Badboy v2.1
User Documentation

Table of Contents

Table of Figures

1 Introduction

Welcome to Badboy!

Badboy is an application which makes testing and building Web applications easier by combining a capture/replay style testing tool with live diagnostics on the activities your browser is performing while you navigate your site.

- If you have a development background then you are probably used to using a debugger to step through your code and see what is happening inside. You can think of Badboy as a debugger for your browser. Read about all the great features for developers!
- If you do testing or Quality Assurance work then you can use Badboy as a powerful test automation tool to help you record and play back tests of your web site or application.

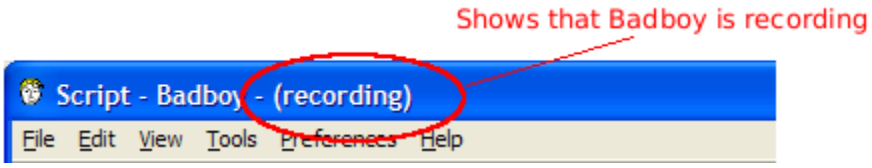
If you are new to Badboy, why not take the tutorial? Select "Help" and then "Tutorial" from the menu to get started! Or, read on to find out more about what Badboy does!

2 Basic Operation

Badboy works by monitoring the activity of Internet Explorer as you browse, and recording interesting events so that you can see them later and, if you wish, replay them.

2.1 Recording

Recording can be turned on and off. You can tell if Badboy is recording by looking at the title bar of the window.



When you press the Play button, recording is automatically turned off. After the play finishes, you can turn recording on again by clicking on the red record button on the toolbar.

While recording, you should let each request completely finish loading before interacting with the browser again. This is important because if you generate a new request while a previous one is still loading, the new request may be treated as a sub frame or other non-recorded result of the first request.

2.2 Creating Suites, Tests and Steps

When working with a web site it is often useful to break up a sequence of browser interactions into logical steps. For example, you might like to have your first step as “Login to Yahoo Mail”, and your second step as “Navigate to my Inbox”, and the final step as “Send a message to my mother”. Each of these steps might contain more than one interaction with the browser. In Badboy, you can create "Steps" to reflect this by clicking on the New Step button on the toolbar.

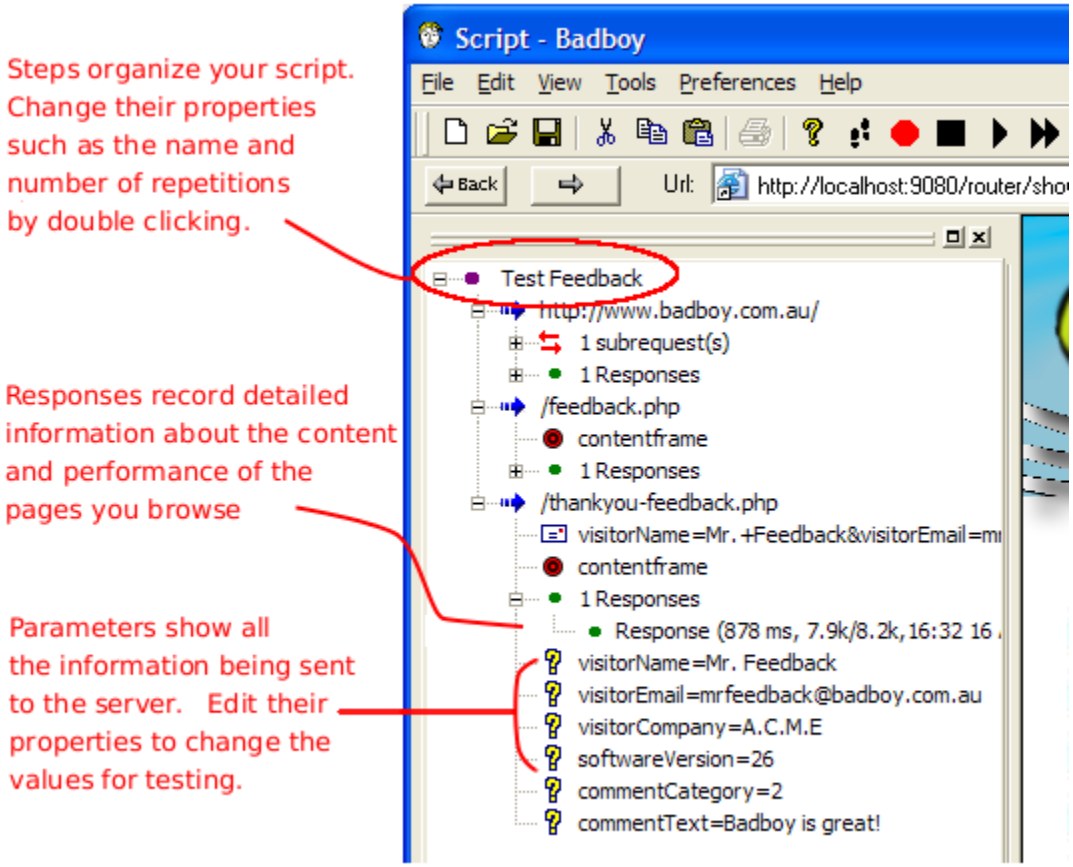
If you want to organize your Script even more, you can create Suites and Tests to place your Steps into. But if you don't want to you don't have to - you can just create Steps with the New Step button if you like. See Suites, Tests and Templates for more information about organizing your Script with Suites and Tests.

When you replay a script, Badboy will automatically stop at each step so you can review the screen. If you push Play again, the script will continue from the next step. If you want to play your whole script without stopping, use the "double" play button instead, which will play your whole script from beginning to end.

2.3 The Script Tree

The Script tree is displayed in the left hand pane of the main window. It displays all the requests that you have recorded, arranged into the steps that you create while recording, along with any other items you have added to your script.

You can customize the items in the tree by double clicking on them.



2.4 Playing

You can replay a sequence you have recorded at any time by clicking on the Play button. If you want to play one Step at a time, use the Play Request button, shown below. There are also stop and rewind buttons to stop a currently playing script and rewind to a previous step respectively. If you want to play your whole script without stopping anywhere, use the "double" play button.

What else can Badboy do?

We've hardly scratched the surface here of all the things that Badboy can do. To learn more, try the following sections

- See Automating Your Scripts to find out more about customizing Badboy's playback.
- See Variables and Looping to see how to use Variables to create loops and sophisticated playback patterns.
- See Assertions to find out how make Badboy automatically check your website for errors.

3 Automating Scripts

Being able to replay back a sequence of browsing activities repeatedly can be a very useful mechanism for debugging and testing your web site. Unfortunately, playing back exactly the same requests that were used on a previous occasion frequently does not satisfy the requirements of complex web sites.

Some examples of scenarios where this can occur are:

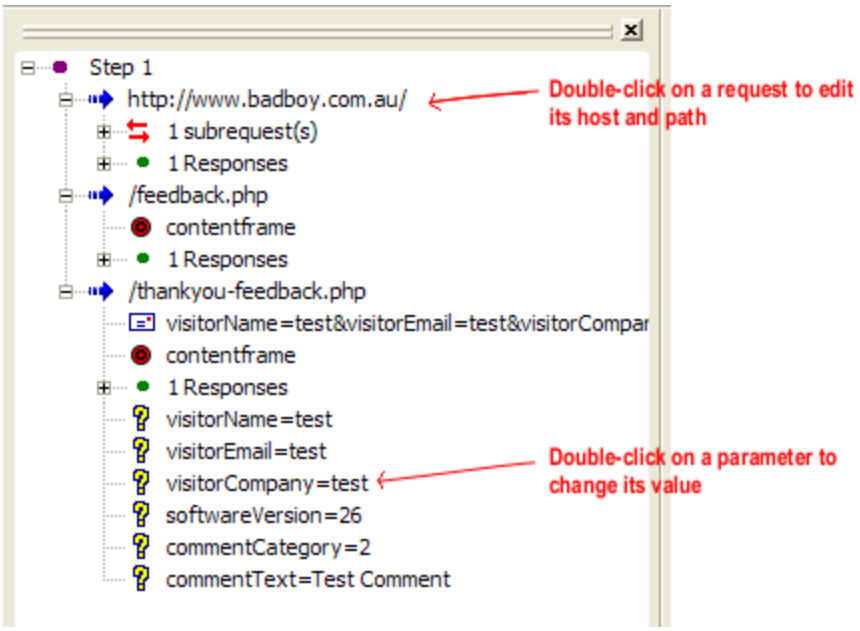
• An identifier that is entered must be unique. Entering the same value twice generates an error. This would occur, for example, if you tried to make a script that would register for an account with a user Id. The first time it ran it would succeed, but on replay trying to sign up for the same Id again would generate an error

• You recorded the script on one server (for example, your local development box), but you would like to run it against a different server. You need a way to replace the host name to which the requests are directed.

Badboy helps you deal with these problems in several ways. These are Editing Parameters, Script Variables, and a variety of tools to help you such as Search and Replace. Read on to find out more about these!

3.1 Editing Parameters and Hosts

Badboy allows you to edit the values of request parameters and host names in your script. This is very easy to do - just double click on the request or parameter that you would like to change the value of.



Using this feature you can find and change any values that need to be updated before you run your script.

3.2 Searching and Replacing

If you have a value that you need to change throughout your script then finding and editing every occurrence of that value would be very long and tedious process. To help with this, Badboy has a Search and Replace feature. You can invoke the Search and Replace function by selecting "Search/Replace" from the Badboy's Edit menu. Alternatively you can just press "F3" to show the dialog. The figure below shows the search/replace dialog:

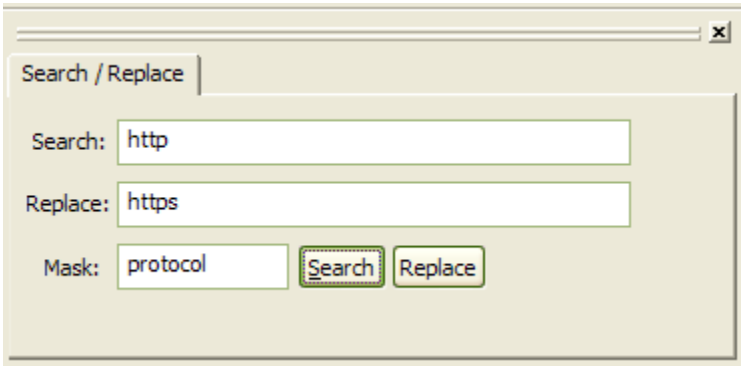


Figure 1: Search/Replace

3.3 Property Mask

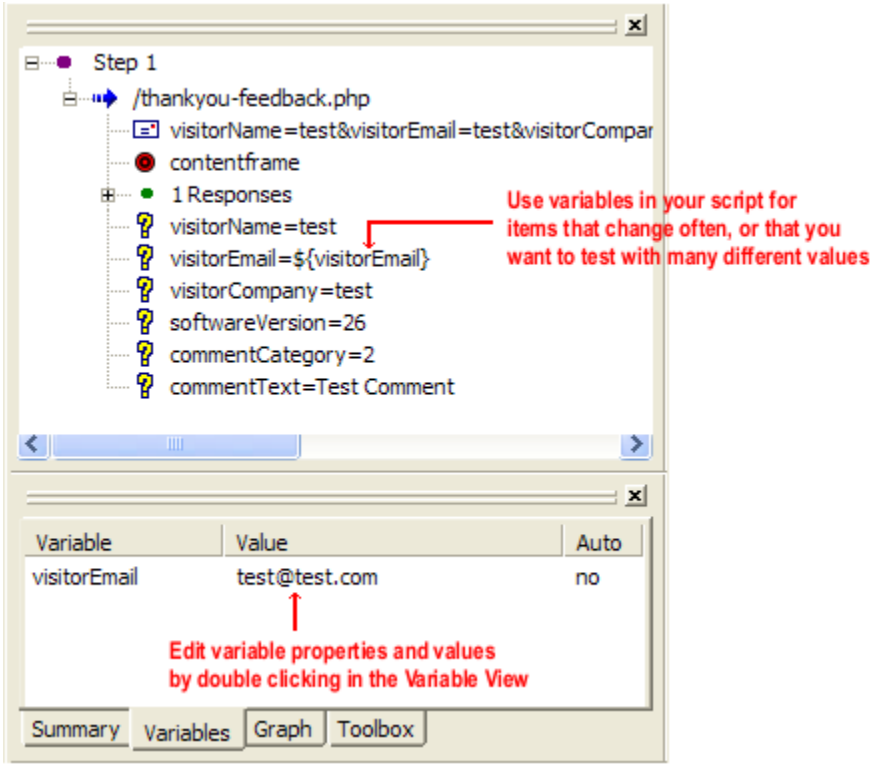
Sometimes when searching and replacing you only want to search for specific properties of an item, such as its name, id, or other property. If you want to filter the properties that are considered in the search, specify the name of the property to match on in the Mask field. Note that you can use a regular expression in this field to match multiple properties or a range of properties matching a particular pattern.

After you have opened the search dialog with F3 and entered your text, pressing F3 again will search for the text. You can keep pressing F3 to find the next item containing your text.

3.4 Variables

If you have values that change often you will quickly get tired of having to search and replace in your script. To alleviate this Badboy provides a feature called “script variables”. Script variables are displayed in the "Variables Tab" (usually in the bottom left corner of the Badboy window) and contain values that you can reference by name throughout your script. This allows you to include references to variables in your parameter names and host names by using the syntax "${variable name}" to include a variable value. In this way you can create scripts that share the same value in many places and you can maintain it in just one place.

To add a variable, just right click on an item in the tree on the item that you would like to become controlled by the variable, and choose "Add Variable". To edit a variable's value, just double click on the variable in the Variable View and changes its value in the properties box.



3.5 Linking Values

Often the same parameter value is sent many times in a script. Badboy offers a feature called "linking" to help you find all the places where it the value is occurs and replace them with variables. Linking automatically joins all the separate instances of a recurring value to a single variable. If you use a linked variable, you only have to create the variable once and from then on all the items in your script with the same value will be converted to use the variable. To do this, simply add a variable by right clicking on a parameter in the script and choose "Add Linked Variable". After adding the variable, Badboy will search for all values that are the same as the one you selected and change their values to reference the new variable. Linking is especially useful because Badboy will also automatically replace values for new items

that you add or record in your script with variable references whenever or however they appear. This way you don't need to remember to add variable references as you work with your script.

Badboy has a special tool to help you link a variable to host names in your script. This is very useful if you would like to test your script against a different host to the one you recorded on. To do this, just choose "Create Hostname Variable..." from the Tools menu in Badboy's main window.

3.6 Using the Date/Dime to Create Unique Parameter Values

Using Variables you can create scripts that can be easily run repeatedly to test a defect, validate code as you write it, or a myriad of other uses. It is a very common occurrence, however, to need variables that are guaranteed to be unique each time the script is played. The scenario of creating a user account is a good example. To help with this, Badboy makes it easy to automatically set variable values to a unique value. You can do this by adding an *Increment* script item (available in the Toolbox) at the start of your script. You can place increments into your script wherever you want new values for your Variable. Increments have many other abilities and features - for more information on Increments, see the topic Incrementing Variables.

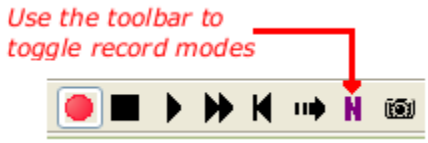
There are many more advanced uses of variables to automate your scripts. To learn more, see the following sections:

- Using Variables
- Using Data Sources
- Using Increments

4 Recording Modes

Badboy offers two different recording modes that you can use. These are:

- Request Mode (the default mode)
- Navigation Mode (activated using toolbar button or by holding down Ctrl-Alt while recording)



You can see which recording mode is being used by looking at the 'N' button on the Toolbar:

4.1 Request Mode

In this mode Badboy records the HTTP requests that are sent by the browser to the server, including all their parameters and other information. In a simplistic sense, you can think about Requests simply as URLs, such as the address for your web page. There are advantages using this mode:

- Requests are independent of the layout and appearance of the web pages you are testing. This means that if the layout of the web pages changes your test will still work.
- Requests work excellently with Badboy's Load Testing features and they also can be exported to JMeter for doing load and stress testing. Although Badboy *can* load test pages using Navigations, they do not work as efficiently - so if you need to do load testing at very high levels then recording your tests with Requests may work better for you.

However there are also some important disadvantages:

- In some cases Requests can be more difficult to get working. This is because Requests completely replace the information sent by the web page to the server with the recorded HTTP information. Sometimes web pages put information into the page which has to be specially calculated for each time a user goes to the page. In that case, Requests need to have their Parameters changed to use variables after you record them (see Using Variables for more information).
- While Requests are independent of the page layout and appearance, this can sometimes be a drawback. For example, imagine that you edit a page and accidentally delete the "logon" button. You would probably like your test to fail so that you could find and fix the problem. However when you use Requests this *will not happen!* Instead, Badboy will play the Request the same way regardless of whether the logon button is there or not. Of course, you could explicitly check whether the Logon button is there by using an Assertion (see Assertions), but to do that you would have to create the Assertion in advance - it would be very laborious to check every item you record this way. Instead, Navigation Mode can help you overcome this problem.

4.2 Navigation Mode

In this mode Badboy will record which *browser element* you clicked on. When you play back, rather than replaying the HTTP request that was sent previously, Badboy will find the original browser element that you clicked on when recording and simulate a click on it.

Navigation Mode has the following advantages:

- For some pages it is much easier to get working. This is especially true for complex pages such as Logon pages. The reason is that the Navigation is replaying the interaction to the browser and letting the browser do the work of creating the request.
- Because Navigations explicitly exercise the user interface they are much better at catching problems if the interface is broken. In the example above for Request Mode, if you had recorded your "logon" button as a Navigation your test would fail if somebody deleted the button from the page.

The main disadvantage of Navigation mode is that you often cannot use this mode for running load tests. This is because the Load Test engine runs without displaying any user interface, and hence it cannot execute Navigations. Another disadvantage is that your tests will depend on the correct items being present in the user interface to work. Thus if your main goal is to just test that the functionality of your web site works without caring about the user interface, Request mode may be better.

The difference between these two modes is very important. The choice you make in recording will have a big effect on how well your scripts adapt to changes in your web site. See below for more information on deciding which mode you should use.

For more information on Navigations, see the Navigations topic.

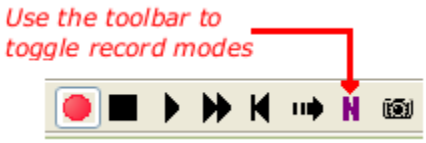
5 Navigations

Navigations are a kind of item in Badboy that records clicks or activation of user interface elements in the browser window. When you record a navigation, it finds the browser element that was clicked on (or navigated in another way, for example, by hitting space or enter) and remembers it. When you play back, Badboy finds the same element again and simulates a click on it again so that the browser performs the same navigation again.

5.1 Recording Navigations

By default, Badboy records Requests instead of Navigations. However you can easily switch recording modes at any time to record Navigations instead. You can do this by the following methods:

- Click the 'N' button on the Toolbar to toggle between modes
- Press Ctrl-Alt-N to toggle between modes
- To switch modes while recording just a single item simply hold the Control and Alt keys down while clicking on or navigating the page. This will change the record mode only while you hold the keys down.



5.2 Types of Navigation Items

Badboy will record different kinds of Navigation Item in your script depending on what kind of element you click on and also how that element is represented in the page you are recording. The table below shows the three different kinds of item you will see:

Navigation Type	Description
a	Represents a click on a link. This may be any kind of link including an image link or other non-textual content.
	A click on a button. These are usually buttons in forms such as Submit buttons.
N	A click on a miscellaneous item on the page. Frequently these occur when the page has used JavaScript to respond to "on-click" clicks on page elements resulting in a page navigation or a form submission.

5.3 Navigation References

Web pages can often change quite significantly in their layout, size and shape, and can often have many elements that look or appear the same. Because of this, Badboy uses several different ways of identifying elements in order to ensure that it finds the correct one that you originally recorded when playing back. The information that Badboy uses to identify the element is called the "Reference" to the element. When you record a click on an element, Badboy uses the following logic to find a reference for it:

- If it has an id then it will record the id of the element
- If it has a unique name assigned then it will record the name
- If it is a link or a button and has distinctive visible text (such as the label on the button) then it will record the text and identify the item via it's text
- If none of the other methods apply, it will identify the element by its position in the browser DOM, using a JavaScript expression.

5.4 Navigation Properties

If you wish you can open the properties of the Navigation and set the reference information yourself. The figure below shows how the properties dialog looks:

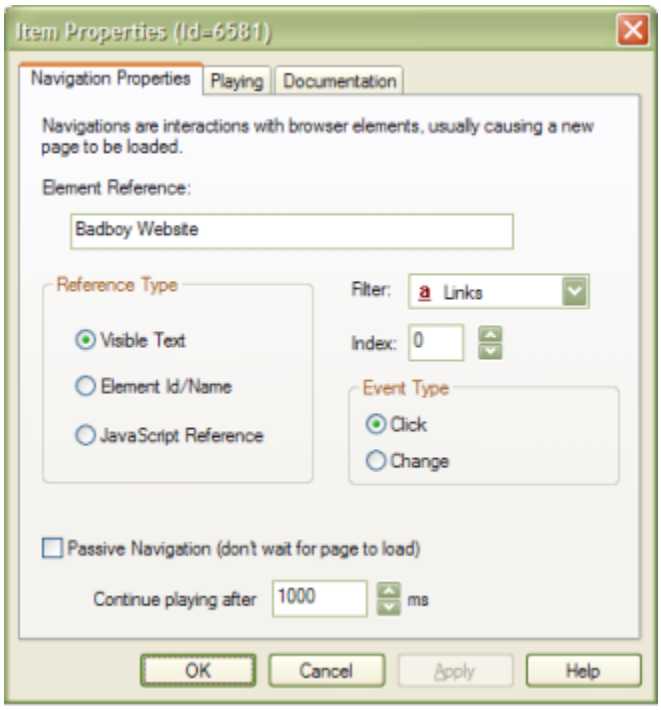


Figure 2: Navigation Properties

The following table describes the different properties you can set:

Property	Description
Element Reference	This identifies the element to be navigated. How this text is interpreted depends on the Reference Type.
Element Type	Determines how the reference text is used. • For Visible Text, the reference is interpreted as the label or visible page content that the element displays (e.g. the displayed content for a link, the text on a button or a tooltip describing the item). • For Id/Name, the reference is interpreted as an id or name set on the element to be navigated. • If you choose "JavaScript Reference", Badboy will execute the text as JavaScript which should return the item to be navigated. You can use this to write more complex logic to find the element to be navigated.
Element Filter	Sets the kind of elements that should be considered for matching the reference text. This helps make sure Badboy chooses the correct element by screening out all elements that do not match the filter. You can choose "Links", "Buttons" or "All Elements"
Index	Causes Badboy to use the specified occurrence of the matched element on the page. For example, if there are three buttons named 'Logout' on the page, you can make Badboy use the third one by specifying "2" in the Index property. Note that this property is "zero-based", so the first occurrence (the default) is specified using "0" and the second is specified using "1" and so on.
Use Regex Match	Causes the Navigation to treat the Reference as a regular expression when attempting to match elements on the page. This is useful if the element you wish to navigation changes for each playback but has a definable form that you can describe with a regular expression. This option only applies to the "Visible Text Reference" reference type.

If you are having problems getting your navigation to locate the right element you can right click on the Navigation and choose "Highlight Element" to show which element on the page will be targeted. If it targets the wrong one then you can increment the "Index" option in the properties to cause it to find the next item in the page until it finds the correct element.

5.5 Auto-Record of Form Populators

When pages contain fields that allow the user to enter information you will usually want any changes that are made to the fields to also be made when your Navigations are played back. For this reason, Badboy automatically records Form Populators for you to set any changed fields on forms on the page. Note that Badboy will only record a Form Populator if one of the forms on the page has been changed from its state when the page was first loaded.

5.6 Passive Navigations

Badboy supports two types of Navigation: Passive and Active.

- **Active Navigations** are actions that cause (or are expected to cause) the browser to load a new page. By default, Badboy will wait for one of the open pages or frames to reload before continuing playing the script after a Passive Navigation is executed. Active Navigations are the most common kind and are typical for normal links in a web site.
- **Passive Navigations** are a special kind of Navigation in that they **do not** expect a new page to load in the browser. These Navigations often perform no interaction with the server at all, although they may still change or activate parts of the browser window. A good example of a Passive Navigation is the frequent case where a user forgets to enter data into a form and a JavaScript check is done to remind them to enter it. The check might show a message box, or highlight the field on the page. However because it does not submit the form, no part of the page reloads from the server. Passive Navigations are especially important for web sites that use the so-called "AJAX" design methodology, since this kind of design often performs many background interactions that do not reload pages.

When Navigations are recorded, Badboy automatically notices whether there was a page load and sets the "Passive Navigation" option for you.

For more information see the topic Recording Modes for more information about Navigations and how Badboy's recording of them works.

6 Tests and Templates

This section explains how Badboy allows you to structure your scripts using Suites, Tests and Templates.

6.1 Suites and Test Items

Complex web applications will often have a large number of separate areas of functionality that can be tested independently of each other. Scripts for testing such web sites can end up being very large, so it is very useful to divide the tests into a hierarchy so that people looking the test can understand the different parts more easily.

In earlier versions of Badboy you could use Steps to structure your scripts, and by placing steps inside each other a whole hierarchy could be modeled. This worked well, but many people wanted to be able to make the structure of their scripts clearer, so to help with this, Badboy 2.0 has introduced two new items:

- *Tests* - shown as a notepad style icon:
- *Suites* - shown as a folder style icon:

Suites and Tests are provided to add to your Badboy script to help to structure it so that it is easy to understand, and to make reports and other output clearer.

Badboy doesn't enforce any rules about how you use and mix Tests, Suites and Steps in your script. If you prefer, you can make your whole script from Steps as in previous versions of Badboy. Or, you can place Suites and Tests inside Steps or Steps inside Tests and Suites. You can choose what kind of item (Test, Suite, Step) you would like Badboy to use at the root of your Script Tree in Badboy's preferences.

6.2 Advantages to using Tests

There are a number of advantages to using Tests to structure your scripts:

- Tests make your script easier to understand
- Tests can extend *Templates* to save you time and make maintaining your tests easier (see below)
- You can add *References* to Tests

6.3 Adding Suites and Tests to your Script

The easiest way to add new Suites and Tests to your script is to use the Toolbar button. This will create a new Test or Suite at the top level in your script. From there you can drag it to the position you want it to go with the mouse.

You can also add Tests and Suites to your script by dragging them from the Toolbox and dropping them in your script where you want the test to go.

6.4 Using Tests with Templates

One of the most powerful reasons to use Tests in your script is their support for *Test Templates*. Test Templates allow you to easily make many tests that follow the same outline without duplicating the structure of the Test many times over in your script. See the section Test Templates for more information about using Test Templates.

7 Variables

Variables are a key feature of Badboy. They let you customize how script items are played back at run time so that you can create scripts that behave intelligently rather than simply repeating the same action over and over again. This section describes how you can add and use variables in Badboy scripts.

7.1 Adding Variables

There are several ways to add variables in Badboy. First, you can add them directly by right-clicking in the variable window at the bottom left corner of the screen.

In order to make adding variables even easier Badboy offers a couple of other ways to do it:

- Right click on a parameter in the script tree and select "Add Variable..." or "Add Linked Variable". The first of these will simply add a variable with the same name as the parameter. The second will add a variable and search for the value throughout all the other parameters in the script, linking all parameters with the same value as the variable to it.
- Use the "Tools->Create Hostname Variable" option in the menu. This is specifically for creating a variable to control the host name of all your requests.

7.2 Viewing Variables

You can see all the variables you have added to your script in the Variable View. This view is a tab in the Summary View which is normally at the bottom left hand corner of your Badboy Window (although you can move it around by dragging it). The figure below shows how the Variable View looks:

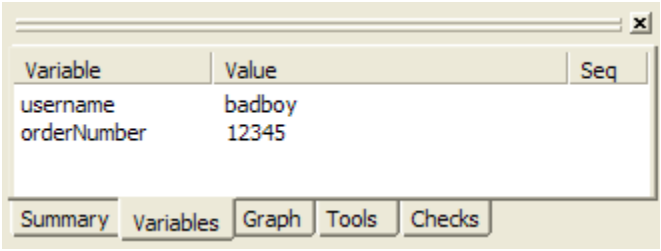


Figure 3: Variable View

7.3 Editing Variables

Editing the value of a variable can be performed by opening its properties dialog - just right click on the variable and select "Properties".

7.4 Using Variables

You can use variables nearly anywhere that you can enter a value into Badboy. To use a variable, refer to it by placing it inside curly brackets with a dollar sign in front. For example, to use a variable "foo", you could set a field in Badboy to be "${foo}". You can also use a variable embedded in the middle of other text. For example, if a variable is called "animal" and has a value "tree frog", then I can make a sentence "the ${animal} is green" using the variable. When Badboy plays your script it will substitute the values of variables into where you have put variable references.

7.5 Variable Value Lists

Variables in Badboy have two components:

- The current value (this is the value that is used when the variable is evaluated in your script)
- A list of future values

You can set the list of values that a variable has by opening the variable properties dialog (double click in the Variable View or right click and select "Properties"). When you do this you can add a list of values to the variable. If you want to make an expression that refers to a particular value in a variable's value list then you can place the index of the value in square brackets after the variable's name. For example, ${animal[3]} would refer to the fourth value in the value list for the variable "animal". (*Note that variable values are numbered starting at zero!*).

7.6 Incrementing Variables

A powerful way of using variables becomes apparent when you combine them with "Increments". Incrementing simply means to change the value of a variable to a new value, based on a strategy that you can specify. By default, Badboy will first look for the next value in the variables Value List (see below). If the variable doesn't have a list of values set then it will just assign a random value to the variable. See Incrementing Variables for more information about using Increments.

7.7 Setting Variables as Part of your Script

Sometimes you may want to set a variable as part of playing your script. Often it is useful for a variable to contain the result of an operation that your script has performed so that it can be used elsewhere. For example, if your script creates an "Order", you might make a variable to contain the order number so that it can be used in other parts of your script later on. To help you do this, Badboy provides a "Variable Setter" item in the Toolbox.

