blueprism

The Operational Agility Software Company

Blue Prism

Browser Automation Guide

info@blueprism.com • +44 (0)870 879 3000 • Centrix House, Crow Lane East, Newton-le-Willows, WA12 9UY

Revision History

Date	Revision	Author	Description
02/03/2015	1.0	JT	Document created
06/07/2016	1.1	JT	More information on HTML attributes and Java applications

Published by:

Blue Prism Limited
Centrix House
Crow Lane East
Newton-le-Willows
WA12 9UY, UK
Registered in England; Reg. No. 4260035
www.blueprism.com
Tel: 0870 879 3000

Contents

1 Introduction

The purpose of this document is to act as a useful source of advice and various helpful tips to Blue Prism developers who are attempting to automate browser applications. This document can be referred to when attempting to troubleshoot issues encountered when automating browser applications. It is important to bear in mind that only Internet Explorer is supported by Blue Prism.

2 Launching and Attaching

In this chapter we will discuss some useful techniques that can be used to launch and attach to browser applications.

It can often be difficult to reliably launch and attach to Internet Explorer sessions due to a feature in IE that allows IE sessions to run in separate Windows processes which are spawned from a primary IE process. The below tips can be used to work around this.

2.1 Child Index

Blue Prism version 4.1.3 and above includes a feature which allows you to specify the Child Index of the process to which to attach. Each session is given a unique index based on the order in which the process for that session was created. For example the main parent process is given the index of 0, whereas the first tab of the process is given the Child Index of 1. The Child Index parameter is specified in the attach action of a navigate stage.

2.2 TabProcGrowth

If the above is not appropriate for your means, as an alternative you can configure Internet Explorer to not create multiple processes. This is done by changing a registry setting, specifically the registry key:

```
HKEY_CURRENT_USER\Software\Microsoft\Internet Explorer\Main
```

A DWORD value called TabProcGrowth should be set to one of the following values:

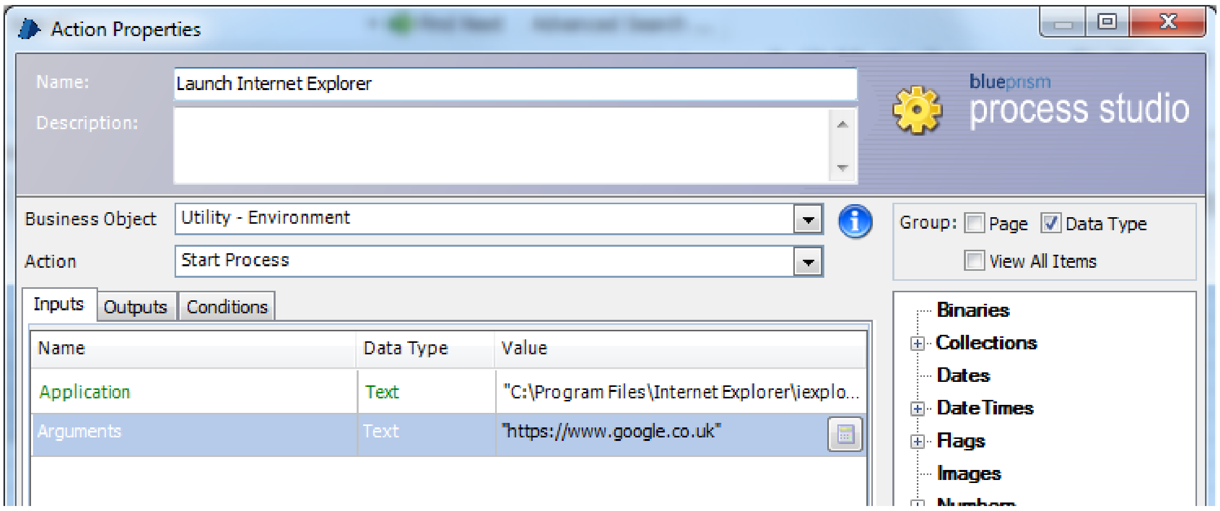
- 0 (Zero) – This disables protected mode for IE Security Zones and tells Internet Explorer to open tabs and frames within the same process.

- 1 (One) - This leaves protected mode enabled, and ensures that all tabs in a given frame run in a single process. Note that this may require the use of process flow to launch and separately attach to the required IE instance. Two processes will still be created, one for the frame and one for all the tabs within that frame.

Changing the value will require Internet Explorer to be restarted before it can pick up the new registry value.

2.3 Launch via Process Name

You should be able to launch a web application like you would any other application. If you are experiencing problems launching a web app, it should be noted that sometimes web apps need to be launched directly by the Internet Explorer process and then attached to. This can be performed by using a built in Blue Prism utility called Utility – Environment. There is an action within this object called Start Process which takes the path of the application you would wish to start and any command line arguments. For example to launch Internet Explorer to display Google, the supplied inputs would be as below.



Action Properties

Name: Launch Internet Explorer

Description:

