)	`crm.key`

5.3 Interfaces Specialized for Light Speed

Light Speed Interfaces

Light Speed Controller Interface

The controller interface for Light Speed exposes the following methods.

Method Name	Description
`CreateLsQuery`	Creates a query object for searching the HTML content for Light Speed controls INPUT PARAMETER: Options ➔ Activates some default filtering on common attributes to restrict the number of elements found when executing the query. • /all – returns all HTML elements that are child of a Light Speed control. This is the default value. • /ls – returns all Light Speed elements – i.e.: elements having a **ct** attribute or a **subct** attributes. • /ct – returns all Light Speed elements having a **ct** attribute. • /subct - returns all Light Speed elements having a **subct** attribute. • /web – returns all HTML elements (even the one that are not child of a Light Speed control) The option set here may have a huge impact on performance when executing a query.
`CreateQuery`	Idem as *CreateLsQuery* method when using this **Light Speed** Controller.
`CreateWebQuery`	Idem as CreateQuery method of the **web control**ler.
`GetWebControlByUri`	Resolves the URI parameter and returns the corresponding WebCUIF control (if any)

Light Speed Query Interface

The Light Speed Query interface exposes the same methods then the web query interface that it extends. The only difference is that the query result returns objects that implement the Light Speed controls instead of web controls.

Light Speed Control Interface

The Light Speed Control interface extends the web control interface. In other words it inherits from the web control interface capabilities.

It exposes the additional methods listed below:

Method Name	Description
`GetRelevantControl`	Returns the so called “relevant” control. This control is the one having HTML attributes (such as the ct or the subct attributes) that define the control type.
`GetRelevantElement`	Returns the “relevant” element. This is the underlying HTML element of the Relevant Control. This element is the one having HTML attributes (such as the ct or the subct attributes) that define the control type.

Light Speed Relevant Control Interface

The relevant control interface is only there to make it easier to access to some meta-information that the Light Speed framework provides.

Method Name	Description	Equivalent when filtering
`GetLsId`	Returns the ID of the control	`ls.id`
`GetLsTag`	Returns the tag of the underlying HTML element	`ls.tag`
`GetLsType`	Returns the type of the control	`ls.type`
`GetLsSubtype`	Returns the subtype of the control	`ls.subtype`
`GetLsComponent`	Returns the component of the control (Web Dynpro ABAP Only)	`ls.component`
`GetLsViewName`	Returns the view of the control (Web Dynpro ABAP Only)	`ls.view`
`GetLsFieldName`	Returns the name of the control (Web Dynpro ABAP Only)	`ls.field`
`GetLsRow`	Returns the row number (when embedded within a cell in a Table)	`ls.row`

6 References

6.1 Related Links

Portals where to find additional information about CBTA:

Title	Link
SAP Support Portal	https://service.sap.com/support ➢ Media Library ➔ Documentation ➔ Independent
SAP Community Network	http://scn.sap.com ➢ Search for CBTA
Wiki Page	http://wiki.scn.sap.com/wiki/display/SM/CBTA

6.2 List of Materials and Documentations

Official materials and documentations:

Title	Link
CBTA – Default Components (EN)	https://service.sap.com/~sapidb/011000358700001306982012E
CBTA - How-To Guide	https://service.sap.com/~sapidb/011000358700000049262013E
CBTA – Test Automation Demo	https://service.sap.com/~sapidb/011000358700000062492013E
Test Management – Application Lifecycle Management Processes	https://service.sap.com/~sapidb/011000358700000697562009E

Documents mentioned in this guide:

Title	Location
CBTA – Runtime Library Manager – CBASE Customization	Wiki Page for CBTA ➢ Search for CBTA

6.3 List of SAP Notes

The following table list the SAP Notes mentioned in this document.

SAP Note	Title
1778899	CBTA - Collective Note
1926307	CBTA – Release Note for version 3.0 SP2
1912801	CBTA – RTL Manager – Runtime Library Customization
1666201	CRM – Collective Note for running CRM Application in Test Mode

www.sap.com/contactsap