The Variable Setter properties dialog offers a number of ways for you to set the value for a variable. These include:

- Setting a list of fixed values separated by commas
- Setting values by extracting content from the browser window (based on a regular expression match)
- Loading content from a file (optionally, parsing that content into variables in CSV form)

The figure below shows the Variable Setter Properties dialog and the options that are available:

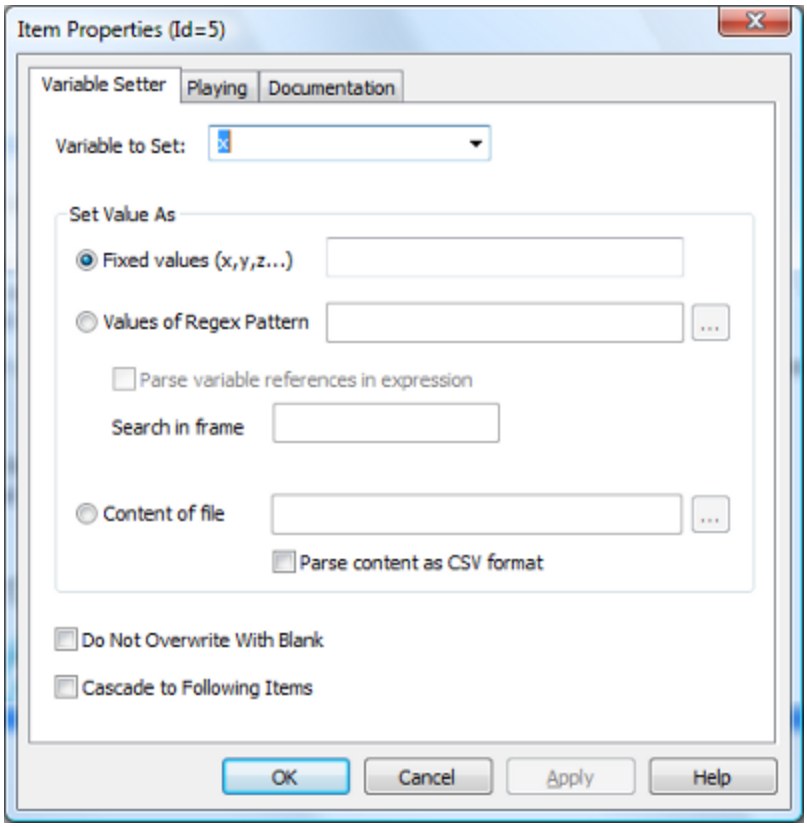


Figure 4: Variable Setter Properties

If you choose the "Content of File" option then you can also optionally choose to parse the content of the file as CSV (Comma Separated Values) which will then set variables in Badboy with the column names found in the parsed file in addition to the original variable you selected to set in your Variable Setter. This can be an effective and quick way to import data into Badboy from many tools which can export in CSV format.

7.8 Regular Expression Notes

Regular expressions are a particularly powerful way to set variables, because they allow you to scan the contents of the browser window and pick specific elements into your variable. The variable can then be used in playing other items later in the script. If the expression matches multiple times then you can loop over the values of the variable by setting the properties in a Step to iterate over the variable.

Sometimes you want to set a variable "opportunistically". That is, you don't know when a value will appear on the page, but if it is you want to store it. To do this, you can select the "Do Not Overwrite with Blank" option so that Badboy will only update the value and not set it to blank if a matching value is not found on the page.

Good references on Regular Expressions can be found by searching the Internet; however some simple notes on how Badboy implements the expressions are as follows:

- Brackets "(" and ")" are automatically special characters. If you really want to match a bracket then you must escape it with a back slash as "\)"
- If you provide a sub-expression using brackets then Badboy will populate the variable with the first sub-expression. If you do not provide a sub-expression then Badboy will place the match of the whole expression into the variable. For example, assume your content is "tree frogs live long". The expression "tree.*long" will return a match of the whole sentence because there is no sub-expression. However the match "tree (.*) long" will return the match of the sub-expression in brackets, "frogs live".
- By default Badboy will match the regular expression against the main Badboy window. If you want to match against a popup window, set the "target window" in the Play Properties for the Variable Setter.
- You can make an expression ignore case in matching by prefixing it with (?i). For example, the expression "(?i)Green Frog" would match "green frog".
- By default wildcards in expressions match across new lines. For example "Green.*Frog" would match even if "Green" and "Frog" appear on separate lines in the HTML source for the page. You can turn this behavior off by prefixing "(?-s)" before your expression. For example: "(?-s) Green.*Frog" would only match if the words "Green" and "Frog" appear on the same line in the HTML source for the page.

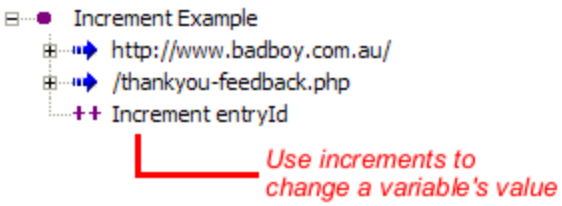
7.9 Automatic Variables

Badboy has a special kind of variable called an "Automatic Variable". See the section Automatic Variables for more information on these.

8 Incrementing Variables

Incrementing means to change the value of a variable to a new value, based on a strategy that you can specify. Incrementing variables let you make your scripts run for many different values of a variable.

To cause a variable to be incremented as part of your script, drag an Increment item from your Toolbox to the place in your script where you would like the increment to occur. The diagram below shows how an Increment looks in your script:



8.1 Increment Strategies

The new value assigned to a variable depends on the "strategy" you choose for the Increment. The default strategy that Badboy uses works like this:

1. Look for the next value from the variable's value list (see Variable Value Lists)
2. Look for the next value from an ODBC Data Source set by the user (see Using Data Sources)
3. If neither of the above is present, a random value will be generated for the variable using the current time and the id of the thread which is running the Badboy script.

If you wish, you can control exactly how Badboy increments the variable by setting the properties of the increment item. The table below shows the different strategies that you can assign and explains how they operate:

Strategy	Description
Default	A combination of the "List Value" and "Random Integer" strategies. If the variable has a value list then it will use the "List Value" strategy, otherwise it will use the "Random Integer" strategy; if it has one.
Random Integer	Appends a random integer based on the thread id and current time to the variable's current value, replacing any other numeric characters at the end of the value. For example, "treefrog1" might become "treefrog65267"
Value List	Uses the next value in the variable's value list. The value list might have been assigned by a Variable Setter, or it might come from an ODBC data source, or it might have been manually set by the user. (see Variable Value Lists)
Sequential Integer	Adds 1 to the value of the number at the end of the current value. For example, if the current value is "treefrog1", then incrementing would make it "treefrog2".

9 Using Data Sources

Sophisticated applications may need to test the same script using a whole range of values for their input. For example you may want to test all the boundary conditions for an operation – what happens when somebody orders quantities of 0, 5, 10, 1000 of a particular item? Or perhaps you want to test that ordering of every item in your warehouse works. Doing these kind of operations where you want to run the same script over many different values for a parameter (or variable) is greatly eased by connecting your scripts to a Data Source such as a database, spreadsheet or text file.

Badboy supports reading values for variables through two different techniques:

- Using a Variable Setter tool from the Toolbox
- Using ODBC Data Sources to read directly from a Database or Spreadsheet

This section describes the second of these options: how you can connect Badboy directly to a source of data such as a database or a spreadsheet. This means that if you set up an ODBC data source (which could be anything from an Oracle database to a comma separated text file) then Badboy can read variable values from it and when variables are incremented in your script they will iterate through the values in your source.

This help uses an Excel file as an example – but remember that you can use any ODBC source.

Another way to read data from Excel is to first export the spreadsheet data in CSV format by selecting File => Save As and then choosing CSV as the file format in Excel. Once exported as CSV, you can read data into your variables using the "Contents of File" option in a Variable Setter tool.

9.1 Data Source Requirements

By default Badboy requires data sources to satisfy certain requirements. (Note: you can change or avoid these requirements if you know how to write SQL for your Data Source. See the information on Using Custom SQL later in this topic for more information.)

1. All the values for all variables must appear in a single table
2. The table must have column headings that match the variable names that you want to be read
3. The values for the variables should appear under the corresponding headings
4. You may optionally include a column called “SEQ”. If this column exists then it will be used to order the values that are read from the data source. It must be a sortable data type (plain integer values are suggested). Note that if the column does not exist then there are no guarantees about the order in which the values will be used.

Here is how an example Excel spreadsheet satisfying these criteria might look:

	A	B	C
1	SEQ	Pieces	weight
2	1	100	5000
3	2	200	4000
4	3	300	3000
5			

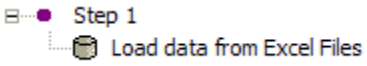
This shows how you would create an excel file to load values for variables “Pieces” and “weight”. Note that all names are *case sensitive*, and that in some cases the ODBC may convert the names to upper case without asking you. You will need to experiment with your data source to get this right.

Sometimes you might find that the wrong part of your spreadsheet gets exported by Excel to Badboy, or even a subset of the rows you want or nothing at all. The solution to this is to define the area you want Excel to export by selecting it and pressing Ctrl-F3. Then you can give it a name and when you configure the data source in Badboy the name should show up for you to choose as a table. This allows you to choose exactly the rows you want from your spreadsheet and ensures that Excel sends the correct data.

9.2 Adding a Data Source

Data Sources are configured in Badboy by adding "Data Source" items from your Toolbox into your script.

The image below shows how a Data Source item looks after adding to a script:



9.3 Setting the Properties

When you add a Data Source item to your script you will be given a choice of all the ODBC data sources on your system. Some data sources must be set up in advance using the Control Panel on your computer (from the Start menu choose Settings->Control Panel and look for the “Data Sources” applet, which may be under Administrative Tools depending on your version of windows). Other data sources, for example Excel files, can be accessed directly from Badboy. For Excel files there should be an “Excel Files” option in the drop down menu (you must have Excel along with its ODBC drivers installed). Select this option and Badboy will read the file and show you the worksheets inside.

You should then select the worksheet from which you would like Badboy to load values. Once you have selected the worksheet, Badboy will return you to the Data Source Properties page where you can select other preferences for how data should be loaded. The diagram below shows the Properties page for configuring a Data Source:

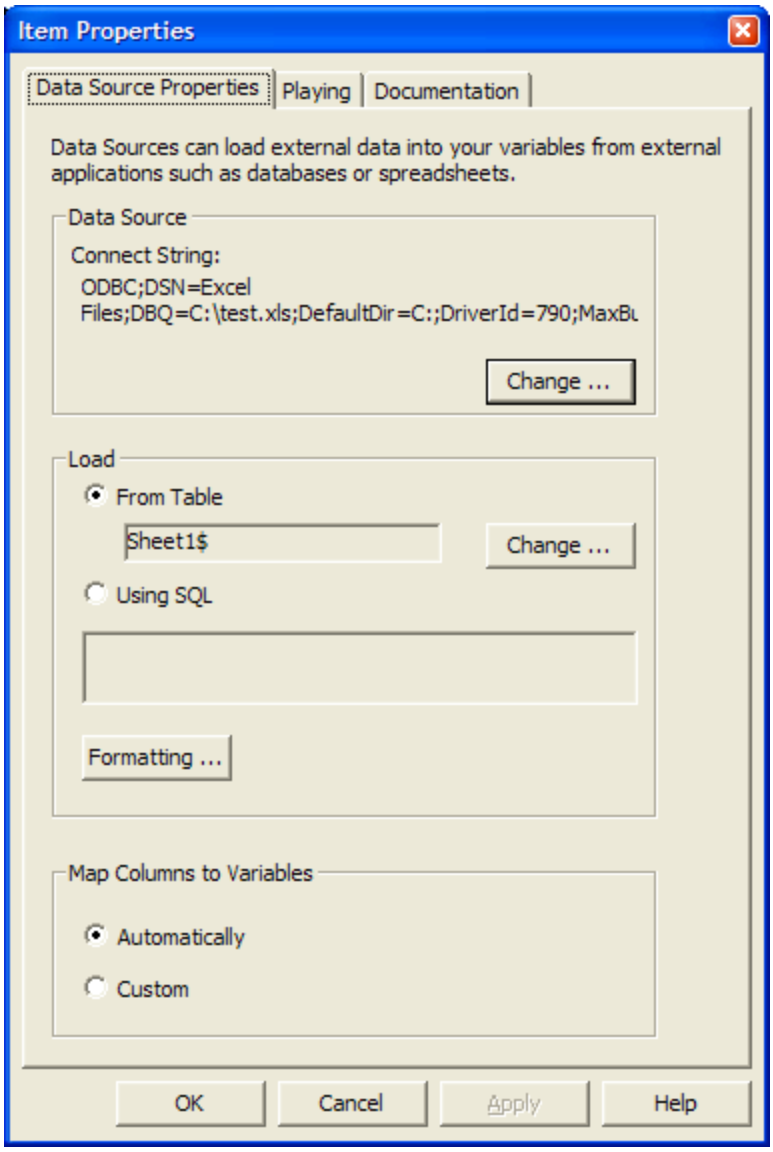


Figure 5: Data Source Properties

9.4 Controlling the Format of Loaded Data

You can review how the loaded data will look by clicking the "Format..." button. This will cause Badboy to load the data from your Data Source and show it to you. If the values do not appear in the format that you want then you can choose from among some format options on this screen to change how they are loaded.

The diagram below shows an example of this:

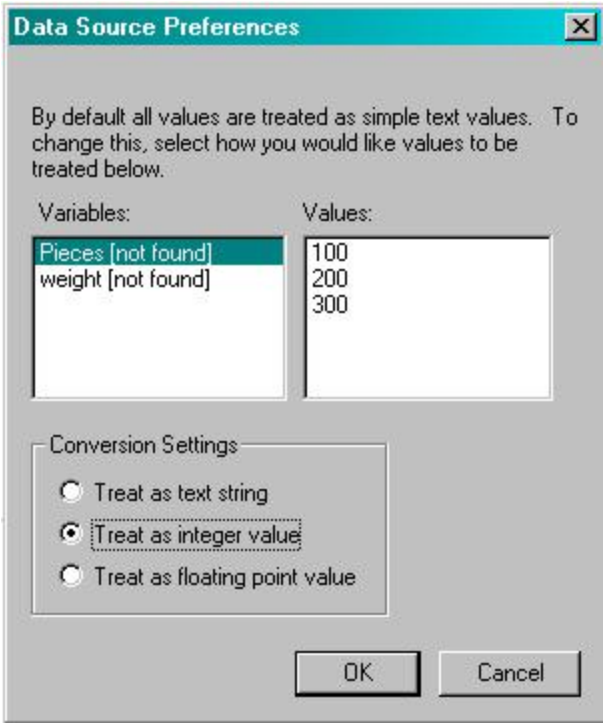


Figure 6: Data Source Preferences

9.5 Playing Data Source Items

Data is loaded when a Data Source item is played as part of the script execution. Hence after defining your data source you won't see any values loaded until you cause it to be played, for example, by right clicking on the item and selecting 'Play'.

Note: prior versions of Badboy loaded data from data sources when Badboy was started. If you have legacy scripts with data sources attached then they will still behave this way. New scripts, however, should use the methods described in this section.

9.6 Using Data Source Values

Badboy will automatically read the next value from your data source each time a variable is incremented. This could be as a result of an auto-increment (performed at the start of the script when a variable is flagged for auto-increment) or as a result of an explicit increment placed in your script.

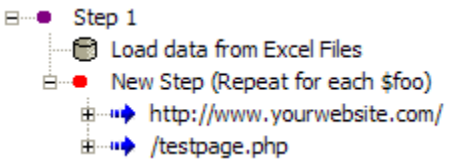
9.7 Looping Over Values in a Data Source

A very common scenario is to load values from a data source and then repeat a Step for each row of values in the data source. Badboy makes this very easy to do by using the looping properties of Steps. You can do this by taking the following actions:

- Add the data source to your script to load the data
- After the data source, add a new Step
- Put the actions that you want to occur for each row of the data set into the new Step

- Play the data source once to load the variables
- Set the new Step to loop over one of the variables loaded by your data source

When you have finished, your Script would look something like the figure below:



9.8 Advanced Options

Badboy offers some advanced options that you can use to load data in more sophisticated ways. These are:

- *Using Custom SQL* If you know how to write SQL that your Data Source understands then you can make up your own SQL statement to load data from your data source. Badboy will try to execute this statement against the Data Source and it will treat the returned columns and rows as a table from which to load data. You should make sure that your SQL statement returns appropriately named columns that Badboy can understand. By default, Badboy will convert the columns names to variable names.
- *Custom Variable Mapping* By default Badboy tries to use the columns returned from your table (or from your Custom SQL, if you have chosen that option) as variable names. If you want to change how the columns get matched to variables, choose the "Custom" option in the lower portion of the Data Source Properties dialog. This will show you a list of the variables you have defined to choose from. You can then "check" the ones that you want Badboy to map data to from your data source. When the Data Source item is played, Badboy will then load each checked variable, in order, from the corresponding column from the Data Source. In this way you can map any column from your Data Source to any variable that you have defined in Badboy.

10 Populating and Submitting Forms

HTML Forms are the core of many modern web applications. Form Populators help you populate and submit forms as part of your Badboy scripts.

You often don't have to use Form Populators at all - Badboy automatically records and sends the data from the Forms as Request parameters, so it typically isn't necessary to actually populate the form on the page. There are some times, however, when certain visual or functional effects are explicitly linked to the population of a form. Form Populators are here to help you when you need to test these kinds of pages.

Some examples are:

- A web site may perform scripted operations based on the content of Forms prior to submitting them. For example, sometimes the content of one field may be dynamically calculated from another field at the time the form is submitted by DHTML scripting. When Badboy operates in Request mode it may bypass the calculation and send an incorrect value.
- A web site may send back many fields in a form pre-populated. In Request mode, Badboy ignores the values that are pre-populated and instead sends a request containing all the values that were recorded originally. Although it is possible to make Badboy extract the values from the page and send them in the Request, it may be much easier to use a Form Populator to populate only the fields you need and submit the form.

- Many web sites take measures to prevent automated access which detect Badboy's Request mode and prevent it working. This is especially common on important pages such as login pages and order submission pages. Since Form Populators are a more realistic simulation of user activity, they often provide a work around when normal Badboy Requests fail.

10.1 Creating a Form Populator Manually

A new Form Populator added to your script appears as shown below:

10.2 Adding fields to a Form Populator

Once a Form Populator has been created, you need to add the fields that you would like populated to it. There are several ways to do this. The easiest is to let Badboy do the work by capturing the form automatically. You can do this very simply in a couple of ways:

- Click in a field of the form you want to capture and hit "Ctrl-Alt-f" on your keyboard. This will capture a Form Populator for the form where you clicked.
- Drag a Form Populator into your script, and click the "Capture" button in the Form Populators Properties dialog.

The Figure below shows how the Form Populator properties dialog appears:

Figure 7: Form Populator Properties

In order to use the Capture button, you should first choose the form that you would like to populate from by selecting either its index (i.e., the position of the form based on the order the forms appear on the

screen), or its name. If you aren't sure which form to populate from then the easiest method is to simple try capturing different indexes until the element you want appears. It is often useful to use the DOM View to help select the correct form. When you have selected the form you would like to populate from, just hit the Capture button and Badboy will capture form elements along with their current values from the form that you selected.

10.3 Using Regular Expressions

Occasionally you may need to populate a form whose name is not the same each time it appears in the page, but follows a fixed pattern. For example, in some applications the form may have a constant name followed by a number that changes. For these cases you can enable the "Use Regex" option and enter a Regular Expression that matches the pattern of the form you wish to populate. For example, if the form is always called "logonForm" followed by a 4 digit number, you can use an expression like this:

```
logonForm[0-9]{4}
```

10.4 Automatic Capture of Form Populators

Sometimes Badboy may automatically record Form Populators for you. This occurs when you record a Navigation (see Navigation Items) and you have modified some fields in a form on the page that you are recording. In this case, Badboy detects the modified fields and creates a Form Populator for those fields so that when you play back they fields will be populated correctly.

10.5 Playing Form Populators

Form Populators are easy to play - they behave the same way as other Badboy elements and can be played either using the right-click menu or as part of the normal flow of your script. You should be aware that if the form that you attempt to populate does not exist on the page, Badboy will generate a warning in the log file, but will continue playing without error. If you want to be assured that the form populated correctly, you may like to use an Assertion.

10.6 Form Values

Form Populators work by recording the values present in forms as "Form Values" that are children of the Form Populator in the script. When a Form Populator is played, each of its child Form Values is populated into the form. If you want to change which values are populated by a Form Populator, you can do that by modifying the Form Values by editing their properties.

10.7 Indexed Form Values

As well as a name for the field, each Form Value has an "index". Indexed Form Values are used when there are multiple elements in a form that have the same name or id. When this occurs, Badboy will record a form value for each form field that has the same name and an index that specifies which instance of the field having that name should be populated.

10.8 Select / Dropdown Boxes

Selection boxes present a minor complication in form population because each entry in the dropdown menu is in fact represented by two different values that are both useful in populating them on playback:

- The value *submitted* by the field to the server - this is often a code or a number
- The value *displayed* to the user - this is usually human readable text

By default Badboy records the first of these options - the value submitted to the server, and on playback it selects the option in the select box that has that same submitted value. This often works well but in some cases it may fail or be more difficult to use. For example, if the page uses arbitrary values for the options that change each time the script is played back then playback will not work because the original recorded value may be incorrect next time the script is played back, even though the text of the option values stays the same. It may be also may easier to select the option based on the displayed value when you are doing data driven testing such as loading the values from a spread sheet or other external source where the codes are not externally known but the text values are.

For these cases where the human readable value is more appropriate, open the Form Value properties and select the option "Match Visible Text for Select / Dropdown Fields". This will change playback to select the option where the text displayed in the dropdown matches the script Form Value rather than the code submitted to the server.

10.9 Sending JavaScript Events

By default Badboy populates fields in forms without any side effects. This means that the normal JavaScript events that occur when a person types into the field do not get fired. In some cases it is desirable to reproduce these events. For example: when the population of a field on the page triggers a change in another part of the page that is then needed for subsequent actions in your test. You can enable sending of these events by checking the box labeled 'Send DOM Events' in the properties for each Form Value.

It is also sometimes the case that a particular effect may only be triggered while a field has the focus. This can mean that even populating a field and sending the DOM Events does not enable the necessary interactions desired on the page. A common example of this is an auto-complete style dropdown. If you wish to test that correct values appear in the dropdown, these values may only appear if the field is populated and also retains the focus. You can cause a field to be left with the focus if you select "Retain Focus" on that field, **and** ensure that it is either the only field in a Form Populator with DOM Events enabled or it is the last field populated.

10.10 Using Form Populators to Submit Forms

If you wish, Badboy can also submit the populated form after it has populated it. This allows you to use Form Populators as an alternative to the usual Request and Navigation Mode playback mechanisms. Used in this way, Form Populators can often assist to automate operations that experience problems using other methods.

11 Using Templates

This section explains how Badboy allows you to structure your scripts using Templates.

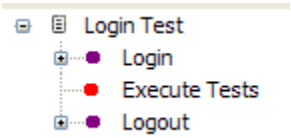
11.1 The Need for Test Templates

It is very common to find that many of your tests share a similar pattern. For example, it's very normal for all of your tests to have to first log in to your application and then log out afterwards. Sometimes there can be quite complicated sequences of actions that are common to a whole class of tests. Test Templates are designed to help you with this problem by letting you create the pattern for your test and save it, and then reuse that pattern for all of your Tests that share the same sequence of actions.

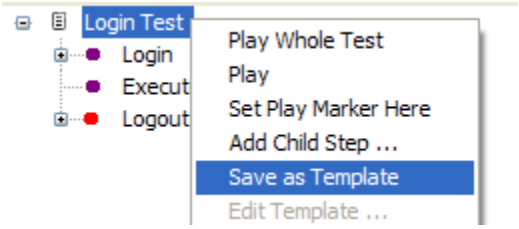
If you simply want to reuse one item or a group of items in a Step then you can save the items as an External Tool. The difference between External Tools and Test Templates is that Templates allow you to reuse the structure of a Test as well as the items in it.

11.2 Creating Test Templates

You create Test Templates by first creating a Test that matches the structure that you want Tests that use the Template to have. For example, if you want to make Tests that log in, perform a specific action and then log out, you might make a Template in the following form:



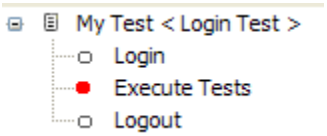
Notice that the Login and Logout Steps in the above example have items recorded in them, but the "Execute Tests" Step is empty. This step is a place holder for where tests that extend your template will place items. Once you have created a Test that matches the outline you want, right click on it and select the "Save as Template" option.



Badboy will then let you choose a file name and allow you to save the template. The filename you choose should end in the extension ".bxt" (this will be the default if you don't enter an extension).

By default Badboy saves templates in a directory called "Badboy Templates" created in your login user's Documents and Settings folder. However if you want to you can also save the template in the same directory as the script you are creating. This way it will always be easy to find the template file and you can simply zip up the folder if you want to send it to someone else and be sure they get all the templates needed by the script. Badboy will always check both these locations when looking for templates that are referenced in your script.

After you have saved your template Badboy will offer to replace the Test with one that extends the template. If you do this you will see the Test change: all the Steps that were previously in your Test will have changed to a white. These Steps are known as "bind points" as they are "bound" to the template and will execute what is in the template. The figure below shows how the Login Test looks after it has been saved and converted:



Notice that the Template that has been extended is shown in angled brackets so that you know it is the source for the Test's bound items. Also note that the Step that you left empty has been automatically made the recording Step - Badboy automatically assumes that empty Steps are meant to be implemented by Tests that extend the Template.

11.3 Overriding Steps

It is common that you may find that some of your Tests that extend a Template need to alter one of the Steps that the Template defines. This is known as "overriding" the Template Step because your own Test's version of the Step will execute instead of the Template version. Overriding a Template Step is easy to do - just right click on the Template Step that you want to override and select "Override/Implement" and Badboy will change the Template Step to a normal Step and let you record inside it.

12 Understanding Playback Results

When you play your script back Badboy records statistics about each page for you so that you can monitor the progress and review what happened later on. Badboy makes it possible to quickly see this information for any item in your script via the *Summary View*.

12.1 Summary View

The Summary View is the tabbed view that displays by default in the bottom left of your main Badboy window. The figure below shows how it looks:

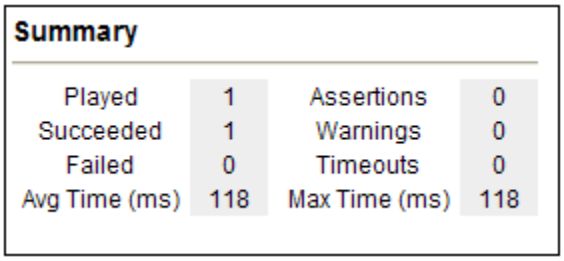


Figure 8: Summary View

The following table explains the numbers that are shown in the summary:

Statistic	Description
Played	The number of script items that played and returned a response
Succeeded	The number of script items that played and returned a successful response
Failed	The number of script items that played and returned an error response
Assertions	The number of assertions that have failed. (see Assertions)
Warnings	The number of Warnings generated. Warnings are problems that occur while playing which don't prevent playback but may indicate problems with your script or your web site. For example, if a page experiences JavaScript errors, or if a Form Populator executes

Statistic	Description
	but cannot find the specified form then warnings will be recorded as part of the response for the item.
Timeouts	The number of Timeouts that have occurred (see Timeouts)
Avg Time (ms)	The average time in milliseconds for items that played and received a response..
Max Time (ms)	The maximum time for any individual recorded response.

12.2 The Summary Hierarchy

Summaries are hierarchical - that is, each summary is actually a summary of its children. When you select an item the Script Tree, Badboy displays the summary for that particular item in the Summary View. If you expand the item in the Script Tree and select its children then you can view the breakdown of that summary for each of its children.

12.3 Adding Information to Summary View

If you like you can add your own documentation or notes to the information displayed in the Summary View. This can be a convenient way to explain what your script is doing. To do this, select the item in your script with the mouse and press "Ctrl-Shift-d".

12.4 Generating Reports

If you want a summary of your whole script in one page, you can generate an HTML Report. This is easy to do by selecting "View" and then "Report" from Badboy's menu. You can customize this report if you like - see Generating Reports.

13 Timeouts and Error Handlers

As everyone who uses a browser knows, things frequently don't go how you expect. Web sites go down, connections fail, systems and computers experience problems. You might be wondering, how can I make my scripts run reliably in the face of all this unreliability? One way that you can manage the unreliable world is by using Timeouts and Error Handlers.

- Timeouts ensure that items in your script do not take excessive amounts of time and ensure that if things go wrong your script keeps running and returns a result to you.
- Error Handlers allow you to retry or abort items that fail automatically.

Timeouts and Error Handlers can be configured on any playable item, including requests, navigations, and even Steps. If you configure a timeout for a Step the timeout will govern the whole Step so that the sum of the times for all the items in the Step will be limited to the timeout period.

13.1 Configuring a Timeout

To add a timeout, simply open the properties for the item on which you would like to configure the timeout and choose the "Playing" tab:

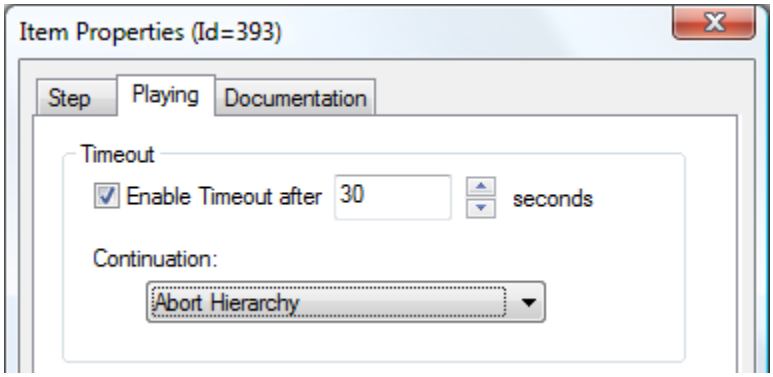


Figure 9: Configure Timeout

Once a timeout is configured, Badboy will monitor the item as it plays and if the time you set is exceeded, Badboy will execute the Continuation action you specified.