Business Object: Utility - Environment

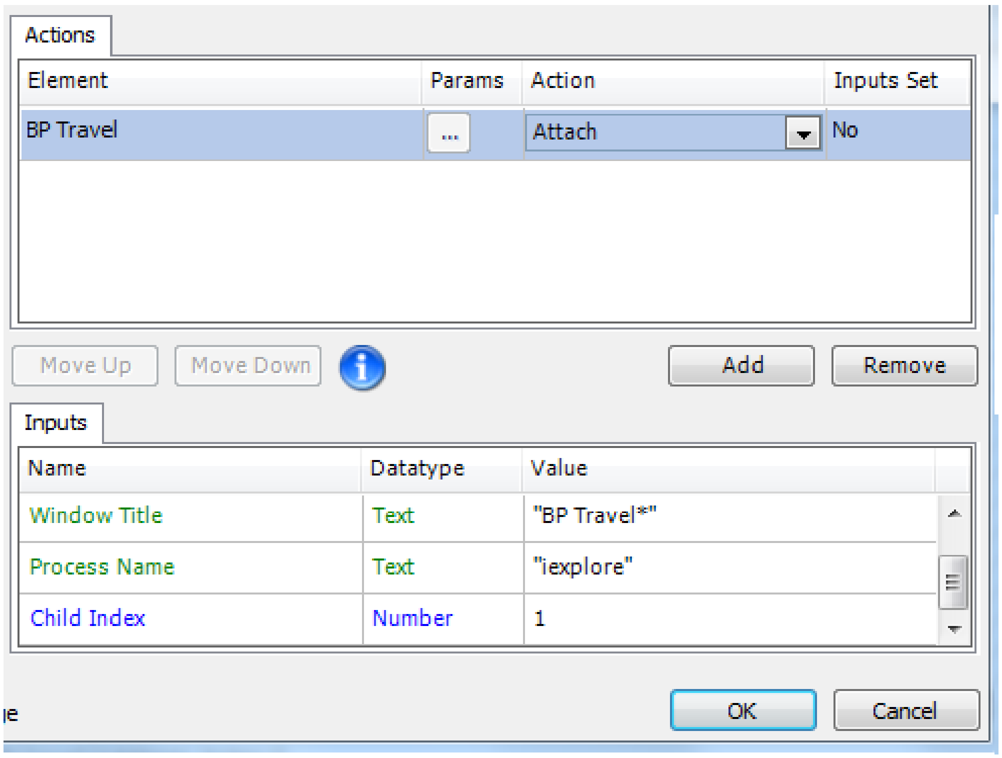
Action: Start Process

Group: Page [✓ Data Type] View All Items

Name	Data Type	Value
Application	Text	"C:\Program Files\Internet Explorer\iexplo...
Arguments	Text	"https://www.google.co.uk"

Figure 1 Launching Internet Explorer via it's process

Once launched like this, your object will not be attached to Internet Explorer, meaning that you will have to manually attach. This can be done by using a Navigate stage with the Attach Action as shown below. You might night to adjust the value for the Child Index input.



Actions

Element	Params	Action	Inputs Set
BP Travel	...	Attach	No

Inputs

Name	Datatype	Value
Window Title	Text	"BP Travel*"
Process Name	Text	"iexplore"
Child Index	Number	1

Figure 2 Attaching to Internet Explorer

3 Interfacing HTML Elements

In this section we will discuss various techniques that can be implemented when interfacing with browser applications.

3.1 Working With Varieties of HTML Elements

HTML web pages are made up of a variety of types of elements and fields, usually more so than Windows applications. This can provide new challenges when attempting to interface with a browser application. When interfacing with HTML elements it is important that you only spy the element you wish to interact with, not placeholder HTML elements or any other hidden elements. For example, when spying a checkbox, ensure only the checkbox itself is highlighted, not the area behind it.

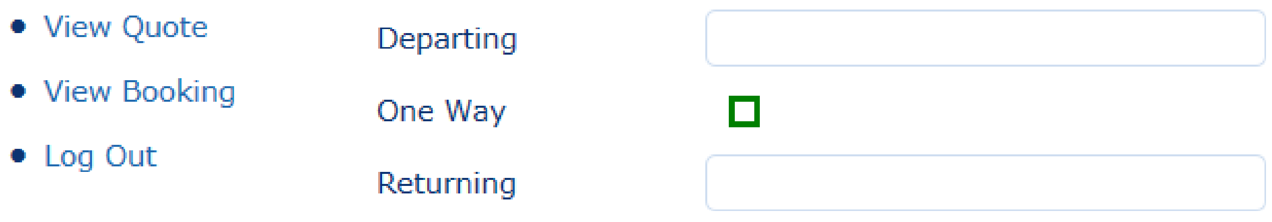


Figure 3 Checkbox correctly spied

This checkbox can then be checked by sending the value “True” to it by using a Write stage.

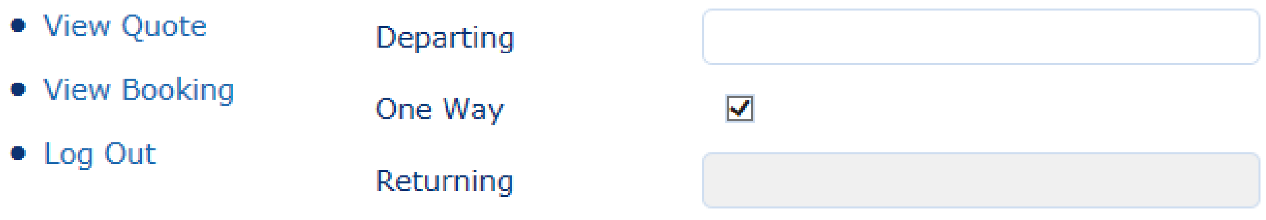


Figure 4 Checkbox correctly set

If you spy any HTML element behind the checkbox such as in the screenshot below, you may not be able to interact with it properly.

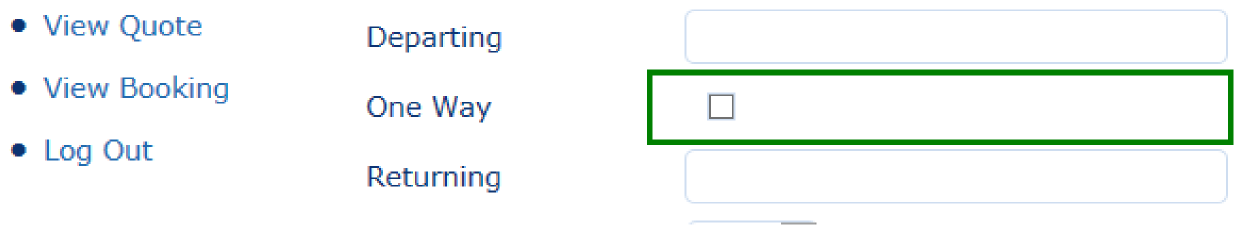


Figure 5 Checkbox incorrectly spied

As shown below, sending “True” to the above element does not produce the desired effect.

