Training Guide for Writing BES Custom Content (Windows)

BigFix, Inc.
Emeryville, CA

Last Modified: June 17, 2005
Version 5.1

Table of Contents

Overview

The purpose of this guide is to equip the reader with the skills needed to create BES Custom Actions, Fixlet® messages, BES tasks, and BES Retrieved Properties for the BigFix Enterprise System (BES). The core elements of all these elements are the BigFix Relevance Language and the BigFix Action Language. This guide outlines the most useful commands from each of these languages and provides examples and exercises to assist with instruction.

In order to get the most out of this guide, the reader should have a working knowledge of the BES Console and the BigFix Development Environment (BDE) if they intend to use it. Since this guide will focus on the relevance and action languages, this information should be applicable to content creation from either the BES Console or BDE.

In order to get the most out of this document, you will need access to the BigFix Query Analyser (QNA), the BES Console, and BDE (if you intend to use it). You can download QNA here:

http://support.bigfix.com/fixlet/documents/qna-5.0.3.0.zip

Note that this document is far from a comprehensive resource on the relevance and action languages. For this, please consult the references listed below.

References

The following documents should serve as valuable resources both while working through this guide and while subsequently beginning to write custom content. They contain more examples, and comprehensive lists of available commands and syntax:

BigFix Windows Inspector Library (http://support.bigfix.com/fixlet/documents/WinInspectors-2004-04-07.pdf)
BigFix Windows Shell Action Commands (http://support.bigfix.com/fixlet/documents/WinActions-2003-05-27b.pdf)
BigFix Windows Relevance Language Quick Reference Guide
BigFix Action Language Quick Reference Guide

The BigFix Windows Inspector Library is a technical document that lists the creation mechanisms and properties of all valid objects in the relevance language. This comprehensive guide is a very valuable resource for writing more difficult Fixlet messages. If you feel like you need a query that isn't mentioned in this guide, or are just curious as to what is available, the Windows Inspector Library is the place to look.

The BigFix Windows Shell Action Commands document contains most of the available actions in the BigFix Action Language. The action section in this document contains most of the same information and is organized in a similar manner.

The help function of BDE gives much information on various relevance commands and actions. It contains many of the tables found in the Windows Inspector Library, along with other useful tips. This is an invaluable resource one may use while writing Fixlet messages.

Also, the Configuration Manager page on the BigFix Support Site contains useful information. It can be found here:

http://support.bigfix.com/fixlet/configmanager.html

Conventions Used in this manual

This document makes use of the following conventions and nomenclature:

Convention	Use
`Mono-space`	A mono-spaced font is used to indicate expressions in the Relevance Language.
bold	Bold is used to denote returned values from relevance expressions
<angle bracket>	Angle brackets are used to indicate an object type. For instance to indicate the creation and usage of a particular object, you might see `default value of` <key of registry> which indicates that a registry key is to follow the default value keyword.

Examples

Square bullets and a mono-spaced font denote examples of Inspectors as used in a Relevance Expression. If you have a color version of this file, these square bullets are also red:

- `exists folder “fonts” of folder “c:\windows”`

Types of Custom Content

What is a Fixlet® message?

The BigFix system is a powerful support technology that allows you to create "Intelligent" messages, which identify problems on a computer, and offer them a single-click solution. Fixlet messages are created using the BigFix Development Environment or the BES Console, and are generally simple and quick to produce. It is useful to think of a Fixlet message as the sum of three constituent parts: Fixlet Relevance, Fixlet Body, and Fixlet Action.

Fixlet Relevance is the "brain" of the Fixlet message that identifies specific problem conditions that the Fixlet describes and fixes. Fixlet Relevance is written in the BigFix Relevance Language, and it allows Fixlet messages to target only computers with specific characteristics that indicate it has a problem. A Fixlet is defined as 'Relevant' for a certain computer if all of its relevance statements return **True** for that specific computer.

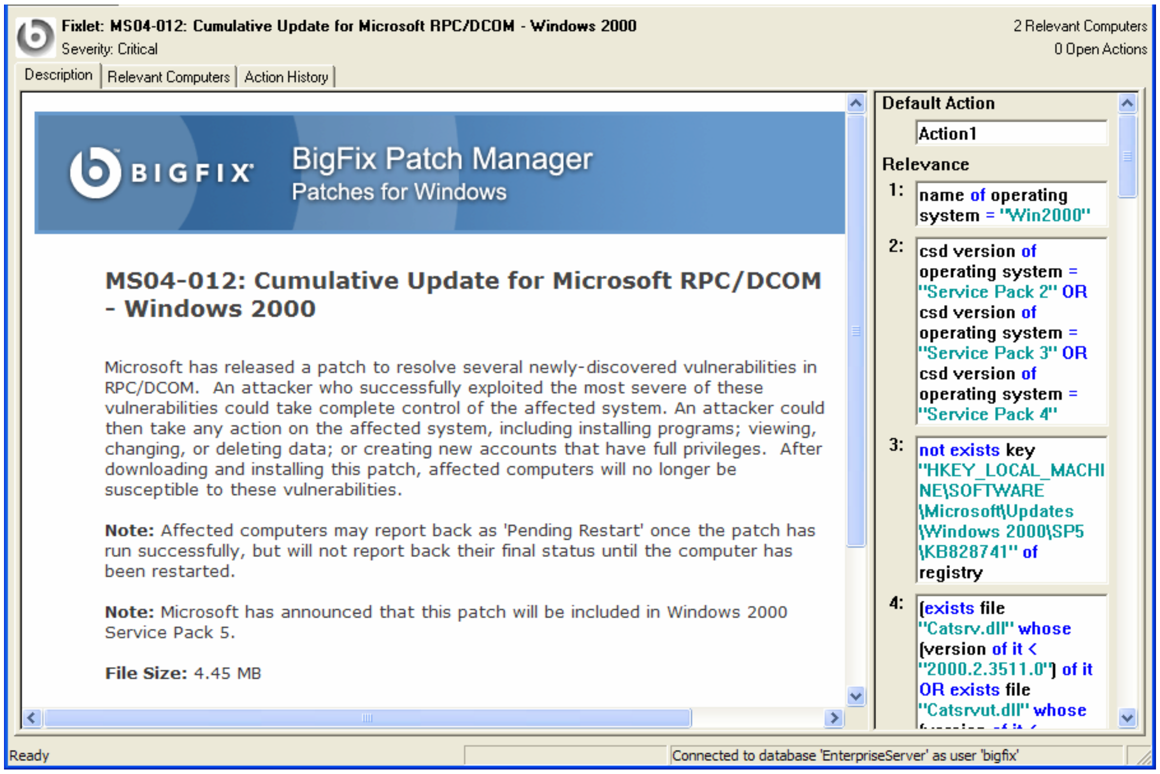
Fixlet Body is the text (which can be formatted using standard HTML) of the Fixlet message, which explains the problem and gives the user any pertinent information they may need. The body should be concise, containing only important information.

Fixlet Action is the process that solves the problem, allowing the user to click on a single link to resolve the issue quickly and simply. A Fixlet action may only be taken on a computer where the Fixlet is relevant.

Fixlet messages Vs. BES Tasks

BES Tasks are essentially the same as Fixlet messages, but they are a different category for organizational purposes. By definition, Fixlet messages are meant to be an identification and remediation of a problem. When the action of a Fixlet message is taken, it should 'solve' the problem. BES Tasks, on the other hand, perform maintenance and management functions that don't necessarily address problems. For instance, BES Tasks can be used to deploy software, restart services, or run backups. BES Tasks have the same makeup as a Fixlet messages (relevance, actions, and body), and they interact with the BES Console and BES Clients in essentially the same manner. The only functional difference is that by default Fixlet actions will report back as 'Fixed' once their relevance has falsified, while by default Task actions will report back as 'Fixed' once all lines of its action have completed. For historical purposes, this guide will generally talk about Fixlets, but all content is applicable to BES Tasks as well.

Here is an example of a Fixlet message as seen by a console operator. The body is the text in the middle, and the two links that say 'Click Here' are actions. The other links and graphics are part of the Fixlet template, can be customized using BDE (template design will not be covered in this guide.)


Fixlet: MS04-012: Cumulative Update for Microsoft RPC/DCOM - Windows 2000
Severity: Critical

2 Relevant Computers
0 Open Actions

MS04-012: Cumulative Update for Microsoft RPC/DCOM - Windows 2000

Microsoft has released a patch to resolve several newly-discovered vulnerabilities in RPC/DCOM. An attacker who successfully exploited the most severe of these vulnerabilities could take complete control of the affected system. An attacker could then take any action on the affected system, including installing programs; viewing, changing, or deleting data; or creating new accounts that have full privileges. After downloading and installing this patch, affected computers will no longer be susceptible to these vulnerabilities.

Note: Affected computers may report back as 'Pending Restart' once the patch has run successfully, but will not report back their final status until the computer has been restarted.

Note: Microsoft has announced that this patch will be included in Windows 2000 Service Pack 5.

File Size: 4.45 MB

Default Action: Action1

Relevance

```
1: name of operating system = "Win2000"
2: csd version of operating system = "Service Pack 2" OR csd version of operating system = "Service Pack 3" OR csd version of operating system = "Service Pack 4"
3: not exists key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Updates\Windows 2000\SP5\KB828741" of registry
4: (exists file "Catsrv.dll" whose (version of it < "2000.2.3511.0") of it OR exists file "Catsrvut.dll" whose
```

Ready   Connected to database 'EnterpriseServer' as user 'bigfix'

BES Retrieved Properties

Retrieved properties are an essential part of the BigFix Enterprise Suite. BES Retrieved Properties are certain pieces of information that are gathered from each BES Client and stored in the database on the BES Server. This information is displayed in the BES Console and can be used to define management rights for BES Console operators and target actions to specific subsets of computers. Retrieved properties are written in the BigFix Relevance Language. Each Retrieved Property is simply a return value of a query written in the BigFix Relevance Language.

BES Custom Actions

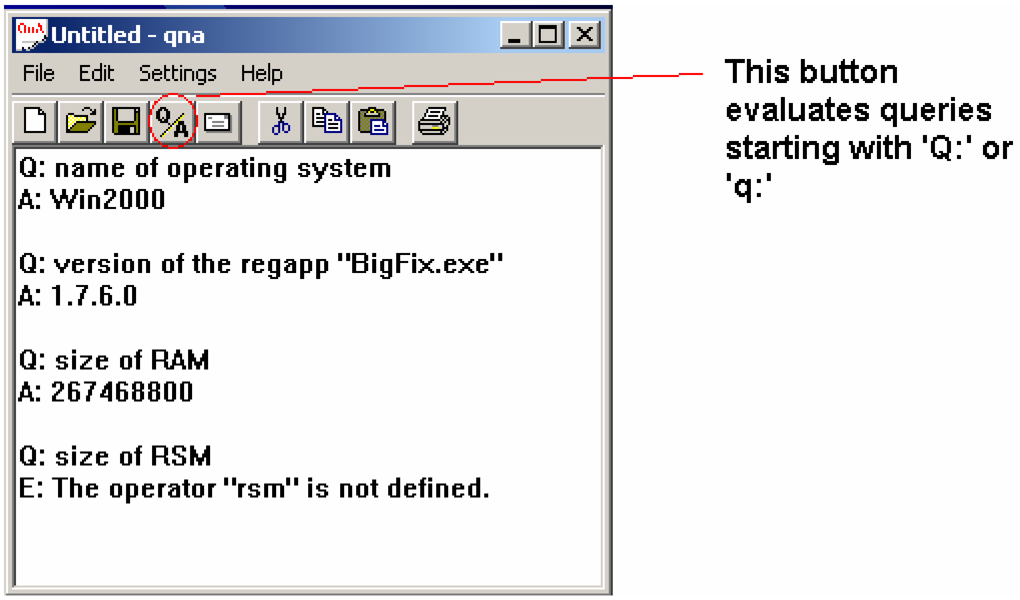
A BES Custom Action contains the main constituent parts of a Fixlet message or BES Task, but is not saved for future application. When you write a BES Custom Action, you write the relevance and action scripts, and immediately send out the action to targeted computers. In general, you should only use a BES Custom Action if you want to perform a quick task that you only need to do once. Otherwise, it would be better to take advantage of the increased functionality of a Fixlet message or BES Task.

Initial Query Analysis

The BigFix Relevance Language forms that basis for most content in BES. BES Retrieved Properties are just evaluations of a relevance query. Whether or not Fixlet® messages or BES Tasks appear relevant in the BES Console is dictated by relevance clauses, which are written in the BigFix Relevance Language. A Fixlet message may contain many relevance clauses, all of which must return **true** for it to become 'relevant' for a certain computer. An expression will only lead to a Fixlet message becoming relevant if it *successfully evaluates* to **True**. A Fixlet action cannot be taken on a machine if any of its relevance clauses are false. This section of the guide will introduce basic relevance queries and techniques. We recommend that the reader use QNA to try out various queries, complete exercises, and get a general feel for the language. The QNA program is also built into BDE as the relevance evaluator. You may access the relevance evaluator by pressing CTRL-E.

The QNA Program

QNA is a simple program that analyzes relevance queries for the computer it is being run on. QNA comes installed with the BDE program. Here is a quick screenshot of the QNA program:



```
Q: name of operating system
A: Win2000

Q: version of the regapp "BigFix.exe"
A: 1.7.6.0

Q: size of RAM
A: 267468800

Q: size of RSM
E: The operator "rsm" is not defined.
```

Use of the QNA program is straightforward. Simply type 'Q:' followed by a relevance clause, and click on the 'Q/A' button for evaluation. The QNA program can evaluate many queries at once, and will ignore all text not preceded by 'Q:'. Here are some useful keyboard shortcuts:

CTRL-A: Evaluate all queries.

CTRL-T: Evaluate all queries and display the time needed to complete each one in microseconds.
CTRL-Q: Remove all query answers.

Although most relevance clauses take only a fraction of a second to evaluate, there are some larger ones that may take a significant amount of time to evaluate. When creating these clauses, it's important to use syntax that will minimize the amount of time needed to evaluate the clause. In many cases, two statements that seem equivalent can have very different evaluation times. Evaluation time can always be checked in QNA by using CTRL-T.

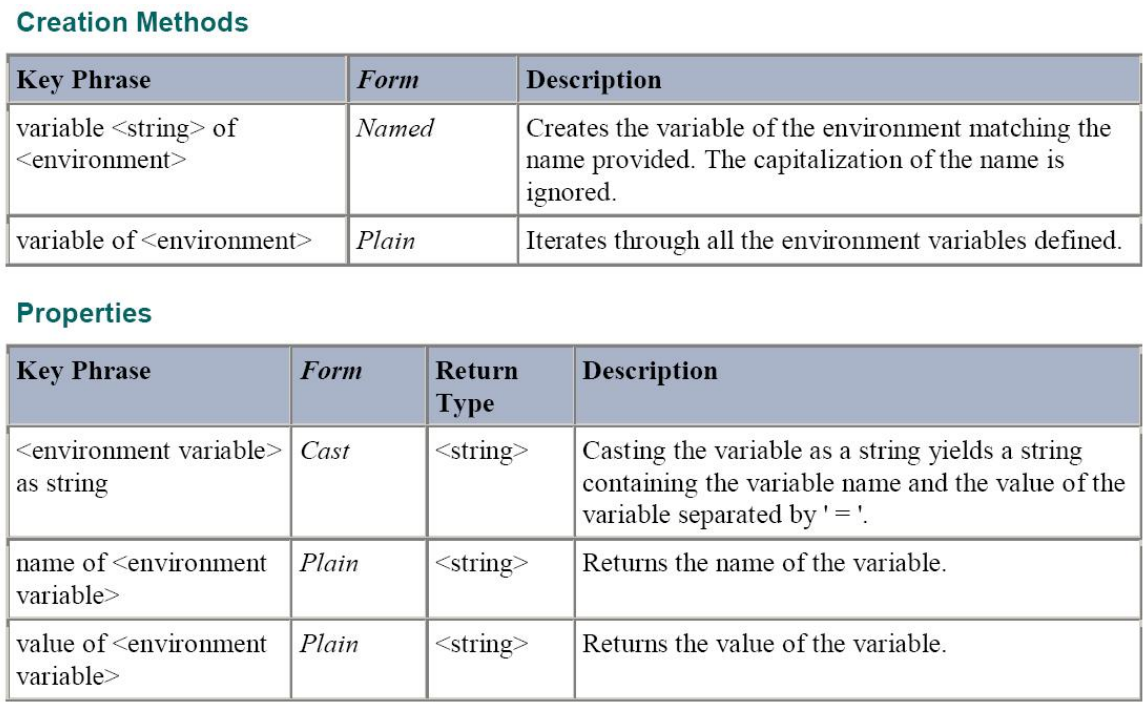
Before we get into specifics, try re-creating some of simple queries above. Get QNA to display the amount of your computer's RAM and the name of its operating system. Also, use CTRL-T to see how long it takes each one of these queries to evaluate.