Notes and Limitations:

- Some items cannot be timed out during certain phases of their execution due to their nature. For example, once the Save item has started writing data to the disk it will continue even if a timeout occurs. When the save finishes the timeout will execute.

- Timeouts have a minimum resolution of 1 second. This means that the actual timeout may occur at plus or minus one second from when you configure it to occur. For low timeouts this may be a significant effect and thus it is not recommended to configure timeouts for less than 2 seconds.

13.2 Error Handlers

Errors include problems like server failures, HTTP protocol errors (for example, 404 Not Found), or network failures. By default, when an error occurs, Badboy will simply note the error and continue playing the next item in the script. You can change this behavior by configuring handling for errors on the Playing tab of the item, or on a parent of the item such as the Step or Test that it resides in.

The figure below shows how the error handling section of the properties dialog appears:

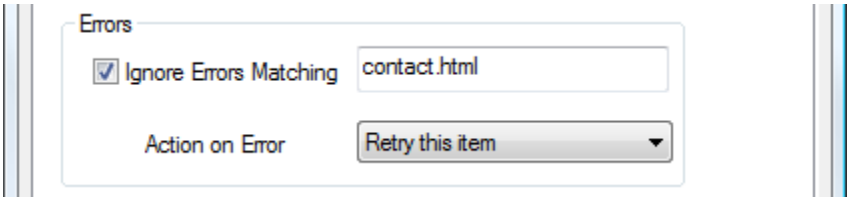


Figure 10: Error Handling

To make Badboy completely ignore an error, you can enter a regular expression pattern ("Regex") that matches the URL for which the error occurs, or which matches the error message that is reported by Badboy. You only need to match a portion of the error for the match to succeed.

*To make Badboy ignore all errors on an item, enter the Regex pattern: .**

To configure other handling for errors, change the "Action on Error" drop down box to specify the action you would like. For example, you can make Badboy abort the current Test if an error occurs, or repeat the containing Step. A particularly useful option is "Retry this item". The Retry option **removes** the error response and re-executes the item on which Retry is set one time before registering the error. For example, you can set a Retry on a Step and if any item fails inside the Step then Badboy will clear the error and try the Step again. If there is no error on the second try then the script will continue without reporting any problem. However if the item fails again on the second execution then Badboy registers the error. This can help you prevent transient or temporary problems from causing your tests to fail in situations where Badboy is running on unreliable networks or is subject to other occasional problems.

13.3 Continuation after an Error or Timeout

When you configure handling for errors or timeouts, you have the option to control what action is taken when the timeout occurs. These include:

Timeout Actions	Description
Continue from same position	Aborts the currently playing item and continues with the next item.
Abort this Step	Aborts the currently playing Step and continues with the item following that Step in the script.
Abort Hierarchy	Aborts the whole hierarchy of Steps right to the top level. Continues from the next item in the script following the top level Step containing the item that timed out.
Stop Playing	Stops the play sequence altogether. Badboy will stop and wait for manual intervention.
Repeat Containing Step	Rewinds back to the Step containing the item that timed out and plays the Step again. Note that any loop counters will be reset so that the Step will re-execute its looping behavior as well.
Retry this item	Discards any error response in the item and replays it *one time only*. If playback fails a second time, record the error and continue playing.

14 Mouse Clicks

Browsers are intimately mouse driven tools, so it is not surprising that there are times when nothing other than a real mouse click will replay exactly the effect you are looking for in a test. Some common examples of situations when this can happen are:

- Your web site uses Java Applets
- Your web site relies on complicated JavaScript effects that are only activated via the mouse
- Popup boxes, JavaScript errors or security dialogs prevent your tests from playing properly.

In all these situations a simple and effective solution can be to control the browser by automating mouse clicks in the window. Fortunately, Badboy makes recording and adding Mouse Clicks easy.

14.1 Adding a Mouse Click

To add a mouse click, just drag a Click Item from the Toolbox into your Script. The figure below shows how a Click Item appears in the script:

When you add a Click Item, the properties dialog box for the Click Item will show giving a number of options that you can choose to determine how the mouse click is performed. The following paragraphs describe these. The figure below shows how this looks:

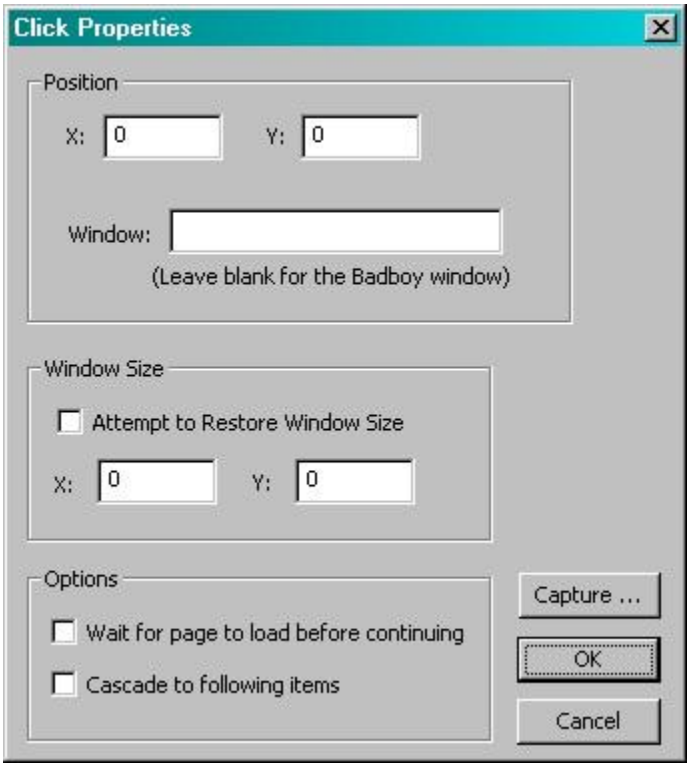


Figure 11: Click Properties

14.2 Window Name, X and Y Position

These parameters determine the window and the location within the window (in pixels) where the mouse click will occur. The mouse click is always relative to the top left corner of the window specified. This means that you don't need to worry about where the window appears on the screen (which can easily change), but only about where in that window the click needs to occur. Note that sometimes looks can be deceiving - some applications use many layers of windows and consequently if you enter the numbers manually you may have to experiment to get the right X and Y position. Sometimes you may even find that the values you need are negative. The best way to set all these parameters at once is to use the Capture Button (see below).

14.3 Capturing the Click from an Existing Window

By far the easiest way to create a Click Item is to capture it from an existing window on your screen. This means that you should set up Badboy (and any other windows you expect to be on the screen and open the Click Item properties and then click the Capture button. The next left click that you do will then be captured, including window title, X and Y position. Note that you are not limited to recording clicks inside Badboy's browser window - you can record clicks on any window on your desktop, even Badboy's menus or the desktop itself!

Tip: *Badboy offers a shortcut way to create Mouse Click Items: just click (anywhere) in a browser window that you want the click to occur in, position the mouse at the right location and the hit keys "Ctrl-Alt-M". Badboy will add a Mouse Click recorded at the selected location to your script tree.*

14.4 Restoring Window Size

One problem with using mouse clicks is that sometimes if windows change their size and shape then they may lay out their contents differently. This is especially true with HTML based views such as Badboy's. For this situation you can check the "Attempt to Restore Window Size" box prior to capturing the click with the "Capture" button. In this case the window size will be captured and then restored prior to executing the Click operation. *Note: although Badboy will always try to resize windows if you check this option some windows may refuse to let Badboy resize them, preventing it from working. Fortunately, the most important window - Badboy's own window - should always work.*

14.5 Cascading Click Items

You might wonder, what do I do if I know that a window that I want to click on will appear, but I don't know *when*? This situation appears more commonly than you might think. Most especially, it occurs when unexpected dialog boxes show during your tests - (examples are JavaScript Alerts, confirmation boxes, security warnings or JavaScript errors). To handle these it can be very useful to be able to say "If this kind of window ever appears, click on it to make it go away". This is exactly what Badboy allows you to do with Cascading Click Items. The term "Cascading" means that the Click will apply for all items following and below the Click Item in the Script, but *not* outside of the Step in which the Click is positioned. When a Cascading Click Item plays, it starts a background "watchdog" that continuously looks to see if a window matching the name you have given is present. If it finds one, it executes the click. When the play position moves outside of the Step in which the Click is positioned, the watchdog is terminated.

Note: if you want Badboy to click on a window, you must often place the Cascading Click Item **before** *the item that causes the popup to occur in the script. If you put it afterward it may not execute until after the actual popup has been closed.*

14.6 A Common Problem: Capturing Modal Dialogs

Sometimes you may want to capture a click on a window that only shows up during playback and you may find that you can't open the Click Properties dialog to capture the Click coordinates. A simple trick will help you deal with this: run a *second* instance of Badboy to cause the window to show, then capture the Click using the first instance. This trick works because Badboy can capture Clicks from the second window, but will still happily play the Click back on windows generated by the first instance . Using this trick you can capture a click for any kind of popup or dialog that may show.

14.7 Disadvantages of Mouse Clicks

You could potentially write your whole test using "Click" items. You might find this a very easy away to get your tests working. However there are some disadvantages of creating test scripts this way that you should seriously consider before basing significant portions of your tests on mouse clicks:

- Because Mouse Clicks are critically dependent on the exact layout of the page to work, they are harder to maintain. A small change to the layout of the page, the size of the font or the width of a table will throw the position of the Click completely off. This means that although your test will be easy to create, as time goes by and your pages change it will begin to break. You will have to go back and re-record the Click items when this happens. When your tests fail, you will have to wonder if they really failed or if perhaps just one of the Mouse Clicks has stopped working.
- Mouse Clicks may depend on your browser's configuration to work. Different users may have different fonts and other browser settings that may subtly affect the layout of the page. If you record scripts with a lot of Mouse Clicks you will find it harder to share them with others because of this.
- Mouse Clicks rely on having a real, true browser window present. This means that if you try to run your scripts with bbcmd, the Badboy command line runner, you will find that they won't work. They also won't get exported if you save your script in JMeter format. You will even find that you can't minimize the Badboy window while the Click Item is playing.

For the reasons above over-use of Click Items is not encouraged. However using them carefully and wisely will let you test things that might otherwise be impossible, and can make your testing life easier when you are confronted with difficult problems

15 Assertions

Testing a web site can be a frustrating, tedious and monotonous task. Badboy helps you ease the task by automating the playback of testing scripts, but you will still get sick of watching every screen to see if it worked the way you expected. You might wonder, can't Badboy help with this too? This is where Assertions come into play.

15.1 How Assertions Work

Assertions are a way that you can tell Badboy to perform automatic checks to make sure that your web site is working as you expect. You can think of an Assertion as a "statement of truth" - something that you say is true about your web site. If you make this statement, Badboy can make sure it is true and warn you if it is not.

Assertions in Badboy are made up of two parts:

- The Assertion item itself - describes how checks are executed and what to do when they fail
- Checks - these are items that are added as children of the Assertion and examine different aspects of the page or the script to determine if your Assertion passes or fails.

15.2 Adding Assertions

You can add an Assertion anywhere in your script just by dragging one from the Toolbox. A new Assertion added to your script appears as a question mark symbol as shown below:

The question mark indicates that the assertion has not been tested yet. When the assertion plays, it will change to either a tick or a cross depending on whether the assertion was found to be true.

15.3 Checks

An Assertion by itself won't check anything and will always pass - you need to add Checks from the Toolbox to it to describe which properties of the page to check. There are several different kinds of Check items that Badboy supports. The table below shows some of the different types:

Name	Icon	Description
Content Check		Checks for the presence of some text on the page.
Response Check		Checks characteristics of the response time and size
Color Check		Checks for particular colors on the page. You can specify a range of colors and approximate location to allow for slight variations.
Summary Check		Checks the summary information for items in the script. For example, it can check the number of times an item has played, errored or timed out.
JScript Check		Executes JScript that you provide and passes or fails the assertion based on whether your JScript returns "true" or "false".
Variable Check		Checks the value of a variable to see if it matches a regular expression that you provide.
Window Caption Check		Searches for a window with the caption you specify and optionally look for a child window (such as a button, text field, checkbox, etc.) with specified text. This check does not match text on web content, only on native windows (including, however Java applets embedded in web content).

15.4 Easy Assertions

The most common kind of Check is the Content Check, so Badboy provides an especially easy way to add Assertions with Content Checks to your script. To use this function all you have to do is select with the mouse the content you would like the Assertion to check and then click the "Easy Assertion" button (shown below).

You can find more information about Content Checks in the topic Content Checks.

*By default the Easy Assertion button adds a positive Assertion, i.e. an Assertion that states that the content **should** exist on your page. If you want to add a negative Assertion, i.e. that the selected content **should not** exist on the page, hold down the SHIFT key while clicking the button.*

15.5 Assertion Properties

An Assertion has a number of properties that control how it works. Below is an example of the Assertion Properties dialog showing the options that you can choose:

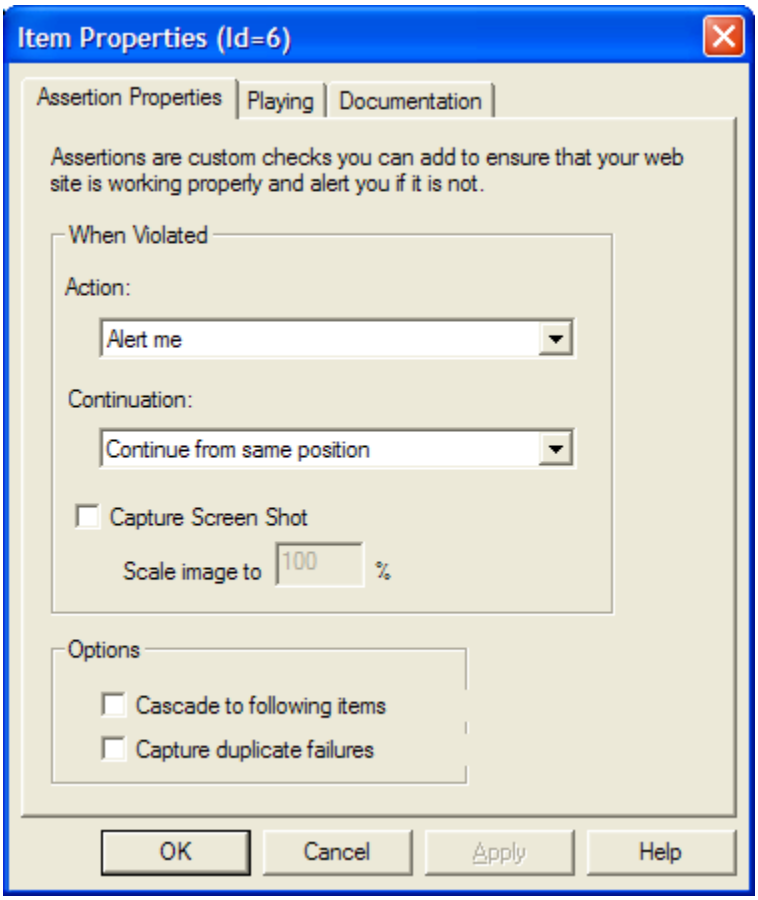


Figure 12: Assertion Properties

15.6 Cascading Assertions

Cascading Assertions differ in that they have a longer lifetime than just the moment they are played. When you check the "Cascade to following items" box Badboy will test the Assertion after every item is played until the parent Step, Test or Suite of the Assertion is exited. Thus Cascading Assertions are useful for detecting general error conditions. For example, you could specify that no page anywhere on your site should ever contain the words "Internal Server Error". By adding this as a "Cascading" Assertion at the top of your script you can ensure that Badboy will flag such an error no matter when or where this text occurs.

The scope of a Cascading Assertion is limited to the Step, Test or Suite that is the direct parent of the Assertion. This means that the Assertion flows "downward" into all the Steps that are children of the Assertion's parent Step. Once the play position moves out of the Assertion's parent item the Assertion no longer applies. If you would like your Assertion to apply across a whole Suite of Tests then you can put it immediately under the Suite item in your script, so that it plays before any of the Tests execute and it stays active for the whole Suite.

15.7 Violation Actions and Continuation

Very often an Assertion failure may represent a severe error in your web site or application. In such a case you may not want your test to continue, but rather to abort either the whole script or the Step that is playing. To achieve this, select the action you would like from the "Continuation" box in the Assertion properties.

15.8 Capturing a Screenshot

Often just the fact that an Assertion failed won't be enough to describe in detail what the problem was that occurred. For example, if you matched "Error 500" in your Assertion text, you would want to see what else was on the screen at the time to tell what the problem really was. Badboy makes this easy by giving you the option to automatically capture a Screen Shot of the browser window any time an Assertion fails. You will then be able to view the Screen Shot at any later time by opening the captured image in your Script. It will also be exported if you save an HTML Report of your Script's execution.

15.9 Waiting for Assertions to Pass

Sometimes the content that an assertion is checking for may be delayed in appearing on the page. In general this should not be the case because Badboy should wait for pages to completely load before checking them with Assertions. If, however, you have a special case where this is not sufficient (for example, a page where some content is dynamically added after some time), you can check the box marked "Wait up to ... seconds for Assertion to succeed" and enter an arbitrary amount of time that Badboy should wait and watch for the Assertion to become true.

If the Assertion succeeds before the time expires then Badboy will continue playing immediately. If, however, the time expires and the Assertion has still not succeeded then Badboy will fail the Assertion.

Badboy checks an Assertion periodically (several times per second) while waiting for it to succeed. Thus the Assertion need only become true for a momentary amount of time for it to pass. If, however, it is only true for a very small amount of time (smaller than about 200ms) then Badboy may not notice the Assertion has passed and may keep waiting.

16 Content Checks

Content Checks are a kind of Assertion Check that examines the content of the web page in the browser to see if it contains particular text that you specify.

16.1 Content Check Properties

A Content Check has a number of properties that control how it works. Below is an example of the Properties dialog showing the options that you can choose:

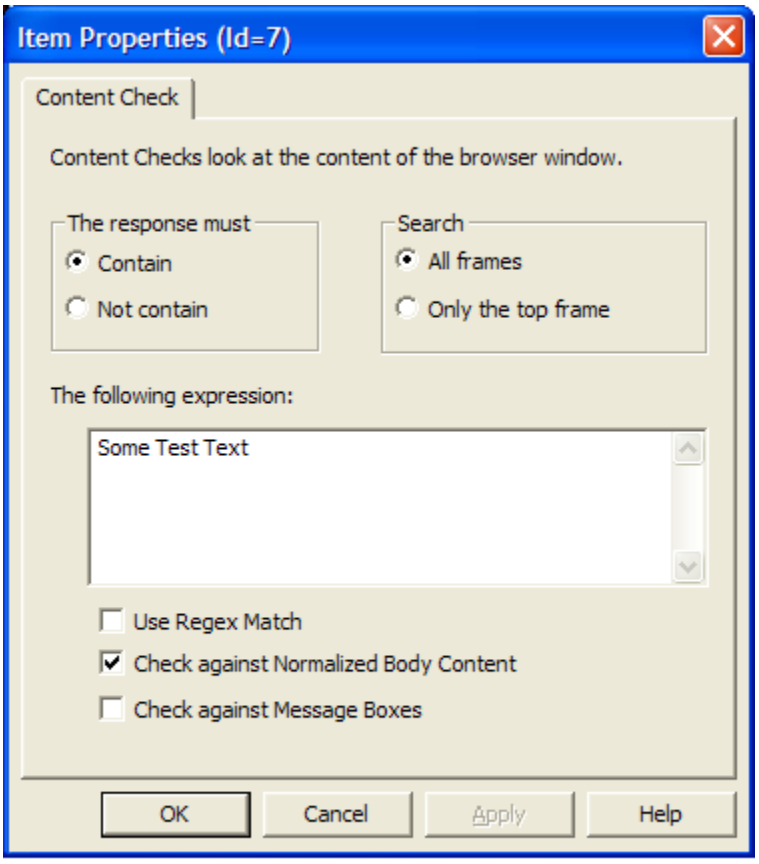


Figure 13: Content Check

In a simple case you will just type in some text that you want to test for and select either "Contain" or "Not Contain" depending on whether the text is supposed to appear on the page or not. Some more complex options are described below.

You can reference Badboy variables inside your expression using the standard ${variable name} syntax.

16.2 Regular Expressions

Sometimes you can't say exactly what a page should look like but you *can* predict it's general form. In this case you can use the "Regex Match" option in the Assertion properties to specify a Regular Expression pattern to match the content that you want to trigger the Assertion. For example, you might want to check that a person logged in successfully by looking for the login message with an Assertion such as "Welcome [A-z0-9]*\. You have logged in successfully to account [0-9]*."

16.3 Matching Against Normalized Body Content

Badboy offers an advanced mode for matching content which can both make it easier to match and also enable matching in situations where it may otherwise be impossible. When the *Normalized Body Content* option is selected:

- HTML is extracted from the Body element of the HTML at the time the Assertion executes instead of the time the page was loaded.
- The HTML is processed in several ways to ensure that it matches uniformly even if the page content varies in certain ways. These include: all new lines are removed, all tags are made upper case, and all attributes are ordered similarly

Using this option can help if you have Pages which write or modify their own HTML using a scripting language (such as JavaScript). Note that this option can only match content inside the HTML's body element on the page.

Badboy's Easy Assertion button always creates Assertions with Normalized Body Content enabled. You should not disable this option if you create an Assertion with the Easy Assertion button unless you are sure of what you are doing!

16.4 Common Problems

A common problem encountered when using Assertions is that the Assertion fails to trigger even though the text it is supposed to find appears visually on the page. The key to understanding this problem is to realize that the HTML source for a web page often contains special characters that do not appear visually in the browser. For example, the words "tree frog" seen visually in the browser might actually be represented as "tree frog" in the HTML source. Because Badboy tests the actual HTML source for the page it may not find the words "tree frog" entered into an Assertion.

One way to avoid these problems is to just highlight the text on the page and use the Easy Assertion button (described above). In this case Badboy will do the work to examine the source of the page and format the content in such a way to ensure that it matches. If you don't want to use the Easy Assertion method (for example, if you are writing a complex Assertion using regular expressions or Badboy variables) then you should right click and use the "View Source..." option to show the HTML source for the page that you are testing. Then you can search for the text you are trying to test and copy it exactly from the HTML source into your Assertion expression to make sure it gets accurately matched.

17 Summary Checks

Summary Checks are a kind of Assertion Check that examines the summary information about an item in your script such as the number of times it has played, succeeded or failed.

17.1 Adding a Summary Check

To add a Summary Check, drag it from the Checks toolbox in the tabbed view and drop it into your script at the position you would like it to execute.

When you drag a Summary Check from the Checks Toolbox, slightly different behavior occurs depending on whether you drop it inside an existing Assertion. If you drop it inside an existing Assertion then Badboy simply creates the Check in there. However if you drop it straight into the script then Badboy will automatically create a new Assertion and put the Check inside it.

17.2 Summary Check Properties

A Summary Check has a number of properties that control how it works. Below is an example of the Properties dialog showing the options that you can choose:

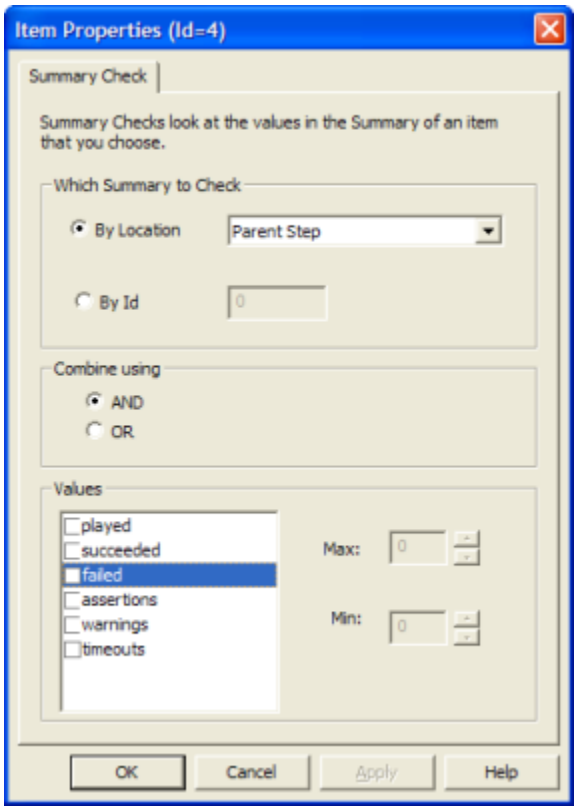


Figure 14: Summary Check Properties

The options you can choose are described below.

17.3 Choosing Which Summary to Check

A Summary Check can only check the summary of one specific item in your script. There are two ways you can identify which item it should check:

- **By Location** - This option determines an item to check by its relation to the position of the Summary Check itself. For example if you want to verify that the current playing Step has not experienced any timeouts, you can choose to check the Parent Step of the Check, and set timeouts to zero in the Values section.
- **By Id** - If you wish you can simply put in an Id to check any item in your script. While this provides a lot of flexibility, it is not usually recommended since you might later change the Id of or remove the item it is referring to, in which case the Check will not play properly.

If you want to check the summaries for a group of items then you can place them in a Step and check the summary of the Step.

17.4 Setting Values to Check

To set the values to check, just check the boxes in the Values list for each summary property that you want to be examined, and then specify a maximum and minimum value for that property. If any property falls outside the range you specify then the check will fail.

17.5 Combining Value Checks

By default the Summary Check requires that **all** of the values selected are in the specified ranges. In this case the values are combined using and strategy. You can, however, choose an OR strategy in which case any one of the values being in range will allow the Summary Check to pass.

18 JScript Checks

Sometimes you need to check your page in ways that may be more complicated than just looking for a fixed piece of content. For example, it might be that the location in the page matters, or the style that it is rendered in is important, or sometimes you may even wish to use sophisticated logic to check your page. For example, you might decide that for all of your order forms, the quantity in the "Total" field should always equal the sum of the quantities of the line items in the order. To make sophisticated checks like this, the best way is to use a *JScript Check*. JScript checks allow you to use any JavaScript expression that you can write to check the content of your page.

18.1 Adding a JScript Check

To add a JScript Check, drag it from the Checks toolbox in the tabbed view and drop it into your script at the position you would like it to execute. You can drop it inside an existing Assertion to add the Check to that Assertion, or you can drop it outside, in which case a new Assertion will be created.

18.2 JScript Check Properties

A JScript Check has a number of properties that control how it works. Below is an example of the Properties dialog showing the options that you can choose:

Figure 15: JScript Response Check Properties

18.3 Selecting the Frame to Use

Some websites may use frames to create separate windows within the page where content is rendered. If that is the case and you want to check a specific frame, locate the correct frame and choose it in the drop down list of frames.

If you have trouble identifying which is the correct frame, try locating the content using DOM View (Press Ctrl-D) and then checking which frame is the parent of the element(s) you are looking for.

If you want to make an Assertion that checks all of the frames in the document then you can select the "All" option. In that case, Badboy will execute the JavaScript in every frame of the document. This is very useful for writing general rules that you think should always be true. For example, if your company policy is that all pages must render in the Standards Compliant mode of the browser, you could write a rule that checks that for every frame of the window by using the "All" option.

18.4 Writing JavaScript for JScript Checks

The JavaScript for JScript Checks is executed as if it was inside the body of a function. The script should use the "return" keyword to return a value of either "true" or "false" to indicate whether the check passes or not.

Example 1

The following example checks that the Title of the window is not blank:

```
if(window.title != '') {
  return true; // passed
}
else{
  return false; // failed
}
```

Example 2

This example checks that IE is rendering in Standards Mode:

```
return window.document.compatMode == 'CSS1Compat';
```

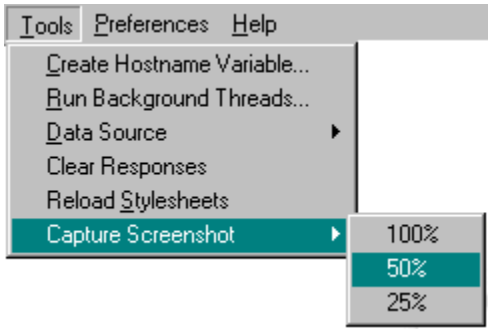
19 Taking Screen Shots

When something goes wrong you often need a way to capture the problem so that you can show it to your colleagues, development team or otherwise document the error. The simplest and fastest way to do this is to take a screen shot of the browser window at the time the problem occurs. Badboy takes this concept and extends it - giving you not only the ability to easily and simply capture screen shots but also to scale them to your preferred resolution, and to capture them automatically when Assertions fail.

19.1 Capturing a Screen Shot Manually

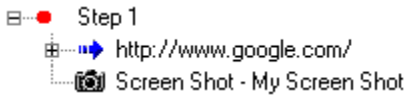
You can easily capture a screen shot manually using either the Toolbar or Badboy's menus. To take a full-size screen shot, just click the camera icon on the toolbar. The screen shot of the browser window will be captured and placed on the clip-board so that you can paste it into another application - for example, a Word document or your defect tracking tool.

You can also capture screen shots using the Tools menu, which lets you capture reduced size images at 50% and 25% size if you wish.



19.2 Capturing a Screen Shot as part of your Script

If you like you can take screen shots at a predetermined point in your script by adding a Screen Shot item directly. Adding a Screen Shot Item is easy: just drag it from the Toolbox to the place in your Script where you would like it to go. The figure below shows how a Screen Shot item looks in your Script:



When you add a Screen Shot item the Screen Shot property dialog will open. Here you can add a label that will be applied to your images when they are captured and you can specify the scaling you would like to apply. It is often useful to reduce the size of captured images so that they take less space. Badboy makes this easy by letting you specify the percentage size to which the image should be reduced. When the Screen Shot item is played it will capture the browser window as an image which appears as an item under the Screen Shot in the Script tree.

19.3 Capturing a Screen Shot Automatically when an Assertion Fails

Badboy can capture a Screen Shot automatically when an Assertion fails. Images captured this way will appear underneath the Assertion in the tree where you can double click on them to view them or copy them to the clipboard by right clicking and selecting 'Copy to Clipboard'.

