BigFix® Relevance Interpreter 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.

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