Besides being a useful training tool, the QNA program will prove invaluable when one is creating a custom content. It's a good way to figure out how relevance can be used to detect a problem, and can help throughout the entire debugging process.

Note! There is a problem with copying text from Microsoft Word into QNA. If you have an electronic version of this document and you try to copy relevance statements into QNA, the quotation marks won't evaluate correctly. You will need to manually erase and retype quotation marks in the QNA program for the evaluation to work.

The Structure of BES Relevance - Object Types

The BES Relevance Language is build upon a multitude of objects and their various properties. Possibly one of the most difficult parts of writing relevance is keeping different objects straight. There are many types of objects (strings, integers, registry objects, versions, etc.), but sometimes it can be difficult to figure out which object type you have and which you need. It is useful to think of the BES Relevance Language in terms of objects and properties. Most queries are properties of an object that produce another object. Since the Windows Inspectors Library is organized according to objects, let's a take a look at a piece of it.



Creation Methods

Key Phrase	*Form*	Description
variable <string> of <environment>	*Named*	Creates the variable of the environment matching the name provided. The capitalization of the name is ignored.
variable of <environment>	*Plain*	Iterates through all the environment variables defined.

Properties

Key Phrase	*Form*	Return Type	Description
<environment variable> as string	*Cast*	<string>	Casting the variable as a string yields a string containing the variable name and the value of the variable separated by ' = '.
name of <environment variable>	*Plain*	<string>	Returns the name of the variable.
value of <environment variable>	*Plain*	<string>	Returns the value of the variable.

This is a snapshot of the part of the Windows Inspector Library that deals with the <environment variable> object. As you can see, it is divided into two sections: Creation Methods and Properties. The ‘Creation Methods’ section tells you all the ways you can produce an <environment variable> object. In order to create an <environment variable> object you form a query of the form:

- `variable <string> of <environment>`

where the <string> is the name of the environment variable, and <environment> is the object corresponding to the environment. You could also get a list of all your environment variables by typing:

- `variables of environment`

If you were unsure how to create an <environment> object, you could look up the section of the Windows Inspectors Library that about the <environment> object, and look at the Creation Methods. As it turns out, you create an <environment> object by just writing `environment`. For instance, you could create an <environment variable> object corresponding to the computer name with the following query:

- `variable “Computername” of environment`

From here you could query the various properties of the <environment variable> object. In order to return the name or value of the environment variable, you would use the following commands, respectively:

- `Value of varaiable “computername” of environment`
- `Name of variable “compuername” of environment`

As you can see from the chart, each of the command returns an object of type <string>. As I introduce the various commands, I’ll try to make a note of what object we’re dealing with. As I mentioned before, a comprehensive list of objects in the BES Relevance Language can be found in the BigFix Windows Inspectors Library. Also, the inspector library lists the object type of the parameter and return value for every command. The inspector library may be extremely useful if you find yourself having difficulties with object types.

TIP! Relevance expressions often fail because you pass the wrong object type to a command as a parameter. A great troubleshooting mechanism for relevance is to look at each command within a relevance clause and make certain that each command is getting passed the correct object type.

Basic Booleans

Many different types of information about a computer can be garnered from a relevance expression. Since for Fixlet messages we eventually want a complete relevance clause to return a Boolean **True** or **False**, let’s first look at operators that return a Boolean value. Although there are a few inspectors that return a Boolean value, there are two main ways to obtain a Boolean value: existence and comparison. You will use one of these tools in most relevance clauses.

Existence

In its most basic form, a relevance expression can check for the existence of an object on the users’ computer. Let’s say that you to write a Fixlet that would be relevant on all computers with “autoexec.bat” located at the root directory of the “c:” drive. The following would be the relevance statement:

- **`exists`** `file "c:\autoexec.bat"`

In this case `file "C:\autoexec.bat"` defines a <file> object. (We’ll go over the <file> object in more detail shortly.) The result of the `exists` operator is a boolean value. In this case, it would return **true** if the file exists or **false** if it doesn't. Therefore, a Fixlet message with the example relevance clause would cause all computers with the file “autoexec.bat” at the root of their “c:” drive to report as relevant. You can also check for a folder using similar syntax:

- `Exists folder “c:\temp”`

Practice with QNA:

Using QNA, try checking for the existence of various files and folders on your computer. Later, we'll learn other objects we can check, and we'll learn tricks to get around typing the complete pathname of a file. Can you answer the following questions about your computer using QNA? (**Note:** Answers to all the practice problems are available at the end of each section.)

Practice Using QNA:

1. Does the folder "c:\winnt" exist on your computer?
2. Does this folder contain "explorer.exe"?

Link to Answers

Comparison

The second basic form of a Relevance clause involves the comparison of two expressions:

Here's an example of this form using an equal sign as the comparator:

- `number of processors = 2`

The result is **true** only if both expressions are equal to each other. If they are not equal -- or if there is some kind of error -- the expression will be **false**. Here's an example of comparing two relevance expressions using a 'greater than' sign:

- `Version of file "c:\temp\prog1.exe" > version of file "c:\temp\prog2.exe"`

We will explore the `version` command a bit later.

Here is a list of all of the comparators that can be used in the Relevance Language, with examples of how they can be used:

=	5/2 = 2.5
!=	5/2 != 3
<	5 < 6
>	6 > 5
<=	5 <= 6
>=	6 >= 6

contains	"xyz" contains "y"
does not contain	"xyz" does not contain "bc"
does not end with	"xyz" does not end with "abc"
does not equal	"abc" does not equal "Abc"
does not start with	"abc" does not start with "bc"
ends with	"abc" ends with "bc"
equals	5 = 4+1
is	1 is 1
is contained by	"y" is contained by "xyz"
is equal to	1 is equal to 1
is greater than	2 is greater than 1
is greater than or equal to	"banana" is greater than or equal to "apple"
is less than	4 is less than 5
is less than or equal to	5 is less than or equal to 5
is not	1 is not 0
is not contained by	"fuz" is not contained by "foo"
is not equal to	"foo" is not equal to "bar"
is not greater than	"foo" is not greater than "bar"
is not greater than or equal to	5 is not greater than or equal 4
is not less than	5 is not less than 3
is not less than or equal to	5 is not less than or equal to 5
starts with	"good" starts with "goo"

Operators

The examples in the comparator list above represent simple relevance clauses. However, larger Relevance clauses can be more complicated, describing very specific computer configurations and processes. When dealing with these more complex Relevance clauses, it is often useful to break them down into smaller expressions. In some cases, more complex expressions can themselves be broken down into simple "sub-expressions". Each of these expressions and sub-expressions must be linked together with "operators". Operators tell the Relevance Language how to relate two expressions or sub-expressions.

For example:

- `1 + 1`
- `length of file "c:\autoexec.bat" / 512`
- `number of files of system folder mod 10`

Here is a list of all of the operators in the Relevance Language:

(	*begin sub-expression*
)	*end sub-expression*
*****	*integer multiply*
/	*integer divide*
+	*integer plus*
-	*integer minus*
&	*string concatenate*
mod	*integer modulo*
not	*boolean negate*
and	*logical and*
or	*logical or*

File System Objects

Using the Relevance Language, a multitude of information can be inspected from a computer. A complete list of inspectors would take up many pages, so instead I'll try to introduce some of the more common inspectors and their uses. For a detailed list of every inspector, please refer to the Windows Inspector Library.

The Folder Object

The <folder> object refers to a folder in the windows operating system. The most common creation methods are:

Creation Methods

Key Phrase	***Form***	**Description**
parent folder of <file>	*Plain*	Creates a folder object of the parent folder of the file.
parent folder of <folder>	*Plain*	Creates a folder object of the parent folder. The parent folder of a root folder does not exist.
folder <string>	*NamedGlobal*	Creates a folder object for the named folder. This is a global property.
system folder	*PlainGlobal*	Creates a folder object of the system folder. This is operating system dependent. Under Win98 it is usually c:\windows\system.
windows folder	*PlainGlobal*	Creates a folder object of the windows folder. This is operating system dependent. Under Win98 this is usually c:\windows.

Here are some syntax examples:

- `Exists folder “C:\Program Files\BigFix”`
- `Exists parent folder of system folder`

The `windows folder` and `system folder` commands are special queries that return a <folder> object corresponding to the windows or system folder. Since these folders are located in different locations for different operating systems, these commands are very useful.

An important note on the <folder> object is that is has no default return value. Therefore, if you construct a query that returns a <folder>, the query will return an error. Instead of returning a <folder>, you have to return a property of a folder. Here are a few common properties of a <folder>.

Properties

Key Phrase	***Form***	**Return Type**	**Description**
parent folder of <folder>	*Plain*	<folder>	Returns a folder object of the parent folder. The parent folder of a root folder does not exist.
pathname of <folder>	*Plain*	<string>	Returns the full pathname of the folder object as a string.
name of <folder>	*Plain*	<string>	Returns the last component of the folder name as a string.
folder <string> of <folder>	*Named*	<folder>	Returns a folder object for the named sub-folder. Trailing slashes should be omitted from the name.

As you can see, `parent folder of <folder>` and `folder <string> of folder` both return <folder> objects, so they are creation methods as well. Syntax examples:

- `Name of windows folder = “WINDOWS”`
- `Pathname of parent folder of system folder = “C:\WINDOWS”`

The File Object

Possibly the most basic object in the relevance language is the <file> object. The most common creation methods for a <file> are:

Creation Methods

Key Phrase	***Form***	**Description**
file <string>	*NamedGlobal*	Creates a file object for the name provided.
file <string> of <folder>	*Named*	Creates the file objects corresponding to the named file within the folder.

In the first example, the <string> is the complete path of the file. For instance,

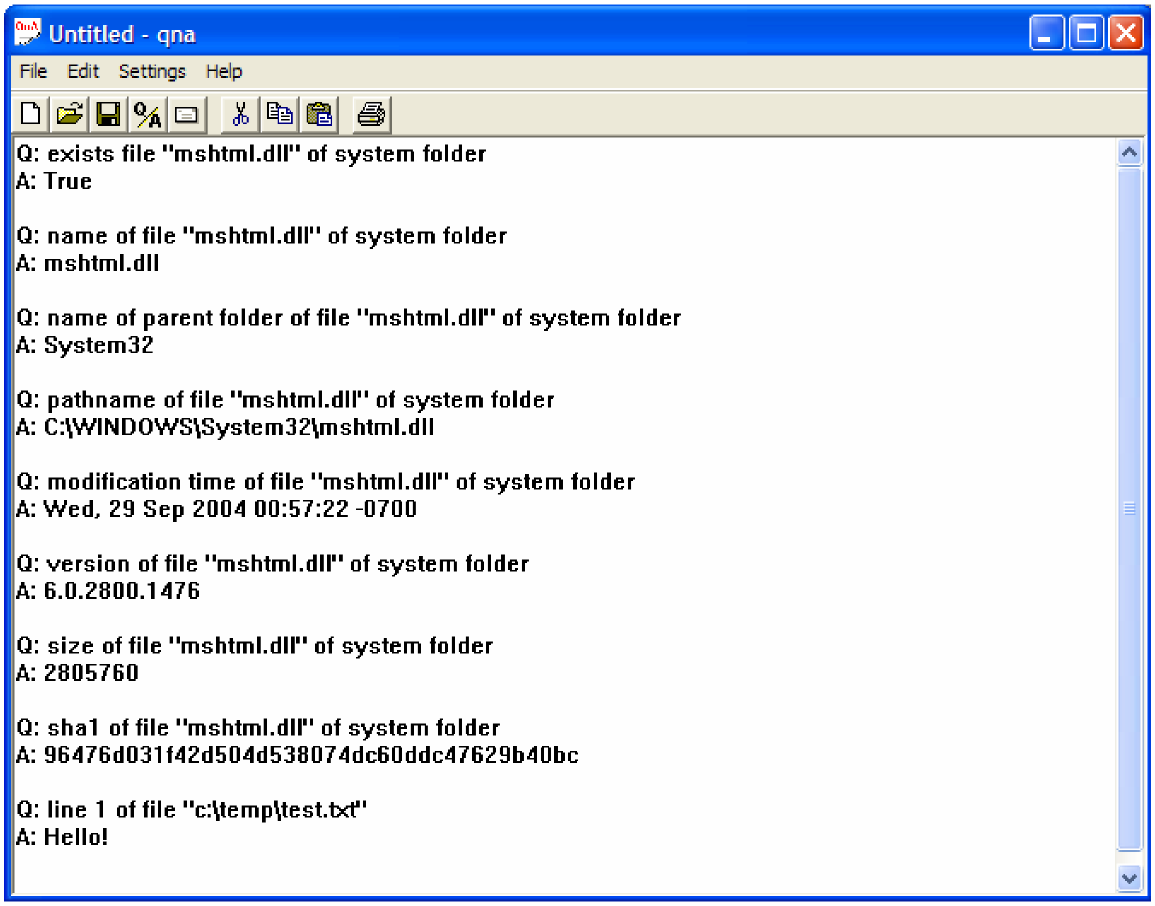
- `Exists File “c:\windows\notepad.exe”`

Since files and folders are not always in the same place on every computer, and it would be difficult to always write relevance that refers to a file that you want to analyze by a specific pathname. Also, you cannot find a file of an unknown location by simply searching through the file system, as this would be an incredible drag on system resources of the computer running the BES Client. Fortunately, while the main ways of creating a <file> object are to use the path of the file or its containing folder, you can generally find this information without needing to hard-code paths. Also, there are special commands for applications that assist with locating many files. We’ll take a look at these shortly. First, however, let’s examine some properties of a <file>.

Properties

Key Phrase	***Form***	**Return Type**	**Description**
name of <file>	*Plain*	<string>	Returns a string that is the last component of the full path.
parent folder of <file>	*Plain*	<folder>	Returns the folder object of the file. This is the directory of the file. See folder.
pathname of <file>	*Plain*	<string>	Returns a string that is the full path name of the file.
modification time of <file>	*Plain*	<time>	Returns the time the file was last modified. See time.
version of <file>	*Plain*	<version>	Synonym for file version of <file>.
size of <file>	*Plain*	<integer>	Returns the size in bytes of a file.
sha1 of <file>	*Plain*	<string>	Returns the sha1 checksum of the file hex encoded as a 40 character long string.
line <integer> of <file>	*Numbered*	<string>	Returns the nth line (specified by <integer>) from the given file.

Here are examples of each of these inspectors, along with some sample return values from QNA:



```
Q: exists file "mshtml.dll" of system folder
A: True

Q: name of file "mshtml.dll" of system folder
A: mshtml.dll

Q: name of parent folder of file "mshtml.dll" of system folder
A: System32

Q: pathname of file "mshtml.dll" of system folder
A: C:\WINDOWS\System32\mshtml.dll

Q: modification time of file "mshtml.dll" of system folder
A: Wed, 29 Sep 2004 00:57:22 -0700

Q: version of file "mshtml.dll" of system folder
A: 6.0.2800.1476

Q: size of file "mshtml.dll" of system folder
A: 2805760

Q: sha1 of file "mshtml.dll" of system folder
A: 96476d031f42d504d538074dc60ddc47629b40bc

Q: line 1 of file "c:\temp\test.txt"
A: Hello!
```

As you can see, the phrase `file "mshtml.dll" of system folder` defines a <file> object, from which various properties are queried. The `sha1` inspector stands for 'Secure Hash Algorithm', and it is a simple checksum that is used to verify the integrity of downloaded files. You will need to use this along with the `size` inspector to create downloads in a Fixlet action.