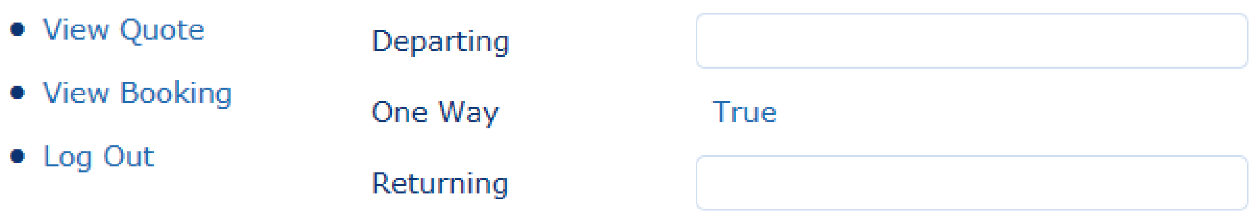


Figure 6 Failure to correctly set a Checkbox

Extra care should be taken when spying HTML elements, as more complex websites may make use of tables and other techniques for formatting fields and other values, which could possibly interfere with interfacing. As outlined in section 3.2, it can be a good idea to make use of more than one spying mode if you encounter issues.

A useful tip to remember is that to read or write to an HTML element, that element does not have to actually be visible on the screen. As long as the element exists on the page you are currently viewing, reading or writing to it should work.

3.2 Spying Modes

The three spying modes available for browser integration are Win32 mode, HTML mode and Active Accessibility mode. Sometimes it is logical to use a mixture of all three modes when you are building your element tree.

- Win32 mode will only work with Internet Explorer windows themselves, not with the web app within. This mode is particularly helpful for manipulating IE windows by performing actions such as Resize, Activate and Send Keys. It can also be used with Wait stages, for example, waiting for a window to appear on the screen.

- HTML mode will only recognise the HTML elements within the web app, nothing outside.

- Active Accessibility mode can detect a mixture of web app elements and Internet Explorer elements. If the standard HTML mode does not work, this mode often does.

If you are using more than one spying mode in your Object, it can be convenient to name your elements with suffixes indicating which mode was used to spy them, such as “Alert Message (Win32)”. In a large application model it would be practical to leave your HTML elements unmarked, and only apply suffixes to elements spied in Win32 or AA mode.

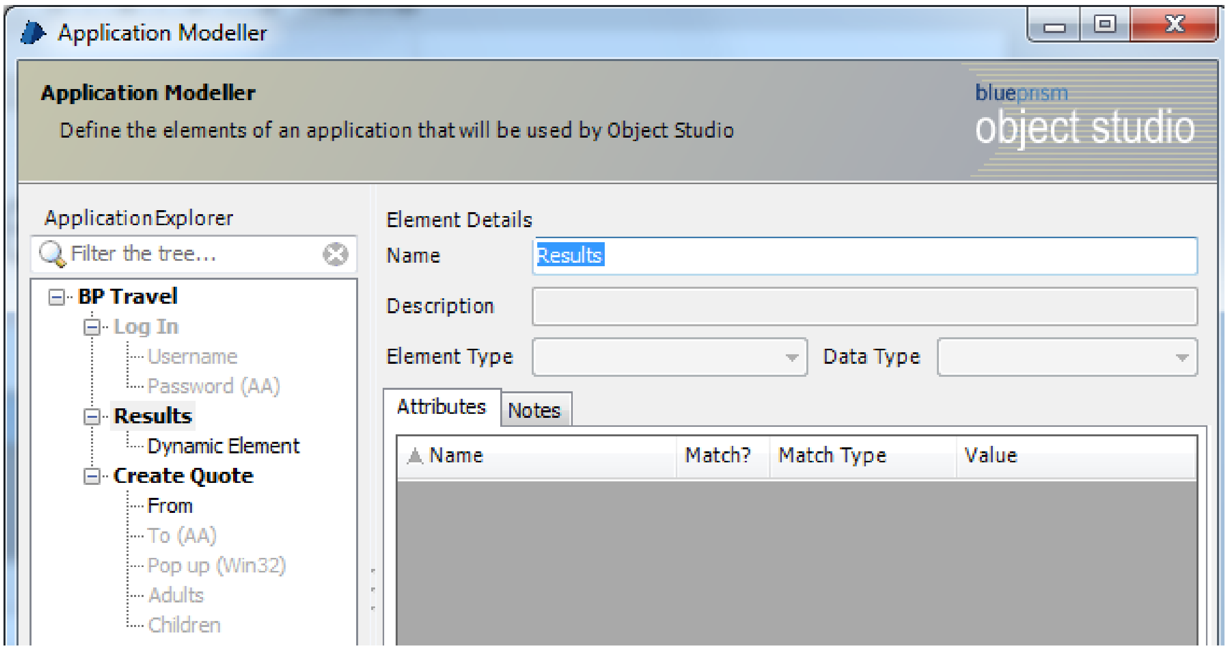


Figure 7 Labelled Application Modeller tree elements

If you encounter any unexpected problems that prevent you from using either of the three spying modes, an alternative exists in the form of the Application Navigator. To open the navigator, you need to click on the drop down arrow next to the Identify button in Application Modeller and select Open Application Navigator.