To enable this feature, select the 'Capture screen shot on failure' in the properties for your Assertion.

19.4 Using Screen Shots for Manual Reviews

Sometimes there are cases where automated testing simply cannot replace the function of a real, intelligent human viewing a page. Reviewing the quality of images, or the exact layout of items on a page, the readability of fonts, or many other possible requirements are things that only a human can do. However this does not mean you cannot use automation for such tests: only the *review* part requires a human, not the processes that generate the pages to be reviewed. Therefore Badboy supports the idea of flagging screen shots as *requiring review*. To do this, just check the box in the Screen Shot item properties window called "Flag this Screen Shot for Review". When executing such a Screen Shot item, Badboy takes the screen shot and notes that it needs manual review. Then when the script is finished running, you can generate a *Review Report* containing all the items requiring review along with the notes from the Documentation in a simple format that allows a human to quickly scan each screen shot and validate that it satisfies the requirements. To view the Review Report, select it from the menu using View => Report => Review Report.

You can enter notes about what requires review in the normal Documentation section of the Screen Shot item's properties window and these will be shown alongside the screen shots captured in the Review Report

You can also send Review Reports by email using a Send Item from the Toolbox. This can make a very convenient mechanism whereby you can run your Badboy tests off line and get a report of items requiring review sent to you by email afterwards.

19.5 Capturing Screen Shots of Response Time Graphs

Badboy can also capture a screen shot of the Graph that is normally displayed in the "Graph" tab next to the Summary View. This is a useful way to capture a display of the response times of items in your test so that you can compare them to previous instances.

Capturing the Response Time Graph is particularly useful if you want to include the graph in HTML reports, as the captured graph will be exported along with other screen shots in the report. See Generating Reports for more information about creating reports with Badboy.

Capturing screen shots of Response Time Graphs requires a valid license key to be entered under Help => Enter License Key

20 Creating Reports

After you have played your Badboy script you can review the results by looking at the Summary View and browsing the script to look at items that have passed and failed. For some people this is perfectly reasonable but for others it lacks some important features:

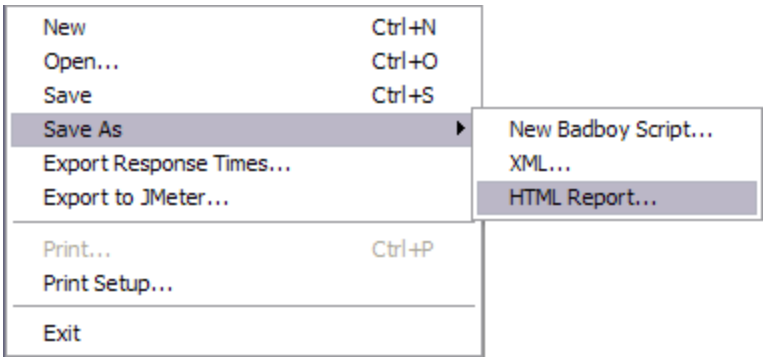
- The summary view isn't very detailed. It only shows 5 top level statistics for a particular item you select in the script.
- You can only view the summary properly by running Badboy so you can't easily email it to other people who might not have Badboy.
- It doesn't support integration with external programs or documents. Many people would like to put the results from their Badboy tests, for example, into a word-processor document or spread sheet.

Fortunately, Badboy offers some great features to generate reports for you which give you an unlimited amount of detail about how your script played. This section describes how you can use Badboy to generate such reports, both manually and also as part of your script execution.

20.1 HTML Reports

The easiest report to use is the HTML Report - you can easily see it any time just by clicking the "View" menu and selecting "Report". When you do this, Badboy will save the an HTML version of your script (including images from screen shots) to a temporary file and will show the report in the main Badboy window. This simple report is easy to use and a great way to get a quick snapshot of your script.

If you want to save the HTML report to keep on your computer or send to someone else you can use the File=>Save As menu:



The figure below shows an example of how the HTML Report looks after you generate it:

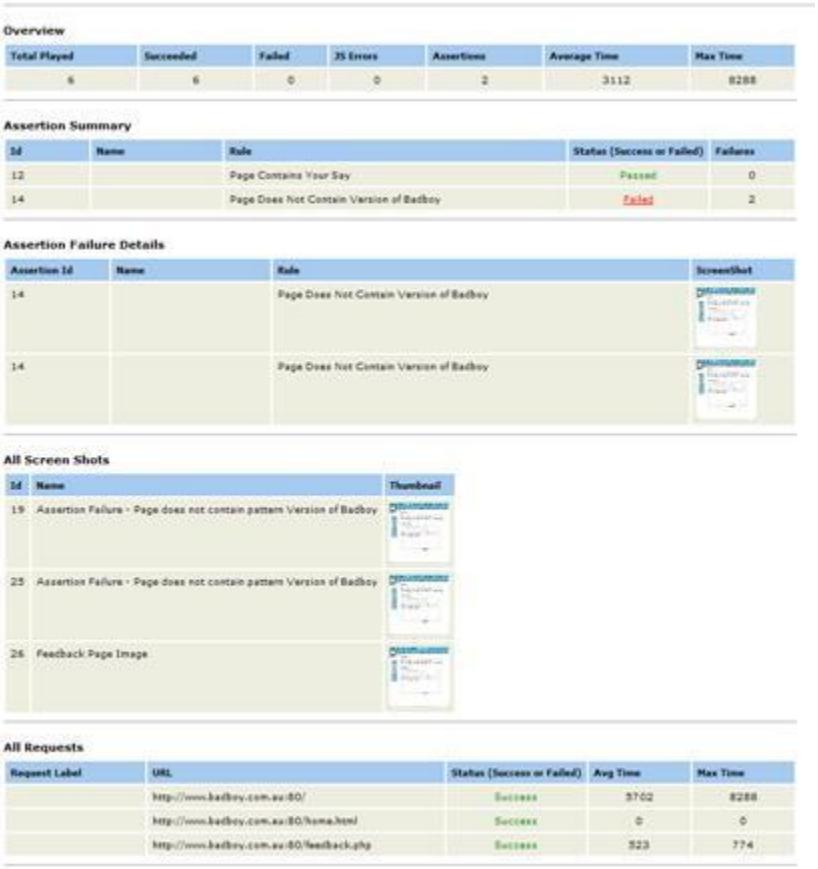


Figure 16: HTML Report

When you save the HTML Report, Badboy saves the images that are used in the report in a sub-directory that is named with the same name as the HTML file with "-images" appended. If you want to copy your report to another location then you need to copy this folder with the images as well. To send to another person, the most convenient way may be to "zip" the files using a Zip utility or a "Compressed Folder" (Windows XP only).

20.2 Saving an HTML Report as part of your Script

Often you want to run a test and have it automatically save a report for you so that you can review the results later without having to remember to manually save the report. This is especially useful when scripts run unsupervised as part of a batch process. You can do this in Badboy by using the Save Item from the toolbox. Just drop a Save Item into your script at the place where you would like it to execute. The Save Item Properties dialog will show (see image below). You should then select the "Script" radio button and choose "HTML Report" from the adjacent drop down menu.

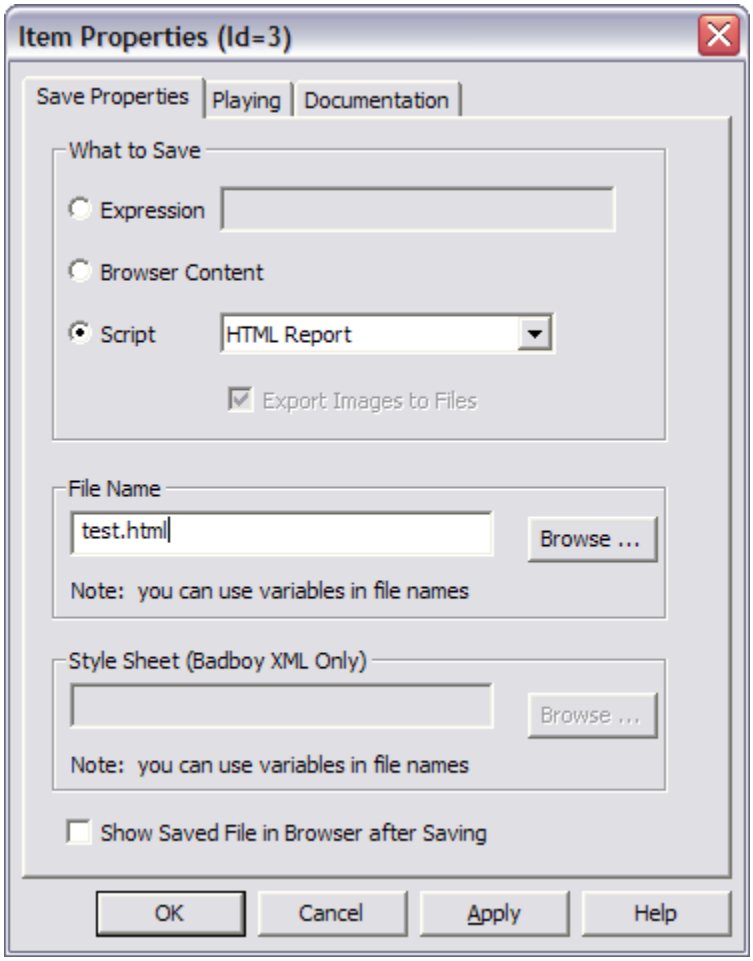


Figure 17: Save Item Properties

If you would like Badboy to display the report in the browser window as part of playing the script, check the "Show Saved File in Browser after Saving" option in the properties dialog box.

20.3 Including Screen Shots in your Report

Badboy automatically includes any screen shots that were captured as part of your script in the saved reports, including those that may have been automatically captured by Assertions. Additionally, if you configure screen shots of Response Time graphs then your report will display graphs of response times for the configured items as well (see Taking Screen Shots for more information).

20.4 Exporting Raw XML

Many tools can read and write XML files to import and export data. If you such tools then you can use them with Badboy by saving the script as XML. To do this, follow the same steps as above for the HTML Report, but choose either "Badboy XML" or "Badboy Script (XML)" instead of "HTML Report" in the menu and/or properties dialog option. Saving as XML can also be a very convenient way to integrate your scripts with other automated processes as the XML is a human-readable format - allowing you to easily write scripts that perform tasks based on the output of Badboy.

Note that Badboy supports two different XML formats - the "Badboy Script (XML)" format is now the preferred format and should be used unless you have a particular reason to use the old format. See the File Formats section for more information.

20.5 Generating Custom Reports

With some simple XML knowledge the Save Item allows you to generate *customized* reports containing exactly the details that you want or even in a completely arbitrary format that you design. This is achieved by specifying an *XSL Style sheet* for Badboy to use to format the saved file. XSL Style sheets are a highly popular and standard way of formatting XML documents. They use a file called a "Style sheet" to describe what should appear in the output document based on the input document (in this case the input document is Badboy's XML file format). You can find many sources of information on how to write XSL on the internet.

Since Badboy itself uses XML Style sheets to perform it's file export functions, the best way to start learning about them is to look in your Badboy install directory for a directory called "xsl". In here you can see all the style sheets that Badboy uses to perform functions such as producing the HTML Report, exporting JMeter files and rendering the Summary View in the Badboy window. If you wish, you can even modify these default style sheets to reflect your own preferences.

Note: if you modify the default style sheets, please remember that re-installing or upgrading Badboy may overwrite these style sheets and you may lose your work. It is strongly suggested to keep a backup so that you can restore them if this happens!

To apply a style sheet of your own, use the Save item and choose the "Script" radio button and then "Badboy XML" as the output format in the properties dialog. Then you will see that the "Style sheet" box becomes enabled and you can choose the style sheet file for Badboy to apply. If the box is blank Badboy will not apply any style sheet and will instead export raw Badboy XML.

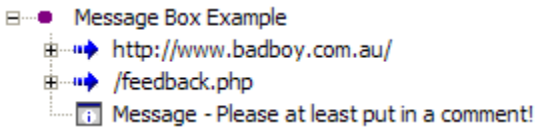
Badboy used to save reports using BadboyXML by default. However in version 2.0 a new XML Format was introduced and the HTML report is now generated by default from this new format. If you still wish to use the old format, you can select Badboy XML in the Save Item dropdown instead of "Badboy Script (XML)" and then the old Badboy XML format will be generated and you can use a style sheet with that, if you wish.

21 Handling Popup Message Boxes

Popup Message Boxes are small dialog boxes that are launched (usually) from scripts in web pages. They display a simple message and sometimes offer buttons that allow the user to select a simple choice such as OK or Cancel. Badboy refers to this kind of window as a "Message Box".

21.1 Recording Message Boxes

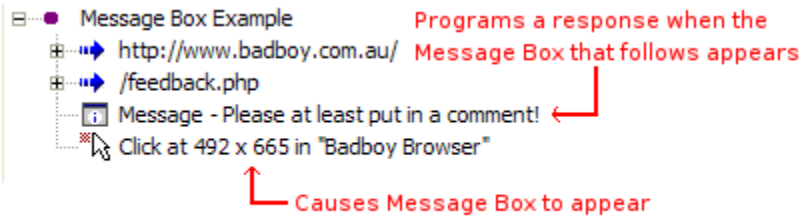
If you are recording when a Message Box shows, Badboy will record a *Message Box Item* in your Script containing the details of the message that appeared and the user response. The figure below shows how this appears in your script:



21.2 Message Box Playback

Unlike most items in Badboy Scripts, Message Box Items don't cause Badboy to do *anything* immediately when they play. You might be surprised by this and think that the Message Box Item should show the Message Box itself. The reason it does not is that the appearance of the Message Box is usually a *response* to other items in your script - for example Requests, Mouse Clicks, Form Populators, JScript, or other items. Thus you do not need Message Box items to show Message Boxes - they will happen anyway.

What Message Box Items actually do when they play is to wait in the background for a Message Box to appear and then provide the response to it for you when that happens. This means that you don't have to push "Ok" or "Cancel" - Badboy does it for you according to how you have configured the Message Box Item. The script below shows an example of how a script might be constructed to respond to a Message Box using a Message Box Item:



After a Message Box Item has played, if a Message Box appears that matches it's message text you will see a slightly different kind of Message Box to usual. The figure below shows how this looks:

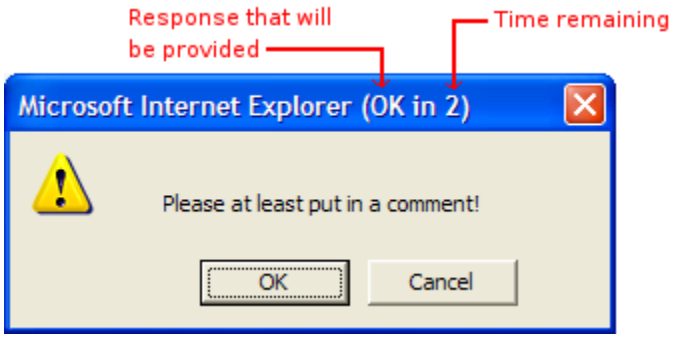


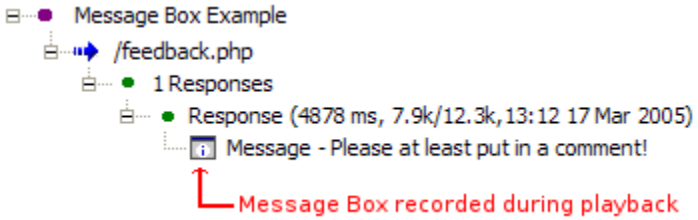
Figure 18: Message Box

The modified Message Box that appears shows in the title bar the response that Badboy is going to supply and also the time remaining before the response will be provided in seconds. This enables you to watch and if you like, provide your own response manually - or otherwise let the Message Box time out and have Badboy provide the response.

*Because Message Box Items instruct Badboy about how to respond to messages they must play **prior** to the item in your script that causes the message to appear. In some situations Badboy might record the Message Box Item after the item that makes the Message Box appear; in this case, just drag the Message Box Item up in the script so that it executes earlier until it executes before the Message Box appears.*

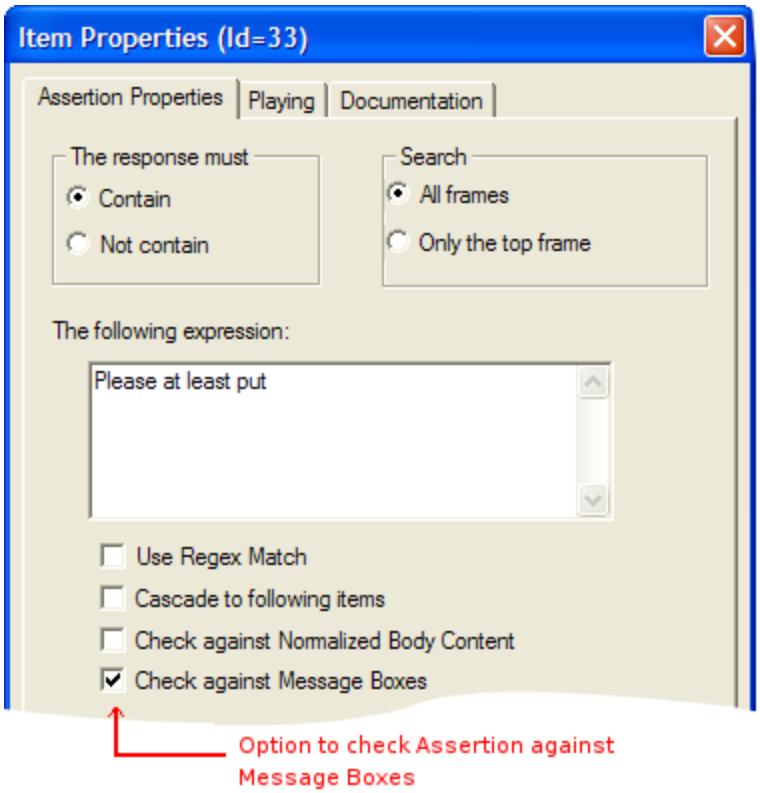
21.3 Viewing Message Boxes in Responses

Badboy also records the appearance of Message Boxes as part of Responses in your script. Thus if you want to see whether a Message Box appeared or the response that was provided you can look at the responses for the item in your script that caused the message to show. The figure below shows a Message Box Item recorded as part of a Response.



21.4 Using Assertions with Message Boxes

If you want to check that a particular Message Box appears or ensure that it *does not* appear when your script is playing, you can use Assertions to do this. To make an Assertion operate on Message Boxes, enable the "Check against Message Boxes" option in the Assertion properties.



Badboy keeps track of all the Message Boxes that appear for a page and records their messages along with the page content. Thus when an Assertion plays, it checks the messages that have appeared on the page similarly to how it checks the content on the page. If the browser moves to a new page, however, Badboy will clear the messages accumulated and begin a fresh list of messages. Consequently, Assertions that you create must play against the same page that the Message Box you want to test appears on.

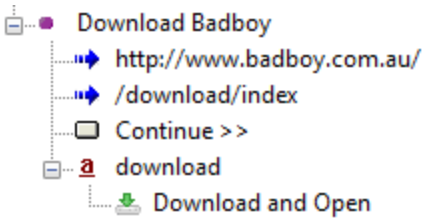
22 Handling File Downloads

File downloads occur when a user clicks on a link or button on a web site that sends back a file to be saved or opened on the user's computer. When this happens, the browser usually prompts the user as to whether they would like to Open, Save or Cancel, and if they wish to save the file, where they would like to save it. These prompts can interrupt your tests and prevent them from playing back smoothly, so Badboy provides you with tools to help in handling them.

While recording, Badboy does not create handlers for File Downloads. When a file is downloaded, Badboy just records the Navigation or Request that causes the file to be sent back as it would for any other page navigation. As a result, on playback you may find that the script stops and is waiting with a prompt open. To deal with this, a tool is provided in the Toolbox that lets you to easily add file download handling to these items so that Badboy automatically handles the prompts for you in the way you desire.

22.1 Adding File Download Handlers

You can add a Download Handler to any playable item in your script simply by dragging it from the Toolbox into your script, and dropping it so that it becomes a *child* of the item that you want it to handle downloads for. The figure below shows how it should appear in your script when correctly positioned:



Note that the Download Handler must be a *child* of the item in the script that you want it to handle downloads for. The script tree will not let you drop a Download Handler except as a child of a playable item.

If the item you drop it into does not initiate a download on playback then no error or warning will be recorded. The download handler will simply have no effect.

22.2 Configuration

A number of possible actions can be configured to occur when a download is handled by a Download Handler. The figure below shows how the dialog for configuration looks:

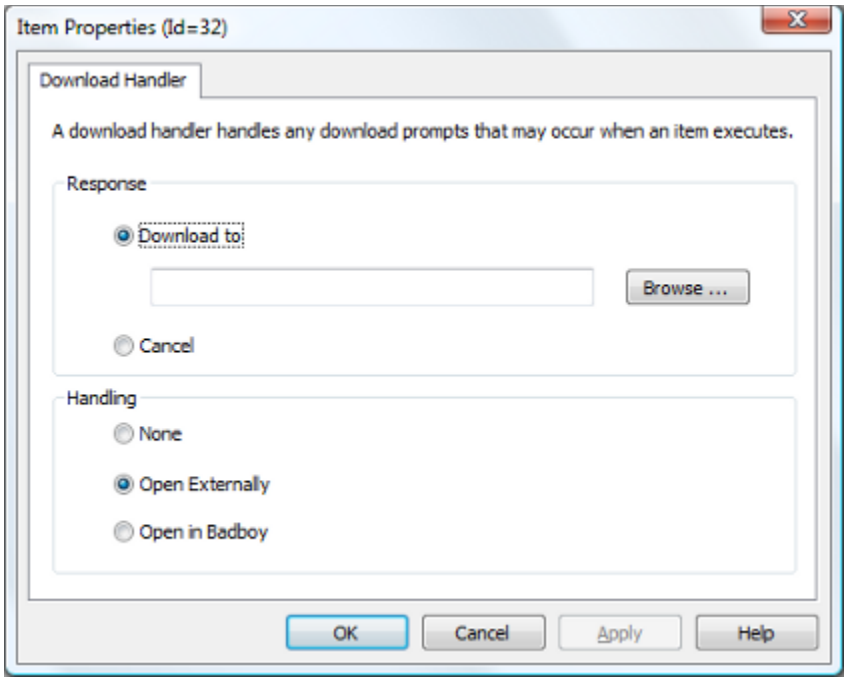


Figure 19: Download Handler

As shown in the diagram above, you can choose two kinds of response:

- **Download the file** - You can provide an explicit location to save it, or you can leave it blank and let Badboy automatically choose a location for you. By default Badboy will save it in the same directory as your script, under a file name indicated by the URL for the download. If you want, you can specify a folder here and Badboy will use that directory, but automatically decide the file name, or, you can specify the complete file name you would like and Badboy will save it there.

- **Cancel** - Ignore the download. Badboy will not transfer the content at all. You should still receive an error if the content does not exist or a server error occurs in providing it, however you will not test the full download process.

Unlike in the normal IE Save dialog, you do not have the option under Response to open the content instead of downloading it. If you want to open it, you must download it and then configure one of the "Open" options under "Handling" as described below.

If you choose to download the file, you can then choose one of the options for handling the file after it has been downloaded, as follows:

- **None** - do nothing at all. The file is saved and left in the folder.
- **Open Externally** - Open the file *outside* of Badboy using the default program configured on your computer for opening such files.
- **Open Internally** - Badboy will construct a file type URL to point to the file it has downloaded and browse the internal browser to that location. For file types that support embedded display inside the browser this will result in the files displaying inside the Badboy window itself. You should not use this option for file types that do not support embedded display.

There is no automatic error if the program fails to start or cannot open the file. You would need to configure a check (such as a Window Caption Check to verify the content after it has opened.) or other means to validate that it loaded correctly.

23 Slowing Down Playback with Timers

Sometimes for various reasons you may want your script to play more slowly than Badboy does it by default. Some situations where this is useful include:

- You want to simulate a realistic time between user actions ("think time"). This is particularly useful when using Threads to test your web site or application under load.
- Badboy plays an item in the script before the web page is ready. Badboy tries to wait automatically for the page to be ready before moving to play the next item in your script. Sometimes, however, the page may still be busy doing activity that Badboy can't detect, and you might want to make the script wait until the page is really ready.

In Badboy the way to do this is to use Timers. This section describes how to use Timers in Badboy.

23.1 Adding Timers

To add a Timer, just drag the Timer Tool from the Toolbox to the place in your script where you want Badboy to pause.

If you can't find the Toolbox tab, just press Ctrl-T to display it.

Once you drop a Timer item into your script, the Timer Properties dialog will show so that you can set the Timer's options

23.2 Waiting for Fixed Time

The simplest use of Timers is just to wait for a predetermined time to expire. To do this, just select the "Fixed" option in the properties dialog and enter the amount of time in the "Time to Pause" box in milliseconds.

23.3 Waiting for a Random Time

Sometimes it is useful to be able to wait for a random time. This is especially the case when you have multiple threads running and you would like to simulate realistic "random" pauses when browsing.

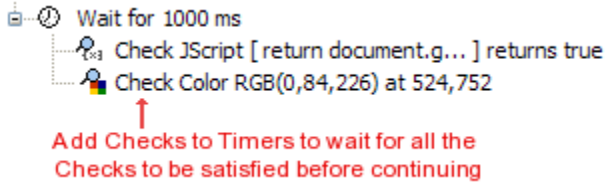
To use random times, just select the "Random" option in the properties dialog and the minimum and maximum amount of time to wait in milliseconds into the provided fields. Each time the Timer plays it will select a random time to wait within the range you specified.

You can use variable references in the time fields to make Badboy wait for a time that is the value of a variable. To do this, enter the variable in the form "${variable name}".

23.4 Using Checks with Timers

Timers can do more than just wait for a fixed time to expire. They can wait for conditions about the page to become true by using *Checks*. To add a Check to a Timer, just drag it from the Checks tab (usually, this is next to the Tools tab), and drop it onto the Timer item. The properties dialog for the Check will show so that you can set its attributes.

When a Timer has Checks added beneath it, it will wait for **all** the Checks to become true before it continues, **and** the time set in the "Time to Pause" box has expired.



If a Check that you have added never becomes true, Badboy will pause forever at the Timer item waiting for it. To avoid this you can set a Timeout on the Timer or the Step that contains it to take appropriate action if this happens.

23.5 Cascading Timers

If you want to make Badboy pause after every item in your script then adding Timers after each item will get very tiresome. To avoid this, you can select the option "Cascade delay to following items". When used in this mode, a Timer is called a *Cascading* Timer.

A Cascading Timer will pause not just when it is played itself, but also after every item that *follows it* or is *beneath* it in the script. This means that you can simply add a Cascading Timer at the start of a Step to make all the items in that Step pause for a fixed amount of time.

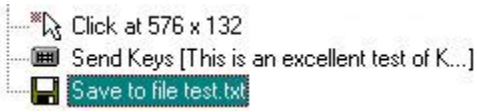
Badboy uses high performance counters built into most computers to measure time for Timers. On some platforms or hardware configurations, these timers may not be supported. If you find Timers are not waiting for the correct amount of time, or if you find that response times recorded by Badboy seem to be inaccurate by an order of magnitude or more, set the environment variable BADBOY_USE_TICKCOUNT_TIMER on your computer. This will cause Badboy to fall back to less accurate but more commonly available timers.

24 Keyboard Input

Sometimes you may want to simulate key strokes as input to one of your web pages or another window. This might be just to enter some simple text, or it might be to control the application in some way (using the control keys or arrow keys, for example.) Badboy makes it easy to do this using the "Keys" Item.

24.1 Adding a Keys Item

Adding a Keys Item is easy: just drag it from the Toolbox to the place in your Script where you would like it to go. The figure below shows how a Keys item looks in your Script:



When you add a Keys Item, the Keys property dialog will open. Here you can type the text for key strokes you would like to be sent and configure other options for how and when the key strokes should be simulated.

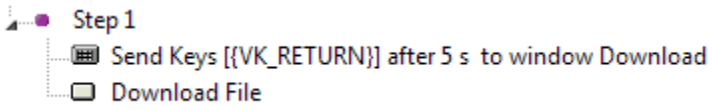
Figure 20: Keys Property

24.2 Window Focus

When the Keys item is played, Badboy will send key strokes matching the text you have entered to whichever window has the focus. In order to ensure your key strokes go to the right window, you will want to ensure somehow that the window you intend has the focus when the Keys item is played. One simple way to do this is to use a Mouse Click item which clicks in the target window prior to the Keys item executing.

24.3 Handling Modal Windows

Sometimes the window that you want to send key strokes to is a *modal* window that blocks Badboy from playing or any other window from taking focus. When these are invoked or created by actions in the Badboy script it is possible that by the time the required window has become visible the Badboy script is paused and cannot continue on to play your Keys Item. To deal with this problem you can configure your Keys Item to play *before* the item that causes the dialog to show, but to run "in the background". This means that when the Keys Item plays it doesn't send the key strokes immediately - instead, it allows the Badboy script to keep playing and then sends them later on once the required window is visible. You can set the criteria for when the key strokes will be sent in the options pane at the bottom of the Keys item configuration dialog (see above). Note: for this process to work you must place the Keys Item in the script *prior* to the item that shows the modal dialog that you want to send key strokes to. The diagram below shows how this might look:



24.4 Sending Special Characters

If you want to send non-textual keys (for example, Ctrl keys, function keys or the Enter key), you can do this using Virtual Key Codes. Virtual Key Codes are special names for keys which describe the keys. You can include a Virtual Key Code in your Keys item by entering the Virtual Key Code surrounded by curly braces. For example, to send the 'HOME' key on your keyboard you could enter '{VK_HOME}'. You can embed this in text, if you like, for example: 'Tree frogs {VK_HOME} are green'.

The table at the end of this section shows the virtual keys supported by Badboy.

24.5 Key Combinations

Sometimes you may want to send a combination of keys all pressed together. For example, you might want to send the "Alt" key plus the "F4" key to simulate closing an application. For such a case you need to send the F4 key in-between the "down" and the "up" action for the "Alt" key. You can do this by adding the modifiers ":UP" and ":DOWN" to the virtual keys in the table above.