Practice with QNA:

3. What is the version of the file "tcpip.sys"? This file is located in the folder "Drivers", which is in the system folder.
4. Write a relevance statement that will only return true if the size of the file "url.dll" is 106496 and its sha1 is b53aadb418dfb956e7aebef60d924ef990f919e6. Url.dll is found in the system folder. Since a sha1 hash is a string, it must be entered in quotes in a relevance statement.

Link to Answers

Plurals

Most objects can be made plural by adding an 's' at the end of them. If a plural query returns multiple results, it will return a list with all the possible return values. Later on, we will look at how to filter a list. Syntax Examples:

- `Files of windows folder`
- `Names of files of system folder`

You can query the number of items in a list using the `number of` command.

- `Number of folders of system folder`

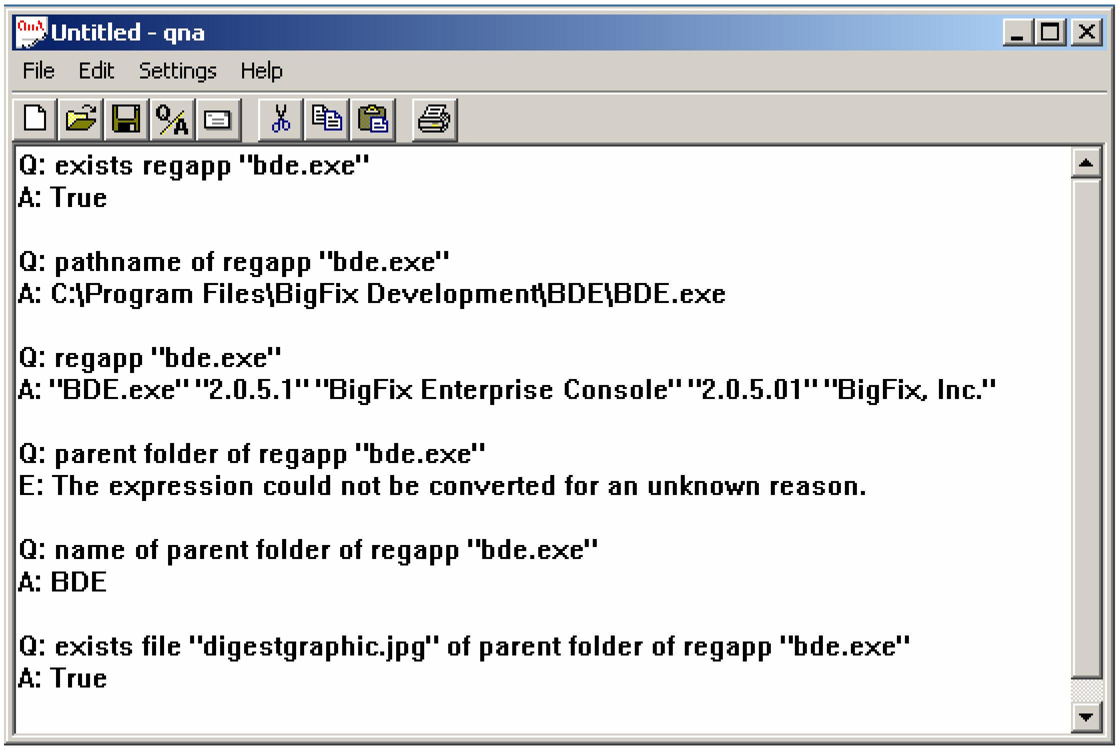
Applications

The relevance language can access certain applications without knowing the path to their executable. An <application> object inherits all the characteristics of the <file> object for the application's executable. For all practical purposes, you can think of an <application> as simply being a nifty way to access the executable file for a program.
The three main types of applications recognized by the relevance language are registered applications (regapps), running applications, and recent applications.

Programs that are installed correctly according to windows guidelines create a key under the key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths. Applications that have a key entry here can be accessed using the `regapp` command:

- `Version of regapp “BesClient.exe”`
- `Pathname of regapp “outlook.exe”`

Here are some examples of the regapp command.



```
Q: exists regapp "bde.exe"
A: True

Q: pathname of regapp "bde.exe"
A: C:\Program Files\BigFix Development\BDE\BDE.exe

Q: regapp "bde.exe"
A: "BDE.exe" "2.0.5.1" "BigFix Enterprise Console" "2.0.5.01" "BigFix, Inc."

Q: parent folder of regapp "bde.exe"
E: The expression could not be converted for an unknown reason.

Q: name of parent folder of regapp "bde.exe"
A: BDE

Q: exists file "digestgraphic.jpg" of parent folder of regapp "bde.exe"
A: True
```

As I mentioned before, the `parent folder` command errors out if used by itself because it returns an object type of “folder”, which has no specified display value. Instead, you have to return a certain characteristic of the folder by using `name of parent folder` or `pathname of parent folder`, etc.

You can use all the same commands for running or recent applications. You can see a list of all your regapps, running, or recent applications by using the following commands:

- `Regapps`
- `Running Applications`
- `Recent Applications`

Note! The running and recent applications inspectors do not quite work correctly in QNA. This list of running and recent applications does not always get updated while running a session of QNA, so you have to close and re-open the program to get an updated list of running applications.

Practice with QNA:

5. Do you have more files in your windows folder than you have recent applications?
6. List all the pathnames of your running applications.
7. List all your regapps. Pick one and display its pathname and parent folder. (Two separate clauses)

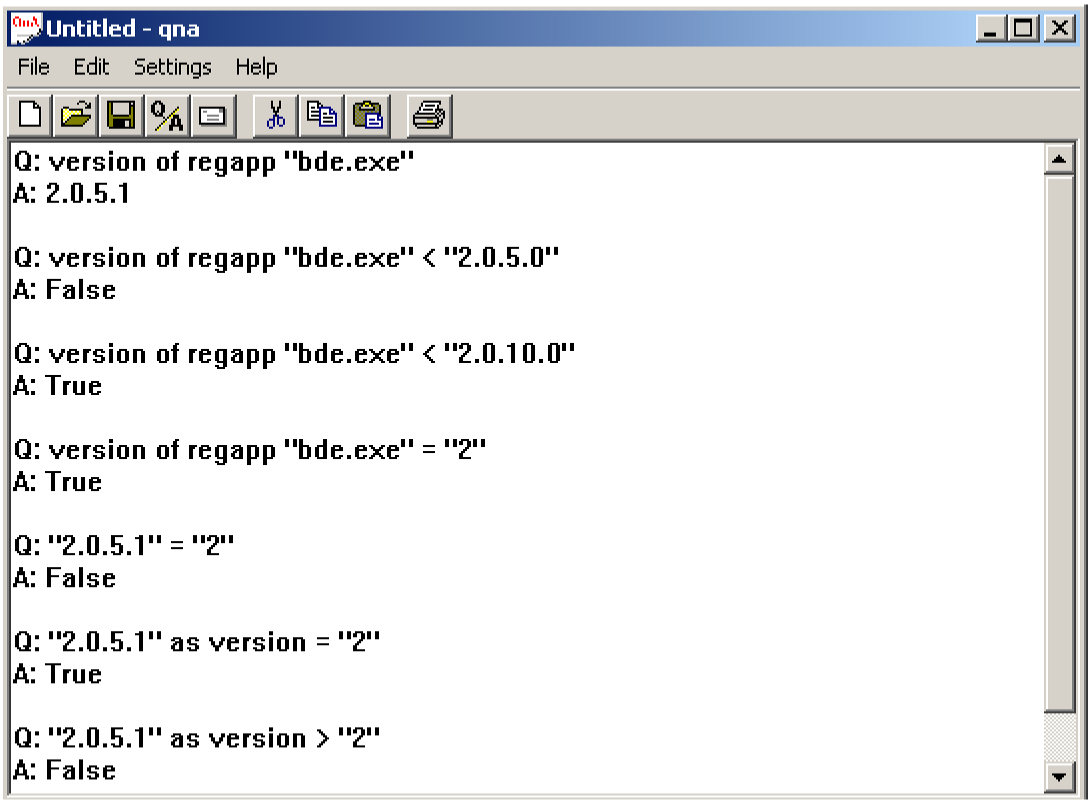
Link to Answers

Versions

As we've seen before, a version is a property of a file. Querying the version of a file returns a <version> object.

- `version of file “cmd.exe” of system folder`
- `version of regapp “bde.exe” > “2.0”`

The <version> object has a few special properties, and since its so commonly used it's worth going into it in some detail. A version consists of up to four integers separated by periods, such as **5.01.235.435**. When comparing versions, the Relevance Language compares the four integers one by one, from the left to the right. It considers two versions as equal if all available integers are equal. For instance, “5.1” < “5.10”, since 5=5 but 1<10. This comparison is somewhat different than if we were dealing with similarly formed strings. This is all a bit easier to explain by example:



```
Q: version of regapp "bde.exe"
A: 2.0.5.1

Q: version of regapp "bde.exe" < "2.0.5.0"
A: False

Q: version of regapp "bde.exe" < "2.0.10.0"
A: True

Q: version of regapp "bde.exe" = "2"
A: True

Q: "2.0.5.1" = "2"
A: False

Q: "2.0.5.1" as version = "2"
A: True

Q: "2.0.5.1" as version > "2"
A: False
```

As you can see, the comparisons work a bit differently if you are working with strings or versions. The main difference is that you can just check the major version of a file by comparing it to a single integer. You can cast a string into a version by using the `as version` command, and vice versa.

One more important thing to note about versions is that the `version` inspector doesn't always return exactly the same version as you would see by right-clicking a file and selecting 'properties'. Usually, the difference is something seemingly trivial such as trailing zeroes, but this can make a big difference when comparing versions. Whenever possible, check a file version using QNA to make sure you know the version as interpreted by the Relevance Language.

TIP! Not all files have versions. You can check with the query `exists version of file <filename>`

Practice With QNA:

8. What is the version of your "regedit.exe" file?
9. What is your version of Internet Explorer? (Hint, it's a regapp)
10. Formulate a relevance expression to check whether one has IE5, and another one to see if they have IE6?
11. Write a relevance expression to check the version of the file "msoe.dll", which is a file used by Outlook Express. This file will be in the same folder as the regapp "msimn.exe". Hint: use `parent folder of regapp`

Link to Answers

Operating System

You can create an <operating system> object corresponding to the local OS by simply writing `operating system`. The two most common expressions involving the <operating system> object are:

- `Name of operating system`
- `Csd version of operating system`

`Name of operating system` returns a string based on the operating system. Possible return values for this query on windows operating systems are **WinNT, Win95, Win98, WinME, Win2000, WinXP, Win2003**.

`Csd version of operating system` returns a string containing the service pack of the operating system. In general, the return value of `csd version of operating system` returns a string of the form of **Service Pack X** where **X** stands for the number of the service pack. Here are some syntax examples:

- `csd version of operating system = "Service Pack 1"`
- `csd version of operating system = ""`

If there is no service pack, the `csd version` is just an empty string, **""**.

Practice with QNA:

12. What service pack are you running?
13. How would you check to see if a computer had any service pack installed?

Link to Answers

Strings

So far we're seen many objects that either require strings for their creation or output their properties as string, so we may as well address them directly. A string is simply a text expression, and it is entered into a relevance statement surrounded by quotes.

Since strings are so prevalent, there are many commands that are used to extract specific information from a string. When you want to use these commands, it is important to make sure you're dealing with a string. More complicated expressions also make use of the <substring> object, which is useful for parsing strings. Other types of objects can be converted to strings using the `as string` command. Here are some useful commands:

- `First/Last <integer> of <string>`

Returns a substring of the first/last <integer> of characters of the string

- `<string1> (contains/starts with/ends with) <string2>`

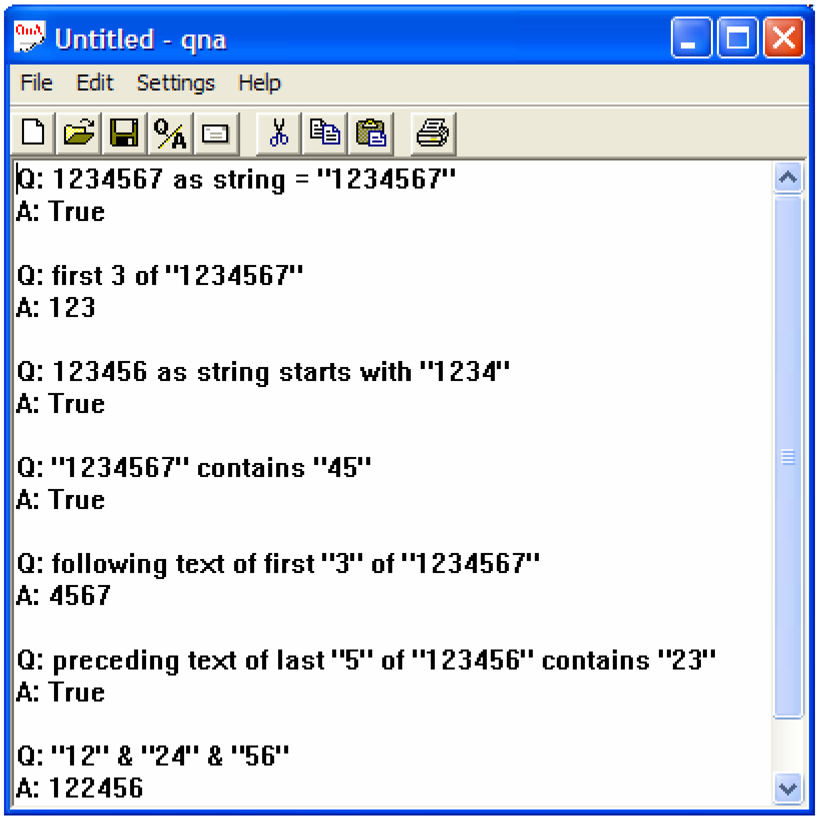
This will return a boolean value based on whether or not <string1> contains/starts with/ends with <string2>.

- `following text of <substring> of <string>`

Returns <substring> corresponding to the part of the original string that follows the named substring. For more information on using substrings, please see the Inspector Library.

Also, you can concatenate a string using the & operator.

The following QNA example should make these commands a bit more clear:


Untitled - qna

File Edit Settings Help

```
Q: 1234567 as string = "1234567"
A: True

Q: first 3 of "1234567"
A: 123

Q: 123456 as string starts with "1234"
A: True

Q: "1234567" contains "45"
A: True

Q: following text of first "3" of "1234567"
A: 4567

Q: preceding text of last "5" of "123456" contains "23"
A: True

Q: "12" & "24" & "56"
A: 122456
```

It can take a little practice to learn how to extract the correct information from a string, but eventually it becomes second nature.

Practice With QNA:

14. Write a relevance statement that will return true on all Windows operating systems.
15. Write a relevance expression that will output just the number of the installed service pack.
16. Create another relevance statement that will output the version of msoe.dll. This time, however, don’t use the `parent folder of regapp` command. Instead, create the full path to the file direct by parsing the pathname of the regapp msimn.exe and concatenating the correct filename onto the end.

Link to Answers

Registry Commands

When writing relevance, the Windows registry can be a gold mine of information. The registry is a database contained in every Windows operating system that stores data about a computer. The registry is organized into a tree of folders called *keys*. A key can have many subkeys, which can have many subkeys, etc. Each key can contain information in the form of *values*. You can see or edit the registry by running the regedit.exe program, which is included with Windows. You can run it by going to Start>Run> and type "regedit".

Note! Be careful when working in regedit or using .reg files. A wrong move could damage your computer irreparably.

Creation Methods

Key Phrase	***Form***	**Description**
key <string> of <registry key>	*Named*	Creates an object for the named sub-key of the key.
key <string> of <registry>	*Named*	Creates an object for the named key. The name may be a full path to a key of the form "HKEY_CLASSES_ROOT\Fixlet.Pool\".