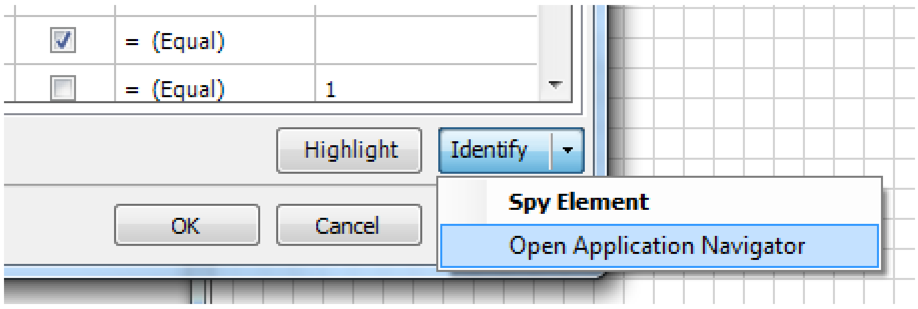
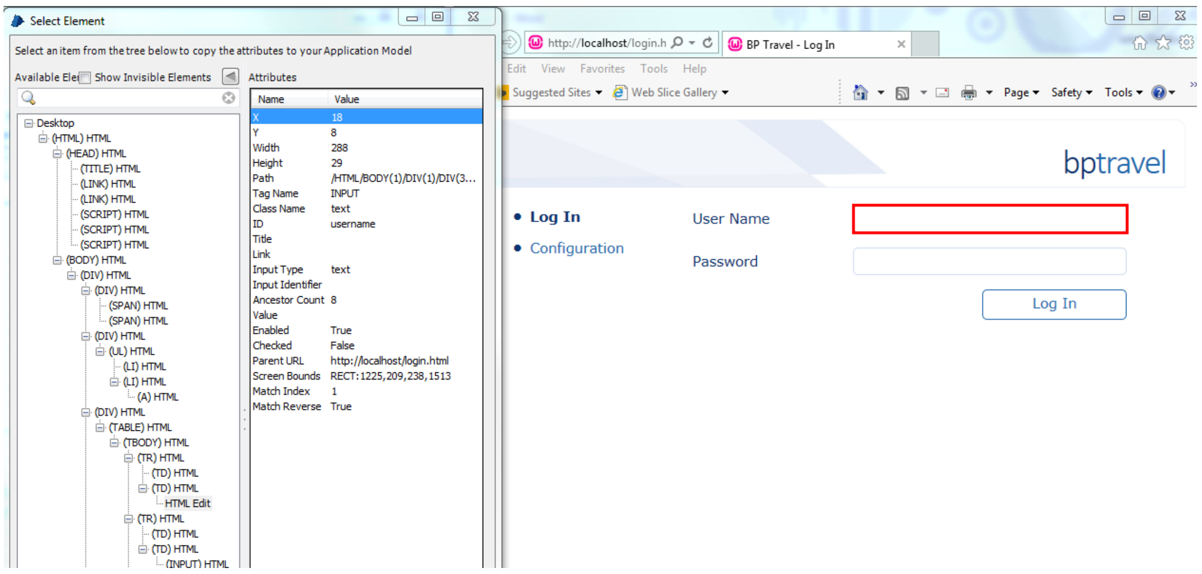


Figure 8 Opening Application Navigator

Once opened, the application navigator will scan the HTML document for all the elements it can find. It will then display a tree structured list of HTML elements from which you can choose from to create your elements in Application Modeller. You will notice that when you select an element in the tree it will be highlighted within the web browser for confirmation and its attributes will be displayed. The screenshot below shows a highlighted HTML Edit element selected in the element tree, resulting in it being highlighted within the browser.



Name	Value
X	18
Y	8
Width	288
Height	29
Path	/HTML/BODY(1)/DIV(1)/DIV(3...
Tag Name	INPUT
Class Name	text
ID	username
Title	
Link	
Input Type	text
Input Identifier	
Ancestor Count	8
Value	
Enabled	True
Checked	False
Parent URL	http://localhost/login.html
Screen Bounds	RECT:1225,209,238,1513
Match Index	1
Match Reverse	True

Figure 9 Application Navigator with Highlighted Element

The navigator also has the option of displaying invisible elements which can be particularly useful if the target application makes use of invisible elements, as described in chapter 3.10.

3.3 Get Table

It can be advantageous to use Get Table to read multiple values from a web page and store them in a Collection. The Collection can then be parsed to extract the required values, which may prove faster than interfacing directly with each individual element due to fewer reads.

If a table cannot be read into a Collection, it could be that the values held within the HTML code are not stored directly in table tags, because more HTML code is being held within the table cell. For example, reading a table structured like this should work:

```
<TD>My Value</TD>
```

But if the table is structured like below, you might have to use another method for reading the values.

```
<TD><DIV>My Value</DIV></TD>
```

If Get Table doesn’t work you could try reading the HTML code of the table and use string manipulation to extract the values, as described in chapter 3.4. Another alternative method would be to use counters for each row and column of the table, and use these counters in conjunction with a dynamic path attribute to read each cell individually. Chapter 4.1 talks in more detail about dynamic paths. You might find having a look at the HTML source code revealing as well, described in chapter 3.9.

3.4 Get HTML

When encountering difficulties reading values from a web form, perhaps due to very slow web pages or elements that you can't interface with easily, it can be helpful to use Get HTML. Get HTML can sometimes return the source code for the entire page, which could contain the values you need. The returned data will need to be parsed to retrieve the desired values using string handling and/or regular expressions.

3.5 Write Stages

Using a Write stage to write to an HTML element such as a text field does not always work properly. For example, you might try writing to a username field, only to see a message appear on the web site saying something like "Please enter a username", even though you can see that the value has been correctly written to the field. This can happen because the data validation functions used by the web form might be expecting keystrokes, and the write stage has "fooled" it. To get around this, you will need to use a Navigate stage to call the Send Keys Action instead of using a Write stage.

Some websites have maximum character limits imposed on some text fields. Using a Write stage can sometimes fool the website into allowing too many characters into the field, because the Write stage "sets" the field value rather than keying characters into it. This is important to bear in mind because if a field has been filled with more characters than the website would usually allow, the website could produce an error when you try to submit or post the data.