For, example, to send "Alt-F4", you could use:

```
{VK_MENU:DOWN}{VK_F4}{VK_MENU:UP}
```

This would cause whichever window had the focus to attempt to close. *Note that the Alt key corresponds to the VK_MENU virtual keycode.*

24.6 Virtual Key Table

Virtual Key Codes		
VK_ACCEPT	VK_F19	VK_NUMPAD9
VK_ADD	VK_F20	VK_OEM_1
VK_APPS	VK_F21	VK_OEM_2
VK_ATTN	VK_F22	VK_OEM_3
VK_BACK	VK_F23	VK_OEM_4
VK_BROWSER_BACK	VK_F24	VK_OEM_5
VK_BROWSER_FAVORITES	VK_HANGUEL	VK_OEM_6
VK_BROWSER_FORWARD	VK_HANGUL	VK_OEM_7
VK_BROWSER_HOME	VK_HELP	VK_OEM_8
VK_BROWSER_REFRESH	VK_HOME	VK_OEM_102
VK_BROWSER_SEARCH	VK_KANA	VK_OEM_CLEAR
VK_BROWSER_STOP	VK_LAUNCH_APP1	VK_OEM_COMMA
VK_CANCEL	VK_LAUNCH_APP2	VK_OEM_MINUS
VK_CAPITAL	VK_LAUNCH_MAIL	VK_OEM_PERIOD
VK_CLEAR	VK_LAUNCH_MEDIA_SELECT	VK_OEM_PLUS
VK_CONTROL	VK_LBUTTON	VK_PA1
VK_CRSEL	VK_LCONTROL	VK_PACKET
VK_DECIMAL	VK_LEFT	VK_PAUSE
VK_DELETE	VK_LMENU	VK_PLAY
VK_DIVIDE	VK_LSHIFT	VK_PRIOR
VK_END	VK_LWIN	VK_PROCESSKEY
VK_EREOF	VK_MBUTTON	VK_RBUTTON
VK_EXSEL	VK_MEDIA_NEXT_TRACK	VK_RCONTROL
VK_F1	VK_MEDIA_PLAY_PAUSE	VK_RETURN

Virtual Key Codes		
VK_F2	VK_MEDIA_PREV_TRACK	VK_RMENU
VK_F3	VK_MEDIA_STOP	VK_RSHIFT
VK_F4	VK_MENU	VK_RWIN
VK_F5	VK_MODECHANGE	VK_SCROLL
VK_F6	VK_MULTIPLY	VK_SEPARATOR
VK_F7	VK_NEXT	VK_SHIFT
VK_F8	VK_NONAME	VK_SLEEP
VK_F9	VK_NUMLOCK	VK_SPACE
VK_F10	VK_NUMPAD0	VK_SUBTRACT
VK_F11	VK_NUMPAD1	VK_TAB
VK_F12	VK_NUMPAD2	VK_UP
VK_F13	VK_NUMPAD3	VK_VOLUME_DOWN
VK_F14	VK_NUMPAD4	VK_VOLUME_MUTE
VK_F15	VK_NUMPAD5	VK_VOLUME_UP
VK_F16	VK_NUMPAD6	VK_XBUTTON1
VK_F17	VK_NUMPAD7	VK_XBUTTON2
VK_F18	VK_NUMPAD8	VK_ZOOM

25 Spidering

Exhaustively testing every function and feature on a web page can be hard and tedious work. While it's easy to try the obvious things, testing *every* link and button on a page can involve hundreds of clicks - and even then, how do you know you didn't miss one? If the links or buttons on your page change frequently then you are even worse off - you might record a script but you would quickly find your script was testing the wrong links!

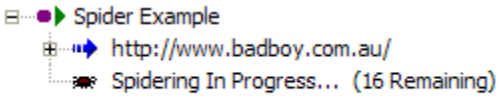
Spidering is a feature to help you efficiently try *every* navigation that is possible on a page with just the click of a button. Using spidering you can quickly find any broken links or other actions that cause problems - without ever having to record them!

25.1 How Spidering Works

The term "Spidering" comes from the idea of a spider crawling across the links in its web. In Badboy, you initiate Spidering by dropping a Spider item from the Toolbox into your Script. When the Spider item

plays, it first surveys all the links and buttons it can find on the page and makes a list of them. From then on, each time the Spider plays it executes *one* item from its list.

The figure below shows how a typical Spider item might appear in your Script:



25.2 Spider Looping

Since a Spider item only executes *one* item from your page each time it plays, in order to execute all the links on a page it needs to execute in a loop. You accomplish this by placing the Spider item in a Step. You don't need to configure looping for the Step - the Spider item will *automatically* loop for you after each spidered link or button is navigated.

If you like you can disable automatic looping and control it yourself by adjusting the Spider's properties.

Each time the Spider plays, it follows the following process:

- First, it executes the next of the navigations on your page from its list
- After the navigation has executed, if there are more navigations to try, it will loop back to the first item in its parent Step.
- If the Spider has exhausted all the links and buttons on the page it will exit and Badboy will continue playing the items after the Spider in your script.

In this way the Spider item will loop inside it's Step until it has browsed all the links on a page.

25.3 Navigation Options

Sometimes you may not always want to execute every navigable element on your page. Badboy offers you the ability to control what content is Spidered in several ways:

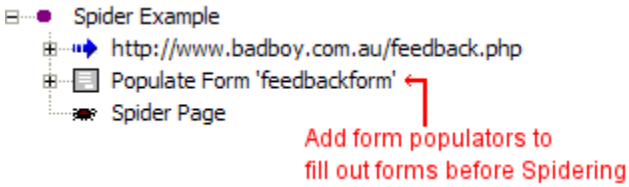
- You can choose what to include: Links, Buttons or both
- Mode - This is an advanced option that lets you choose whether to browse links by *name* (Navigation Mode) or to browse them by their URLs (Request Mode). For highly dynamic links browsing as Navigations is usually better, but for pages where many links have the same name, Request mode may be necessary.
- Which browser frames to include - you can choose a specific frame or you can just let Badboy browse all of them
- Specific exclusions - you can provide a list of regular expressions which will filter out content that should not be browsed. For example, if you don't want Badboy to browse the "logout" link then you might enter "logout" here.

25.4 Setting Assertions

You may like to configure Assertions for your Spider item so that you can check as each page is Spidered that your web site is working correctly. To add Assertions, place them before the Spider item in the script and configure them as *cascading* Assertions. By making them cascading Assertions you ensure that they will execute each time an item is Spidered. The figure below shows how an example of how you might configure a cascading Assertion for a Spider item in your script.

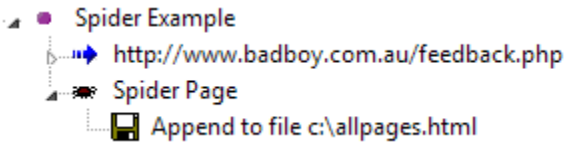
25.5 Populating Forms

If your pages contain Forms then you may wish to ensure that they get populated with data so that Badboy can spider them correctly. To do this, just add Form Populators for any forms on the page prior to your Spider item. The figure below shows how this would appear:



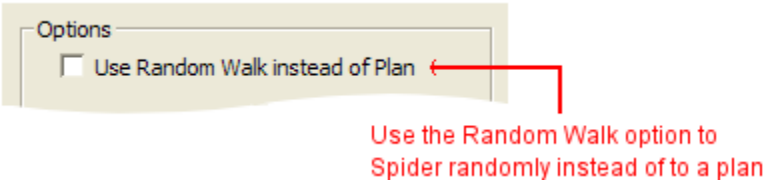
25.6 Performing Actions on Spidered Pages

Badboy will automatically check for errors on the pages visited by the spider, and you can also add cascading Assertions to check conditions as described above. Sometimes, however, you might want to do more actions on each spidered page: execute JavaScript, conditional logic, populate variables or other actions. To do this you can add children to the Spider Item itself: the spider will execute its child items for every page that it visits. The figure below shows how this looks in your script:



25.7 Random Walking

By default the Spider Item creates a plan that it uses to make sure it tries every link on your page when it executes. This makes sure that the each navigation is covered in an orderly manner. If you like, however, you can tell the Spider item to just pick a navigation (link, or button) to perform at random. In this case no plan is created and each time the Spider just randomly picks a link or button on the current page to navigate. Note that if you use the "loop automatically" option in combination with Random Walking, Badboy may loop forever because in Random Walk mode Badboy does not remember which links it has already traveled and may execute the same links over and over again. It will only finish in that case when it comes to a page where it cannot find any navigation to perform!



25.8 Controlling Looping Yourself

In some situations you may prefer not to have Badboy jump to the previous step immediately after executing a Spider. For example, you might like to perform some other operations - Mouse Clicks, Variable Setters etc. In this case, just uncheck the "Loop Automatically" box in the Spider Item properties. This will cause the Spider to execute and then carry on with subsequent items in your script without looping. You can control looping yourself by setting a loop on the Step or using other features that change the script flow (for example, using Badboy's OLE seek() method).

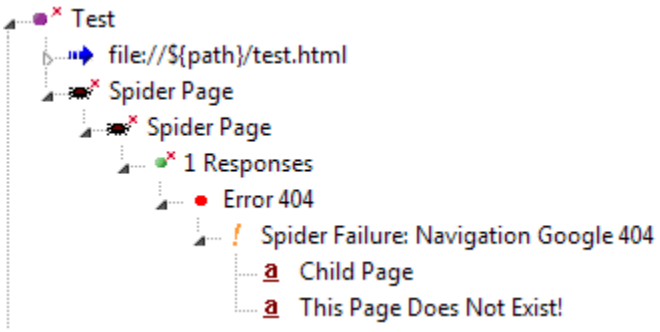
25.9 Detecting Errors

If there are errors during Spidering (for example, due to broken links on your page), usually you will want to locate them and fix them. Badboy records the errors by adding them as red Responses under the Spider item. If you expand such responses then you can see detailed information about the actual navigation that caused the problem. You will also see an item created for you by Badboy which reproduces the problem. To reproduce the failure you can play this item manually, or copy or move it elsewhere to play it as part of your Script execution. The figure below shows how errors are captured during spidering:

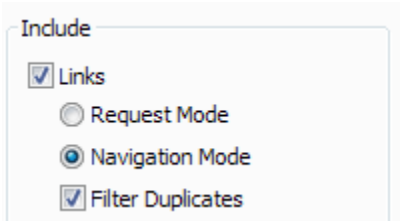
25.10 Recursive Spidering

A Spider in its default configuration will test all the links and buttons on a single page. However it will not descend into the links and buttons on the child pages that are visited to test those. When a spider descends more than one level into a website it is known as "recursive spidering".

Badboy does not support infinite depth recursion, however if you want to spider multiple levels you can do that by adding a Spider Item as a child of another Spider Item. In this configuration Badboy will Spider each link on the top level page and then for each visited page it will spider each link or button found. When errors are found, instead of a single error being reported, Badboy will record all the navigations or requests that preceded the problem from the starting page (so, for example, if you have 2 levels of spiders, you will see 2 navigations recorded when a problem occurs). The diagram below shows how a 2 level recursive spider with some errors appears in a script:



It is important to understand that when a Spider is recursive it may end up back on a page it has previously visited. For example, if you start at the Home Page and then every child page from the Home Page has a link *back* to the home page then a recursive Spider may well find itself back on the Home Page again, and if it descends further it will end up re-testing the whole Home Page. This may not be a problem, but it will waste a lot of time. To help avoid this, you can configure Spider Items to keep track of which links have been visited and not revisit ones that have already been navigated. To do this, check the box in the Spider Item labeled "Filter Duplicates", as shown below:



The "Filter Duplicates" option will stop a Spider from revisiting the same URL multiple times. It will not stop the spider from navigating a link or button with the same name multiple times, if that link actually points to a different URL. When a Spider is a child of another Spider in the script it will check its parent Spiders and filter duplicate links that its parents have visited as well as links it has already visited itself.

Although the "Filter Duplicates" option will try and stop your Spider from going in circles, it is not perfect - it cannot prevent the Spider from looping in cases where the URL to be navigated to is not clear from the element on the page, for example if a link is responded to dynamically using JavaScript, or is redirected or if the loop occurs as a result of a button rather than a link. Hence you may still find it beneficial to add some Exclusions to prevent this happening for cases that you discover.

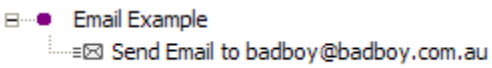
26 Sending Email

If you would like to be notified about the results of your tests, or if you would like to be able to send emails based on information created or extracted from your website, Badboy makes this easy with the Send Email item.

Sending Emails is restricted in the free version of Badboy to emails of less than 50 characters. In order to send larger emails you need to purchase a license and enter your registration key using the "Enter License Key" option under Badboy's Help menu.

26.1 Creating Send Email Items

To create an item to send email in your script, just open the Toolbox and drag a "Send Email" item into your script at the point where you would like the email to be sent. The figure below shows how this looks:



26.2 Setting the Email Content

To set the content of the mail as well as the subject and recipients, open the Email Item properties by double clicking on it in your script. The figure below shows how this looks:

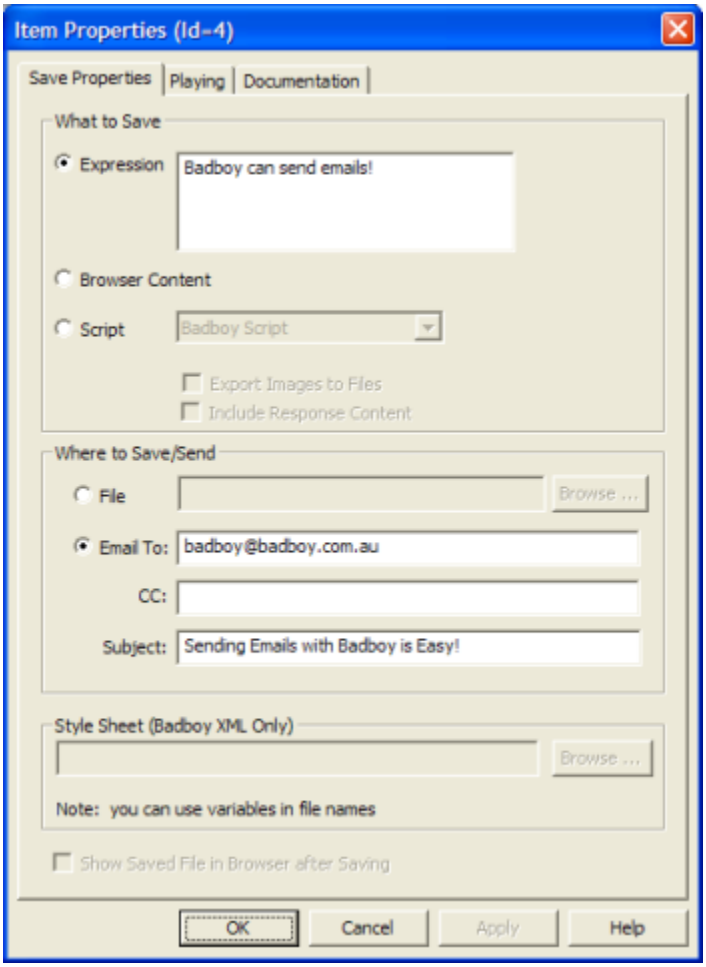


Figure 21: Email Item Properties

The Send Email item is actually the same as the Save item, but it is configured to send content as an email instead of saving it to a file. You can change any Save Item to a Send Item at any time just by changing the option from "File" to "Email" in the properties dialog.

Note that you can choose several different kinds of content:

- To send a plain text message, select the "Expression" option in the properties and type your email into the field next to it.
- To send the current page showing in the browser page by email, select the "Browser Content" option.
- To send an HTML report of how your script played, select the "Script" option and then choose "HTML Report" in the drop down menu next to it. Badboy will send a beautifully formatted HTML summary of your script including all failures, assertions and screen shots as well!

26.3 Configuring your Email Settings

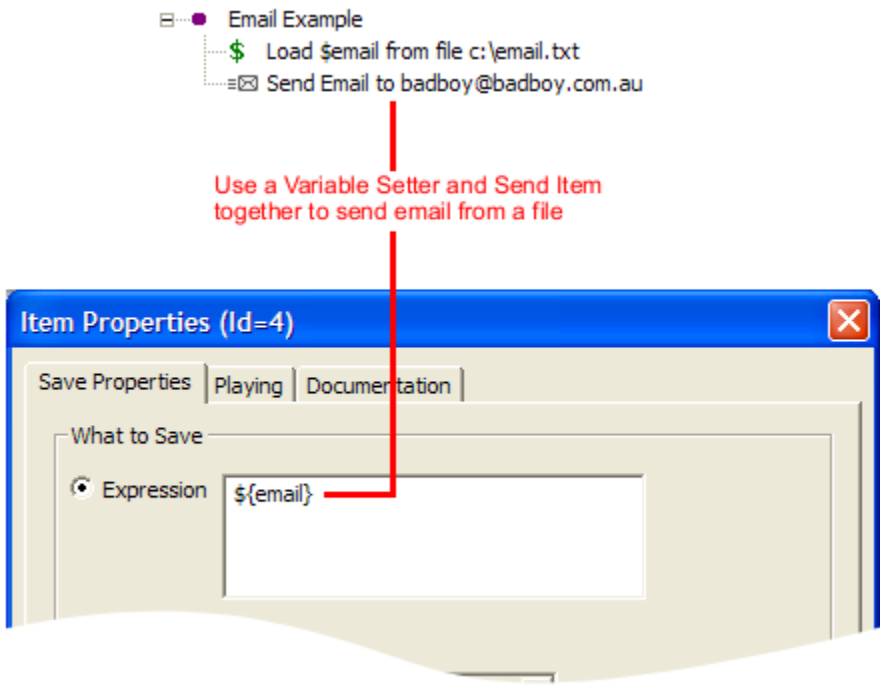
In order to send emails from your computer, Badboy must be configured with an SMTP server that it can use. To do this, open the preferences from the Preferences menu and choose the "Programs" tab, and then enter the details for your SMTP server. You should also enter a display name (this is the name that the recipient of the Email will see the email as being "From"), and a "From" address. Please note that if “From” address is not a real address belonging to the same domain as your network or your computer then your mail server may refuse to send the email.

If you do not know the address of your SMTP Server, ask your ISP or Network Administrator and they should be able to tell you.

26.4 Sending an Email from a File

A common need is to send an email that has its content loaded from a file. This is easy to do by using a Variable Setters. First, add a Variable Setter and choose the "Load from File" option so that it loads it's content from a file. Then, add your "Send Email" item and choose the Expression option and set it to be the content of the variable.

The figure below shows how this might look:



27 Using JavaScript/JScript

27.1 Using JScript

Badboy can execute JavaScript (or JScript) against the visible Browser window as part of your tests. This can be a very useful technique in some cases where you need to do advanced operations or where other mechanisms for playing back aren't sufficient to automate your web site correctly.

27.2 Adding JScript Items to your Script

You can add JScript to your tests by right-clicking in the Script tree on a Step or Request and selecting Add-> JScript or Insert->JScript.

You can also add JScript items to your script by dragging them from the Toolbox (press Ctrl-T to show the Toolbox) and dropping them into your script at the point where you would like the JavaScript to be executed

27.3 Editing JScript Properties

When adding a JScript item or if you edit it's properties, you will be shown a dialog where you can enter the JScript for Badboy to run when it plays the given item:

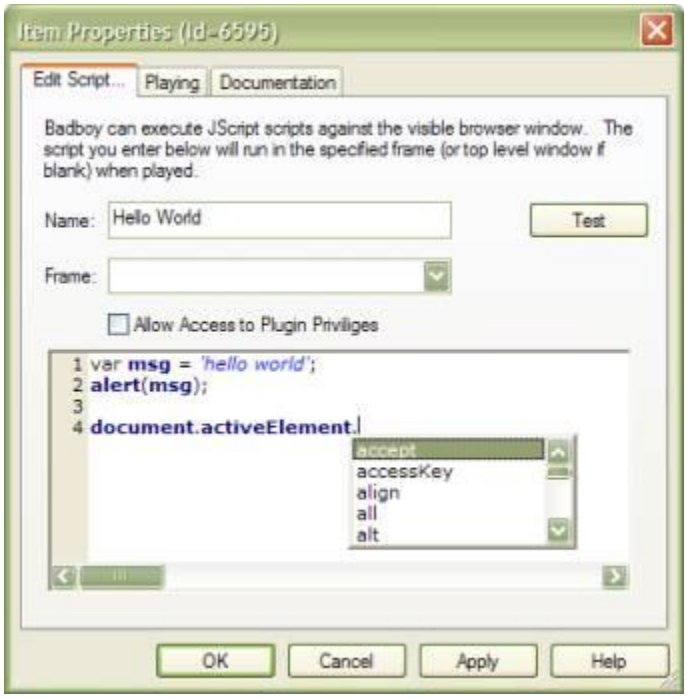


Figure 22: JScript Edit Properties

Badboy supports auto-completion for properties and functions in the Browser window. Press Ctrl-Space to see the available properties displayed when you are creating your JavaScript.

You can also choose the frame that the JavaScript should run in. If your web site has only one frame or you want the script to run in the top frame then you can just leave this box empty. If, however, there are child frames in the page then it can be convenient and occasionally essential to comply with browser security constraints to run the script in the context of the correct frame.

27.4 Plugin Privileges

Badboy offers additional features and functions to Badboy Plugins. However you don't need to write a plugin to access these special functions: you can make it possible by checking the box marked "Allow Access to Plugin Privileges". If you do this then your script can access the plugin API by creating a "plugin" object as follows:

Example 1 - Read Badboy Preferences:

```
  var plugin = badboy.plugin();
  alert("Badboy is configured to use " + plugin.getPreference("Undo Levels")
+ " undo levels");
```

It is important to be mindful of security: Badboy's plugin API can be used to modify your script and could even be manipulated to launch malicious programs on your computer. Therefore you should make sure you do not set references to the "plugin" object outside of your script in case they might be accessed by web pages you do not trust.

28 Advanced JScript

JScript can be used to perform advanced operations and thereby extend the functionality of Badboy. Here is an example JScript that will let you save a file to your local hard disk when it executes:

Example 2 - Write a Text File:

```
  var ForWriting = 2;
  var fso = new ActiveXObject("Scripting.FileSystemObject");
  var textFile = fso.OpenTextFile("c:\\test.txt", ForWriting, true);
  textFile.Write ("This is a test.");
  textFile.Close();
```

This script has a difficulty: in order for it to work you must set your Internet Explorer security level to Low for the site that you are accessing so that ActiveX controls can run and access the file system. Otherwise you will get security warnings shown, or it might be simply refused by the browser. To get around this you can use the Badboy Plugin API instead to create the ActiveX object, as is shown in Example 3, 4, 5 and 6 below.

*For all these examples to work you **must** check the box marked "Allow access to plugin privileges" in your JScript item.*

Example 3 - Write a Text File without Modifying Security Settings:

```
  var ForWriting = 2;
  var fso = badboy.plugin.createObject("Scripting.FileSystemObject");
  var textFile = fso.OpenTextFile("c:\\test.txt", ForWriting, true);
  textFile.Write ("This is a test.");
  textFile.Close();
```

Example 4 - Terminate a Process:

```
    var service =
badboy.plugin.createObject("WbemScripting.SWbemLocator").ConnectServer(".");
    var procs = service.ExecQuery("Select * from Win32_Process Where Name =
'notepad.exe'")
    var e = new Enumerator(procs);
    for (;!e.atEnd();e.moveNext ()) {
      var proc = e.item();
      proc.Terminate();
    }
```

Example 5 - Run a Batch Script:

```
    var shell = badboy.plugin.createObject("WScript.Shell");
    shell.Run("c:\\test.bat")
```

Example 6 - Reading and Writing to a Database:

```
    var con = badboy.plugin.createObject("ADODB.Connection");

con.open("ODBC;DATABASE=testdb;DSN=MyTestDB;OPTION=0;PWD=mypassword;PORT=0;SE
RVER=localhost;UID=testdbuserId");

    // Insert some data
    con.execute("insert into test_table values (1, 'test' 3)");

    // Query some data
    var rs = badboy.plugin.createObject("ADODB.Recordset")
    rs.open("select count(*) from test_table", con);
    alert(rs.fields(0));
    con.close();
```

If you are a programmer, you can even write automation objects of your own using your favorite language and access them in the same way.

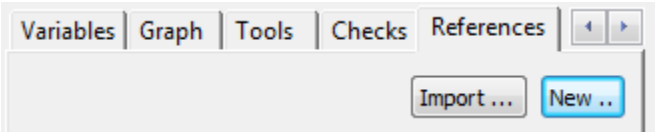
For more information on controlling Badboy with JScript, see Automating Badboy with OLE. You can call any of these OLE function from JScript itself and thereby control Badboy itself as part of your scripts.

29 Using References

Often, functional tests are written closely in conjunction with external documents. For example, it is common to build tests to match corresponding items in a requirements specification, or to build tests based on entries in an issue or bug tracking system. Badboy calls such external entities *References* and offers features to let you define, track and view the associations between tests and external references within Badboy. This allows, you to know, for example, when a test fails, which requirements, bugs or features are impacted by the failing test. Creating or importing references first can also be a very practical way to create tests, as you can then use them to drive which tests you create and ensure complete coverage by checking the mapping of references to tests is complete.

29.1 Creating and Deleting References

There are two ways to create References within Badboy: manually or by importing them from another source. Manual creation is performed from the References View which is usually found as a tab in the bottom left corner of the Badboy frame window. You can click the 'New' button to create a new reference, as shown below:



When the button is clicked, Badboy shows a window to let you enter details about the reference:

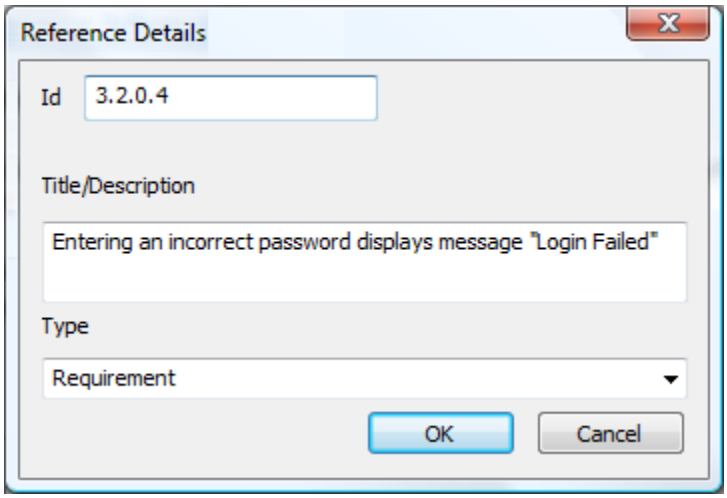


Figure 23: Reference Details

There are three fields you can specify:

- **Id** is a unique identifier for the reference. Usually this will correspond to an identifier from an external system such as a Job Number, Bug ID or similar. However Badboy does not enforce any particular format or constraint on these Ids, so you can use whatever you like here. Making it unique will assist in certain situations, so it is useful, but not mandatory to ensure that no two entries have the same Id.
- **Title / Description** is a textual description of the reference. This can be any text you like and once again, Badboy does not enforce any constraint.

- **Type** is a category for the reference. This allows you to group similar categories of reference such as all bugs or all requirements. Although Badboy suggest the values 'Requirement' or 'Defect', you can actually enter any category you like (e.g.: BUG).

References can be deleted simply by selecting them in the References View and then pressing the 'Del' key. This removes the reference and un-maps it from any Tests that are mapped to that reference.

29.2 Importing References

Adding references manually to your script can be very tedious and time consuming. To avoid this work Badboy supports features to let you import references from external systems. To support interacting with external systems for managing references, Badboy uses the concept of a 'Reference Provider'. Two reference providers are provided in the stock Badboy install: Microsoft Word (tm) Reference Provider and an Open Office Reference Provider. These two providers both allow you to import headings from word processor documents. So, for example, if you have a .doc file that has all your requirements and they are numbered using Word Headings, Badboy will import them all using the heading numbers from the Word document.

If you want, you can easily write your own reference importer. References can be added using some simple JavaScript, allowing you to import references from any source into your script by writing a JavaScript routine to read them from the source and add them to Badboy. Here is an example snippet that shows how to add a reference to Badboy using JavaScript:

```
badboy.script.References.add("12345", "BUG", "The web site does not work");
```

29.3 Mapping References to Tests

To benefit from defining or importing References in Badboy you must *map* them to tests. Any number of References may be mapped to a test and any given Reference may be mapped to any number of tests. Mapping can be performed in two different ways:

- Select any number of references in the References View and then click with the mouse and drag them into the script tree and *drop* them on the Test that you would like them to be mapped to.
- You can also map References to tests by opening the Test properties dialog and selecting references in the drop down box and clicking 'Add'.

Once a reference has been mapped to at least one Test, a tick appears next to it in the Reference View:

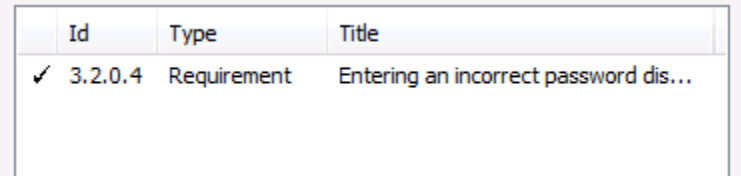


Figure 24: Reference View

29.4 Viewing Reference Information

Once you have mapped one or more References to a Test you will see the Reference information displayed in the Summary display for the test:

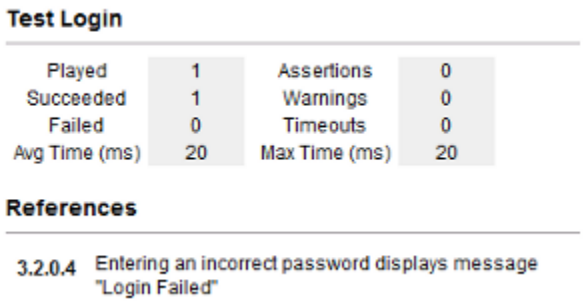


Figure 25: Reference Summary Display

30 Using Badboy with JMeter

While Badboy offers some elementary load testing features, to do real stress testing you often need much more. Rather than trying to offer all of the features of a complete load testing product, Badboy integrates with a free, open source product called JMeter. JMeter is run by the Apache Software Foundation and is free and open source.