All registry commands must originate from a <registry> object. Fortunately, a <registry> object is created by simply writing `registry`.

Properties

Key Phrase	***Form***	**Return Type**	**Description**
key <string> of <registry key>	*Named*	<registry key>	Returns a key for the named sub-key.
name of <registry key>	*Plain*	<string>	Returns the name of the key as a string.
value <string> of <registry key>	*Named*	<registry key value>	Returns the named value stored under the key. See registry key value.

Like the <folder> object, the <registry key> object has no default return value. Therefore, a registry command that returns a <registry key> object will error out. Instead, you need to query the name of the key or a value in the key. Here are some syntax examples.

- `exists key “HKEY_LOCAL_MACHINE\SOFTWARE\BigFix\Enterprise Server” of registry`

- `value "IsInstalled" of key "HKLM\SOFTWARE\Microsoft\Updates\DataAccess\Q318203" of registry = 1`

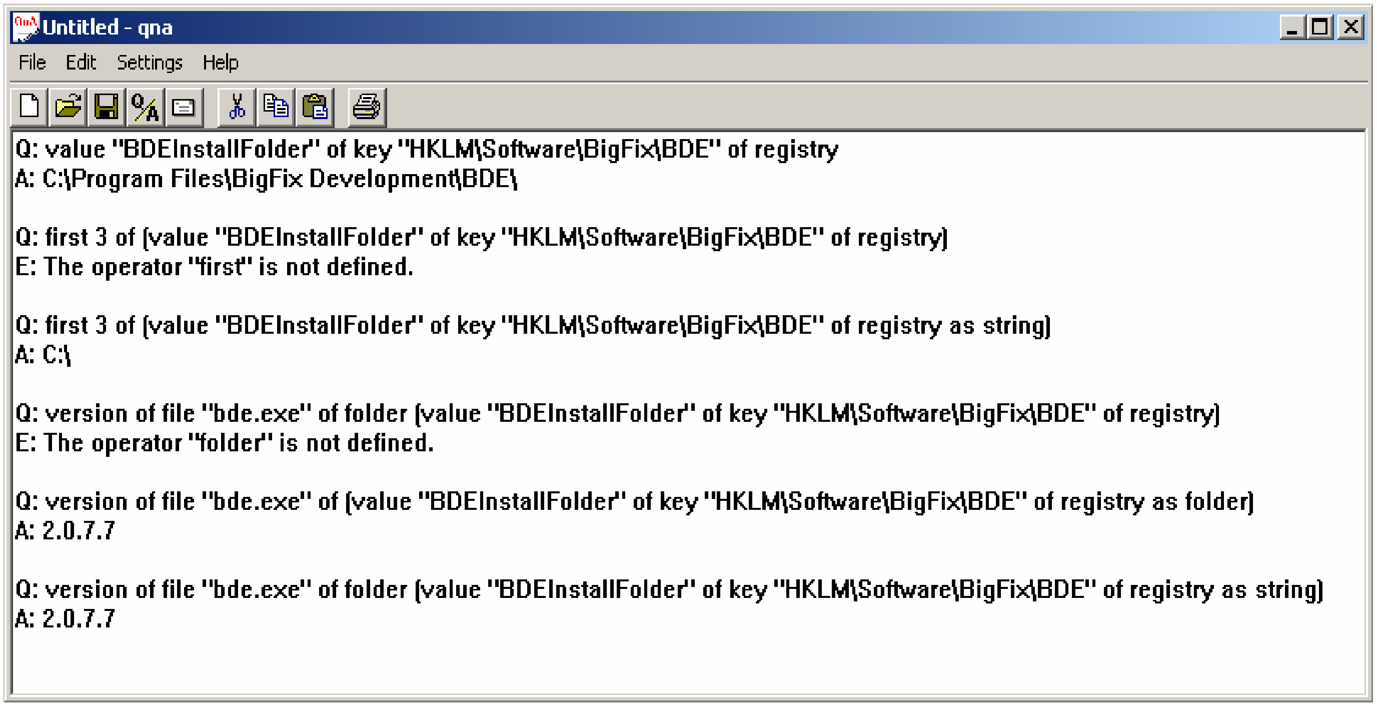
Basically, you just need to keep two things in mind to avoid making syntactical mistakes while forming registry queries.

1. All values and key names must be surrounded by quotes.
2. It's necessary to write `of registry` after the key name.

As you may have noticed from the examples, you can abbreviate the hives of the Registry. Here are the abbreviations:

HKEY_Classes_Root = HKCR
HKEY_Current_User = HKCU
HKEY_Local_Machine = HKLM
HKEY_Users = HKU
HKEY_Current _Config = HKCC

Tip: Values from the registry are pre-defined as registry objects. Basically, this just means that, even if they look like a string, folder, or time, you have to cast the value into the type you want. You can do this by using the commands `as time`, `as folder`, and `as string`. Take a look at this QNA example:



```
Q: value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry
A: C:\Program Files\BigFix Development\BDE\

Q: first 3 of (value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry)
E: The operator "first" is not defined.

Q: first 3 of (value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry as string)
A: C:\

Q: version of file "bde.exe" of folder (value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry)
E: The operator "folder" is not defined.

Q: version of file "bde.exe" of (value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry as folder)
A: 2.0.7.7

Q: version of file "bde.exe" of folder (value "BDEInstallFolder" of key "HKLM\Software\BigFix\BDE" of registry as string)
A: 2.0.7.7
```

As you can see, in order to perform a string operation (`first 3 of`) on a registry value, I had to use the `as string` command. In order to look for a file within the folder, I could either cast the value as a string and use the `folder` command (which expects a string as its argument), or I could simply cast it as a folder. One exception to this rule is Reg_Dword values, which return as integers.

Note! The BigFix Enterprise client runs its queries as the System user. Therefore, a Fixlet message inspecting HKCU will report values for the system user, not the currently logged in user.

Practice with QNA:

17. Write a command that will return true if a user has ever installed a certain Internet Explorer patch on their computer. The following key will get created once the patch in installed: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components\{2298d453-bcae-4519-bf33-1cbf3faf1524}
18. Now check to see if the patch is actually installed at the moment. If it is, it will have a Dword value of "IsInstalled" set to 1.
19. Find the version of wab32.dll, a component of the Windows Address Book. You can find the complete path to this file by checking out the following key- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WAB\DLLPath. Hint: You can query the default value of a key using `default value of <registry key>.`

Link to Answers

Time

There are a number of times that can be analyzed using the Relevance Language. The simplest command is the `now` command, which just returns the current time according to the operating system's clock.

Q: `now`
A: **Mon, 03 Feb 2003 16:30:08 –0800**

Time is displayed in the following format (MIME standard):

ddd, DD mmm YYYY HH:MM:SS sZZZZ

d = Day of the week
D = Day of the month
m = Name of the Month
Y = Year
H = Hour
M = Minute
S = Second
sZZZZ corresponds to the time zone. For instance , -0800 corresponds to 8 hours before Greenwich Mean Time. (US Pacific Time)

Here are a few inspectors that return a time:

- `Modification time of file`
- `Creation time of file`
- `Accessed time of file`
- `Boot time of operating system`

The `modification time` inspector is probably the most useful, as it can be used to compare two different files that either have the same version or no version. When two times are compared, the greater time is defined as being more recent. The sum or difference of two times gives an answer in days, hours, minutes, and seconds. (of the form dd:hh:mm:ss.sss) Here are some examples:

- `modification time of regapp "NsRex.exe" < "Thu, 01 Jun 2000 00:00:00 -0800" as time`
- `modification time of file “file.dll” of system folder > creation time of “file.dll of system folder`
- ` now – modification time of file “c:\bigfix\subscribe.dll” > 3*day`

One can compare the difference of two times to days, hours, or seconds. As shown above, this data type is constructed by `<integer>*<time interval>`. Available time intervals are microsecond, millisecond, second, minute, hour, day, week. Also, if you want to type a time into a relevance clause (or extract one from the registry), as in the first example, you need to cast it as a time by writing `as time`.

Services

The Relevance Language can query installed services on a computer.

Creation Methods

Key Phrase	***Form***	**Description**
running service	*PlainGlobal*	Creates objects corresponding to all the running services.
running service <string>	*NamedGlobal*	Creates the running service object for the specified name.
service	*PlainGlobal*	Creates objects for all the services.
service <string>	*NamedGlobal*	Creates the service object matching the specified

Key Phrase	*Form*	Description
		name, regardless of its running state.

You can see a list of services or running services by evaluating the queries `services` and `running services` respectively. Here are some examples:

- `Exists service “BesClient”`
- `Exists running service “Messenger”`

Properties

Key Phrase	*Form*	Return Type	Description
display name of <service>	*Plain*	<string>	Returns the display name of the service.
login account of <service>	*Plain*	<string>	Returns the login account under which the service is configured to run.
state of <service>	*Plain*	<string>	Returns one of Continuing, Pausing, Paused, Running, Starting, Stopping, Stopped, Unknown.

Here are some more examples:

- `State of service “DCHP” = “Running”`
- `Login account of service “Messenger”`

Note: You can gain additional information about services and make changes in their settings using the registry keys located at:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\

Practice with QNA:

20. What time is it now?
21. How long has it been since explorer.exe has been modified?
22. Has it been more than a day since the computer has been started. Hint: `boot time of operating system` gives the time of last boot.
23. What is the state of the Task Scheduler service on your computer? (Note, 'Task Scheduler' is the Display Name, not the service name.)

Link to Answers

Hardware

There are a number of inspectors that investigate the hardware of a machine. Since these are generally used less frequently than the software inspectors, I'll just go over few of the more common examples.

RAM

You can return the amount of RAM on your computer using the command:

- `Size of ram`

This command returns an <integer> corresponding the amount of Ram in bytes. If you want to return the amount of Ram in MB, you can divide by (1024*1024).

- `Size of ram / (1024*1024)`

Processor

The <processor> object refers a processor on the computer.

Creation Methods

Key Phrase	***Form***	**Description**
main processor	*PlainGlobal*	Creates the object associated with the 'Primary' processor. On most systems this is the only processor.
processor	*PlainGlobal*	Iterates through the processors in the system.
processor <integer>	*NumberedGlobal*	Creates the processor object for the number specified. The first processor is processor number 1.

On single-processor machines, the easiest way to create a <processor> object is simply to write `processor`.

Properties

Key Phrase	***Form***	**Return Type**	**Description**
family name of <processor>	*Plain*	<string>	Returns the family name of the CPU, e.g., • Athlon, • Duron, • Pentium 4.
vendor name of <processor>	*Plain*	<string>	The manufacturer of the CPU. Names include: • GenuineIntel, • AuthenticAMD, • CyrixInstead,
speed of <processor>	*Plain*	<hertz>	Returns the speed of the processor in Hertz. This inspector measures the speed if the vendor name is GenuineIntel.

Here are some syntax examples:

- `vendor name of processor = “GenuineIntel”`
- `Family name of main processor = “Pentium M”`

It is important to notice that the speed of <processor> command returns not an <integer>, but a <hertz> object. The <hertz> object is especially formulated for easy conversion to Mhz, Khz, or Ghz. For instance, if

- `Speed of processor`

Returns **600000000 hertz**, then

- `Speed of processor / Mhz`

Will return **600.**

Network Adapter

You can create a <network adapter> object by writing `adapter of network`. Since there are frequently multiple network adapters present on computers, you may want to make this statement plural to see all the adapters:

- `Adapters of network`

Properties

Key Phrase	***Form***	**Return Type**	**Description**
address of <network adapter> ‡	*Plain*	<ipv4 address>	Returns the ip address of the network adapter (returns the first address if it is a list).
description of <network adapter> ‡	*Plain*	<string>	Returns the description of the network adapter.
dhcp server of <network adapter> ‡	*Plain*	<ipv4 address>	Returns the ip address of the dhcp server of the network adapter.
dns server of <network adapter> ‡	*Plain*	<network address list>	Returns a list of DNS servers used by the specified adapter.

Here are some syntax examples:

- `Addresses of adapters of network`
- `Description of adapter of network`

Practice with QNA:

24. How much RAM do you have on your computer? Output the results so they look like this: 256 MB RAM
25. Write a relevance expression that will only be true if your processor is made by Intel and its speed is less than 800 Mhz.
26. Write a query to output the dhcp servers of all your network adapters.

Link to Answers

Practice Problem Answers

1. Does the folder “c:\winnt” exists on your computer?

   - `Exists folder “c:\winnt”`

2. If so, does this folder contain “explorer.exe”?

   - `Exists file “explorer.exe” of folder “c:\winnt”`

     OR
   - `Exists file “c:\winnt\explorer.exe”`

   These statements will always return the same value, but the second example evaluates in significantly less time. Referencing a file by complete pathname is usually the quickest way to access it.

3. What is the version of the file “tcpip.sys”? This file is located in the folder “Drivers”, which is in the system folder.

   - `Version of file “tcpip.sys” of folder “Drivers” of system folder`

4. Write a relevance statement that will only return true if the size of the file “url.dll” is 106496 and its sha1 is b53aadb418dfb956e7aebef60d924ef990f919e6. Url.dll is found in the system folder. Since a sha1 hash is a string, it must be entered in quotes in a relevance statement.

   - `Size of file “url.dll” of system folder = 106496 AND sha1 of file “url.dll” of system folder = “b53aadb418dfb956e7aebef60d924ef990f919e6”`

Note: Later we’ll get around having to say `file “url.dll” of system folder` multiple times.

5. Do you have more files in your windows folder than you have recent applications?

   - `number of files of windows folder > number of recent applications`

6. List all the pathnames of your running applications.

- `Pathnames of running applications`

7. List all your regapps. Pick one and display its pathname and parent folder. (Two separate clauses)

   - `Names of regapps`
   - `Pathname of regapp “bde.exe”`
   - `Pathname of parent folder of regapp “bde.exe”`

8. What is the version of your “regedit.exe” file?

   - `Version of file “regedit.exe” of windows folder`

9. What is your version of internet explorer? (Hint, it’s a regapp)

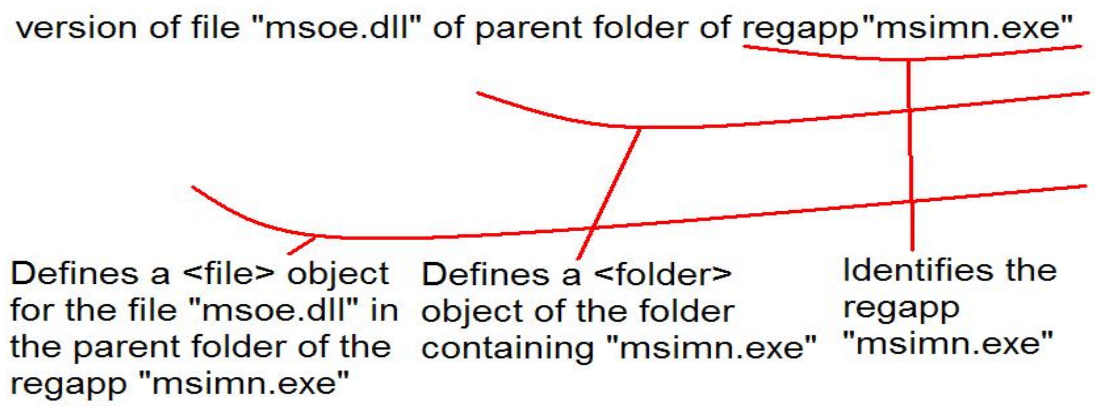
   - `Version of regapp “iexplorer.exe”`

10. Formulate a relevance clause to check whether one has IE5, and another one to see if they have IE6?

    - `Version of regapp “iexplorer.exe” = “5”`
    - `Version of regapp “iexplorer.exe” = “6”`

11. Write a relevance expression to check the version of the file “msoe.dll”, which is a file used by Outlook Express. This file will be in the same folder as the regapp “msimn.exe”. Hint: use `parent folder of regapp`

    - `Version of file “msoe.dll” of parent folder of regapp “msimn.exe”`


version of file "msoe.dll" of parent folder of regapp"msimn.exe"
Defines a <file> object for the file "msoe.dll" in the parent folder of the regapp "msimn.exe"
Defines a <folder> object of the folder containing "msimn.exe"
Identifies the regapp "msimn.exe"

12. What service pack are you running?

- `Csd version of operating system`

13. How would you check to see if a computer had any service pack installed??

- `Csd version of operating system != “”`

14. What relevance would return true for all Windows Operating Systems?

- `Name of operating system contains “Win”`

15. Write a relevance statement that would output just the number of the service pack

- `Last 1 of csd version of operating system`

16. Create another relevance statement that will output the version of msoe.dll. This time, however, don’t use the `parent folder of regapp` command. Instead, create the full path to the file direct by parsing the pathname of the regapp msimn.exe and concatenating the correct filename onto the end.

