BigFix Action Language Reference

A Guide to the BigFix® Action Shell Commands for the BigFix Enterprise Suite (BES)

BigFix, Inc.

Emeryville, CA

Last Modified: September 8, 2007
Compatible with
BES Version 7.0

All inquiries regarding the foregoing should be addressed to:
BigFix, Inc.
1480 64th Street, Suite 200
Emeryville, CA 94608

Contents

Preface

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 examples of actions and expressions in the Relevance Language.
bold Italics	Bold Italics are used for the titles of manuals and other cited literature.
[brackets]	Brackets are used to indicate *optional* items in an expression. For instance, [of <parameter>] means that an "**of"** statement may be included in the expression at your option.
{braces}	Braces indicate a substitution syntax. Items in braces are evaluated and the result is substituted in the action expression. The braces are literal, that is, they should be typed in the expression.
<angle bracket>	Angle brackets are used to indicate action parameters. These can be static or formed from substituted relevance expressions.

Examples

A mono-spaced font denotes examples of Actions as used in a Fixlet message:

- `delete "c:\updates\q312456.exe"`

Introduction

This manual details the properties and operations of the BigFix Action Shell Commands. After a Fixlet message locates a potential problem on a computer, it may offer to fix the problem by executing an **Action Shell Command**. This allows the user to quickly cure the problem, often with a single mouse-click.

This manual lists all the BigFix Action Shell Commands, with examples of each.

Many Action Shell Commands allow or require parameters. Those parameters can either be hard-coded (static) values or expressions that are evaluated by the BigFix relevance language. These “substitution variables” endow actions with great power and flexibility.

This guide is current for BigFix Enterprise Suite (BES) version 7.0 for both Unix and Windows. At the bottom of each topic is a version number, such as BES 5.1 and above. This represents the first version that is compatible with the given topic.

Using Action Scripts

Creating Action Scripts

You can create custom actions to fix problems or address issues across your network that are not covered by the standard content. Although the process is simple to describe, there are a large range of actions and targeting techniques at your disposal. To create a custom action:

1 Log on to the BigFix Console as a Master Operator.

2 Select **Tools > Take Custom Action**.

3 The **Take Action** dialog pops up. At the top is a place to provide a **Name** for your custom action. This field can be sorted and filtered, so a good naming convention will let you get the most out of your reports.

4 Under the Name field is the **Preset** pull-down menu that allows you pick a preset customized action, saving you time and ensuring accuracy. You can also save your current input as a preset for later use. The Preset interface includes these fields and buttons:

- **Preset:** Select a preset from the pull-down menu.
- **Show only personal presets:** Filter the list of presets to just your personal ones.
- **Save Preset:** Save the current set of action options for later use. This button isn't active until you make a change to one of the options somewhere in this dialog. When you click this button, a dialog pops up prompting you for the name of your preset. A check box below that lets you save it as a public or private preset.
- **Delete Preset:** Removes this preset from the selectable list. It brings up a confirmation dialog allowing you to cancel this command.

5 Under the Presets, there are several tabs:

- **Target:** Select the targets from the provided list, or use properties or a specific list of computers to target the action.
- **Execution:** Specify the deployment options and constraints, including repeated application and failure recovery.
- **Users:** Determine how this Action will respond to the presence or absence of users.
- **Messages:** Provide a message to precede and accompany the Action.
- **Offer:** Create an Action offering, allowing the user to choose whether or not to apply the Action.

- **Post-Action:** Describe what actions need to be done to complete the action, including restarts or shutdowns.
- **Applicability:** Allows you to override the original Action relevance.
- **Success Criteria:** Create specific criteria that you can use to determine if your Action was successful.
- **Action Script:** This tab allows you to create or modify an action script.

6 Click on the **Action Script** tab and type in your script. This guide will help you with a description of the Action commands and multiple examples.

7 Click on the **Applicability** tab if you would like to fine-tune the targeting of your action script. For more information about the Relevance language, see the ***BigFix Relevance Language Reference*** and the ***BigFix Inspector Guides***.