Badboy integrates with JMeter by letting you save your scripts in the JMeter file format so that you can open them and run them in JMeter. This support means that you can use Badboy to record your scripts to do functional testing, and then save the same script as a JMeter file to do performance testing using the full power of JMeter!

Exporting a script in JMeter format is very simple. Simply record your script and navigate to File->Export to JMeter

30.1 Limitations

There are, unfortunately some limitations to Badboy's JMeter support. These limitations exist because the features of JMeter and Badboy do not exactly match each other and in some cases it is not possible to export items directly.

The following limitations should be noted:

- **JScript cannot be exported**. JMeter cannot run JScript because it does not have an embedded browser window. JScript items will be ignored in the script when you do your export.
- **Data sources are not exported**. JMeter has its own concept to replace Data Sources. When you export your file, all of your variables will be created in JMeter as "User Parameters". You can, if you wish, provide a file for JMeter to read variable values from. Please read the JMeter help on the "HTTP User Parameter Modifier" to see how to do this.
- **Increments are not exported.** JMeter does not use increments to modify variables. Rather, variables are automatically modified by the iteration of a loop and the presence of a User Parameter Modifier.

- **Navigations are not exported.** Navigations also rely on having a browser window present and thus cannot be directly supported by JMeter. You can simulate them if you wish by using JMeter features.
- **Assertions are exported but may need adjustment**. Badboy will attempt to export Assertions to JMeter but due to small differences in how JMeter and Badboy handle Assertions it is possible that they will not always work the same way. Badboy automatically scans for these problems and will alert you with a message after export containing details about what may need to be adjusted.

Important Note

Badboy's integration with JMeter is entirely supported by Badboy Software. If you have issues with JMeter support, please find help from Badboy's support channels (for example, the Badboy Forum). Please do not bother the JMeter mailing lists with Badboy questions!

31 Using the Command Line Runner

While Badboy's GUI interface is very powerful and simple to use, sometimes advanced users may wish to run scripts without showing a window at all. There are some common scenarios where this can be very useful:

- You want to run Badboy tests as part of a script or an automatic schedule. Using the command line runner, you can create a batch or shell script to run Badboy at a scheduled time each day to monitor the performance or stability of your web site.
- You want to create a large load on a server. Running Badboy's GUI inevitably takes some resources and can slow down your test. If you run the script via the command line you may be able to simulate more users than otherwise.

31.1 Limitations of the Badboy Web Test Engine

Because Badboy uses its own test engine to run your scripts in command line mode you may experience some different behavior in this mode to that of a normal browser. The most noticeable issue that most users will experience is that Badboy's Web Test Engine cannot run JScript or VBScript. However even if your site uses these languages there is still a good chance that scripts you create will work with Badboy's command line runner anyway. This is because most sites only use client side scripting for user interface effects, which do not affect the command line runner. There are cases, however, where the operational functioning of a site may require the use of JScript and thus these operations may fail when you run them in the command line runner even though they succeed in Badboy's GUI. If your scripts must run in a GUI to execute, you may like to try automating Badboy through OLE instead of using the command line runner (see Automating Badboy with OLE).

31.2 Running a Command Line Script

Running a command line script is easy. The most important thing that you will need to do is to make sure that the Badboy install directory is in your system's PATH variable. Alternatively you can run the scripts from the Badboy install directory. A command to run a script will usually take the form:

Example:

```
  bbcmd -i 1 test.bb
  Badboy Command Line Runner Version 1.3
  Copyright (C) Badboy Software 2001-2003. All rights reserved.

  Loading archive...Loaded.

  Launching 1 threads...
  Url=http://www.google.com/:status=200:704 bytes
  Url=http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-
8&q=test&btnG=Google+Search&bbrand=18467:status=200:883 bytes
  All threads finished

  ============== Summary Statistics ======================

  Url                        Response Time         Responses       Errors
  ------------------------------------------------------------------------
-------
  www.google.com/ (5)          243                  1               0
  www.google.com/search (8)    460                  1               0
```

31.3 Command Line Options

The command line runner supports many command line options to allow you to customize how the script is played. To view these options, just type "bbcmd" without any options.

Example

```
    Badboy Command Line Runner Version 1.3
    Copyright (C) Badboy Software 2001-2003. All rights reserved.

    Usage: bbcmd.exe [options]

    Options are:

     -n <number of threads>
     -s <stagger time between threads>
     -d <duration to run for in seconds (all threads terminated after this
time)>
     -i <iterations to perform (per thread)>
     -D <name>=<value> : defines or sets a variable with this name and value
     -c : causes report to collate statistics for identical Urls together
     -v : causes verbose logging of responses (extremely verbose)
```

31.4 HTTP Authentication and Proxy Authentication

Badboy has elementary support for Basic Authentication in bbcmd. You can enable this by setting environment variables as described in the table below:

Environment Variable	Effect
BADBOY_AUTH_USER	User name supplied for HTTP Basic Authentication
BADBOY_AUTH_PASSWORD	Password supplied for HTTP Basic Authentication
BADBOY_PROXY_AUTH_USER	User name supplied for Proxy Authentication
BADBOY_PROXY_AUTH_PASSWORD	Password supplied for Proxy Authentication

32 Aggregating Scripts

If you have a large number of tests, or conversely, if you have tests that are very large then you can take advantage of Badboy's ability to call one Badboy Script from another Script. This feature allows you to break your tests into individual Scripts and then write a Master Script that calls all your other Scripts. This is called "Script Aggregation".

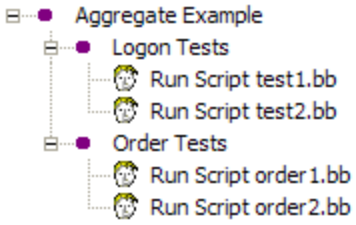
32.1 Creating an Aggregate Script

To create a script that Aggregates another script:

- Create a new script and drag an "Aggregate Script" item from your Toolbox into your script at the point where you want the aggregated script to run.
- A properties dialog should show, allowing you to choose the Script you would like to run in a text box that is provided.

When played, an Aggregate item launches a separate instance of Badboy and plays the aggregated Badboy Script all the way through, automatically exiting the created Badboy instance when it is complete. Note that the Aggregate item will wait for the called script to finish before continuing. When the aggregated Script is finished the Aggregate item will load the summary statistics from the called Script as its own so that you can see the statistics for all played scripts from your Master Script. As with other playable items, you can see statistics for any group of Aggregate items by selecting the Step that contains them.

The figure below shows how an Aggregate Script appears:

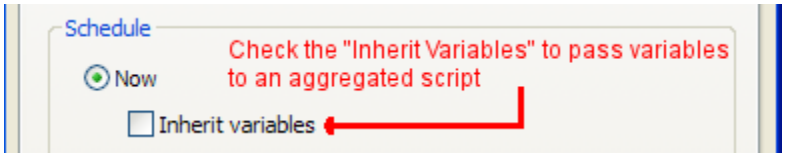


Aggregate items are just a special kind of Schedule Item where the scheduled script is configured to run immediately. Hence you can turn an Aggregate item into a Schedule item and vice versa just by changing the Schedule option between "Now" and "Later" You can read more about Scheduling in help section Scheduling Badboy Scripts.

32.2 Passing Variables to the Aggregated Script

Often you will want to control variables from your master script and have any aggregated scripts inherit those values. For example, if you have created a "host" variable to control which website you are testing, you would naturally want aggregated scripts to also use that same value.

To make variables in your master script get passed to a called script, check the "Inherit Variables" option in the Aggregate Item's properties.



33 Automatic Variables

Automatic Variables are variables that are available in all your scripts without you having to add them manually. These variables are useful in a number of circumstances:

- Sometimes there is a common variable that you add to all your scripts. You can avoid having to do this by instead making an Automatic Variable.
- There might be a variable that you would like to have in your script but you don't want it to be saved with the script. For example, you might not want to save passwords into your script since then if you send the script to someone else they will be able to see it. You can avoid this by making the password into an Automatic Variable.
- Automatic Variables can also be very useful for handling dynamic content on your site when you use the ability of Variables to update their values from page content. If you have a particular value that you would like to hold in a variable no matter where or when it appears on your web site you can make an Automatic Variable for it and then the value will be available to all your scripts without having to add anything to the scripts themselves.

33.1 Adding Automatic Variables

Automatic Variables are configured in the Badboy Preferences dialog. To add an Automatic Variable, go to the Preferences menu and select "Preferences" and then click on the "Variables" tab. The figure below shows how the Automatic Variables tab looks:

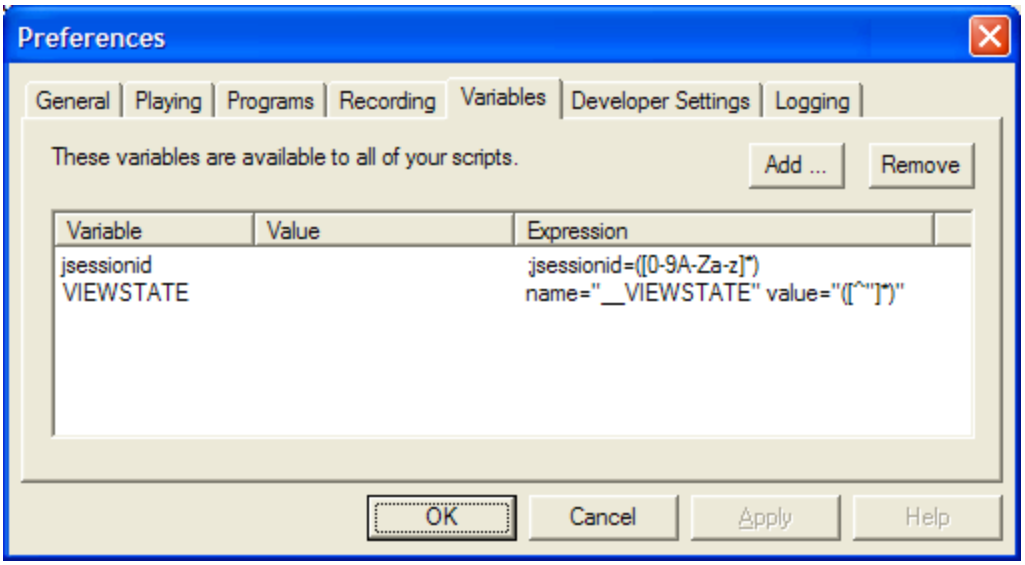


Figure 26: Automatic Variables Tab

33.2 Changes to Automatic Variables

When you play your script you can make changes to Automatic Variables just like you can to ordinary variables. In fact, Automatic Variables work just like ordinary variables in nearly all respects. One difference, however, is important: values of Automatic Variables are not saved with your script. Instead they are saved as part of the Badboy Preferences. This means that although your script can modify the values of automatic variables, other scripts will not see the modifications. If you want to make a permanent change to the value of an automatic variable then you have to do it by changing the variable in the Preferences.

If you are familiar with the concept of environment variables, you can think of Automatic Variables as being a bit like environment variables for Badboy. Just like environment variables, scripts get the values of the Automatic Variables when they are opened. After that they can be modified but the changes are only visible to the Badboy process that is running the script.

33.3 Predefined Automatic Variables

Badboy comes with some Automatic Variables built in when you install it. These are common variables that are helpful to applications built on web frameworks such as Java Servlets and ASP.NET. They help by capturing session id's and form states to ensure scripts play back properly, even when the server uses different unique values for each web session. If you don't need these variables you can delete them, however they shouldn't cause you any trouble and can help make testing applications built on these common frameworks much easier.

34 Custom Toolbox Items

As you begin to build bigger and more complex scripts you will notice that you use similar sequences of Script Items over and over again. You may find, for example, that very many of your tests need to execute a common sequence of actions to "Login" before running tests. You would find that you had those same actions recorded all throughout your tests, not only would you get sick of recording the same sequence over and over again, but your scripts would get bloated and overcomplicated by all the unnecessary items. Even worse, if your web site changed you might have to go through all your tests to update every place where you had recorded these common items.

Badboy provides a solution to these problems in the form of *Custom Items*. Custom Items work the same way as the other items in your Toolbox do, except that instead of Badboy providing them, you create them yourself. A Custom Item can contain many regular script items grouped together. Custom Items can even contain other Custom Items, letting you build your tests from a hierarchy of reused components.

34.1 Creating a Custom Toolbox Item

Creating a Custom Toolbox Item is very easy - simply drag an item from your script into your Toolbox. Badboy will prompt you for a name for your new Custom Item.

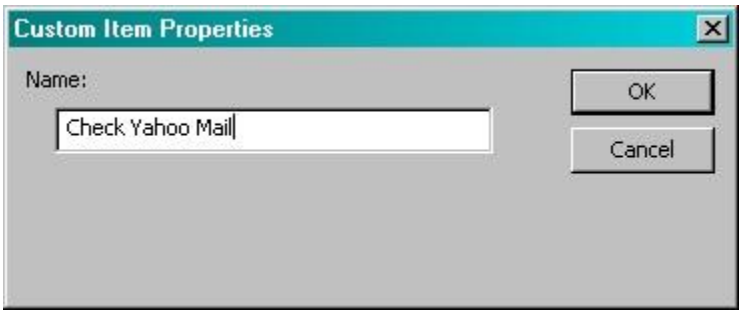


Figure 27: Custom Toolbox Item

Once you have created a Custom Toolbox Item it will appear in your Toolbox with the name that you gave it.

34.2 Using Custom Toolbox Items

Once you have created a Custom Toolbox Item, using it is as easy as using any other Toolbox Item - just drag it from your Toolbox into your script at the location you want it to go. When you drop the item into your script, Badboy will prompt you for a name for it. You can just use the default name if you like, or you can give it a new name of your own. The name you give it is just for display purposes and won't affect its operation.

Badboy also lets you edit the name of the Toolbox Item that the script item is linked to. By changing this you can switch the item to point at a different Custom Item in your Toolbox. *Note: If you change the name of the Toolbox Item a custom item in your script is linked to a non-existent item, it will still work, but it will lose its link to the original item. This means that if you update the item in the Toolbox then the item in your script will not pick up the change.*

34.3 Updating a Custom Toolbox Item

Once you have created a Custom Toolbox Item, you may wish to update it. You can do this by simply re-dragging the original item you used to create the Custom Item into your Toolbox again. If you give it the same name, Badboy will save over the original item and all the Custom Items in your script linked to that item will use the new version instead of the old one. If you don't have the original item that you saved then just place the Custom Item that you want to update in your script and then right click on it and select "Expand". The item will expand to show the original contents of the Custom Item. You can then edit the items and drag them back into the Toolbox to update them.

34.4 Your Toolbox File

Badboy stores all of your custom tools in a file. You may like to save and backup this file, or to share it with your colleagues.

In the current release, Badboy always stores the Toolbox in the file "toolcache.bbt" in the Badboy installation directory. Later releases will allow you to choose the location of your Toolbox file, and to switch between different Toolboxes if you wish.

35 File Formats

This section explains the file formats that Badboy uses to save scripts.

35.1 Binary Format

The binary format is the original Badboy file format and in this format files are saved with the extension ".bb". The binary format has the following characteristics:

- It is fast to load and save
- It is compact
- It is not human readable and is hard for other tools to read
- It is hard to merge changes from two scripts together using text-based merging tools

35.2 XML Format

Because the binary format is difficult for other tools to read and write, a second file format was created. This format is based on XML so that any software program that processes XML can read and write the files. The XML format is saved using a file extension of ".bx", and has the following characteristics:

- It is slower to load and save
- It is much larger - script files may be 3 - 4 times larger when saved as XML than they are in the binary format
- It can be easily opened and read by other programs
- If you have a team of people editing the same script then it is possible to merge scripts together using text or xml based merge tools. This is particularly useful when you wish to check your scripts into source control systems.

35.3 Badboy XML

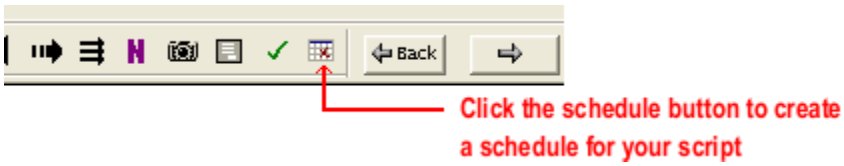
In versions prior to 2.0, Badboy supported an XML format called "Badboy XML". Although Badboy can write files in this format, it does not support loading them back in. This format is still supported in its current form, however new features and functions will be built based on the new XML format.

36 Scheduling Badboy Scripts

If you need to monitor your website or application or perform any kind of regular task using Badboy then you can take advantage of Badboy's scheduling functions. Scheduling allows you to program Badboy to run at a future time and date you specify, if you like, at repeating intervals.

36.1 Scheduling Manually

If you want to set up a schedule for your script to run based on fixed criteria then you can do this simply using the Schedule button on the Toolbar. The figure below shows this button:



When you click the Schedule button Badboy will create a Task in the Windows Task Scheduler that is pre-configured to run your Badboy script. Badboy then shows the properties box for the Schedule so that you can set the times you want the task to run.

Badboy schedules scripts using the Windows Task Scheduler. In order for Scheduling to work, you need to have the Task Scheduler component installed and it must be accessible from the user account that is running Badboy. If you don't have this component installed then Badboy will show an error message when you try to use the Scheduling features.

In the Schedule Task properties dialog you can set when you want your Badboy Script to run. You can choose to have it repeat at various different intervals. The figure below shows how this looks:

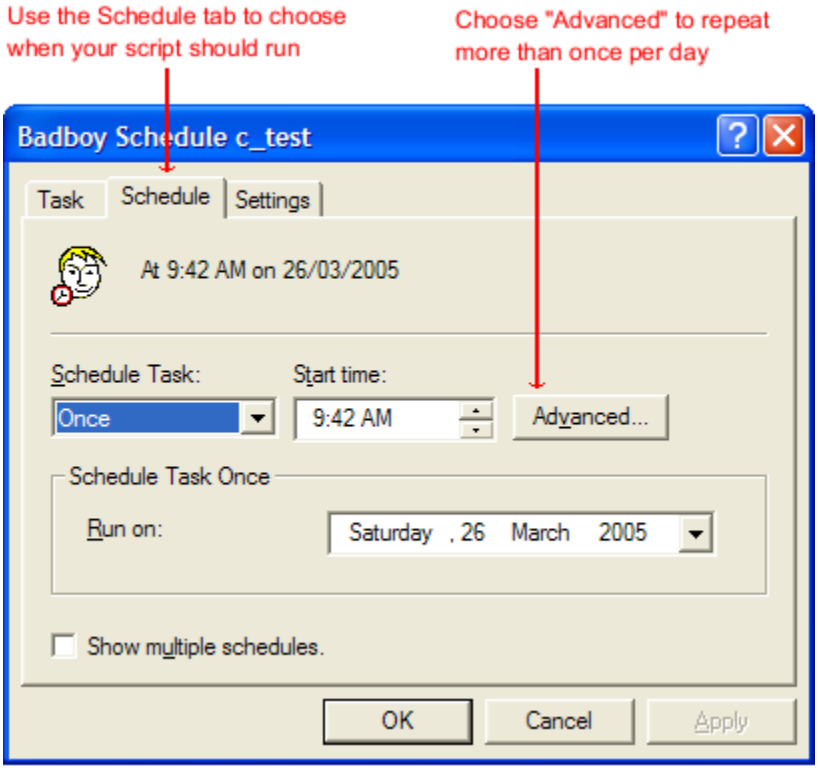


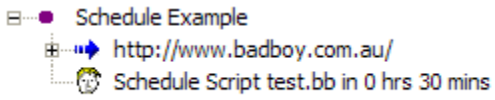
Figure 28: Schedule Task Properties

If you want to run your script more than once per day (e.g. every 30 minutes, for example) click the "Advanced" button to choose this option.

36.2 Scheduling as Part of your Script

Sometimes you may want to schedule scripts based on dynamic behavior - for example, if your ordering system takes 30 minutes to process an order you might want to schedule a test to check the order 30 minutes after your "Create Order" test runs. Of course you would only want the check to occur if your prior test has succeeded. Badboy's dynamic scheduling feature allows you to incorporate this kind scheduling as part of your Scripts.

To perform dynamic scheduling, drag a Schedule Item from the Toolbox into your Script. The figure below shows how the Schedule Item appears:



To set the delay time before the script runs, open the Schedule Item properties and choose "Later" in the Schedule options and choose the time you want the Scheduler to wait before it runs the script.

Badboy will not pause playing to wait for a Script that is scheduled later. It will schedule the script as a Task and continue running the current script.

You can include variable references in the "hours" and "minutes" fields if you want to schedule for an amount of time that depends on other outputs from your script.

36.3 Deleting a Schedule

Badboy does not provide a built in feature for deleting a scheduled script execution. Instead, you can easily delete tasks by using the features built into the operating system. To delete a task that you have scheduled:

- Choose "Settings=>Control Panel" on your Start Menu
- Double click on the "Scheduled Tasks" icon
- Identify the Badboy task and delete it by right-clicking and choosing "Delete".

Badboy tasks are always named "Badboy Schedule". Items scheduled by Schedule Items in your Script are named with a number that starts with the Id of the Schedule Item that created them (you can find the id by looking at the properties for the Schedule Item in your script.)

37 Using Badboy with AJAX Web Sites

Many modern web sites and applications use a technique known as "AJAX" for updating content on a page in the background without refreshing the page. Badboy is designed to support recording and playing back this type of application. This section explains how Badboy handles AJAX web pages and gives some tips for creating successful tests.

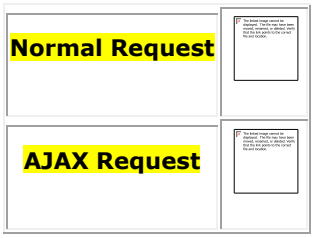
It's possible to record and play back AJAX web pages using Badboy in either Request Mode or Navigation Mode, however for functional testing in general you'll get better results if you use Navigation Mode. Read on below for discussion of why this is and how Badboy handles AJAX pages in these two modes.

37.1 Understanding AJAX Requests

AJAX requests differ in important ways from regular requests. The most important aspect of AJAX requests is that they do not directly cause the web browser to load a new page when they complete. Instead, when AJAX requests complete they invoke an existing JavaScript function on the page that launched the AJAX request. This JavaScript function can do all kinds of things - it can update the page dynamically, set JavaScript variables, fill out forms on the page - it may even launch more AJAX requests or cause a real page load to occur.

37.2 Recording AJAX Pages in Request Mode

When you record in Request Mode Badboy detects and records AJAX requests directly in your script. AJAX requests look similar to regular requests but they are colored gray. The table below shows how AJAX requests look compared to normal requests:



37.3 Playback of AJAX Requests

Because of the different nature of AJAX requests, Badboy plays them differently to normal requests. *Badboy sends the AJAX request to the server but it does not invoke the subsequent JavaScript call that handled to the original request.* This important difference means that when Badboy plays an AJAX request you may not see the screen update as it does when same action happens manually. However what *has* happened is that the web server has processed the request and now believes and behaves as if the page is updated. Since updates may be missing from the screen after an AJAX request executes, you might not be able to use Assertions that check the content of the page to verify that the AJAX request succeeded.

Since there can be occasions when the user interface becomes out of date with the server when playing back AJAX Requests, it can be useful to deliberately record actions that cause a page load after you have executed the AJAX request. Then you can do the usual content check Assertions to check the results of the request.

You might worry that your test will then not function correctly because the subsequent activity caused by an AJAX request is not executed. However this is generally not problematic because Badboy records *all* the interactions with the web server: if the AJAX request went on to make another AJAX request then Badboy will have recorded the subsequent request and will play it back. On the other hand, if the AJAX request actually invoked a page load then Badboy will have recorded that as well. So although there may be cases where the user interface does not reflect the updates caused by an AJAX request, the server is always kept correctly updated, and the next page load will cause correct content to be displayed.

The possibility that the screen may not be updated after an AJAX call, is the primary reason that it's generally better to record AJAX pages in Navigation Mode. Request Mode is mainly useful for two purposes: executing Load Tests with the Raw Browser Engine, or exporting your script to JMeter. Outside of these purposes you should usually be using Navigation Mode for testing AJAX based web sites and applications.

37.4 Recording AJAX Pages in Navigation Mode

For functional testing, Navigation mode is generally the best mode to use in recording AJAX based web applications, at least for those specific actions in the application that use AJAX requests. In fact, you probably won't notice any difference to how Navigations are recorded for AJAX interactions. However if you inspect closely you will see some small differences:

- If you open the properties you will see the "Passive Navigation" option is selected
- Underneath the Navigation item you may see an AJAX Request item () recorded to show the AJAX request that occurred when the item was clicked

37.5 Playback of AJAX Pages in Navigation Mode

When AJAX interactions recorded in Navigation Mode are played back Badboy does not do anything specific to initiate the AJAX request. Rather, it simulates a mouse click on the element so that the web browser initiates the AJAX interaction just as if the user had done it.

Once the AJAX request has been initiated by the browser, Badboy observes it, waits for it to finish, and then records the response time, size and other attributes are recorded as a Response for the Navigation.

After the AJAX request completes the browser invokes all the normal handling that occurs when the page is navigated manually. Because of this the page updates with new content and you can use Assertions to check the displayed content on the page just as you would in a non-AJAX application.

38 Integrating Badboy with Your Server's Log File

If you are a web developer or Software Engineer, you will be very familiar with the concept of a log file. Most applications (especially web based applications) write a file containing information about all the actions occurring as they run. This file usually contains detailed information including informational messages, warnings and errors. Because they show everything that is going on, application log files can be a tremendously useful tool in understanding, reproducing and diagnosing problems in a web site.

Unfortunately, log files are frequently lost, deleted or inaccessible to the person who needs them. Even when found the person reading it often has to search exhaustively to find the right part of the file before they can use it.

- *Wouldn't it be nice if the log file was captured automatically as you browsed, each part snipped out and associated with the browsing action recorded in your script?*

Badboy offers exactly this feature with *Server Log File Integration*! If you are responsible for QA or Support you can record with Badboy and put the log file in your defect submission - and know that the log is captured forever for convenient and easy review. If you're an Engineer or Developer you can bring up the log file at the press of a key and see the lines related to the last browsing action or for any item you have played or recorded in your script.

38.1 Setting Up Server Log File Integration

Setting up server log file integration is easy. To make Badboy start capturing your log file, go to the Preferences menu and select Preferences=>Developer Settings. Then check the box labeled "Enable Capture of Server Log File", and enter the location of your server's log file in the box provided.

The figure below shows the Server Log File Configuration page in Badboy's preferences:

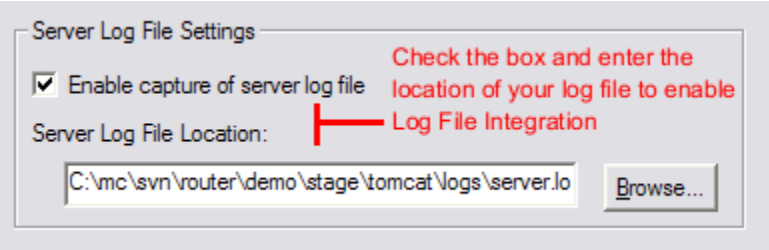


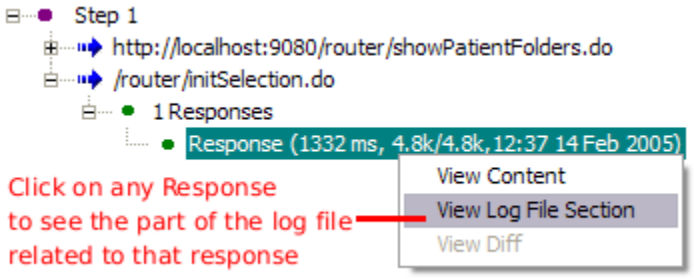
Figure 29: Server Log File Configuration

In the current release you must have access to the Application's log file in order to use Server Log File Integration. If the log file is on another computer then you may need to share the file system on that computer and map a drive to it on your own computer so that you can enter the location in Badboy. In a future release, Badboy will offer a feature to allow you to monitor any log file - even ones that you cannot map a drive to.

38.2 Using Server Log File Integration

Once you have configured Badboy to read your server's log file you can try it out by browsing around your web site to generate some new lines in the log file and recording some items in a script. You can then look at the log file in several ways:

- Select "View=>Server Log File" from the main menu, or hit **Ctrl-L**
- Right-click on a recorded response in your Script and select "View Log File Section"



In each case you should see lines that appeared in the log while the browsing occurred. In the first case they will be the lines related to the *most recent browsing activity*, while in the second they will be the lines related to the specific item that you clicked on. Because Badboy only shows the lines related to your last browsing activity you never have to load the whole log file and search through it for the right location!

You can choose the editor program that Badboy will use to show you the log file by setting as your preferred editor under the "Programs" of Badboy's Preferences.

You can also view the latest log file entries at any time by opening DOM View (Ctrl-D) and clicking on the "Server Log" tab. You can even leave it open and watch the server log scroll past as you browse.

You can pause the scrolling of the log file in DOM View any time by hitting the Space key. You can un-pause it by hitting the space key again. This is very useful if you need to read it or copy / paste information from it while the log is still scrolling by!

38.3 Trouble Shooting

If the log file always appears to be blank when Badboy shows it to you, check the following things:

- Did you point Badboy to the right log file in the Preferences dialog?
- Did your browsing cause any activity in the log? Sometimes browsing that doesn't cause any errors won't cause any log activity: this is quite normal and depends how your application writes its log files.

39 Badboy's Developer Features

If you are a designer, developer or engineer who builds web pages or the software that supports them then you are probably used to spending lots of time manually browsing your pages to test and debug your work. If this is you, read on to find out more about some of the great features that Badboy offers specifically for developers!

39.1 Editing Source Files Directly From Badboy

Most web sites are based on source documents that are used to generate content for rendering back to the browser. In the simplest case these are static HTML files. For more dynamic web sites, PHP, JSP, or ASP files may be used. The normal workflow of a person building such a site is:

- Browse until the page to modify is found
- Look at the URL in the browser try to work out what the source file would be. This might involve viewing the page source if it contains frames.
- Go to the source folder and search for the file in there.
- Open the file in an editor and make the change

It doesn't take much time, but it certainly adds up when done over and over - wouldn't it be nice if you could just hit a key from your browser and it would find the right source file for you and open it in your preferred editor? This is what Badboy does for you!