3.6 Navigate URL

If a website is particularly complex or if the current method of navigating through pages is long winded, it might be better to use Navigate URL to directly go to the required page. Using this could speed up navigation substantially as an alternative to navigating around multiple screens. Be careful however as it also has its cons, for example if the URL of the required page changes.

3.7 Wait Stages

Among the conditions that can be checked for in a Wait stage are Parent Document Loaded, Document Loaded and Check Exists. Each of these conditions check for separate things and should be usually appropriately.

- Parent Document Loaded. This checks the element exists and that the entire page and all of its child frames are fully loaded. Parent Document Loaded includes an implicit Check Exists on the element as well as a Document Loaded check. It is usually good practice to use this at the start of a browser action.

- Document Loaded. This checks if the current document has loaded. Do not use this unless you know you are already on the page which you are waiting to load, otherwise it is more appropriate to use Parent Document Loaded on an element on the target page.

- Check Exists. This simply checks that the specified element exists on the page, regardless of if any other elements exist.

3.8 Pop-up Windows

Depending on how the web application has been built, you might need to create a separate business object to interface with a popup window. This will be the case if you see the “A spying error has occurred” error message. A new object will need to be configured to attach to a running instance of the popup, rather than launching something. Pop-up windows may need to be spied using Win32 or AA mode instead of the usual HTML mode. To speed up development, it might be best to build a separate generic object to handle IE pop-up windows.

3.9 Inspecting HTML Source Code

Sometimes it can be useful to look at the HTML source code to understand the structure of the web page such as when dealing with tables, as described in chapter 3.3. This isn’t always helpful but it could be revealing. To view the source code, right click on the web page and select “View Source”.

3.10 Invisible Elements

It is sometimes possible to manipulate HTML elements that aren’t on the visible screen. For example, it might be possible to click a link on a sidebar even though it isn’t currently visible to a human user. After some experimentation you can often save time by using this technique.

After spying an element and saving it in Application Modeller, sometimes it won’t be highlighted on the screen by the red rectangle when you press highlight, even though the “Unable to find element” error message was not displayed by Blue Prism. After closer inspection, sometimes you will notice a tiny 1x1 red pixel highlighted at the top left corner of the page. This is often indicative of an invisible, zero-size element that the web page manipulates with client-side code such as JavaScript. Using the Application Navigator as described in chapter 3.2 might be useful in such circumstances.

It is also possible to interact with AA elements that are invisible. Often, it can be hard to reliably match on an AA element because invisible elements are being highlighted in Blue Prism as well as the intended element. To avoid this, try matching using the Invisible attribute available in the Application Modeller for AA elements. For example, you might want to tick the Invisible attribute and set its value to False, meaning only visible matching elements are highlighted.

4 Application Modeller Attributes

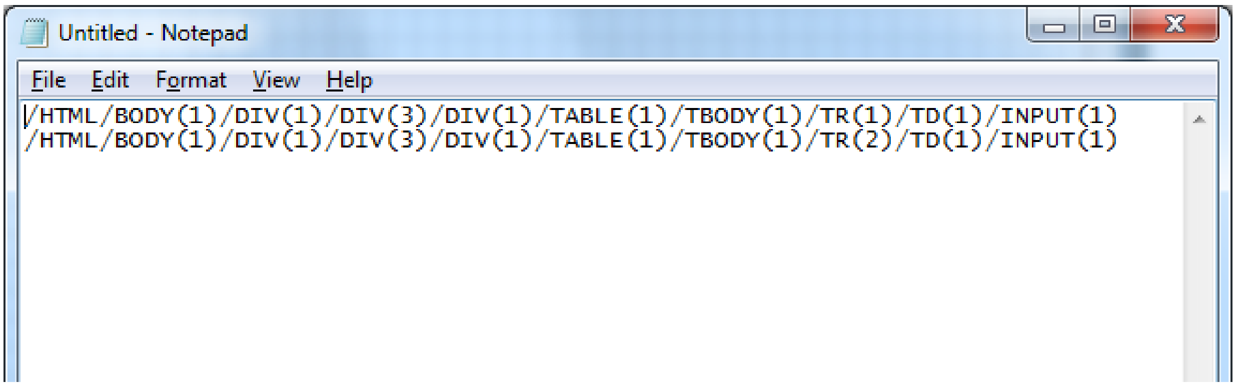
This chapter provides useful tips regarding HTML element attributes in Application Modeller.

4.1 Path and Dynamic Paths

The Path attribute should only be used to match elements if the web form is simple, i.e. not dynamic in nature and unlikely to change. Using the Path attribute to match elements will speed up the interface but requires that the web page to be static in its underlying structure.

Using a text editor like Notepad is useful when you need to spot where a path has changed. For example if you have spied an element that used to work but now it doesn't, you could paste the path that used to work and the current path into Notepad and compare them. Often the only difference is just a single digit. You can then work around the issue by using a dynamic value for the Path attribute of your element.