8 Click on the **Execution, Users, Messages, Offer** or **Post-Action** tabs to further customize your action.

9 When you are ready to deploy your custom action, click **OK**.

10 Your custom action will be distributed to all the computers that have been selected or targeted. The actions will be applied using whatever constraints and schedules you've specified.

You can also create actions when you **Create Tasks** or **Create Fixlets**. See the BES manual for more information on these topics.

Using Substitution

Substitution allows the Fixlet author to include relevance expressions in an Action. This is accomplished by placing the relevance expression in curly braces:

- `run "{pathname of regapp "excel.exe"}"`

  This example runs a program without knowing where it is located. A relevance expression evaluates the pathname automatically using the ‘regapp’ inspector.

- `pause while {exists running application "c:\updater.exe"}`

  This action pauses until a program finishes executing, using the ‘running application’ inspector.

Substitution is not recursive, although any particular command may have one or more expressions to evaluate before execution. The BigFix application is expecting to find a single expression inside the curly braces. If it sees another left brace before it encounters a closing right brace, it treats it as an ordinary character:

- `echo {"a left brace: {"}`

  would send this string to output:

  a left brace: {

  Therefore no special escape characters are necessary to represent a left brace. To output a literal right brace without ending the substitution, use a double character:

- `echo {"{a string inside braces}}"}`

  would send this string to output:

  {a string inside braces}

  Or consider this example:

- `appendfile {{ name of operating system } {name of operating system}`

  When this example is parsed, the double left braces indicate that what follows is not a relevance expression. Only a single right brace is necessary when it's outside of a relevance expression (inside a relevance expression, a double right brace is necessary to specify a literal one). This would output the following line to __appendfile:

  { name of operating system } WinXP

File System Commands

copy

Copies the source file to the named destination file. An action 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

copy <Source_FileName> <Destination_FileName>

Where **Source_Filename** and **Destination_Filename** are the names of the files to copy from and to respectively (typically enclosed in quotes).

Examples

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

  This command copies the win.com file to the bigsoftware folder.

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

  This pair of Action Shell Commands deletes the target file (if it exists) before it performs the copy action.

BES 5.1 and above

move

Moves the source file to the named destination file. This command also gives the action author the ability to rename a file. An action script with the move command terminates if the destination already exists, if the source file doesn't exist, or if the move fails for any other reason.

Syntax

move <Source_FileName> <Destination_FileName>

Where **Source_Filename** and **Destination_Filename** are the names of the files to move from and to respectively (typically enclosed in quotes).

Examples

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

  This command moves and renames the mod.dll file. Note that quotes are necessary for file names and folder names with spaces in them.

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

  The command lines above first delete the file, then move it back in place from another location.

BES 5.1 and above

---

delete

Deletes the named file. Any Action 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 action script will continue to execute.

Syntax

`delete <FileName>`

Where **FileName** is the name of the file to delete (typically enclosed in quotes). Relevance substitution is performed on the arguments of delete action command lines.

Examples

- ```
  delete "c:\program files\bigsoftware\module.dll"
  ```

- ```
  delete "{name of drive of windows folder}\win.com"
  ```

These examples delete the specified files. Note that you can use variable substitution (in curly brackets) to specify pathnames.

Note

It’s good practice to enclose filenames in quotes to preserve spaces in the filenames. Without quotes, the file system will not be able to access those files with spaces in the path or file name.

BES 5.1 and above

download

Downloads the file indicated by the URL. This command is included for backward compatibility for the BigFix Client Edition, version 2.0 and it continues to be supported in BES 6.0 to properly handle existing BES actions. For all other applications, this command has been superseded by the **download as** and **download now as** commands.

After downloading, the file is saved in a folder named "__Download" (the folder name begins with two underscores) relative to the local folder of the Fixlet Site that issued the download command.

If the download fails, the action 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.

The filename is derived from the URL. Everything after the final '/' or '\' character is considered to be the filename. This may occasionally generate a problematic filename, for instance:

URL: http://skdkdk.ddddd.com/cgi-bin/xyz?jjj=yyy

results in a file named xyz?jjj=yyy, which is not a valid filename. You can usually work around this inconvenience by adding a dummy argument to the end of the URL:

http://skdkdk.ddddd.com/cgi-bin/xyz?jjj=yyy?file=/ddd.txt

which will result in a file named ddd.txt being saved to the __Download directory. The **download as** and **prefetch** commands can also be used to address this situation.