To use this feature:

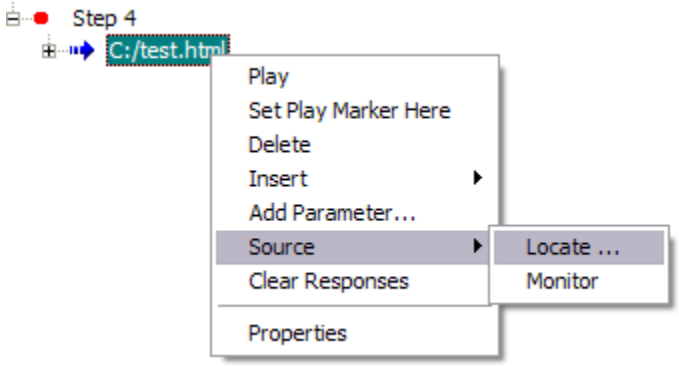
- Browse using Badboy to a page you would like to edit.
- Click in the browser on the content you want to edit (this sets the focus in the right place helps Badboy know what frame you want to edit)
- Press Ctrl-J or choose Tools=>Locate Current Page Source

Badboy will search for the file and open it in your editor!

The first time you use this feature Badboy will ask you to configure the source location for your web pages in the Preferences=>Developer Settings tab. This determines where Badboy will search for your source files. At this point you can also specify your favorite editor (otherwise Badboy will just use Notepad). If you want Badboy to search in multiple locations you can enter them as absolute paths into the source location field separated by semicolons.

39.2 Editing Source Files for Requests

It is also easy to edit the source file for any request that you have recorded in your Script - just right click and choose "Source=>Locate Source"



If Badboy can't find your source file it will prompt you to let you manually find the source file. You can choose any file you like. Badboy will remember this source file and save it with your script so that you don't need to manually locate it again.

39.3 Monitoring Source Files

Wouldn't it be nice if when you saved your file in your editor your browser could automatically load the new one - or even execute a whole sequence of actions to test out the new page?

Badboy can do this with its Monitoring features!

Activating Source Monitoring is easy: any request or Step in your Script can automatically play when one of its source files is monitored. The fastest way is just browse to a page you want to edit and press "Ctrl+Shift+J" - this will cause Badboy to edit the source file for the current frame *and* also add a monitored request to your script so that it updates the frame whenever you change the source! You can activate monitoring for any existing request in your script: just right click in the script and select "Source=>Monitor". Badboy will watch the source file for the request and play it automatically any time the source file changes. This means you can work continuously in your editor, changing the file and watching it update in Badboy without ever having to switch between programs.

Badboy can monitor sub-requests as well as top level requests! If you are working on a frame or iframe that belongs to a larger page, just select the corresponding sub-request and start monitoring it: Badboy will reload only that sub request in its correct frame each time you change the source.

39.4 Monitoring Steps

Sometimes things are more complicated than just reloading one page. For example, perhaps when you save your file you'd like a whole test to run, to make sure the whole scenario you are modifying on still works. For this situation, you can use *Step Monitoring*.

Step Monitoring works in a very similar way to Request Monitoring however instead of monitoring a single source file it monitors all the source files for the Requests contained in the Step. In addition to this, you can manually specify additional dependencies yourself - so for example, you can add any files that might affect the pages you are working on including images, Java Script files or CSS style sheets

When you activate monitoring for a Step it will cause the **whole Step** to play if *any* of the dependencies of the Step are modified. This means you can write a whole test with Assertions, JScript or other items to perform complex navigations and re-validate your site automatically each time a dependency changes!

39.5 Capturing Your Log File

Most web servers and application servers write a log file that is very useful for diagnosing problems when your pages are not operating correctly. Badboy makes it easy to see your log file at any time: just configure the location of your log file under Preferences=>Developer Settings, and then hit "Ctrl-L" to see the most recent lines added to your log file at any time!

Once you configure your server log file, you can also see the latest entries in it any time by opening DOM View (Ctrl-D) and selecting the "Server Log" tab. This can be a great way to grab any errors that may have been reported there to include in bug reports or other tracking mechanisms that you may have.

Badboy does more than just show your log file: because it is observing the actions of browser it is able to know which browsing actions cause which lines to appear in the log file. Hence, when you press "Ctrl-L", Badboy is smart enough to show you only the log lines that were caused by your last navigation in the browser!

For more information about Badboy's support for log files, read the separate topic, Server Log File Integration.

39.6 DOM View

Understanding complex pages can be a nightmare without some kind of guide to their structure. To help with this, Badboy offers a convenient DOM View feature:

- To activate DOM View just hit "Ctrl-D" in any Badboy browser window

The DOM View shows you a complete overview of all the elements in the page and lets you easily filter them, manipulate them and display them on the page. You can just select an element in the DOM View to see it highlighted on the page with a border.

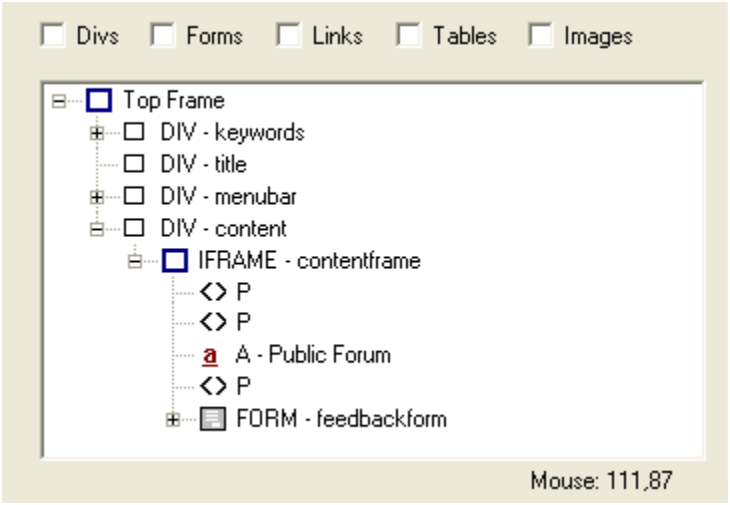


Figure 30: DOM View

DOM View provides another handy feature: a continuous display of mouse coordinates while you hover your mouse over the browser. By pressing and holding down the "Shift" key you can make the display use coordinates relative to any position on the page. This feature is great for developers get placement of elements correct without wasting time with trial error positioning.

39.7 Syntax Highlighting JavaScript Editor with Auto-Complete

Creating JavaScript either for your web page or for your Badboy Tests is much easier when you can edit it *in context* with a real editor. Badboy offers the unique ability to let you edit JavaScript in the context of the browser right where it will run for real. Badboy even offers a great Syntax Highlighting editor to help you understand the functions you are writing better and an auto-complete function (just hit Ctrl-Space). Since the auto-complete runs in real time on the page as it actually exists in your browser it can see dynamic properties that no normal editor can show you!

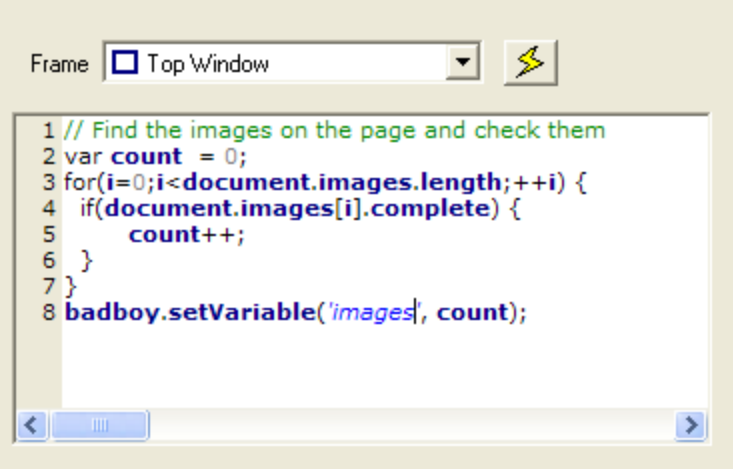


Figure 31: JavaScript Editor Auto-Complete

39.8 JavaScript Logging

With applications becoming increasingly driven by client side scripting, it's getting harder and harder to know what is going on inside them. To help with this, Badboy now gives you an additional tool in the form of JavaScript logging. The code below shows how you can add logging so that in a normal browser it has no effect but in Badboy you can read your statements from the Badboy log file:

```
/**
 * An example of how to put logging into your web pages so that
 * it appears in the Badboy log when run inside Badboy.
 */
var enableLog = true;
function log(msg) {
  try {
    if(enableLog) {
      window.external.info(msg);
    }
  }
  catch(er) {
    enableLog=false;
  }
}

function myTestFunction() {
  log("This is a log message for Badboy!");
}
```

You can configure the Badboy log file in the Preferences dialog on the "Logging Tab". Once you have configured it you can easily view it at any time by opening DOM View (Ctrl-D) and selecting the "Badboy Log" tab. If you wish you can drag this tab to another position in the Badboy window so that you can see it even when DOM View is not open. You can pause / un-pause the view from scrolling by pressing Space.

40 Shortcut Keys

Badboy supports some shortcut keys for convenience. These can be used while playing to play and repeat sections of script that you are testing.

Key Combination	Description
~	Inverts a Content Check, Color Check, or Assertion
F2	Toggles Record Mode On/Off
F3	Displays the search/replace dialog, or searches if the dialog is already open.
F4	Opens DOM View and tries to locate element under mouse cursor in DOM tree if mouse is in the browser window. Displays box containing element info.
F6	Single steps (plays next single item in script.)
F8	Shows/Hides Script Tree
F9	Shows/Hides Summary Tab View
F12	Shows/Hides both Script and Summary Views together (gives browser full window space)
Ctrl-Alt	Changes Record Mode temporarily to Navigation Mode while held down
Ctrl-Alt-B	Toggles a break point on the selected item in the Script Tree.
Ctrl-Alt-Down	Moves the play marker to the previous item in script.
Ctrl-Alt-N	Toggles record mode between Navigation and Request mode
Ctrl-Alt-Right	Starts playing from the current item.
Ctrl-Alt-Space	Stops Playing
Ctrl-Alt-Up	Moves the play marker to the previous item in script.
Ctrl-Enter	Replays the item current item in the script
Shift-Enter	Sets currently selected item in the Script Tree as the current play position, without playing it. Only when Script Tree view has focus.
Ctrl-Shift-Enter	Replays the current step in the script
Ctrl-F5	Plays entire hierarchy from the current item. (Note: if focus is inside the browser, IE will intercept as "Refresh").
Ctrl-D	Toggles DOM View On/Off for the active window.

Key Combination	Description
Ctrl-J	Attempts to find and edit the source code file for the current page and/or frame that have focus in the browser.
Ctrl-K	Clears all responses from the Script.
Ctrl-L	Displays the lines Server Log File related to the most recent browsing activity.
Ctrl-Page Down	While in DOM View changes to next frame in frame list
Ctrl-Page Up	While in DOM View changes to previous frame in frame list
Ctrl-Shift-D	Opens the documentation editor for the item currently selected in the Script Tree.
Ctrl-Shift-J	Attempts to find and edit the source code file for the current page and/or frame that have focus in the browser and also adds the URL for the frame to your script as a Monitored request.
Ctrl-Shift-L	Opens DOM View and shows Badboy Log (Badboy log must be enabled in Preferences)
Ctrl-Shift-Left	Rewinds the play marker to the previous step.
Ctrl-Shift-M	Adds URLs for all frames in the current browser as Monitored requests.
Ctrl-Shift-R	Opens DOM View and shows Request Log
Ctrl-Space	While in Log View clears current content from view and only displays new content.
Shift-F2	Toggles Record Mode On and sets the parent Step of the currently selected item in the Script Tree as the current recording Step.

41 Load and Stress Testing with Badboy

A key part of validating your web site is to know not just how it behaves when you alone are using it, but how it behaves when a large number of people are accessing it all at once. This is generally known as "load" or "stress" testing. Badboy offers some highly useful features to help you do this kind of testing.

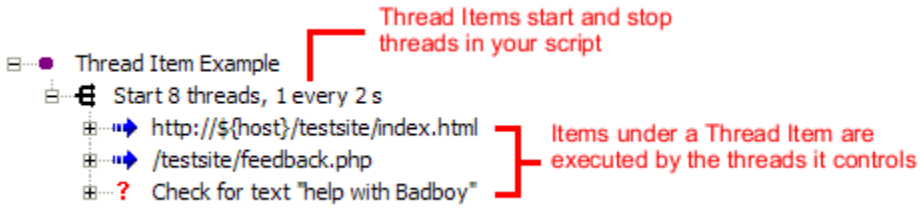
41.1 Threads

In order to understand Badboy's load testing features it is important to understand the concept of *"threads"*. The term "thread" is borrowed from computer programming and simply means a separate virtual instance of Badboy executing your script. For example, if you have five threads running then it is as if you had five separate copies of Badboy running, all executing your script - just like threads in a cloth, all the threads run in parallel, doing the same thing. Each thread creates an additional unit of load on your server, simulating the effect of one or more users navigating your web site.

41.2 Thread Items

Badboy makes it easy to create Threads as part of your script by using *Thread Items*. Thread Items play as part of your script, but they behave a bit differently to normal items. Instead of directly executing the items they contain, Thread Items instead create "virtual", invisible Badboy instances and use those to play the items underneath them in the script. Each such thread executes as if one user was browsing, creating additional load on your web server.

The figure below shows how a Thread Item looks in your script:



42 Creating and Running Thread Items

42.1 Creating Thread Items

There are two main ways to create Thread Items in your script:

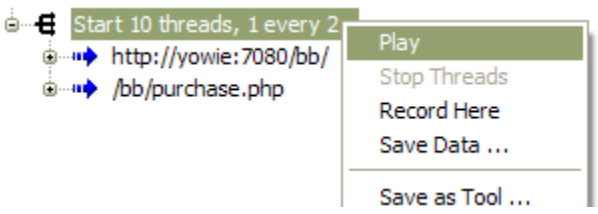
- Drag a Thread Item into the Script Tree from the Toolbox (press Ctrl-T to see the Toolbox) - this creates a new, empty Thread Item and sets it to be the recording point for your script. If you then browse using the browser, the thread item will record requests to be executed.
- Convert an Existing Step - you can right click on any Step in your script and choose "Convert to Thread Item". This will take all the child items of that step and put them into a new Thread Item that you can use to run threads.

Of these two approaches, the second one is usually more convenient because it allows you to record your script as a normal Step which you can test by playing back first. You can then easily add Assertions and other items to the Step before converting it to a Thread Item when you are finished.

42.2 Running Thread Items

Thread Items are intended to be executed in the normal flow of your script - that is, they are launched by your script as it plays through. This allows you to add several different Thread Items to your script to simulate different activities and have Badboy launch them all.

For testing, the easiest way to run your Thread Item is to right click on it and select "Play" - this causes just the individual Thread Item to play.



42.3 Stopping Thread Items

Normally, Thread Items stop by themselves after the period of time for which they have been configured to run. Note that this applies even if the rest of your script finishes playing - so your threads may continue running even after everything else finishes. For example, if you have configured your Thread Item to run for 30 seconds then after you select "Play" it will continue running for 30 seconds even if the rest of your script finishes. There are various options for how to configure the run time - see Configuring Thread Items for information about how to configure the run time of your Threads.

During testing you may often want to stop your threads manually before they are scheduled to finish. To do this, press the "Stop" button in the toolbar or hit the Ctrl-Alt-Space keys to forcibly stop all your Threads from running. Note that even then it may take a few seconds for the Threads to completely stop, especially if you are running threads using the External MSHTML Browser engine (see Choosing a Browser Engine for more information on Browser Engines).

43 Browser Engines

43.1 Browser Engines

Badboy supports different "engines" for running Background Threads, which correspond roughly to different kinds of browsers to use for executing your tests. Each engine has different characteristics and capabilities. After you create a Thread Item, one of the most important decisions to make is which Browser Engine to use for your load test. Which one you choose would depend on the nature of your web site, how you have recorded your script and kind of test you want to conduct.

The three kinds of thread engine are:

- **Raw** - Raw threads are very simple. They do not attempt to render the pages downloaded by the browser at all. All they do is download the data and let you do Assertions to check that the page content returned contains expected text. Due to Raw Threads being so simple Badboy can execute them *very* efficiently and can hence can generate a very large load on your web server from only a single instance of Badboy. The biggest disadvantage of Raw Threads is that because they do not parse or render the HTML they cannot perform any operations that involve the user interface of the browser. For example, and they cannot execute JavaScript, and most kinds of Navigations cannot be executed because they are performed by acting on the browser user interface.

  Because of these limitations normally the best way to create scripts for Raw Threads to execute is to only record your browsing actions using Request Mode as this mode does not rely on the browser user interface and thus is well supported by Raw Threads.

- **External MSHTML** - MSHTML is the browser component that Microsoft ™ includes with Microsoft Windows ™ and Internet Explorer ™ . When you use the MSHTML Engine Badboy creates invisible instances of MSHTML that it uses to navigate your web site. Although you can't see them they are actually downloading, parsing and rendering the HTML as if a real browser was doing it. The biggest advantage of this is that if your web site contains JavaScript, or if you want to control the virtual Badboy instances using JavaScript (e.g. to manipulate variables or to perform JScript Checks to validate your web site) then MSHTML Threads can do that. When you use External MSHTML, Badboy in fact runs each virtual browser in its own process. This ensures that the virtual browsers will not interact, for example, by sharing session cookies.

*External MSHTML threads are actually run using the Badboy command line runner, bbcmd. When they run they load the script from the **saved file**, not from the script that you may be editing. This means that it is very important to ensure your script is saved before launching the thread item as otherwise it may play different items to the ones you expect!*

- **MSHTML** - MSHTML is the same as External MSHTML except that the virtual browser windows are run *inside* Badboy itself. The main advantages to this are that performance is better and the tests are less resource intensive on your computer - just as it is less expensive in memory to run one copy of a browser with several windows instead of several copies of the whole browser, so it is less expensive to run virtual windows in a shared Badboy instance than it is to launch many separate copies of Badboy. However a big disadvantage of using internal MSHTML windows is that they can interact with each other. For example, all threads created internally share cookies with each other, and even with the main (visible) Badboy window. Thus all such threads may appear to be logged in as the same user in some applications.

44 Configuring Thread Items

Thread Items offer a range of sophisticated options for configuration of how they start, stop and execute threads. This section explains these configuration options in detail.

For reference, a diagram below is shown of the Thread Item properties page:

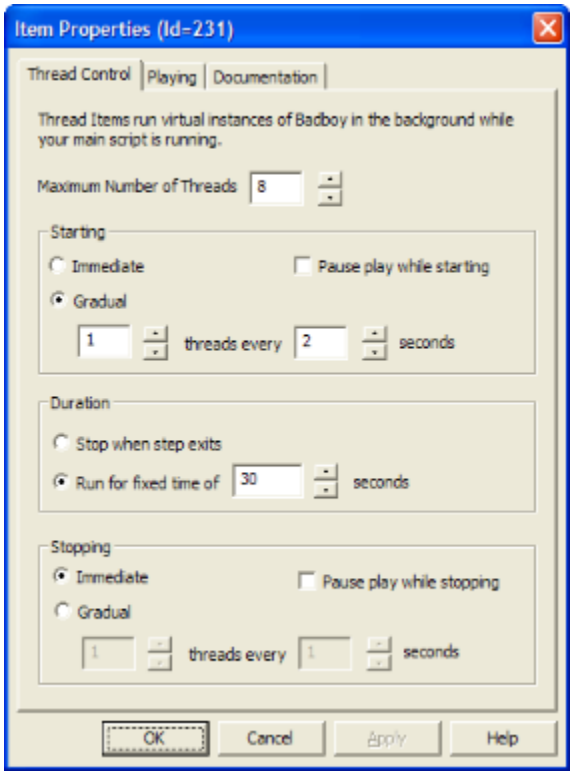


Figure 32: Thread Item Properties

44.1 Setting the Number of Threads

When you begin load testing a web site you need to determine how much load you wish to try and create. This in turn depends on many factors and constraints, and of course, on the requirements of your web site. The following are points to consider when setting the maximum number of Threads:

- **Actual Load Created per Thread -** Many people begin by assuming that each thread corresponds to one real-world person browsing your web site. However this is false in many situations, because when Badboy browses your web site, it does so as fast as it possibly can. It does not wait between playing items in the script the way humans do - to read the page contents or to fill out forms, for example. Because of this, one Thread usually corresponds to more than one real world user. In fact, when using the Raw Browser Engine, a single thread can create a comparable load to that of dozens of real world users, depending on your web site. If you wish to make each Thread more closely match the behavior of a real-world user, add Timers to your script to add a realistic "think time" between each interaction in the script. In that case you can begin to consider a single Thread as approximating a single user.
- **How much Load can your Computer Produce?** - Each Thread you create doesn't just add load to your web server - it also uses resources on your own computer and your network. At some point you may find that the Threads you create slow down because the Threads have saturated the available resources on the client computer rather than those on the server. This is especially true for the "heavy weight" MSHTML Browser Engines that simulate a real browser as part of the load test. Once you have reached this point your load test will no longer produce realistic results because you are using one computer where in real life your web site will have to deal with many hundreds or thousands of client computers. For this reason you should watch the amount of CPU, RAM and network bandwidth used by your load test and ensure that they stay within the limits that your computer is capable of producing. If you need to produce more load than a single client computer can create then you may need to use multiple computers running Badboy simultaneously to create that load.

44.2 How long Threads Run

When you start Threads, they do not continue running forever. Rather, they run for a programmed amount of time that is controlled by settings in the Thread Item properties. The primary setting for this is the **Duration**. Badboy offers three options for how to control the duration of a Thread Item's threads:

- Until Step Exits - If you choose "until step exits" they will loop over and over, executing until the parent Step finished executing. This makes it easy to coordinate a test by putting all the items you want to test inside a single parent step. As soon as the step exits your threads will automatically begin stopping.
- Fixed Time - This option will cause your threads to run for a fixed amount of time, regardless of all other factors. This means that your threads will keep running even if the step exits, or even if the whole script is finished playing.
- Fixed Iterations - This option will cause each Thread Item to run exactly the specified number of iterations of the children of the Thread.

If you find your threads don't seem to be running, check if you chose "Stop when step exits" for duration. If you choose this option and your Script finishes playing your threads may be aborting before they even started. Try setting a fixed interval to experiment with playing your threads.

44.3 Gradual Starting/Stopping

Often it is not desirable to have threads all start at once because all the steps are then "synchronized" which does not mirror normal user behavior. For example, while it might be common for your web site to have 100 users at any one time, it would be very unusual for all those users to decide to login at exactly the same instant. Another problem is that starting all the threads at the same time does not allow you to observe the effects of how the system behaves as load increases. Badboy allows you therefore to start and stop threads gradually by setting a number of threads to start and an interval in seconds to space each group of threads by.

44.4 Thread Limitations

In addition to the limitations mentioned for the different Browser Engines, there are some obvious limitations when you are running items in threads as opposed to executing them in a real browser, since many items in Badboy depend on the presence of a real browser interface in order to work. These include

- Mouse Clicks - since there is no browser window, mouse clicks in the window will not work. However Mouse clicks on other windows than the browser window may still work.
- Keys - You generally cannot send key strokes to items in background threads. Note that the keys items will still play but they will send key strokes to whichever window is active. Usually this would be undesirable and thus it is not recommended to use these at all inside Thread Items

To see if items are not playing correctly in background threads, enable the log file and set it to INFO level in the Preferences. Then after playing your Thread Item you can look for any warnings in the log file which will tell you any items that are unable to play properly because of being embedded in a Thread Item. Of course, you can also add Content Checks to validate the returned HTML as well.

44.5 Using Different Data Across Threads

You may encounter situations where your script needs to ensure different data is used by each thread. For example, each thread might need its own login account if your system only allows a user to login to a single session at a time.

To achieve this easily, Badboy provides a special *"threadNum"* variable that you can use to access different data in different threads. An example of how to do this is illustrated in the following steps:

- Create a variable (for example, "foo") and add a list of values to it. In this example, one value will be used by each thread that you run.
- When you reference your variable in your script, do so in the form:
- `${foo[${threadNum}]}`

This will cause your script to access the value of "foo" according to the index $threadNum, which is set by Badboy for each thread automatically for you.

44.6 HTTP Authentication and Proxy Authentication

Badboy has elementary support for Basic Authentication in threads. You can enable this by setting environment variables as described in the table below:

Environment Variable	Effect
BADBOY_AUTH_USER	User name supplied for HTTP Basic Authentication
BADBOY_AUTH_PASSWORD	Password supplied for HTTP Basic Authentication
BADBOY_PROXY_AUTH_USER	User name supplied for Proxy Authentication
BADBOY_PROXY_AUTH_PASSWORD	Password supplied for Proxy Authentication

45 Viewing and Understanding Thread Results

45.1 Accessing Thread Data

After you have run a load test the next step is to analyze the data and look for potential problem areas that may not be performing sufficiently well. There are several different ways to analyze the data from a Badboy load test:

- Viewing the Response Time Graph
- Saving the Response Time Graph
- Save Raw Response Time Data
- Save Time Average Data

45.2 Viewing Response Time Graphs

A key element of load testing is visually tracking the results so that you can easily identify problematic areas of your application. Badboy makes it easy to see at a glance the performance of all the items inside a Thread Item by viewing them in the Graph View, which is normally available as a tab next to the Summary View in the lower left hand corner of the Badboy window.

To see thread results in Graph View, just select the Graph tab in Badboy and then click on the Thread Item that you would like to see results for in the Script Tree. The Graph View will then show you a graph displaying the average response time for each item in the Thread Item at periodic intervals over the time the Thread Item has been running. This graph shows running time (time since the test began in seconds or minutes) along the bottom (X) axis and the *average response time* for various items along the vertical (Y) axis. The average response times are values that are calculated at regular intervals during the test based on the responses received during those intervals.

The response time graph is updated "live" as your test runs so you can track how your threads are performing in "real time" by watching this graph.

The following picture shows how Graph View appears:

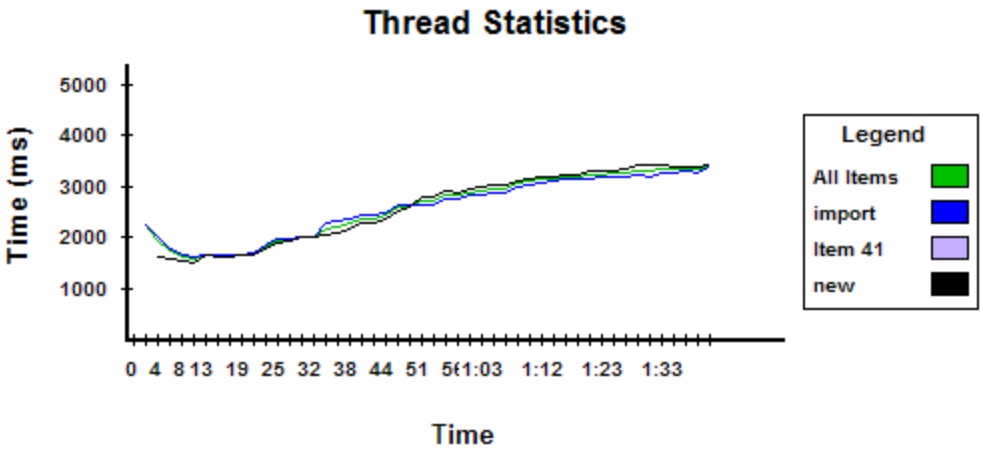
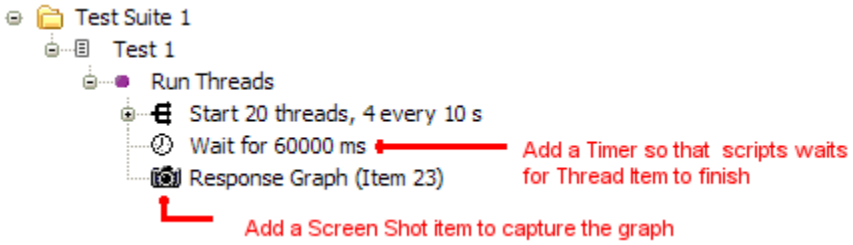


Figure 33: Graph View

45.3 Saving Response Time Graphs

If you would like to save a copy of the graph, for example, to display in the HTML Report for the test, add a Screen Shot item from the toolbox to your script so that it executes after the Thread Item has finished. You can then configure the Screen Shot item to capture a Response Time Graph, and if you enter the id of the Thread Item as the item to capture the response graph for then Badboy will capture an image of the thread results for display in your report.

The following picture shows an example of how to create a script that automatically captures a snapshot of the response time graph for a Thread Item:



45.4 Saving Raw Response Time Data

If you wish you can download all the raw response times received for items in your script. To do this, open the File menu and select the option "Export Response Times". This will let you save the response times for all the items in your script in CSV format, allowing you to further analyze them in a spreadsheet program or other analytical tool.

45.5 Saving Time Average Data

In addition to saving the raw thread data you can also save time averages of the data. Rather than containing an entry for every response to every item, this report instead contains the average response time for each item at regular intervals of the test. This allows you to easily plot how the average time responded to various conditions changing in your test.

To save Time Average data, right click on the Thread Item and choose "Save Data".

46 Global Threads (Legacy Function)

46.1 Global Threads - Thread Control Dialog

Global Threads are a Legacy Feature of Badboy and may not be supported in future versions!

In previous versions of Badboy Threads were controlled using the Thread Control Dialog which is accessed from the "Tools" menu. When you run threads this way, your script is played all the way through without pausing at steps. When it reaches the end it will automatically repeat from the beginning.