- `Version of file (preceding text of last “\” of pathname of regapp “msimn.exe” & “\msoe.dll”)`

17. Write a command that will return true if a user has ever installed a certain Internet Explorer patch on their computer. The following key will get created once the patch in installed: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components\{2298d453-bcae-4519-bf33-1cbf3faf1524}

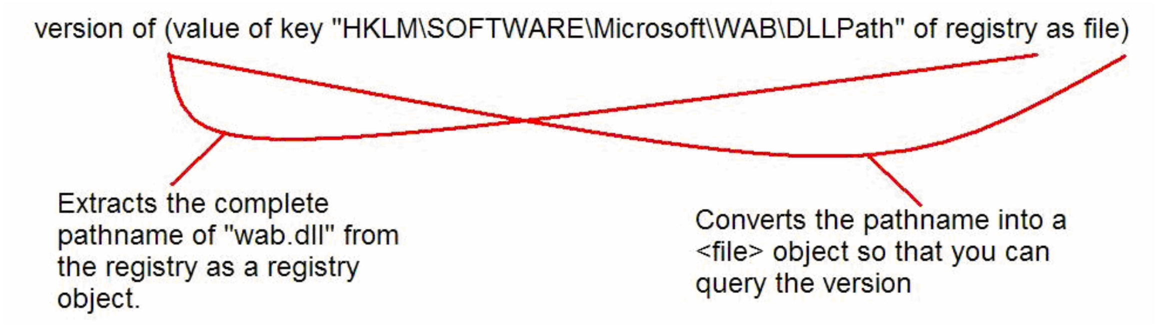
- `Exists key “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components\{2298d453-bcae-4519-bf33-1cbf3faf1524}” of registry`

18. Now check to see if the patch is actually installed at the moment. If it is, it will have a Dword value of “IsInstalled” set to 1.

   - `Value “IsInstalled” of key “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Active Setup\Installed Components\{2298d453-bcae-4519-bf33-1cbf3faf1524}” of registry = 1`

19. Find the version of wab.dll, a component of the Windows Address Book. You can find the complete path to this file by checking out the following key- HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WAB\DLLPath. Hint: You can query the default value of a key using `default value of <registry key>`.

   - `Version of (value of key “HKLM\Software\Microsoft\WAB\DLLPath” of registry as file)`


version of (value of key "HKLM\SOFTWARE\Microsoft\WAB\DLLPath" of registry as file)
Extracts the complete pathname of "wab.dll" from the registry as a registry object.
Converts the pathname into a <file> object so that you can query the version

20. What time is it now?

   - `Now`

21. How long has it been since explorer.exe has been modified?

   - `Now – modification time of file “explorer.exe” of windows folder`

22. Has it been more than a day since the system has been restarted?

   - `now – boot time of operating system > 1*day`

23. What is the state of the Task Scheduler service on your computer? (Note, ‘Task Scheduler’ is the Display Name, not the service name.)

- `State of service “Schedule”`

24. How much RAM do you have on your computer? Output the results so they look like this: 256 MB RAM

    - `((Size of RAM / (1024*1024)) as string) & “ MB RAM”`

25. Write a relevance expression that will only be true if your processor is made by Intel and its speed is less than 800 Mhz.

    - `Vendor name of processor = “GenuineIntel” AND speed of processor /Mhz < 800`

26. Write a query to output the dhcp servers of all your network adapters.

    - `Dhcp servers of adapters of network`

Advanced BES Relevance Syntax

If/Else/Then

The BES Relevance Language supports basic if/then/else logic within a single relevance statement. The general syntax is:

- `If <boolean> then <object> Else <object>`

The <objects> returned after `then` and `else` can be the result of a longer query, but they must both return the same object type. Here's an example:

- `If exists regapp “besconsole.exe” then version of regapp “besconsole.exe” as string else “BES Console Not Installed”`

This statement will output the version of the BES Console if it is installed. Otherwise it will return **BES Console Not Installed**. As I mentioned before, the return types from the clauses after `then` and `else` must be the same. In order to ensure this, I had to cast the version into a string in the `then` clause. You can nest multiple if/then/else statements together. Here's an example:

- `If exists regapp “wmplayer.exe” then version of regapp “wmplayer.exe” as string else if exists regapp “mplayer2.exe” then version of regapp “mplayer2.exe” as string else “Media Player Not Installed”`

This statement will output the version of Windows Media Player for recent versions that use the ‘wmplayer.exe’ executable or older versions that use the ‘mplayer2.exe’ executable. If/Then/Else formulations are most useful for creating retrieved properties, since you can use them to ensure that the property never returns an error.

Practice With QNA

1. Write a query that will return the service pack of the operating system if it exists. Otherwise, the query should return "No Service Pack"
2. Write a query that will return the number of processors on the machine. However, if the system has 1, 2, or 4 processors the statement should return 'Single', 'Dual' or 'Quad' respectively. Here are some examples of the output format

   1 processor – "Single Processor"
   2 processors – "Dual Processor"
   3 processors – "3 Processor"
   4 processors – "Quad Processor"
   5 processors – "5 Processor"

   Link To Answers

Whose/It – Filtering a list

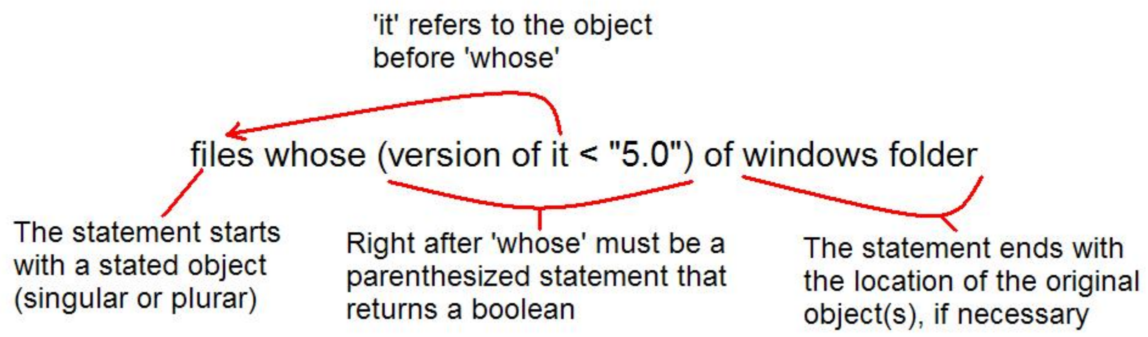
Iterative relevance clauses allow one to look for objects with certain characteristics out of a group of objects. The group can consist of files, registry keys, registry values, and more. To make an iterative relevance clause, you will need to use special syntax built into the Relevance Language using the keywords `whose` and `it`. In its most general case, a `whose` clause is used to filter a list. First, let's start with a general list:

- `Files of windows folder`

This query, of course, outputs a list of all the files in the windows folder. Now say that instead of all the files, you just wanted files with a certain characteristic. That's where the `whose` clause comes in. Let's say that we just wanted to output the files in the windows folder that have a version of less than 5.0:

- `Files whose (version of it < "5.0") of windows folder`

This clause iterates through all the files in the windows folder and only returns the ones for which the `whose` clause returns true. In this example, `it` refers to the <files> in the windows folder. The following graphic should be helpful:

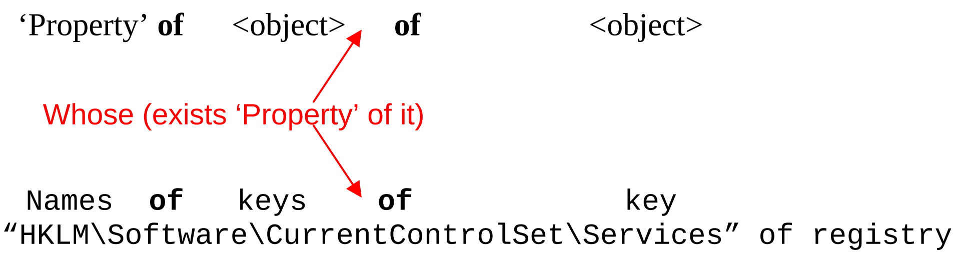

'it' refers to the object before 'whose'
files whose (version of it < "5.0") of windows folder
The statement starts with a stated object (singular or plurar)
Right after 'whose' must be a parenthesized statement that returns a boolean
The statement ends with the location of the original object(s), if necessary

Next, I'll try to clarify further with a more general discussion on the formulation of `whose` clauses. As we've mentioned many times before, the relevance language us built upon querying properties of objects (resulting in other objects). A general clause may look something like this:

'Property' **of** <object> **of** <object>

- `Names` **`of`** `keys` **`of`** `key "HKLM\Software\..." of registry`

In a query such as this, the expression will return the final property(ies) that are queried. In the above example, this corresponds to the `names` of all the subkeys of the key "HKLM\Software..." Now, say we wanted to filter our results so that we only returned the names of keys with particular characteristics.


'Property' of <object> of <object>
Whose (exists 'Property' of it)
Names of keys of key "HKLM\Software\CurrentControlSet\Services" of registry

There are a couple important notes to make about the contents of the parenthetical statement following `whose.` First, of all, it must return a boolean value. Next it needs to contain `it`, where `it` refers to the object directly preceding `whose`. (Technically speaking, the clause doesn't have to contain `it`, but otherwise the formulation is useless.) Continuing with the above example:

'Property' **of** <object> whose (exists 'Property' of it) **of** <object>

- `Names` **`of`** `keys whose (value "Start" of it = 2)` **`of`** `key "HKLM\Software\CurrentControlSet\Services" of registry`

The above example will output the names of the subkeys of the key "HKLM\Software\CurrentControlSet\Services" that have a value named "Start" set to 2. This corresponds to all services that start automatically when the computer starts.

Here are a couple more examples:

- `Names of files whose (name of it as lowercase contains ".dll" AND version of it <"4.4.0.3396") of parent folder of regapp "conf.exe"`

This example would return the names of all the **.dll files** in the same location as "conf.exe" who have a version less than **4.4.0.3396**. In this example, `it` refers to all files in the parent folder of the file conf.exe. As you can see, the parenthetical statement after `whose` can have multiple expressions linked together; they just must return a boolean value in the end. Here's another example:

- `number of keys whose (name of it contains "Microsoft SQL Server 2000" and name of it does not contain "Analysis Services") of key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" of registry`

This example would return the number of SQL Server 2000 instances installed on the computer. Here, `it` refers to all subkeys of the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Uninstall.

Practice With QNA:

3. How many executables do you have in your Windows folder?
4. How many files have the same modification and creation time in your windows folder?
5. What is your version of Microsoft office? Try to formulate this using a whose clause. Again, you can look at the following location in the registry: "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" Try to see if you can iterate through the subkeys of this key, finding the one corresponding to Microsoft Office, then output the version. This technique is useful because the name of the key tends to differ based on the version. Hint: When analyzing the Name value of the keys, make sure the value exists and is a string.

   Link To Answers

Whose/It – Beyond Filtering

While in the examples above we were using a whose/it formulation to filter a list, we can use the same syntax to check for the existence of a single object. In this case, you apply the filter to a single object instead of a list in order to find out if this object complies with the demands of the filter. Here's an example:

- `Exists` **`file "mshtml.dll"`** `whose (version of` **`it`** `< "5.0.2600.1528") of system folder`

This example checks to see whether there is any file named 'mshtml.dll' in the system folder that has a version less than 5.0.2600.1528. Here, `it` refers to the file `"mshtml.dll"`. The nice thing about this formulation is that will not error out if mshtml.dll does not exists or if it doesn't have a version. Let's quickly compare this to the previous method for doing this type of query. In most cases, the below statement will evaluate the same way as the above one

- `Version of file "mshtml.dll" of system folder < "5.0.2600.1528"`

However, there are some key differences. The second statement will error out if the file "mshtml.dll" does not exists in the system folder, if it doesn't have a version. On the other hand, the `whose` statement will never error out (on windows machines). In order to maintain the same level of error-checking robustness without a `whose` clause, you would need to write

- `Exists file "mshtml.dll" of system folder AND exists version of file "mshtml.dll" of system folder AND version of file "mshtml.dll" of system folder < "5.0.2600.1528"`

As a point of reference, the `whose` statement evaluates in about a third of a time as the above clause.

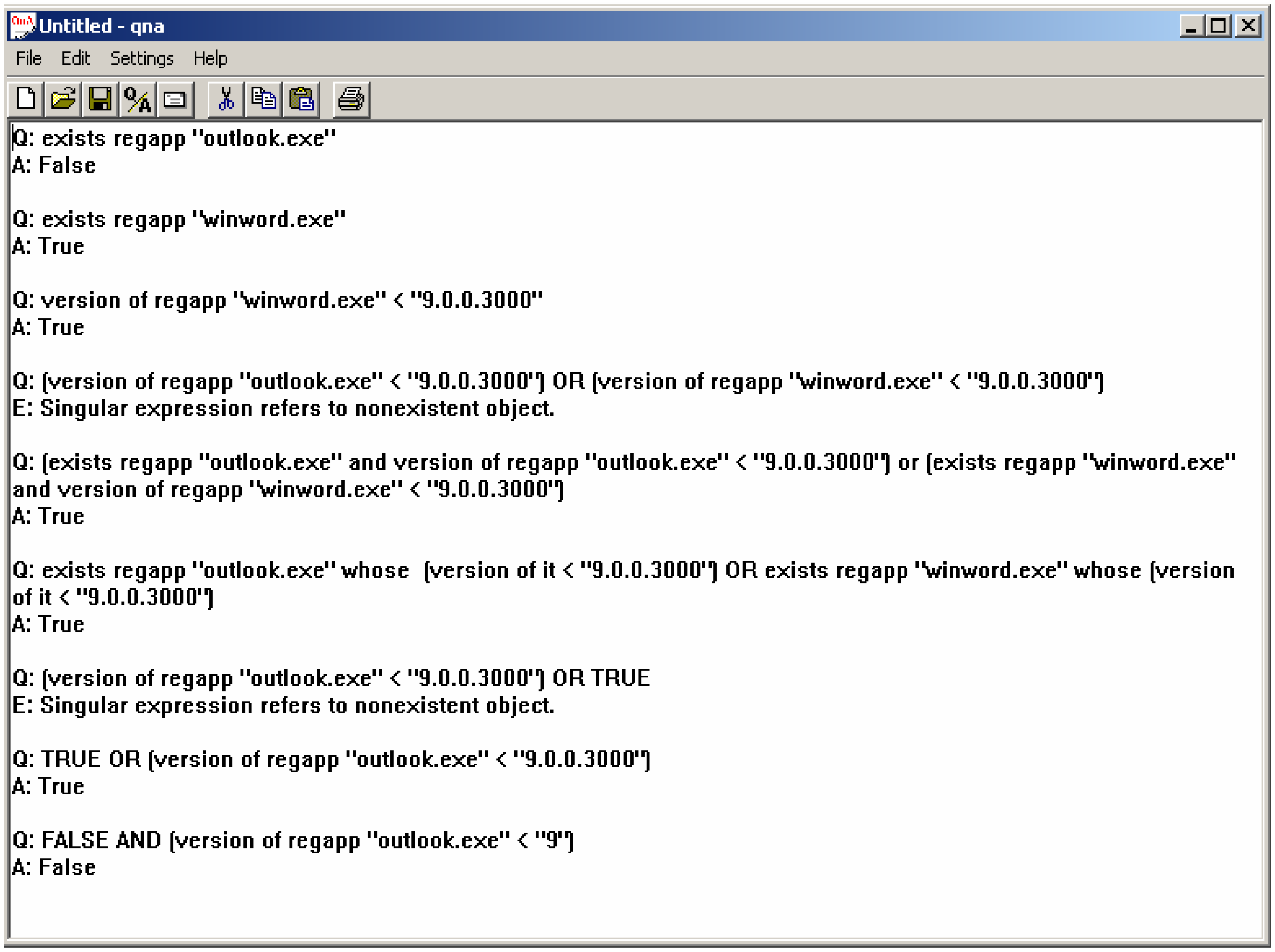
Practice With QNA

6. Write a relevance statement that will return **True** if a computer has a version of the BES Console less than 5.0.3.0. The BES Console is a regapp named 'BESConsole.exe'. Write the statement using a `whose` clause such that it won't fail if the BES Console doesn't exist.