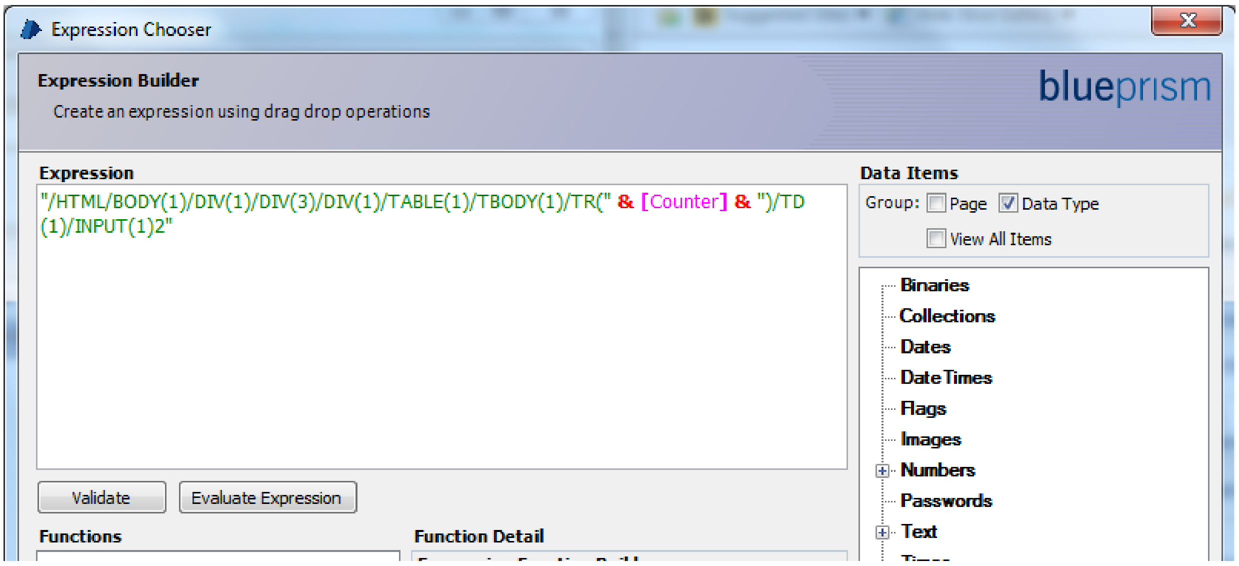
For elements that have dynamic path attributes, it can be useful to get the path of one element in a section of the web page and then use this to dynamically set the path for other elements in the same section. For example, you could use a single element to read the values in a column of a table. The screenshot below shows the path attribute of two separate elements within a column, one on the first row and the other on the second row.



```
/HTML/BODY(1)/DIV(1)/DIV(3)/DIV(1)/TABLE(1)/TBODY(1)/TR(1)/TD(1)/INPUT(1)
/HTML/BODY(1)/DIV(1)/DIV(3)/DIV(1)/TABLE(1)/TBODY(1)/TR(2)/TD(1)/INPUT(1)
```

Figure 10 Path Comparison in Notepad

You will notice that the only difference between the paths of both elements is the digit held within the HTML tag TR(). The increase by one is a common pattern for each row of this particular HTML table. This means you can create a single element and set its Path value to be dynamic, and then use this single element to navigate down the table while reading the values. In order to do this, you will need to use a counter, in the form of a Data Item with its initial value set to 1. The counter will go in place of the increasing number, as shown below.


Expression Chooser

Expression Builder

Create an expression using drag drop operations

Expression

```
"/HTML/BODY(1)/DIV(1)/DIV(3)/DIV(1)/TABLE(1)/TBODY(1)/TR(" & [Counter] & ")/TD
(1)/INPUT(1)2"
```

Data Items

Group: [ ] Page [✓] Data Type

[ ] View All Items

Figure 11 Use of a counter in an expression

The value of your counter could be increased with each iteration. The diagram snippet below shows how your Business Object Action might look if you were looking for a particular value in the column.

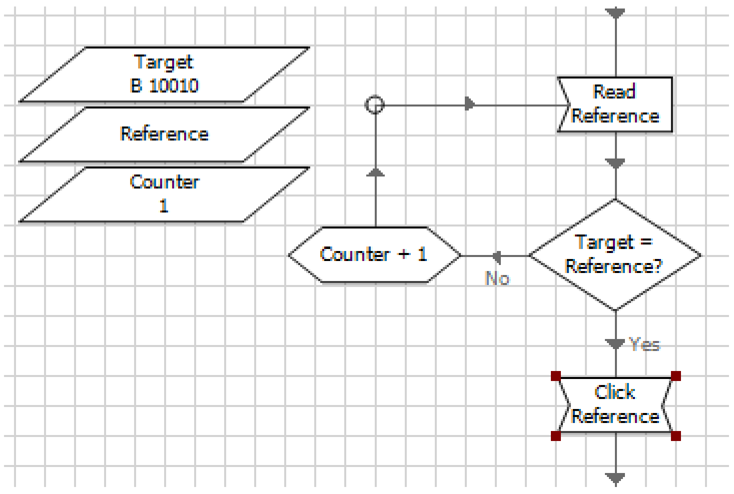


Figure 12 Example of Dynamic Path logic

A Block could be placed around the Read stage to capture any exception thrown. This would typically indicate that the path has been incremented too many times and the element at the new path does not exist. For example, the end of a list or table may have been reached. Using the Block you could capture the exception and take appropriate action.

4.2 Parent URL

Depending on the structure of the web site, the Parent URL of an element may not be consistent. It is usually helpful to un-tick the Match column for the Parent URL attribute of the spied element in Application Modeller, or else problems will arise when you attempt to use the element after the Parent URL has changed.

4.3 Element Mask

Element Mask is a feature in Application Modeller that enables you to copy the attribute selection of one element and apply it to another. This can be very useful once you have found a winning combination of attributes on one element that you want to apply to more elements. To use Element Mask simply right-click on an element and select Copy Element Mask or Apply Element Mask.

4.4 ID Attribute

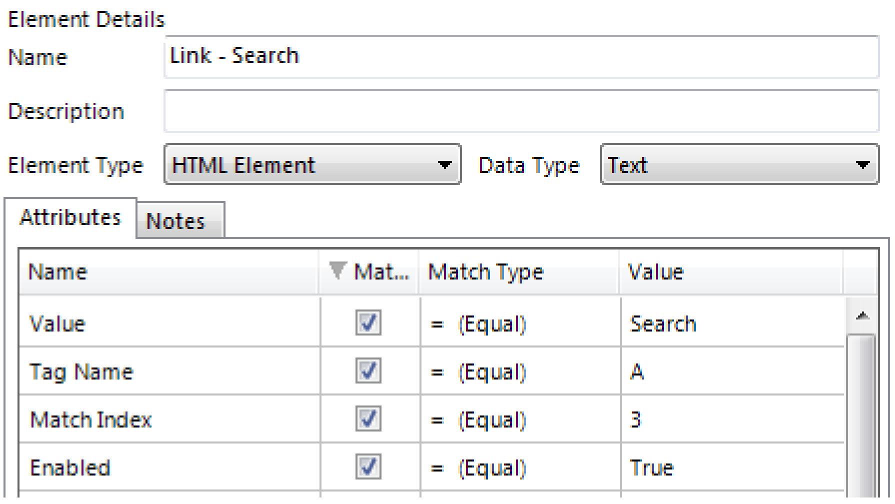
Using the ID attribute to match elements can speed up interfacing with the target application. Some web applications allocate unique IDs to elements meaning the ID attribute can be used by a Blue Prism developer to create a reliable application model. However, with some applications the ID attribute can be dynamic, so matching with this attribute might not work reliably.