The diagram below shows how the Thread Control Dialog looks:

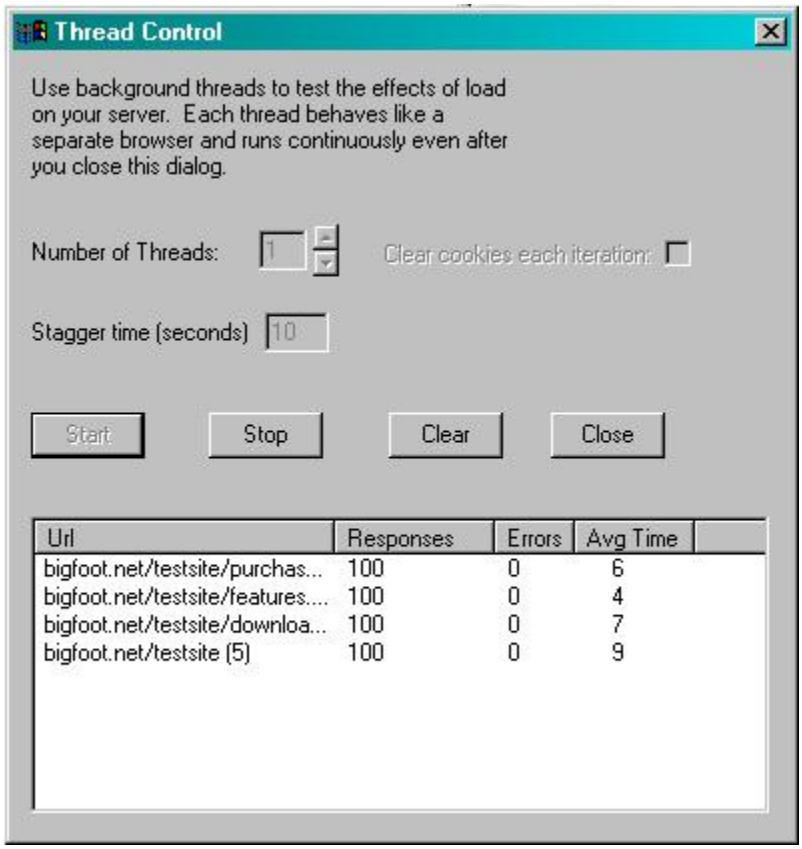


Figure 34: Thread Control Dialog

The Thread Control Dialog offers you several options that control how the Threads run.

Attribute	Meaning
+	How many threads you want to run. You can enter very large numbers here if you wish, however you should be aware that your computer might be unable to run fast enough support all the threads, in which case each thread will slow down to within the limits of your machine. To monitor this, check the level of CPU being used on your computer to make sure that it is below 80% while the load test is running.
Stagger Time	Normally when you start a load test, all the threads will start together at exactly the same time. Frequently however this creates an unrealistic load pattern on the server whereby there is an instantaneous jump in the load to a very high level. To avoid this Badboy lets you "stagger" the ramp up of users by starting up threads one at a time with some time separating each thread. You can specify here the length of time to wait between starting each thread in seconds. You should be aware that this will mean that it will take some time before your load test is running at maximum load. For example if you have ten threads and 30 seconds stagger time then it will take 300 seconds or five minutes for the load to ramp up.
Clear Cookies	By default each Thread will behave as if the same user is using a single browser to navigate your script over and over again. However if your site uses cookies then the user may have some state left over from their previous iteration when they start a new iteration. Sometimes this can cause problems for some applications, so you have the option for Threads to clear out the cookies before it starts every iteration. You should be aware of some subtle differences when you clear cookies, however. Some websites will associate dedicated memory on the server to each unique cookie that they receive. If you clear cookies with each iteration then that memory may get allocated over and over again each iteration, causing the server to (eventually) run out of memory. You need to decide what is a realistic scenario for your application when using this option.

46.2 Thread Statistics

The Thread Control Dialog also shows you statistics about your threads. The information provided is as follows:

Name	Meaning
URL	This is the URL that the statistics refer to. Since several different items can be played it may not always be the actual URL but descriptive text of the item instead. Note that the URL is simplified to remove parameters and the protocol information to improve the readability of the display. When the item is a Request the id of the request will be displayed so that you can distinguish the request from others that may have the same URL.
Responses	This is a count of how many times the item has been played. Note that Badboy only tracks the most recent 100 responses in order to conserve memory. Therefore once you reach 100 iterations, this number will no longer increase.
Errors	A count of how many of the responses have returned errors. These may be HTTP errors such as Error 500 or 404 errors or they may correspond to Assertions or other items in

	the script reporting that they did not play correctly.
Average Time	This is the average time in milliseconds (ms) that the response to the request took. Note that where a request has many sub requests (e.g. redirects or child frames) the reported time includes the total time for all the requests to occur.

47 Automating Badboy with OLE

Badboy is built as a first-class COM object so that you can access it and automate it from external programs using your choice of programming or scripting language. For example, Badboy can be automated from VBScript, VBA (Visual Basic for Applications), and even from JScript. Badboy also exposes its OLE interface to JScript items running within Badboy scripts as the "window.badboy" object, so you can make advanced scripts that can introspect Badboy itself including all the items in the script tree, preferences and variables.

See Adding JavaScript to Scripts for some examples of what you can do with JScript items inside your Badboy scripts.

47.1 OLE Interface API

The following table lists the operations you can call to control Badboy through OLE.

Operation / Property	Description
current	The id of the current play item (Read/Write property)
boolean openFile(BSTR* fileName)	Causes the given file to be loaded. Note: this method is deprecated.
boolean openFile2(BSTR fileName)	Causes the given file to be loaded.
void play()	Causes playing to start (as if the user had clicked the "Play" button).
boolean isPlaying()	Returns true if a play operation is in progress. If called after play finishes, returns false.
void sleep(LONG timeMs)	Sleeps for the given time in milliseconds. This is useful when accessing via JScript as JScript does not have a native sleep operation.
ULONG saveResponseTimes(BSTR fileName)	Saves response times to the given file.
void playAll(void)	Starts playing but will play all the way through rather than pausing at Steps in the script.
LONG record(LONG id)	Sets the recording point to the step with the given id, or the step containing the given item if it is not a step itself. If no step is found, fails and does not change the recording point or the recording state.

Operation / Property	Description
	To turn off recording, pass zero (0) as the recording id. If successful, returns the id of the item that became the recording Step as a result.
BSTR getVariableNames()	Returns comma separated list of all defined variables
BSTR getVariable(BSTR variableName)	Returns the value of the given variable.
void setVariable(BSTR variableName, BSTR variableValue)	Sets the value of the given variable.
void getVariableProperty(BSTR variableName, BSTR propertyName)	Returns the value of the specified property for the specified variable, if it exists. • name • currentValue • valueCount - number of values in variable's value list • value • obfuscate - "true" or "false" • autoVariableExpression - Regex to use with auto update function • isAutoVariable - "true" or "false" • autoLink - "true" or "false"
void setVariableProperty(BSTR variableName, BSTR propertyName, BSTR propertyValue)	Sets the value of the specified property for the given variable. Property values include: • name • currentValue • value • obfuscate - "true" or "false" • autoVariableExpression - Regex to use with auto update function • isAutoVariable - "true" or "false" • autoLink - "true" or "false"
void deleteVariable(BSTR variableName)	Deletes the named variable including its value list.
Object getResponse(LONG id)	Search for and return the newest response under item with given id
void quit(void)	Causes the Badboy instance to close/exit. NOTE: may be held in memory still depending on how the client holds the resources (it must release the reference).
void clearResponses(void)	Deletes all responses in the Script. Use this in long running scripts to prevent Badboy from using an increasing amount of memory as it runs due to captured responses.
LONG seek(LONG id)	Attempts to move the Badboy play engine to a position such that the given item (specified by its Id) will play next. Returns the Id of the actual item that will be played next after the seek() operation has

Operation / Property	Description
	been performed, or -1 if the seek() operation was unable to execute.
BSTR eval(BSTR expression)	Evaluates the given expression, substituting variables according to Badboy's variable expression syntax. Variables may be referenced in the form "${name}" and values in variable value lists can be referenced using syntax "${name[index]}".
BOOL addValue(BSTR variableName, BSTR value);	Adds the given value to the end of the given variables' value list.
LONG summary(LONG id, BSTR attribute)	Returns the value of the requested summary attribute for the item in the script with the given id. Valid values of the attribute parameter are: • played • succeeded • failed • assertions • warnings • averageResponseTime • maxResponseTime • timeouts
void setUIOption(BSTR option, BSTR value);	Controls aspects of the UI. Valid values of "option" include: • toolbars - "on" or "off" • windowstate - "maximize","minimize","normal","hide"
void setSilent(BOOL isSilent)	Sets "silent" mode on or off. Silent mode prevents any messages or other dialogs from showing that will halt the script from playing. This **overrides** settings that may exist on items such as Assertions. This is useful for playing scripts in an unsupervised setting without having to manually change configuration of items in the script.
BOOL save(BSTR format, BSTR fileName, BSTR options)	Saves the script (or data exported from it) in the requested format. The "format" parameter can be one of: • BadboyScript - save script in binary format (*.bb) • Xml - save script in XML format (*.bx) • BadboyXml - export script to BadboyXML (*.xml) - not openable by Badboy • JMeter18 - export JMeter version of script • Report - save standard HTML Report • BrowserContent - content of current browser window Options are a comma-separated list of attributes in the form "name1=value1,name2=value2". The only currently supported options are: • exportImages=true/false • includeResponseContent=true/false
version	Property indicating the version of the installed version of Badboy.
BSTR progress(LONG id)	returns a floating point value between 0 and 1 indicating the position of playback within the specified item.

Badboy also supports an OLE interface on the script items in the script. The script itself can be accessed as the "badboy.script" object and it and the children returned by its operations support the Script Item interface in the following table:

Name	Description
long id	Read-only property returning the id of this item in the script
Object find(LONG id)	Find a given script item in the script, specified by its Id and return it as an object.
Object next	Return the next item after this one in the script
Object parent	Return the parent of this item in the script
long length	Read-only property returning the number of direct children belonging to this item.
BSTR get(BSTR propertyName)	Return the value of the named property.
BOOL set(BSTR propertyName, BSTR propertyValue)	Set the value of the named property.
Object at(LONG index)	Returns the child of this item at specified index, if any.
BOOL removeAt(LONG index)	Removes child at specified index, if it exists.

The above properties are supported on all items in a Badboy Script. Each type of item also has a list of named properties that are specific to that type, and which can be retrieved or set using the "get()" and "set()" methods. See the Script Item Property Reference for full details of the properties available on individual item types.

Example 1:

This example shows how to set the name and value of a Form Value (a child of a Form Populator), in Badboy's script. The FormValue has the id 71:

```
window.badboy.script.find(71).set("name","This is the new name!");

window.badboy.script.find(71).set("value","This is the new value!");
```

Example 2:

This example shows how the Badboy OLE interface can be used to control Badboy with VBScript via the WSH (Windows Scripting Host). To run the example, save it as a file called "test.vbs" in your desktop, then double click the file. Make sure first that you have a script called "c:\test.bb" saved on your computer for it to open.

```
'
' Example Program to Demonstrate OLE Scripting Access to Badboy
'
' This program shows how you can manipulate Badboy as an OLE object
' via the Windows Scripting Host.  You can load scripts and play them
' in a scripted environment this way.
'
' Expect many more OLE methods & properties to be exposed in the future!
'

' Create a Badboy instance
Dim badboy
Set badboy = WScript.CreateObject("Badboy.Document")

' Open a file to play
badboy.openFile2("c:\test.bb")

' Note: at this point the window may be invisible.  If you want to see it
' then you can set the visible property directly:
' badboy.Visible = true

' Tell badboy to start playing the script
badboy.playAll

' If the script exits then the badboy instance will be destroyed.
' so sleep while it is playing to let it finish
while badboy.isPlaying
    WScript.Sleep(1000)
wend

' Get the number of failures
failures = badboy.summary(1, "failed")

' Save response times
badboy.saveResponseTimes("c:\\test.csv")

MsgBox("Done with " & failures & " failures")
```

48 Script Item Property Reference

This section explains the properties can be accessed on the various items in Badboy Scripts. There are 4 different property types that are recognized by Badboy. These are:

- String (text)
- Integer
- Boolean - set as "true" or "false"
- Date / Time - set in format YYYY-MM-DD Hour:Minute:Second

48.1 Accessing Properties

Properties are accessed using the `get` and `set` functions that are available on all script item objects. For example, to access the type of the item with id 32 in your script you could use the "get" function from JScript as follows:

```
badboy.script.find(32).get("itemType");
```

48.2 Available Properties

The following table lists the properties that are accessible on each type of item.

All Items	Property Name	Notes
	itemType (readonly)	
All Playable Items	**Property Name**	**Notes**
	timeoutSeconds	
	timeoutEnabled	
	timeoutAction	
	targetWindow	
	itemName	
	disabled	set to 'true' or 'false' to disable or enable the item.
	documentation	
	waitForPageToLoad	
	sourceLocation	
	recordResponses	
	quiescenceTimeMs	

	isInherited	
Assertion	**Property Name**	**Notes**
	captureScreenShot	
	scalingPercent	
	captureDuplicates	
	passed	0 if the Assertion failed last time it played, 1 if the Assertion passed last time it was played
AssertionFailure	**Property Name**	**Notes**
	description	
	timeStamp	
BitmapCheckItem	**Property Name**	**Notes**
	RGBTolerance	
	yTolerance	
	xTolerance	
	b	
	g	
	r	
	y	
	x	
	type	
CaptureItem	**Property Name**	**Notes**
	name	
	timeStamp	
ClickItem	**Property Name**	**Notes**
	xPosition	
	yPosition	

	windowName	
	windowSizeX	
	windowSizeY	
	enableRestoreSize	
	waitForPageLoad	
	cascade	
	raiseToTop	
	clickType	
	upXPosition	
	upYPosition	
	dragMouse	
	waitForWindow	
ContentCheckItem	**Property Name**	**Notes**
	pattern	
	type	
	searchMode	
	bUseRegex	
	checkMessageBoxes	
	checkRuntimeBodyContent	
Custom	**Property Name**	**Notes**
	name	
	toolboxName	
	modifiedTimeStamp	
DataSourceItem	**Property Name**	**Notes**
	connectString	
	loadString	

	loadType	
	mapType	
	tableName	
	dataSourceName	
DownloadHandlerItem	**Property Name**	**Notes**
	responseType	0, or 1 for whether to save or cancel the download respectively.
	location	The location in which to save the downloaded file. If an existing directory then the file name is determined automatically, if a file then the exact file name is used.
	openAction	0,1,2 for whether to not open the file, open externally or open internally.
FormPartItem	**Property Name**	**Notes**
	headers	
	substituteVariables	
	sourceType	One of either "file" or "script"
	sourceFile	Used only if sourceType is set to "file".
FormPopulator	**Property Name**	**Notes**
	headers	
	substituteVariables	
	formName	
	submit	
	formIndex	
	actionType	
	ignoreHiddenFields	
	useRegex	
FormValue	**Property Name**	**Notes**
	name	

	value	
	index	Index of element to populate when duplicate names exist.
	sendEvents	true or false. Set to 'true' to cause DOM events such as keyup, keydown, onchange to be sent when field is populated.
	retainFocus	true or false. Set to 'true' to cause field to keep its focus after population. Only applies when sendEvents is set to true.
Increment	**Property Name**	**Notes**
	variableName	
	strategy	algorithm to use for changing value of variable, default = 0 random integer = 1, next list value = 2, next integer = 3
	precision	number of digits to use for random part of value (only applies to random strategy).
JScriptCheck	**Property Name**	**Notes**
	script	
	frameType	
	frameName	
KeysItem	**Property Name**	**Notes**
	keys	characters / virtual key codes to send
	runInBackground	true / false - whether to execute asynchronously
	delaySeconds	number of seconds to wait before sending keystrokes
	windowCaption	window to wait to have focus before sending keystrokes
	autoFocus	'true' or 'false'. If 'true', window with name corresponding to windowCaption will be automatically raised to the top and focused as soon as it is found.
Navigation	**Property Name**	**Notes**
	reference	Reference for the active reference type.

		Note: A referenceType must be set prior to setting the reference property.
	targetFrame	
	continueOnFailure	
	referenceType	Integer from 1 to 3, indicating type of reference
	index	Integer, zero-based, indicating index of reference to Navigate for the active reference type when executed. Note: each reference type maintains its own index, setting the index using this property only sets it for the current active reference, as controlled by referenceType property. This property can be set but is ignored for JavaScript references.
	filterType	One of the values "All", "Links", or "Buttons". The referenceType must be set prior to setting filterType.
	eventType	
	isPassive	
	passiveTimeoutMs	
	useRegex	Boolean property - set to 'true' or 'false'
MessageBoxItem	**Property Name**	**Notes**
	message	
	type	
	response	
	useRegex	'true' if 'message' should be interpreted as a regular expression, false otherwise.
	waitTimeSeconds	
Parameter	**Property Name**	**Notes**
	name	
	value	
	method	

PlayContext	Property Name	Notes
	playPosition	
Request	**Property Name**	**Notes**
	host	
	path	
	postData	
	protocol	
	resource	
	targetFrame	
	label	
	headers	
	multipartBoundary	
	newWindowWidth	
	newWindowHeight	
	formCharset	
	defaultMethod	
	bUseLenientSubrequestCount	
Response	**Property Name**	**Notes**
	timeMs	
	size	
	content	
	totalSize	
	errorFlag	
	errorCode	
	timeStamp	
	logContent	

ResponseCheckItem	Property Name	Notes
	minSeconds	
	maxSeconds	
	minSizeKB	
	maxSizeKB	
	checkSize	
ResponseError	**Property Name**	**Notes**
	description	
	category	
	level	Integer - 0=ERROR, 1=WARNING, 2=INFO
	url	
SaveItem	**Property Name**	**Notes**
	writerClass	
	fileName	
	variableExpression	
	styleSheet	
	exportImages	
	showSavedFile	
	includeResponseContent	
	sendType	
	emailTo	
	emailCC	
	emailSubject	
	appendMode	
	sendAsAttachment	
	frameName	

ScheduleItem	Property Name	Notes
	scriptType	
	scheduleDelayType	
	scriptFile	
	delayHours	
	delayMinutes	
	inheritVariables	
ScreenShot	**Property Name**	**Notes**
	label	
	scalingPercent	
	fileName	
	saveToFile	
	addLabel	
SpiderItem	**Property Name**	**Notes**
	browseLinks	
	clickButtons	
	captureResponses	
	randomPopulate	
	navigationMode	
	targetType	
	targetFrame	
	loopAutomatically	
	randomWalk	
Step	**Property Name**	**Notes**
	name	
	repeatVariable	

	repetitions	
	repetitionCount	Read only - returns current repetition when a looping over a fixed number of iterations.
	repeatType	
	incrementAutomatically	
	incrementAll	
SummaryCheckItem	**Property Name**	**Notes**
	locatorType	
	summaryId	
	summarySelector	
	combineType	
TestItem	**Property Name**	**Notes**
	title	
	template	
Thread Item	**Property Name**	**Notes**
	maxThreads	
	pauseWhileStarting	
	startGradual	
	startIncrement	
	startInteveralSeconds	
	pauseWhileStopping	
	stopGradual	
	stopIncrement	
	stopInteveralSeconds	
	durationType	
	durationTimeSeconds	
	engine	

	maxIterations	
	runningCount	Read-only property, returns number of threads currently in running state.
	stoppedCount	Read-only property, returns number of threads currently in stopped state.
Timer	**Property Name**	**Notes**
	waitMs	
	cascade	
Variable	**Property Name**	**Notes**
	dataSourceId	
	isAutoVariable	
	autoVariableExpression	
Variable Check	**Property Name**	**Notes**
	variableName	The name of the variable that will be checked
	Regex	Regular expression to be matched against variable contents
Variable Setter	**Property Name**	**Notes**
	Regex	
	variableName	
	valueList	
	Regex	
	targetFrame	
	sourceType	Whether to extract content from original page source or from runtime content (only applicable when Values of Regex option is active). Valid values are 0 (original) and 1 (runtime).
	parseVariableReferences	
	isCascading	

	noOverWrite	
	fileName	
	type	Specifies how variable is populated. Values are: • 1 - Fixed Values • 2 - Regex Expression • 3 - Content of File
	parseCSV	It set to true and type is 'Content of File' then the content of the loaded file will be parsed and variables created for each column found in the file. *Only applies when type is set to Content of File option.*
Window Caption Check	**Property Name**	**Notes**
	text	
	searchWindowType	
	useRegex	
	searchWindowName	
	windowClass	
WindowControlItem	**Property Name**	**Notes**
	targetType	0,1,2 for options close Active / Target Window, All Popups or None
	focusType	0,1,2 for options focus Main, Popup or None
	focusWindowName	Name of window to focus, when "Popup" is selected for focusType

49 Badboy Plugins

Badboy supports a simple plugin framework that allows you to make your own extensions to Badboy. Badboy Plugins are very simple and easy to write using only basic knowledge of JavaScript and HTML. Plugins can do the following things:

- Create and display HTML based views that display in the main Badboy window
- Interact with the open browser windows including examining and modifying the DOM and HTML
- Control Badboy, the Script and Variables through Badboy's OLE Automation API

- Create and use any ActiveX controls on the local computer (without the normal security restrictions placed on browsers). This allows almost any interaction with the desktop and native programs on the computer.
- Bind hotkeys to perform JavaScript actions that control Badboy or the browser windows that are open
- Add tools to the Toolbox that are implemented by JavaScript with all the plugin capabilities listed above.
- Retrieve and Set Badboy Preferences

49.1 Installing a Plugin

Plugins are installed in Badboy's installation directory in the "plugins" folder. Each plugin is contained in a directory inside this folder. Installing a plugin is done simply by unzipping the plugin (if necessary) and placing it's folder in the Badboy Plugin directory.

49.2 Structure of Plugins

The only file that a plugin *must* have is a JavaScript file called "init.js" which should be at the top level inside the plugin's folder. When Badboy starts it scans all the plugin folders found inside its plugins directory and executes init.js for each one. The init.js file is executed as regular JavaScript inside the main Badboy window. Plugins that wish to continue running after Badboy has started should add views and keyboard and menu commands inside their init.js script.

Plugins may also include other files in their folder which can be referenced by the plugin later on. A typical example would be to include HTML files to be displayed by the plugin when it opens a Plugin View.

49.3 Plugin Tools

As well as adding views and options to Badboy's toolbars and menus, Plugins can create tools in the Toolbox, to be used in scripts. This is done by calling the "addTool()" function on Badboy's plugin object (see below).

A Badboy tool is implemented as a single JavaScript file. The file need only contain a single function called "execute()" which must return either 'true' or 'false'. The 'true' value indicates that the item is finished playing and the script should continue playing the next item. The 'false' value indicates that the item is not finished and is waiting for some activity to occur. Therefore the script will pause and wait for the item itself to resume play.

Other optional functions and attributes can be defined in the tool's JavaScript file and are listed in the table below:

Operation	Description
tick()	If this function is defined then it will be called asynchronously by Badboy while the item is in scope
scriptTreeLabel	If defined this will be used as the label for the item in the script tree.

49.4 The Badboy Plugin Object

Badboy gives Plugins access to a special "plugin" object that can do more powerful things than the regular JavaScript access to the "badboy" object can. The plugin object can be accessed in one of two ways:

- As a property of the "badboy" object itself. For example, JavaScript could refer to it as "badboy.plugin"
- *Only for the case of JavaScript executing inside a plugin's window* the plugin is available as a property of the window. Hence it can be accessed as "window.plugin" or even just "plugin"

The Plugin object exposes the following functions for plugin scripts to call:

Operation	Description
int addView (String strTitle, String strPlugin, String strHTML)	Adds a view with the given name and loaded with html from the given plugin.
void bindKey (int key, int dwModifiers, String strPlugin, String strCommand)	Binds the given key, specified as an ASCII character code, to the specified plugin and JavaScript command. If the plugin has a window then the JavaScript will be executed in the context of that window. The dwModifiers parameter must be one or more of the following values summed, indicating which additional keys apply to the binding: • 4 - Shift key must be pressed • 8 - Control key must be pressed • 16 - Alt key must be pressed Example: to bind Ctrl-U to 'alert("hello world")': `plugin.bindKey(85,8,"testplugin",'alert("hello world")');`
BSTR BrowseFile (String type, String filter, String initialLocation)	Shows an open/save file dialog to browse for the location of a file. • type - "open" or "save" (affects title, prompting for overwriting) • filter - filter string, leave empty for any file, otherwise supply standard format filter string (see MSDN documentation) • initialLocation - initial directory, leave blank to use default.
Window browser (String windowName)	Return dispatch pointer to requested window. Window may be specified by name or by index in form ":n" where n is index.
Object createObject (String progId)	Creates an instance of the named ActiveX Control.
int addTool(String plugin, String name, String scriptSrc, String iconSrc)	Adds a tool to the toolbox that is executed using the script defined in the given source file. The source file is searched for within the plugin's directory and should define a function named 'execute()' that badboy will call run the action when it is played in the script. The execute() function must return true or false as follows: • true - continue playing the script immediately • false - continue playing after a page has loaded The script may also define a global variable called 'scriptTreeLabel'. If it does so, this variable will be displayed as the label of the item in the script

Operation	Description
	tree.
Object addChild(LONG id, LPCTSTR strType, UINT nIndex)	Creates an instance of the requested item type as a child of the given item with given id in the script and returns it. If nIndex is greater than zero then the child is created at the specified index, otherwise it is created as the last child of the item. Example: to add a new Step to your script as a child of item 234: `plugin.addChild(234,"Step",-1);`
void addToolBarButton(String strPlugin, String strCommand, String strIconSrc, String strDescription, String strOptions)	Adds a button to the toolbar that executes the given JavaScript, using the given image for the button and the given description as a tooltip. The icon should be a 16x15 size non-indexed PNG file. Example: to add a toolbar button that shows an alert box using an icon called 'test.png': `badboy.plugin.addToolBarButton("testplugin","alert(1); ", "test.png","My Button", "");`
String getPreference(String strName)	Attempts to retrieve and return named preference. Example: to find the number of undo levels - `badboy.plugin.getPreference("Undo Levels");`
Int setPreference(String strName, String strValue)	Attempts to set the named preference with the given value. Example: to set the number of undo levels to 25 - `badboy.plugin.setPreference("Undo Levels",25);`
String showPopup(String strPluginName, String fileName)	Displays a popup window loaded from the given filename in the given plugin's directory.
String execScript(Object window, String strScript, String strOptions)	Executes the given script in the context (domain, etc.) of the given window. It is also allowable to pass a JavaScript function object to be executed. strOptions is reserved for future use.

Example 1: A simple Plugin

To make a simple example plugin, perform the following steps:

- Find your Badboy Installation directory and the "plugins" folder inside it. Inside the "plugins" folder, make a new directory called "testplugin".
- Inside the "testplugin" directory, make a new file called "init.js" with a text editor (such as Notepad).
- In the init.js file, place the following JavaScript command:

```
badboy.plugin.addView("testplugin","testplugin","test.html");
```

- In the same directory, add a file called "test.html", and place in it the following HTML:

```
<html>
  <head>
    <style type="text/css">
      * { font-family: arial; }
      body { overflow: auto; }
    </style>
    <script type="text/javascript">
      function init() {
        window.setInterval(update,2000);
      }
      function update() {
        var played = window.external.summary(1,"played");
        document.getElementById('played').innerHTML = "Played: " +
played;
        document.getElementById('linkcount').innerHTML = "Links: " +
window.plugin.browser("").document.links.length;
      }
    </script>
  </head>
  <body onload="init();">
    <h2>Example Plugin</h2>
    <p id="played"> </p>
    <p id="linkcount"> </p>
  </body>
    </html>
```

- Restart Badboy

When you restart Badboy you should find that the plugin adds a new view to your Badboy window which contains information that updates as you play your script and tells you how many items have played and also the number of links on the current page in the main browser window.

You will also see that a new entry has been added to the "View" menu to show the plugin in case the user closes the plugin's view.

Example 2: Adding a tool to the Toolbox

We can extend the previous example to add a tool to the toolbox. The tool will simply set a Badboy variable named "hello" with the value "world":

- Add a file called "mytool.js" to the testplugin directory created in the previous example with the following contents:

```
function execute() { badboy.setVariable("hello","world"); return true;
}
```

- Edit the init.js file created in the previous example and add a new line at the end as follows:

```
badboy.plugin.addTool("testplugin","my tool", "mytool.js","test.png");
```

- Add an image file called 'test.png' to be the icon for the tool in the toolbox. You can copy an example image from the following location: http://www.badboy.com.au/images/test.png. If you use another image then you should make sure that it is exactly the same size and type as this image. Place the image file along side the other files in your plugin folder. Note that the image file name is specified in the addTool() call in the previous step.
- Restart Badboy

After performing these steps you should have a new item in your toolbox called 'my tool'. By dragging it to your script you can make it play as part of your script.

49.5 Appendix: List of Badboy Preferences Accessible via setPreference and getPreference functions

Preference Name			
Custom User Agent	Enable Server Log Capture	Record Passive Navigations	SMTP Server
Diff Capture Output	Force Resynchronization	Register Popups	Start Page
Diff Program	Label Manual Screen Shots	Register Popups	Summary Style Sheet
Disable Response Recording	Legacy Report XML	Results Listener Port	Suppress Bbrand
Document Root	Load Previous	Root Item Type	Suppress JScript Errors
Editor Path	Log Threshold	Server Log File Location	Suppress Popups
Enable Custom User Agent	Logfile Location	Show Statistics In Tree	Undo Levels
Enable Direct Downloads	Play Delay	Show Summary Descriptions	XMLHttpRequest Interception
Enable Logging	Record JavaScript Requests	SMTP Display Name	
Enable Recording on Startup	Record Passive Navigation Timeout	SMTP From Address	

50 Using Badboy's GUI from the Command Line

If you wish, you can run the full graphical version of Badboy from the command line (for example, from a command prompt or in a batch script). To do this, just put the Badboy install directory into your system's PATH environment variable. Then you can run badboy as:

```
badboy
```

If you want to open a script called "myscript.bb" then you can give it as an argument:

```
badboy myscript.bb
```

There are a number of options that you can use to load and run scripts in different ways from the command line.

Command Line Flag	Description
/play	Starts the script playing after opening the document.
/playAll	Plays the entire after opening the document, without stopping at Steps.
/autoExit	Causes Badboy to automatically quit when it reaches the end of the script.
/D <name>=<value>	Defines a variable for the script to use. Example: badboy /D foo=bar myscript.bb
/nosplash	Prevents the splash screen from being shown on startup.

Using these options you can create automated Badboy scripts that can run as part of larger batch processes on your computer, or abbreviate running certain scripts by creating shortcuts on your desktop to run them.