Syntax

`download [option] <File_URL>`

Where the **[options]** preface can be one of two optional keywords:

open: calls the ShellExecute API, passing the resulting filename once the download completes.

now: tells the BES agent to start the download at that point in the execution of the action, as opposed to pre-fetching it before the action begins. The agent will attempt to collect the download directly from the specified URL instead of going through the relay system.

The **File_URL** is the location of the file to download.

Examples

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

  Prefetches the bfxxxx.exe file from the BigFix site, and directs the downloaded file to the default site “__Download” folder.

- `download open http://download.bigfix.com/update/bfxxxx.exe`

  Prefetches and saves the bfxxxx.exe file to the default site “__Download” folder and executes the program once the download completes.

- `download now http://download.bigfix.com/update/bfxxxx.exe`

  Downloads the bfxxxx.exe file from the BigFix site as soon as the command is executed.

- ```
  download "http://download.microsoft.com/download/prog.exe"
  run "__Download\prog.exe"
  ```

  This set of actions automates the download process, reducing the application of an executable patch to a single click. Note that the downloaded program is run from the ‘__Download’ directory of the Fixlet site, where the download command places it. The Fixlet site directory is the working directory for all commands and the __Download directory is located there.

Note:

Relevance substitution is **NOT** performed on the **download** action command lines. This is because these actions are scanned by other components that deliver the downloads and these other components run on different machines which do not share those client's evaluation context. This restriction, however, allows BES to prefetch downloads through a relay hierarchy to the clients.

BES 5.1 and above
  ```

download as

Downloads the file indicated by the URL and allows you to rename it. After downloading, the file is saved in a folder named "__Download" (the folder name begins with two underscores) relative to the local folder of the Fixlet Site that issued the **download as** command.

For instance, consider the command:

- `download as intro.txt 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 intro.txt. If the download fails, the action script terminates.

This command, when accompanied by a **continue if** with an sha1 value, allows the file to be pre-fetched.

Syntax

`download as <name> <url>`

Where **name** is a simple filename, without special characters or path delimiters. If the name violates any of the following rules, the download command will fail:

- Name must be 32 characters or less.
- Name must only be composed of ASCII characters a-z, A-Z, 0-9, -, _, and non-leading periods.

Here **url** is the complete URL of the specified file.

Examples

- `download as myprog.exe http://www.website.com/update/prog555.exe`

Downloads the prog555.exe file from the specified folder on the web site, directs the downloaded file to the action site "__Download" folder and renames it to myprog.exe.

- ```
download as patch1 http://www.download.windowsupdate.com/msdownload/update/v3-19990518/cabpool/q307869_f323efa52f460ea1e5f4201b011c071ea5b95110.exe
continue if {(size of it = 813160 and sha1 of it = "92c643875dda80022b3ce3f1ad580f62704b754f") of file "patch1" of folder "__Download"}
```

Downloads the specified file, renames it patch1 and continues only if the size and sha1 are correct.
```

Note:

Relevance substitution is **NOT** performed on the **download as** action command lines. This is because these actions are scanned by other components that deliver the downloads and these other components run on different machines which do not share those client's evaluation context. This restriction, however, allows BES to prefetch downloads through a relay hierarchy to the clients.

BES 6.0 and above -- Windows Only

---

download now as

Downloads the file indicated by the URL and allows you to rename it. After downloading, the file is saved in a folder named "__Download" (the folder name begins with two underscores) relative to the local folder of the Fixlet Site that issued the **download now as** command.

If the download fails, the action script terminates.

For instance, consider the command:

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

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

Syntax

`download now as <name> <url>`

Where **name** is a simple filename, without special characters or path delimiters. If the name violates any of the following rules, the download command will fail:

- Name must be 32 characters or less.
- Name must only be composed of ASCII characters a-z, A-Z, 0-9, -, _, and non-leading periods.