4.5 Class Name

Like the ID attribute, some web applications contain elements that have static class names. However, some applications contain elements where the class name changes so be careful when using this attribute to build an application model.

4.6 Match Index

When the highlight button in the application modeller is pressed, Blue Prism searches for any elements within the target application that have matching attributes with those selected in the application model defined by the developer. If more than one element matches more than one elements are highlighted. When the Match Index attribute is selected, only one element will be highlighted. If the value of Match Index is set to 1, Blue Prism will stop searching the application for elements as soon as it finds the first element that matches. If set to 2, the search will halt after a second element that matches is found and that element will be highlighted, ignoring the first. The value of Match Index can be adjusted by the developer if required. This can be very useful when working with multiple elements that are difficult to distinguish from each other. In the example below, the third element to match the selected attribute criteria will be taken and the search will stop.



Element Details

Name: Link - Search

Description:

Element Type: HTML Element

Data Type: Text

Attributes | Notes

Name	Mat...	Match Type	Value
Value	✓	= (Equal)	Search
Tag Name	✓	= (Equal)	A
Match Index	✓	= (Equal)	3
Enabled	✓	= (Equal)	True

Figure 13 Using Match Index

Using this attribute to match elements can result in significant performance gains by reducing the scope of a query to the target application.

4.7 Match Reverse

When Blue Prism searches an application for an element, it has to search through a “tree” structure of elements. The search starts at the top of the tree unless the Match Reverse attribute has been selected, in which case Blue Prism will search for the element starting at the bottom (last element in the application) progressing upwards. The match reverse attribute is only relevant when used in combination with the Match Index attribute.

5 Diagnostics and Rich Internet Applications

In this chapter we will discuss how Blue Prism can be used to produce diagnostics information and how Rich Internet Applications which make use of technologies such as Flash and Silverlight can be automated.

5.1 Diagnostics

Blue Prism offers a Diagnostic snapshot feature that can be revealing when troubleshooting problems. Whilst the business object is running if Blue Prism fails to find a particular element the Diagnostic snapshot can take full capture of all the details of every single UI feature that Blue Prism can identify. This can help build a picture of the state of the application when the element couldn't be found.

To use diagnostic snapshots you must first turn on the feature for your business object. Within the Application Modeller window select the root node in the application model, and press the Diagnostics button. After clicking the button a dialog will appear allowing you to select one of the available snapshot methods. The Take Now button allows you to create an elements snapshot text or source capture text immediately based on the current application state. The format of the snapshot text uses tab indentation to denote the tree structure of the elements, followed by a single line of attribute value pairs. The format of the HTML snapshot is chunks of HTML embedded within an XML document. The snapshot text will be displayed in a pop up dialog window.

With the feature enabled you can right click any element within the application model which corresponds to a UI feature. Right clicking the element will display a context menu in which there will be there will be a Create Snapshot on Error menu item. Selecting the menu item will check or uncheck a checkbox next to the menu item. When stepping through an object, if the application manager fails to identify any selected elements it will now take a snapshot of the entire model and dump the snapshot text into the logs. In the case of the browser application type it will also dump a source capture of the HTML if that option was chosen.

5.2 Microsoft Silverlight

Silverlight applications have a property which allows them to be displayed as a windowless plug-in, which is usually set to true. There is no accessibility support for this so in order to automate Silverlight applications easily the windowless property needs to be set to false. One way this can be achieved is by injecting some JavaScript into the browser to change the property to false and then invoking that JavaScript, which will allow the browser to expose the Active Accessibility properties of the elements within that application to Blue Prism. If you are using Windows XP, Windows Automation 3.0 will need be installed which will allow Silverlight to expose Active Accessibility properties. If you require more information on this, contact Blue Prism Support. If all else fails, Surface Automation techniques can be used to automate the application.

5.3 Inserting and Invoking JavaScript

Sometimes it can be convenient to call a JavaScript function which is defined in the HTML of the web page you are interfacing with yourself. You can do this by using the Invoke JavaScript Function action. The action takes two inputs: the name of the function to be called and any arguments which are to be passed to the JavaScript function. The specified function will be run within the browser when the action is stepped over.

It is also possible to write your own JavaScript code which can be a mixture of methods and variables, and then run your code on the web application. Once you have written the code you would like to run, into a Data Item for example, you can use the action Insert JavaScript Fragment. This action places the supplied code into the target document, allowing you to then call it using the Invoke JavaScript Function action.