Link To Answers

Avoiding Error Messages

So far, we've generally assumed that all relevance queries will return good, clean results. Yet with all languages, there are some queries that will simply return errors. What if you ask the version of a file that doesn't exist? What if you query the value of a key that doesn't exist? These are things you'll need to do, but you need to make sure these errors don't hurt the intended usage of your relevance. Let's take a look at an example. Say you have a patch that will close a security vulnerability that exists if you have an older version (less than 9.0.0.3000) of either Microsoft Word or Microsoft Outlook on your computer. You'll need to check both programs, but you need to be careful. For the following QNA example, the user has Word installed but not Outlook.



```
Q: exists regapp "outlook.exe"
A: False

Q: exists regapp "winword.exe"
A: True

Q: version of regapp "winword.exe" < "9.0.0.3000"
A: True

Q: (version of regapp "outlook.exe" < "9.0.0.3000") OR (version of regapp "winword.exe" < "9.0.0.3000")
E: Singular expression refers to nonexistent object.

Q: (exists regapp "outlook.exe" and version of regapp "outlook.exe" < "9.0.0.3000") or (exists regapp "winword.exe"
and version of regapp "winword.exe" < "9.0.0.3000")
A: True

Q: exists regapp "outlook.exe" whose  (version of it < "9.0.0.3000") OR exists regapp "winword.exe" whose (version
of it < "9.0.0.3000")
A: True

Q: (version of regapp "outlook.exe" < "9.0.0.3000") OR TRUE
E: Singular expression refers to nonexistent object.

Q: TRUE OR (version of regapp "outlook.exe" < "9.0.0.3000")
A: True

Q: FALSE AND (version of regapp "outlook.exe" < "9")
A: False
```

Some of these results may seem a bit strange. Since the regapp "outlook.exe" doesn't exist, the Relevance Language returns an error message when we query its version. If this error is the first expression in a Boolean statement with 'or' or 'and', the whole statement will return an error. It is important to note that if an entire relevance clause returns an error, a Fixlet message will never appear relevant. Let's take a look at the results:

- `(version of regapp "outlook.exe" < "9.0.0.3000") OR (version of regapp "winword.exe" < "9.0.0.3000")`

This is BAD RELEVANCE! While it obviously wants to see if either Word or Outlook is out of date, it will error out if Outlook doesn't exist. It is good form to always check for the existence of a file or registry key **before** you query it. Here are some better examples:

- `(exists regapp "outlook.exe" and version of regapp "outlook.exe" <`

```
”9.0.0.3000”) or (exists regapp “winword.exe” and
version of regapp “winword.exe” < “9.0.0.3000”)
```

-
  ```
  exists regapp “outlook.exe” whose (version of it
  < “9.0.0.3000”) OR exists regapp “winword.exe”
  whose (version of it < “9.0.0.3000”)
  ```

Both these examples check for the existence of both programs instead of just querying the versions. They are equivalent in what value they will return, but take somewhat different time to evaluate. In fact, the second clause takes about 75% of the evaluation time of the first clause. Under normal circumstances, they both are so quick that the evaluation time doesn’t really matter, but if we were checking 200 files instead of two, we’d need to be very careful. Try plugging the two above statements into QNA to check how long they take to evaluate (CTRL-T).

Note! You can troubleshoot errors in QNA using the reference in Appendix A of this document.

Practice with QNA:

7. Let’s say you want relevance to check for a version of Internet Explorer less than 6.0.2800.1106. For the purposes of this example, we’ll want it to be true if either the regapp or the registry says the version is less than desired. Write relevance with the proper error-checking to check for this. You should be able to find the version of internet explorer at the value “Version” of the following key: “HKLM/Software/Microsoft/Internet Explorer”

   Link To Answers

Using `it` without a `whose`

Querying Multiple Properties of an Object

Another use of this syntax involves using the `it` command without a `whose` command to make a list of desired properties. First, let me show a trivial use of the `it` command that will hopefully make this technique more clear. You can use `it of <object(s)>` to refer to an object or number of objects. For instance,

- `it of files of system folder`

is the same as

- `files of system folder folder`

While at first glance this doesn't seems very useful, it has a number of useful consequences. Let's take a quick example. Say that you wanted to check that a computer had Windows XP Service Pack 1 installed. We've learned that you can check for this with:

- `Name of operating system = “WinXP” AND csd version of operating system = “Service Pack 1”`

You may have notices that we had to write `operating system` twice, which seems necessary. We can write relevance more easily using `it`.

- `(Name of` **`it`** `= “WinXP” and csd version of` **`it`** `= “Service Pack 1”) of` **`operating system`**

In this example, `it` refers to `operating system`. This may be somewhat confusing because in the `whose/it` clauses, `it` referred to the object before the `whose`, which was before the parentheses. When there is no `whose`, `it` refers to the object after the parentheses. As is shown in the example above, you usually use `it` within a parenthetical statement that has multiple expressions. In many cases, similar relevance clauses can be written in multiple ways. For instance, the following three relevance statements will usually evaluate the same:

- `Version of file “mshtml.dll” of system folder < “6” AND size of file “mshtml.dll” of system folder < 54321`
- `(Version of it < “6” AND size of it < 54321) of file “mshtml.dll” of system folder`
- `Exists file “mshtml.dll” whose (version of it < “6” AND size of it < 54321) of system folder`

As long as 'mshtml.dll' exists in the system folder, these expressions will all evaluate the same. The last expression has the advantage that if mshtml.dll doesn't exists it will evaluate to **false**, while the others will error out.

Outputting Multiple Properties

Often it is useful to output multiple properties of an object in either a retrieved property or in a QNA query. For instance, say you wanted to query the names and versions of all your regapps.

- `(name of it & “ – “ & version of it as string) of regapps`

When evaluated in QNA or used as a retrieved property, this will give you a list of the names and versions of all the regapps separated by a hyphen. Here `it` refers to `regapps`. Technically speaking, this statement iterates through all the regapps, and for each of them it evaluates the parenthetical statement. This parenthetical statement concatenates the name of the regapp, a hyphen, and the version of the regapp, and then outputs the final string.

Answers to Practice Problems:

1. Write a query that will return the service pack of the operating system if it exists. Otherwise, the query should return “No Service Pack”

   - `If csd version of operating system != “” then csd version of operating system else “No Service Pack Installed”`

2. Write a query that will return the number of processors on the machine. However, if the system has 1, 2, or 4 processors the statement should return ‘Single’, ‘Dual’ or ‘Quad’ respectively. Here are some examples of the output format

   1 processor – “Single Processor”
   2 processors – “Dual Processor”
   3 processors – “3 Processor”
   4 processors – “Quad Processor”
   5 processors – “5 Processor”

   - `(if number of processors = 1 then "Single" else if number of processors = 2 then "Dual" else if number of processors = 4 then "Quad" else number of processors as string)& " Processor"`

3. How many executables do you have in your Windows folder?

   - `Number of files whose (name of it as lowercase contains “.exe”) of windows folder`

   For whatever reason, some files will return their names in capital letters and some in lowercase. To find all the executables, you’ll generally need to use the `as lowercase` command.

4. How many files have the same modification and creation time in your windows folder?

   - `Number of files whose (modification time of it = creation time of it) of windows folder`

5. What is your version of Microsoft office? Try to formulate this using a whose clause. Again, you can look at the following location in the registry: “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\

Uninstall" Try to see if you can iterate through the subkeys of this key, finding the one corresponding to Microsoft Office, then output the version. This technique is useful because the name of the key tends to differ based on the version. Hint: When analyzing the Name value of the keys, make sure the value exists and is a string.

- `Value "DisplayVersion" of key whose (exists value "DisplayName" of it and value "DisplayName" of it as string contains "Microsoft Office") of key "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall" of registry`

6. Write a relevance statement that will return **True** if a computer has a version of the BES Console less than 5.0.3.0. The BES Console is a regapp named 'BESConsole.exe'. Write the statement using a `whose` clause such that it won't fail if the BES Console doesn't exist.

   - `exists regapp "Besconsole.exe" whose (version of it < "5.0.3.0")`

7. Let's say you want relevance to check for a version of Internet Explorer less than 6.0.2800.1106. For the purposes of this example, we'll want it to be true if either the regapp or the registry says the version is less than desired. Write relevance with the proper error-checking to check for this. You should be able to find the version of internet explorer at the value "Version" of the following key: "HKLM/Software/Microsoft/Internet Explorer"

   - `(exists regapp “iexplorer.exe” and version of regapp “iexplorer.exe” < “6.0.2800.1106”) OR (exists key “HKLM\Software\Microsoft\Internet Explorer” of registry and exists value “version” whose (it < “6.0.2800.1106”) of key “HKLM\Software\Microsoft\Internet Explorer” of registry)`

The BigFix Action Language

Scripts written in the BigFix Action Language allow you to remediate problems or change the configuration of computers running the BES Client. Along with relevance, actions are the other main part of BES Tasks, Fixlet® messages and Custom Actions. Actions run like a script, executing the code of the action language one line at a time. In general, the Action language is very simple yet allows you to perform varied and complex tasks. If you want to use other scripts/programs to remediate issues, you can always call these programs from within a BES Action.

There are a few caveats that we should cover before we dive into the language.
First of all, an error in one line of an action script will usually terminate the script, and the rest of the lines will not run. Because of this, it's very important to utilize error prevention methods in action scripts. Next, debugging actions can sometimes be difficult because there is no QNA-type tool to evaluate Fixlet actions. Instead, you'll need to make a custom Task or Fixlet in order to test your actions while they are in development. In order to do the exercises in this section, it would probably be helpful to create a BES Custom Action, Task, or Fixlet message to test them out. As with relevance, becoming proficient at Fixlet message actions takes some time. Fortunately, the common actions most people will use (such as downloading and applying a patch) are very simple.

One nice thing about Fixlet message actions is that all parameters can be defined by relevance. You can include relevance expressions in actions by surrounding them with curly braces {}. A working knowledge of the Relevance Language will make writing actions much easier.

This guide doesn't give a complete list of all the action commands. Instead, it just includes the most useful ones, which a beginning BES Custom Content author will probably use most. A more comprehensive guide is the BigFix Windows Shell Action Library (http://support.bigfix.com/fixlet/documents/WinActions-2003-05-27b.pdf). That document contains much of the same information as here, but contains more commands and technical details. There is also a comprehensive action library contained in the BDE help feature.

I'd also like to stress the fact that it is important for your actions and relevance to work together. Remember that applying a action should falsify the relevance of your Fixlet message or BES Task.

Relevance Substitution

As I mentioned above, one of the main strengths of the BigFix Action Language is its ability to evaluate relevance statements as part of its operation. You can include relevance statements in an action script by enclosing them in curly braces {}. When an action script is run that contains relevance, this relevance is interpreted and placed into the script before the script is executed. For instance, say you wanted to run a script by calling cscript.exe, which is located in the system folder. You could write

- `Run {pathname of system folder}\cscript.exe`

On Windows XP, the actual script that would run is

- `Run c:\windows\system32\cscript.exe`

This technique is extremely powerful because it allows you to run scripts on computers with many different configurations.

File System Commands

The **File System Commands** will enable you to define a button or link to process files on the user's computer.

The following pages present the `delete`, `copy`, `move`, `open`, and `download` commands. One important thing to note is that in a Fixlet action, there is a default location for the working directory of the action. This default location is the location of the site directory on the hard drive.

C:\Program Files\BigFix Enterprise\Enterprise Client__BESData\<sitename>

Note! For all commands that take a path as a parameter, you must use quotation marks around the path if it has any spaces in it. Otherwise, the quotation marks are optional.

delete

The `delete` command deletes the named file. Any script with the `delete` command will terminate if the file exists but cannot be deleted. This can happen due to write protection or an attempt to delete from a CD-ROM, for instance. If the file does not exist at all, however, the script will continue to execute.

Syntax

Keyword	***Parameters***
delete	*"FileName"*

Example

- `delete “c:\program files\bigsoftware\module.dll”`
- `delete “{name of drive of windows folder}\win.com"`

copy

They `copy` command copies the source file to the named destination file. A script with the `copy` command terminates if the destination already exists or if the copy fails for any other reason such as when the destination file is busy.

Syntax

Keyword	***Parameters***
copy	*"Source_FileName" "Destination_FileName"*

-
  ```
  copy “{name of drive of windows folder}\win.com" “{name of
           drive of windows folder}\bigsoftware\win.com"
  ```

-
  ```
  delete “c:\windows\system\windir.dll”
  copy “__Download\windir.dll”
  “c:\windows\system\windir.dll”
  ```

It is important to note that the `copy` command will kill the entire action script if it fails. For this reason, it is recommended that you use a `delete` command before every `copy` command, just to make sure the script doesn’t error out. The second example above illustrates this.

move

The `move` command moves the source file to the named destination file. This command also provides “rename” type functionality. A script with the `move` command terminates if the destination exists or the move fails.

SYNTAX

Keyword	***Parameters***
move	*“Source_FileName” “Destination_FileName”*

EXAMPLE

-
  ```
  move “c:\program files\bigsoftware\module.dll”
  “c:\temp\mod.dll”
  ```

-
  ```
  delete “c:\updates\q312456.exe”
  move “__Download\q312456.exe”
  “c:\updates\q312456.exe”
  ```

-
  ```
  delete “c:\temp\new_name.doc”
  move “c:\temp\old_name.doc” “c:\temp\new_name.doc”
  ```

Again, it is good practice to use a `delete` command before every move command. The first two examples are actually moving files around, while the third example renames a file by moving it to the same directory.

Practice Problems

1. Write a custom action to perform the following actions:

- Rename the file “test1.txt” within the directory “C:\Windows” to “test2.txt” (You can rename a file my moving it to the same directory with a new name)

- Create a backup copy of “test2.txt” called “test2bak.txt” within the same directory (Remember to use a ‘delete’ command before you move or copy)

You’ll have to create the file “test1.txt” to test this action. Write relevance such that the action will only run if “test1.txt” exists and will fix once test1 goes away.

2. How would I copy the file “acpi.sys” from the <system>\drivers folder to “c:\”? Write relevance to ensure “acpi.sys” exists in the system folder and that it does not exists in “c:\”. (Remember you can substitute relevance in curly braces {} to find variables like the system folder)

Link to Answers

Execution Commands

run

The `run` command executes the indicated process or program. If the process can't be created, the script is terminated. `Run` does not wait for the process to terminate before executing the next line of the script. The command line contains the name of the executable and may optionally contain parameters. If you want to wait for one program to finish before starting another one, use the `wait` command (see below).

Syntax

Keyword	***Parameters***
Run	<command line>

Example

- `run “c:\program files\Microsoft Office\Office\excel.exe”`
- `run “c:\winnt\ftp.exe” ftp.sonic.net`

Technical Note

This command has the same effect as issuing a CreateProcess("CommandLine") statement from Windows API. This is also the same as using CommandLine in the Windows RUN dialog.

See the Windows documentation on CreateProcess for a discussion of the method used to locate the executable from a CommandLine.

wait

The `wait` command behaves the same as the `run` command, except that it waits for the completion of its process or program before continuing. One thing to keep in mind is that the `wait` command only waits for the explicit process it calls to finish before continuing. For instance, if you use `wait` to open an executable which in turn opens another executable, the action script will continue after the first executable completes.