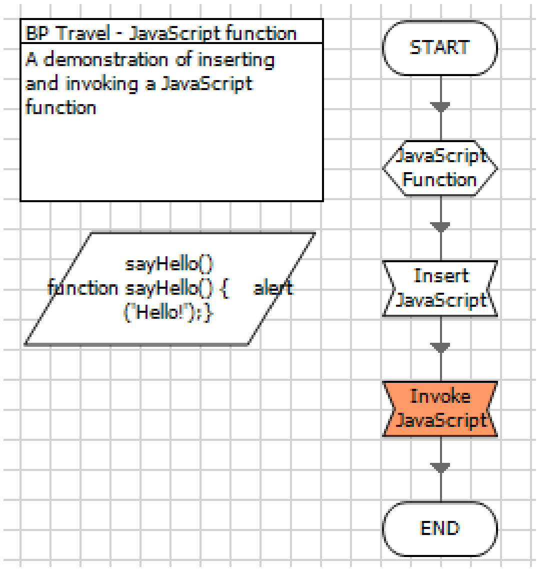
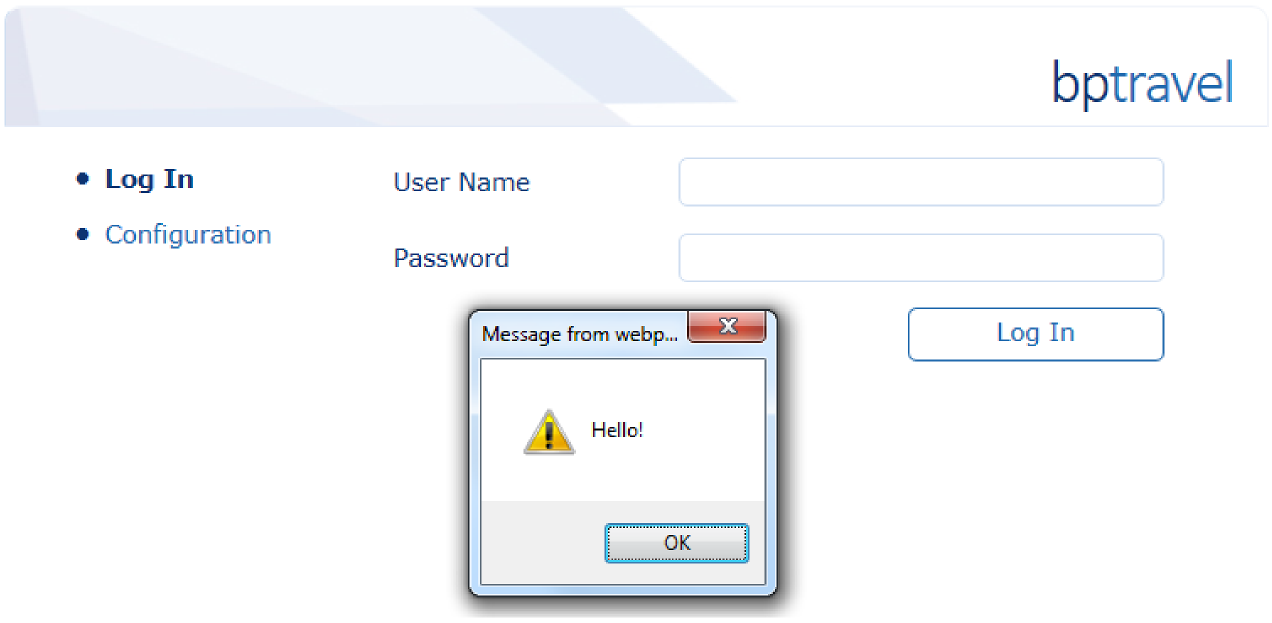

BP Travel - JavaScript function
A demonstration of inserting and invoking a JavaScript function
sayHello()
function sayHello() { alert('Hello!');}
START
JavaScript Function
Insert JavaScript
Invoke JavaScript
END

Figure 14 Inserting a JavaScript Function into a Browser Application

The Insert JavaScript stage shown in figure 5a takes the input [sayHello()], which is the text Data Item holding the custom JavaScript code. The Invoke JavaScript stage takes two inputs, the name of the function held within the browser application source code that you wish to call, in this case “sayHello”, and any Arguments you wish to pass to that function. If you do not want to pass any arguments, you need to set the input to “[]”. Running the highlighted stage results in the browser application running the inserted function.



Hello!

Figure 15 Running the inserted JavaScript fragment

You can invoke any JavaScript function that is available in the source code, but web developers often obfuscate their JavaScript source code so it might be time consuming to determine which functions you need to call as well as risky. Both techniques should be used with caution as you may be forcing the application to do things it wasn't designed to do.

5.4 Flash Applications

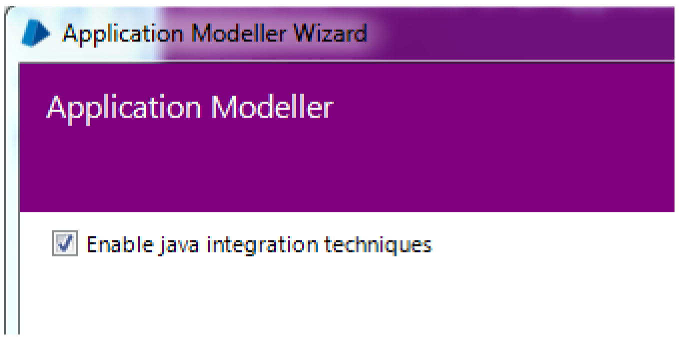
The ability to interface with a Flash elements in a browser application is dependent on whether the Flash developer built the elements with Microsoft Active Accessibility in mind. If AA properties have been included, Blue Prism will be able to spy the elements while in AA spy mode. If the Flash application has not been built with accessibility in mind, Surface Automation techniques will have to be used. If this is the case, it is important to check whether font smoothing on the Flash elements can be turned off if data needs to be read from the application.

5.5 ActiveX

If you come across any ActiveX components within your target browser application that need to be automated, you will need to use Surface Automation techniques.

5.6 Java Applications

Sometimes Java applications are launched via a web application. If the Java application launches in a new window and starts a separate process on the PC, you will need to build a separate business object to attach to it in order to automate it. If the Java application remains open in the Internet Explorer window, you might be able to interface with it by enabling Java integration techniques (checkbox available in the Application Modeller Wizard).



☑ Enable java integration techniques

Figure 16 Enable java integration techniques