SYNTAX

Keyword	***Parameters***
Wait	<command line to execute program>

EXAMPLE

```
wait scandskw.exe
run {pathname of system folder & “\defrag.exe”}
```

TECHNICAL NOTES:

This has the same effect as issuing a CreateProcess("CommandLine") statement from the Windows API, and then waiting for completion.

The working directory for all actions is:

%install path of BES Client%__BES Data\[Fixlet Site Name]

For instance, for Fixlet in the Enterprise Security Site where the BES Client is installed into the default directory, the path is:

C:\Program Files\BigFix Enterprise\BES Client__BES Data\Enterprise Security

The ‘site’ for actions created using custom content is the actionsite. Therefore, the working directory for an action originating from custom content is:

C:\Program Files\BigFix Enterprise\BES Client__BES Data\Actionsite

Practice Problems:

3. Write a custom Task that will add the contents of a .reg file ‘c:\test.reg’ to the registry. This is done by running the regedit.exe program, passing in test.reg as a parameter. Regedit.exe is in the windows folder. In order to make regedit.exe run silently, you can use the –s switch. Write relevance to make sure test.reg exists. In order to test this action, you’ll have to create test.reg on a test computer. Note: this relevance will not falsify.

Link to Answers

dos

The `dos` command issues a standard DOS command. If the DOS command fails, the script that contains it is terminated.

SYNTAX

`Keyword`	***Parameters***
`Dos`	<DOS command line syntax>

EXAMPLE

- `dos rmdir /Q /S {pathname of windows folder & “\temp\BigFixQ322273}`
- `dos scandisk.exe e:`

In the last example, e: is a parameter passed to the scandisk program.

TECHNICAL NOTES:

This has the same effect as issuing a system("DosCommandLine") statement from the Windows API. It is also the same as typing the DosCommandLine to a DOS prompt.

The `dos` command uses the PATH environment variable to locate the command on the user's hard drive. If you want any it to look elsewhere, you must specify a complete pathname.

The most commonly used DOS commands are:

- `Dos net start`
- `Dos net stop`
- `Dos mkdir`

- `Dos rmdir`

Practice Problems:

4. Write a custom task to restart the DHCP service. Send 'net start' and net stop output to a log file located at "c:\BESlogs\DHCPlog.txt". You can send output to a file using the '>' and '>>' dos commands. You'll need to create the directory "c:\BESlogs". For this exercise, just get the action right- we'll worry about relevance next.

5. Write relevance for the above task. This task should be relevant as long as the DHCP service is running. Also, have the relevance check the log file so that once the service has restarted successfully, the task becomes non-relevant. You can do this by checking that line 5 of the file contains "The DHCP Client service was stopped successfully".

Link to Answers

Downloads

continue if

The `continue if` command will enable execution to continue on to the next command in the action script if the value provided as a parameter evaluates to **true**. It will stop without error if the specified expression evaluates to **false**. You can use relevance substitution to compute the value.

SYNTAX

`Keyword`	***Parameters***
`Continue if`	relevance condition to evaluate

EXAMPLE

- `continue if {name of operating system = "Win2k"}`

-
  ```
  download http://www.real-time.com/downloads/win98/dun40.exe

  continue if {(size of it = 325904 and sha1 of it = "013e48a5e71acb10563068fbdd69955b893569dc") of file "dun40.exe" of folder "__Download"}

  wait __Download/dun40.exe /Q:A /R:N
  ```

```
action requires restart
```

download

The `download` command downloads the file indicated by the URL. The file is saved in a directory named "__Download" (the name begins with two underscores) relative to the local folder of the Fixlet® Site that issued the download command.

For the BigFix Enterprise Client, the default download directory is:
C:\Program Files\BigFix Enterprise\Enterprise Client\ __BESData\ <sitename> __Download

If the download fails, the script terminates. The name of the file is derived from the part of the URL after the last slash.

For instance, consider the command:

- `download ftp://ftp.microsoft.com/deskapps/readme.txt`

The action example above downloads the readme.txt file from the Microsoft site and automatically saves it in the local __Download folder as **readme.txt**.

Syntax

Keyword	***Parameters***
Download	[options] *File_URL*

Example

- `download http://download.bigfix.com/update/bf1504.exe`

This command directs the downloaded file to the default site __Download folder.

-
  ```
  download
  http://download.microsoft.com/download/a/8/c/a8c904e2-
   7955-47e2-a2ae-e7f6490eee95/WindowsServer2003-
   KB840374-x86-ENU.EXE

  continue if {(size of it = 658672 AND sha1 of it =
   "fadc782accb64bbec95027d353e2903129d40eb3") of file
   "WindowsServer2003-KB840374-x86-ENU.EXE" of folder
   "__Download"}
  ```

```
wait __Download\WindowsServer2003-KB840374-x86-ENU.EXE
 /passive /quiet /norestart
```

This is the standard format for performing a download and then running the downloaded program. Downloads are actually "prefetched" by the BES Server and delivered to the clients before the action begins to run. The `continue if` statement (which I will formally introduce later) is used here to ensure that the downloaded file has the expected size and Sha1 hash. When the `continue if` statement shown above is used after a download, the downloaded file is saved on the BES Server and referenced by its sha1 hash. If a file with the correct Sha1 is already on the BES Server, the clients will just get the file from it. BES Relays also cache downloads.

Tip! In order for the mirror server to correctly download a file, the syntax of the `Continue If` statement must be exactly as above. Any minor error will cause the server to think that the downloaded file is incorrect and the action will fail. I recommend that you simply copy and paste the appropriate size, and sha1, and file name into every `continue if` statement that you write. There is also a program called "sha1.exe" that will output a complete `continue if` statement for a given file.

You can download "sha1.exe" here:

http://support.bigfix.com/fixlet/documents/sha1.exe

You can get sha1.exe to output a complete `continue if` statement using the following syntax:

- `Sha1.exe –r <filename>`

Practice Problems

6. Try writing a Fixlet message to install a Microsoft security patch . Write the download command, the `continue if` statement, and then run the patch. Use the patch found here: http://download.microsoft.com/download/a/f/a/afa937e7-e7f4-4fe8-8324-6e322f7ab542/WindowsXP-KB840374-x86-ENU.EXE. You can run it silently by using the following command line switches: /silent /norestart. The Fixet message should be relevant if the following key does not exist: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Updates\Windows XP\SP2\KB840374

   Link to Answers

Registry Commands

Note: When using registry action commands, you can't abbreviate the root key like you could when writing relevance. For instance, you must write out 'HKEY_LOCAL_MACHINE" instead of using "HKLM".

regset

The regset command sets a registry key to the given name and value. If the key doesn't already exist, this command creates the key with this starting value.

SYNTAX

`Keyword`	***Parameters***
`Regset`	<"registry key"> <"value name">=<value>

These values are entered just as they are in a registry file, in keeping with the rules for Regedit, the Windows program that edits the registry. String entries are offset by quotes, and the standard 4-byte integer (dword) is entered in hexadecimal with leading zeroes.

EXAMPLE

- `regset "[HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security]" "Level"=dword:00000002`
- `regset "[HKEY_CURRENT_USER\Software\BigFix Inc.]" "testString"="bob"`

NOTE: Registry commands must be formulated exactly as above. The existence or lack of white space can cause the command to fail. For instance, there must be no white space around the equals sign.

TECHNICAL NOTES

Notice in these examples that square brackets [ ] are used to enclose the name of the registry key. Again, this is in keeping with the rules for Regedit files. This syntax is necessary for the `regset` command, but not for registry Inspectors.

When you use the BigFix `regset` command, keep in mind that the BigFix client dynamically builds the .reg file that you would have had to create manually to update the registry and then it executes that resulting .reg file for you. One of the rules of the .reg file is that any \'s in the data value need to appear as double slashes, that is \\. So if you were trying to assign the value SourcePath2 of the registry key HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows

NT\CurrentVersion to c:\I386, the command that you would define would look like this:

- `regset "[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion]" "SourcePath2"="c:\\I386"`

To get around this annoyance, you can use the 'Escapes' relevance command. This command takes a string and puts in the extra slashes. For instance:

- `Escapes of “c:\program files\bigfix” = “c:\\program files\\bigfix”`

regdelete

The `regdelete` command deletes a registry key value of the given name. If the value doesn't already exist, this command will fail and all subsequent commands will not be executed.

SYNTAX

Keyword	***Parameters***
Regdelete	<“registry key”> <“value name”>

Example

-
  ```
  regdelete
  "[HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\Vx
  D\IOS” "NoIDE"
  ```

-
  ```
  regdelete “[HKEY_CLASSES_ROOT\ShellScrap]” “NeverShowExt”
  regset "[HKEY_CLASSES_ROOT\ShellScrap]"
  "AlwaysShowExt"=""
  ```

-
  ```
  regdelete "[HKEY_CURRENT_USER\Software\Microsoft\Windows
  NT\CurrentVersion\AppCompatFlags\Layers]" "{name of value
  whose (name of it contains "BigFix.exe") of key
  "HKEY_CURRENT_USER\Software\Microsoft\Windows
  NT\CurrentVersion\AppCompatFlags\Layers" of registry}"
  ```

Note: You cannot use `regdelete` to delete an entire key, just values. To delete a key, you will have to use a .reg file.

Practice Problems:

7. Write a custom Fixlet message that will create the Key HKEY_Local_Machine\Software\Corporate with the value “Department” = “Engineering”. Write relevance so that this Fixlet will only apply to computers without this registry value.

8. Write a script that will add an “InstallLocation” value to the BES Client registry key. The value should contain the current path of BES Client.

Link to Answers

Comments

//

Lines beginning with // are comments and are ignored during action execution.

Syntax

`Keyword`	***Parameters***
`//`	

Example

-
  ```
  // The following command will replace the file on the C
  drive
  copy “{name of drive of windows folder}\win.com" “{name of
  drive of windows folder}\bigsoftware\win.com"
  ```

Other Action Commands

action requires restart

The `action requires restart` command informs the client that the current action will not be completed until the next restart completes. Once this action has been completed on a computer, the inspector `pending restart` will return **true**. If there is an `action requires restart` command in an action, the BigFix Enterprise Console will report **Pending Restart** until the affected computer is restarted. This command is useful when running installations that will not complete until the computer is restarted.

Syntax

Keyword	***Parameters***
action requires restart	

Example

-
  ```
  wait __Download\323255USA8.EXE /q:a /r:n
  action requires restart
  ```

Action may require restart

The `action may require restart` command will set the `pending restart` inpector to **true** if it detects that files will be replaced on the system upon reboot. This is useful in cases where a patch will requires a restart in certain circumstances but not in others. If it detects that files will be replaced, it will also cause the client to report its action status as **Pending Restart** until the computer is rebooted

Syntax

Keyword	***Parameters***
action may require restart	

Example

-
  ```
  wait __Download\323255USA8.EXE /q:a /r:n
  action may require restart
  ```

action parameter query

This `action parameter query` command allows data entry of parameters to be available via relevance during action execution. Parameter names may include blanks, and are case sensitive. The parameter name, description, and value must each be enclosed inside double quotation marks (").

Syntax

Keyword	***Parameters***
`action parameter query`	"<parameter name>" [with description "<description>"] [and] [with default [value] "<default value>"]

Example

- `action parameter query "Registry key" with description "Please enter your desired registry key" and with default value "null"`
- `action parameter query "InstallationPoint" with description "Please enter the location of the shared installation point:”`

  `regset “[HKEY_LOCAL_MACHINE\Software\BigFix]” “InstallInfo”=”{escapes of (parameter “InstallationPoint” of action)}”`

Technical Notes

The parameter value may include %## where ## are hex digits to specify the character you want to embed. You can find the hex code for characters using any ascii table. To embed a percent sign, use %25. To embed a double quote, use %22.

To retrieve the action parameter value, for example in relevance substitution use: {parameter "paramname" of action}

set clock

The `set clock` command causes the client to reregister with the registration server, and to set its clock to the time received from the server during the interaction. This is useful when the client’s clock is out of sync. This command is not available when the client is operating under an evaluation license.

Syntax

Keyword	***Parameters***
`set clock`	

Example

- `set clock`

restart

The `restart` command causes the Windows client to restart the computer. If a user is logged in, it places a dialog up and waits for the user to press a button in the dialog labeled 'Restart'. This allows the user to save his/her work before restarting. Once the user presses the restart button, the client politely uses an API to request a system restart. If an application refuses the request, the dialog stays up and waits for the user to press the 'Restart' button again. If a user is not logged in, the command will force a restart.

In order to force a restart, you can give a numerical parameter after the command. This will force the computer to restart in the number of seconds you have inputted.

SYNTAX

`Keyword`	***Parameters***
`Restart`	<time to force restart>

EXAMPLE

-
  ```
  run __Download\IE55SP2.exe
  action requires restart
  restart
  ```

Practice Problems:

9. Write a custom task that will force a restart of computers. Use an action parameter query to prompt the user to give the number of seconds in which to force the restart. Have the default value be 120. This task should be relevant on computers pending a restart.

Link to Answers

Practice Problem Answers:

1. Write a custom action to perform the following actions:

   - Rename the file “test1.txt” within the directory “C:\Windows” to “test2.txt” (You can rename a file my moving it to the same directory with a new name)

   - Create a backup copy of “test2.txt” called “test2bak.txt” within the same directory (Remember to use a ‘delete’ command before you move or copy)

   Action:

   - `Delete c:\windows\test2.txt`
   - `Move c:\windows\test.txt c:\windows\test2.txt`
   - `Delete c:\windows\test2bak.txt`
   - `Copy c:\windows\test2.txt c:\windows\test2bak.txt`

   Relevance:

   - `Exists file “c:\windows\test2.txt”`

2. How would I copy the file “acpi.sys” from the <system>\drivers folder to “c:\”? Write relevance to ensure “acpi.sys” exists. Since acpi.sys won’t go away, this relevance won’t falsify, so it would be appropriate for this to be a Task. (Remember you can substitute relevance in curly braces {} to find variables like the system folder)

   Action:

   - `Delete c:\temp\acpi.sys`
   - `Copy {pathname of system folder}\drivers\acpi.sys c:\acpi.sys`

   Relevance:

   - `Exists file “acpi.sys” of folder “drivers” of system folder AND not exists file “acpi.sys” of folder “c:\”`

3. Write a custom Task that will add the contents of a .reg file ‘c:\test.reg’ to the registry. This is done by running the regedit.exe program, passing in test.reg as a parameter. Regedit.exe is in the windows folder. In order to make regedit.exe run silently, you can use the –s switch. Write relevance to make sure test.reg exists. In order to test this action, you’ll have to create test.reg on a test computer. Note: this relevance will not falsify.

   Action:

- `Run {pathname of windows folder}\regedit.exe –s c:\test.reg`

Relevance:

- `Exists file “c:\test.reg”`

4. Write a custom task to restart the DHCP service. Send ‘net start’ and net stop output to a log file located at “c:\BESlogs\DHCPlog.txt”. You can send output to a file using the ‘>’ and ‘>>’ dos commands. You’ll need to create the directory “c:\BESlogs”. For this exercise, just get the action right- we’ll worry about relevance next.

- `Dos Mkdir c:\BESlogs`
- `Dos net stop “DHCP” > c:\BESlogs\DHCPlog.txt`
- `Dos net start “DHCP” >> c:\BESlogs\DHCPlog.txt`

5. Write relevance for the above task. This task should be relevant as long as the DHCP service is running. Also, have the relevance check the log file so that once the service has restarted successfully, the task becomes non-relevant. You can do this by checking that line 5 of the file contains “The DHCP Client service was stopped successfully”.

- `(Exists running service “DHCP”) AND (Not Exists file “c:\BESlogs\DHCPlog.txt” OR line 5 of file “c:\BESlogs\DHCPlog.exe” as lowercase does not contain “the dhcp client was stopped successfully”)`

6. Try downloading a Microsoft security patch . Write the download command, the the correct `continue if` statement, and then run the patch.

Action:

- `download http://download.microsoft.com/download/a/f/a/afa937e7-e7f4-4fe8-8324-6e322f7ab542/WindowsXP-KB840374-x86-ENU.EXE`
- `continue if {(size of it = 874264 and sha1 of it = "74275b7835faf252ee522d1f8dff49ac43a82a90") of file "WindowsXP-KB840374-x86-ENU.EXE" of folder "__Download"}`
- ` wait __Download\WindowsXP-KB840374-x86-ENU.EXE /quiet /norestart`

Relevance:

-
  ```
  not exists key
  "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Updates\Windows XP\SP2\KB840374" of registry
  ```

7. Write a custom Fixlet message that will create the Key HKEY_Local_Machine\Software\Corporate with the value “Department” = “Engineering”. Write relevance so that this Fixlet will only apply to computers without this registry value.

   Action:

   -
     ```
     Regset “[HKEY_Local_Machine\Software\Corporate]”
     “Department”=”Engineering”
     ```

   Relevance:

   -
     ```
     Not exists key "HKLM\Software\Corporate" whose
     (value "Department" of it = "Engineering") of
     registry
     ```

8. Write a script that will add an “InstallPath” value to the BES Client registry key that contains the current path of BES Client.

   Action:

   -
     ```
     Regset
     “[HKEY_Local_Machine\Software\EnterpriseClient]”
     “InstallPath”=”{escapes of pathname of regapp
     “Besclient.exe”}”
     ```

   Relevance:

   -
     ```
     Not exists key “HKLM\Software\EnterpriseClient”
     whose (exists value “InstallPath” of it) of
     registry
     ```

9. Write a custom task that will force a restart of computers. Use an action parameter query to prompt the user to give the number of seconds in which to force the restart. Have the default value be 120.

   Action:

   -
     ```
     action parameter query "RestartTime" with
     description "Please enter your desired restart
     time" with default value "120"
     ```

   -
     ```
     restart {parameter “RestartTime” of action}
     ```

   Relevance

   -
     ```
     Pending restart
     ```

Appendix A: QNA Error Messages

This document is intended to be a quick reference on how to use relevance interpreter error message to debug queries written in the BigFix Relevance Language. This document contains the most common error messages with explanations or each and suggestions on how to troubleshoot them.

Singular Expression Refers to Non-Existent Object

This is probably the most common error message you will receive. It usually results from querying a property of an object that does not exists, or querying a non-existent property of an object.

For instance, the query:

- `Version of file “misspelled.dll” of folder “c:\temp”`

will return **Singular expression refers to non-existent object** if any of the following conditions are true:

1. The folder “c:\temp” does not exists
2. There is no file named “misspelled.dll” in the folder “c:\temp”
3. The file “misspelled.dll” located in the folder “c:\temp” does not have a version

Singular Expression Refers to Non-Unique Object

This error message arises when you try to query a singular property of multiple objects. For instance:

- `Version of files of system folder`

Will return the version of the first file it finds and then the error message **Singular Expression refers to non-existent object**. If you want to output a list of all the properties of a list, make sure to make the queries plural. For instance,

- `Versions of files of system folder`

will return a list of all the versions. If you want a single return value, you have to make sure to just query one object. If you want to return a list of properties, the inspector must be plural.

A Singular Expression Is Required

This error message results from trying to compare two lists or a list to an object. In general, all comparisons must be made between two singular objects. For instance:

- `Versions of files of folder “c:\temp2” = versions of files of folder “c:\temp”`

Will return **A singular expression is required** because comparing two lists is undefined. This will error out even if both folders contain exactly the same files. Similarly,

- `Versions of files of folder “c:\temp” = “4.5”`

Gives the same error because you can’t compare a list to a single value. You will get this error even if there is only one file in the folder “c:\temp” whose version is 4.5.

The Operator “<bad_command>” is Undefined

You will receive this message if you use a word that relevance interpreter does not recognize, or if you use invalid commands on an object. Here are some examples:

- `Exists executable “file_name.exe” of system folder`

This will return **The operator "executable" is not defined** since the word ‘executable’ is not a valid command in the relevance language.

- `Version of key “HLKM/Software” of registry`

This will return **The operator "version" is not defined.** Even though the relevance language knows the word ‘version’, it does not recognize it as a valid property of registry key, and therefore errors out.

The Operator “String” is not defined

This is a very common error message that indicates that you are trying to return an object that has no default return value. In order to remedy it you just need to query a property of that object. For instance.

-
  ```
  Key
  “HKEY_LOCAL_MACHINE\SOFTWARE\BigFix\EnterpriseClient”
  of registry
  ```

Will return **The Operator “String” is not defined.** Although the statement above correctly defines an object that does exist, the relevance language just doesn’t know what you want to know about that object. Instead, you have to either query a property of the object or ask its existence.

-
  ```
  Exists Key
  “HKEY_LOCAL_MACHINE\SOFTWARE\BigFix\EnterpriseClient”
  of registry
  ```

-
  ```
  Value “Version” of Key
  “HKEY_LOCAL_MACHINE\SOFTWARE\BigFix\EnterpriseClient”
  of registry
  ```

This Expression Could Not Be Parsed

The first step of interpreting a relevance statement is parsing the expression into its various components. The above message results from a failure of the parsing engine. This is usually caused by unmatched parentheses or by syntax errors involving certain ‘reserved words’ used by the parsing engine. Reserved words are syntactical statements like ‘of’, ‘and’, ‘equals,’ and so on. Here are a couple of examples:

-
  ```
  Name of (file whose (version of it = “2.6”) of
  system folder
  ```

This will return **This expression could not be parsed** because it has more open parentheses than closed ones. The expression:

-
  ```
  Exists file “name” of or system folder
  ```

Will return the same error message due to the nonsensical use of the reserved words ‘of’ and ‘or’.

A Boolean Expression is Required

This error message is produced when a statement needs a boolean value to evaluate, but instead the expression returns a different return type. A boolean value is required after ‘if’ and ‘whose’. For example, the parenthetical statement after the ‘whose’ in a whose/it clause does must return a Boolean value. For instance:

- `Names of files whose (version of it) of system folder`

The parenthetical statement after whose must return a boolean value for expression to make any sense. Since the relevance interpreter is expecting a Boolean value and instead finds a version, it returns the error **A boolean expression is required.** Instead the statement should be something like this:

- `Name of files whose (version of it = “5”) of system folder`

The same problem exists in if/then/else statements.

- `If regapp “besclient.exe” then version of regapp “besclient.exe” as string else “N/A”`

This will error out because a boolean expression is required after the ‘if’. Instead, the statement should read.

- `If exists regapp “besclient.exe” then version of regapp “besclient.exe” as string else “N/A”`

It Used Outside of Whose Clause

This is an especially confusing error message because it is perfectly legal to use ‘it’ without a ‘whose’ clause as long as you form the syntax correctly. This message just means that interpreter does not know what ‘it’ refers to, meaning that there is some syntax error related to the word ‘it’. For example:

- `system folder (name of it & pathname of it)`

Returns **It used outside of whose clause** since the interpreter does not know what ‘it’ is, as the statement is formulated incorrectly. The correct statement would be:

- `(name of it & pathname of it) of system folder`

To ensure that ‘it’ points toward an object, you must make sure that you include ‘of <object>’ after any statement involving ‘it’ (but not ‘whose’).

A String Constant Had No Ending Quotation Mark

This message is fairly self-explanatory. It simply means that there was an uneven number of quotation marks in the expression. Here is an example:

- `Version of file “mshtml.dll of system folder`

A String Constant had an Improper % Sequence

You can insert characters into a string by entering a percent sign and then the Ascii hex value of the character. If you use enter a percent sign in a string, the relevance engine looks at the next two characters to see if they correspond to an Ascii hex value. For instance:

- `“I’m telling the %22truth%22”`

Returns **I’m telling the “truth”.** A string with a percent sign in it will return **A string constant had an Improper % Sequence** if any of the following conditions are true:

1. There are less than two characters in the string after the percent sign.
2. Any of the two characters following the percent sign are not standard letters or numbers.

You won’t get the error message if the percent sign is followed by letters or numbers but they don’t correspond to an Ascii value. For instance:

- `“hello said %7 ”`

Will return **A string constant had an Improper % Sequence** since the second character after the percent sign is white space. However, the following command:

- `“hello said %9dgsn”`

Will return **hello said %9dgsn** since even though ‘%9d’ isn’t an Ascii hex value, since both characters are letters and numbers the statement won’t error out.

This Expression Contained a Character Which is Not Allowed

This error message is given when the relevance interpreter finds a character that it does not recognize. You can use any character you want in a string (except “), but outside of a string a random character will break the relevance statement. For instance:

- `{pathname of regapp “besclient.exe”}`

Will return **This Expression Contained a Character Which is Not Allowed** because curly braces are not valid in the relevance language. (Although they do signify relevance substitution in an action script.)

Incompatible Types

There are certain inspectors that look for return values of the same type. If different types are returned, the relevance interpreter returns **Incompatible Types**. The first example is with an if/then/else statement. An if/then/else statement will either return the expression after 'then' or after 'else'. Both of these expressions must return the same object. For instance:

- `If exists regapp "Besconsole.exe" then version of regapp "Besconsole.exe" else "Not Installed"`

Returns **Incompatible Types.** This is because the 'then' expression returns a version, while the 'else' expression returns a string. Instead you need to make sure that both statements return the same type by converting the version to a string.

- `If exists regapp "Besconsole.exe" then version of regapp "Besconsole.exe" as string else "Not Installed"`

The same issue exists when you create lists with semicolons.

- `running applications; names of recent applications`

This returns **Incompatible Types** since it is trying to create a list made up of running applications and strings.

No Inspector Context

Certain inspectors can only be evaluated by the BigFix Enterprise Client and therefore will not work in QNA. If you try to evaluate one of these in QNA, you will receive **No Inspector Context.** The most common example is:

- `Pending restart`

In general, in order to evaluate statements that return **No Inspector Context** you must define them as retrieved properties in the BigFix Enterprise Console.

Appendix B: Strategies for Fixlets

While this guide gives an overview of the relevance and action lanaguages, sometimes the greatest difficulty in writing a Fixlet message comes from breaking down a complicated issue into pieces and learning how to convert these pieces into relevance statements. This section helps you develop strategies on how to go through this process.

There's also the issue of patches/software updates that require a restart. This section will go over techniques to deal with this situation appropriately.

The information in this section applies to both Fixlets and tasks, but the difference between the objects is relevant. By default, a Fixlet will report 'Fixed' only if its relevance has been falsed after the application of its action. Therefore, you have to make sure that at least one relevance clause will turn to **false** once the action has been run. Tasks, on the other hand, will report 'Fixed' once all the lines of the action have been run. Therefore, you may worry less about the relevance falsifying.

Using relevance to check for a certain Problem

When confronted with a complicated problem, it can sometimes be difficult to see how it directly translates into relevance. The first task is to figure out what are the deterministic characteristics of a problem. As an example, let's study Microsoft Security Bulletin MS02-023. You can find it here:

http://www.microsoft.com/technet/treeview/default.asp?url=/technet/security/bulletin/MS02-032.asp

This bulletin warns of a security vulnerability in Windows Media Player, and advises that all users with Windows Media player apply the patch. Since there are different patches for each version of media player, we'll need to write a separate Fixlet messages for each version. For now, let's try to write a Fixlet message for Windows Media Player for Windows XP. First of all, we need to determine exactly which characteristics cause this problem. After reading the security bulletin, we notice three things we need to check for:

1. The operating system must be Windows XP.
2. Windows Media Player for Windows XP must be installed.
3. The vulnerability can't have been closed. This can happen in two ways:
    a. They installed the patch already.
    b. They installed Windows XP Service Pack 1.

As an intermediate step, lets try converting these requirements into pseudocode, just to practice thinking about problems in terms of relevance.

a. We'll use `name of operating system` and `csd version of operating system` to make sure the Windows XP is installed with no service packs.
b. In order to find Windows Media Player for Windows XP, we can either look for a registry key identifying the product, or we can check for the existence and correct version of its regapp.
c. In order to make sure the patch hasn't already been installed, we can either check the file version of a file that's been updated by the patch (maybe the regapp) or look for evidence of the patch in the registry.

Now, we just have to convert these characteristics into relevance clauses.

1. First, use the `name of operating system` command

    - `name of operating system = “WinXP”`

    There are a few ways to check that there is no service pack:

    1. csd version of operating system != “Service Pack 1”
    2. csd version of operating system = “”
    3. csd version of operating system does not contain “Service Pack”
    4. not (csd version of operating system >= “Service Pack 1”)
    5. not exists csd version of operating system

    Again, this is partly a matter of style, but the different options aren't exactly the same. First of all, it is highly possible that the user will at some point in the future upgrade to Service Pack 2 (even though it might not exist yet). The first option will revert to TRUE once they upgrade past Service Pack 1, which isn't the desired behavior. Another thing to worry about is if Microsoft has released a version of Windows XP with that will return strange non-standard CSD values. For example, this may happen for a beta release of an operating system of service pack. Unfortunately, this is hard to know because you do not have every release of Windows XP to test with. In cases like these, you should try to use a relevance clause that covers the most possible cases, but will not cause a problem on potential non-standard responses.

2. There are a few ways to check for Windows Media Player for Windows XP. First of all, you could check for a registry key:
    - `Exists key “HKLM\Software\Microsoft\MediaPlayer\8.0” of registry`

Also, you could try finding the regapp. Since there are different versions of Media Player, we'll need to check the version also. Media player for Windows XP is version 8.

- `Exists regapp "wmplayer.exe" AND version of regapp "wmplayer.exe" = "8"`

In this case, either of these statements should work. Under certain conditions, it could be better to check either the regapp or the registry key, but usually it doesn't matter.

3. According to the security bulletin, the patch creates a registry key when it is run, so we can check for that:

   - `not exists key "HKLM\SOFTWARE\Microsoft\Updates\Windows Media Player\wm320920" of registry`

   Alternatively, we could check that files are updated. By extracting the patch, you can find out that it replaces the 'wmplayer.exe' executable with version "8.0.0.4482"

   - `Version of regapp "wmplayer.exe" < "8.0.0.4482"`

   In this case, it seems like either of these clauses would work fine. One thing to consider when checking a file is that the file will not be replaced until the computer is restarted if it is in use. Therefore, if the user is running media player when the patch is run, the relevance will not evaluate to false until the computer restarts. This isn't necessarily a problem, especially with the use of the 'action requires restart command', but it's something to keep in mind.

Ok now let's review our final relevance for this problem:

1. Check for correct operating system:
   - `Name of operating system = "WinXP" AND not exists csd version of operating system`

2. Check for Windows Media Player:
   - `Exists regapp "wmplayer.exe" AND version of regapp "wmplayer.exe" = "8" and version of regapp "wmplayer.exe" < "8.0.0.4482"`

3. Make sure the patch isn't installed:

-
  ```
  not exists key
  “HKLM\SOFTWARE\Microsoft\Updates\Windows Media
  Player\wm320920” of registry
  ```

If you are using BDE to develop the Fixlet, these will be your three Relevance clauses. In terms of actual evaluation, the three causes will bee ANDed together. If you are authoring from the BES Console, you will need to create the whole relevance clause yourself by connecting all the parts with ANDs. Total relevance:

-
  ```
  (Name of operating system = “WinXP” AND not exists
  csd version of operating system) AND (Exists regapp
  “wmplayer.exe” AND version of regapp “wmplayer.exe”
  = “8” and version of regapp “wmplayer.exe” <
  “8.0.0.4482”) AND (not exists key
  “HKLM\SOFTWARE\Microsoft\Updates\Windows Media
  Player\wm320920” of registry)
  ```

Although it may have seemed complicated at first, the final relevance is really quite simple. This is usually the case. It takes a little while to get familiar with the Relevance Language, but with a little experience writing good relevance should be quick and easy. In the fourth section, I’ll give some examples such as this where you can craft your own relevance and actions in response to an issue.

BigFix, Inc.
6121 Hollis Street
Emeryville, California
94608
[t] 510 652-6700
[f] 510 652-6742
[e] info@bigfix.com

Worldwide Sales
[t] 510 652-6700 x116
[f] 510 652-6742
[e] sales@bigfix.com