Here **url** is the complete URL of the specified file.

Examples

- `download now as myprog.exe http://www.website.com/update/prog555.exe`

  Immediately downloads the prog555.exe file from the specified folder on the web site, directs the downloaded file to the action site "__Download" folder and names it myprog.exe.

- ```
  download now as patch2 http://www.download.windowsupdate.com/msdownload/update/v3-19990518/cabpool/q310507_2f3c5854999b7c58272a661d30743abca15caf5c.exe
  continue if {(size of it = 845416 and sha1 of it = "c964d4fd345b6e5fd73c2235ec75079b34e9b3d2") of file "patch2.exe" of folder "__Download"}
  ```

  Immediately downloads the specified file from the web site, directs the downloaded file to the action site "__Download" folder and names it patch2. The action continues only if the size and sha1 are correct.

Note:

Relevance substitution is **NOT** performed on the **download now as** action command lines. This is because these actions are scanned by other components that deliver the downloads and these other components run on different machines which do not share those client's evaluation context. This restriction, however, allows BES to prefetch downloads through a relay hierarchy to the clients.

BES 6.0 and above -- Windows Only

---

prefetch

The prefetch command allows a file to be downloaded before the action begins. You do not need a matching **continue if** statement for the file to be downloaded and checked in advance. The prefetch command is preferred over the **download** command.

For instance, consider the command:

- `prefetch a.exe sha1:0123456789012345678901234567890123456789 size:11723 http://x/z.exe`

  The action example above prefetches the z.exe file from the specified site and automatically saves it in the local __Download folder as a.exe.
  ```

Syntax

prefetch <name> sha1:<value> size:<value> <url>

Where **name** is a simple filename, without special characters or path delimiters. If the name violates any of the following rules, the prefetch command will fail:

- Name must be 32 characters or less.
- Name must only be composed of ASCII characters a-z, A-Z, 0-9, -, _, and non-leading periods.

Here **sha1:value** represents the secure hash algorithm value, **size:value** represents the size of the file in bytes and **url** represents the location of the site, including the filename.

Example

- ```
  prefetch patch3 sha1:92c643875dda80022b3ce3f1ad580f62704b754f size:813160
  http://www.download.windowsupdate.com/msdownload/update/v3-19990518/cabpool/
  q307869_f323efa52f460ea1e5f4201b011c071ea5b95110.exe
  ```

  This line of code prefetches the given file from the specified folder on the web site, directs the downloaded file to the action site "__Download" folder and renames it to patch3.

- ```
  if {name of operating system = "WinXP"}

    prefetch patch.exe sha1:92c643875dda80022b3ce3f1ad580f62704b754f size:813160
    http://www.download.windowsupdate.com/msdownload/update/v3-19990518/cabpool/
    q307869_f323efa52f460ea1e5f4201b011c071ea5b95110.exe

  else

    prefetch patch.exe sha1:c964d4fd345b6e5fd73c2235ec75079b34e9b3d2 size:845416
    http://www.download.windowsupdate.com/msdownload/update/ v3-
    19990518/cabpool/q310507_2f3c5854999b7c58272a661d30743abca15caf5c.exe

  endif

  utility __Download\patch.exe

  wait __Download\patch.exe
  ```

  This code prefetches a file based on the operating system, saves the file to the utility cache as patch.exe and waits for its completion to continue the action.

Note:

Relevance substitution is **NOT** performed on the **prefetch** action command lines. This is because these actions are scanned by other components that deliver the downloads and these other components run on different machines which do not share those client's evaluation context. This restriction, however, allows BES to prefetch downloads through a relay hierarchy to the clients.

BES 6.0 and above -- Windows Only

utility

The utility command can be used to place commonly used programs into a special cache. As an example:

- `utility __Download/RunQuiet.exe`

This places the common **RunQuiet** program into the utility cache to avoid downloading it multiple times.

The 6.0 clients maintain two disk caches; one for utility programs and another for action payloads. Files arriving in the action payload cache will not push files out of the utilities cache and vice versa.

The 6.0 clients use the sha1 value of an action download to locate any matching utility (such as 'RunQuiet') that already exists on the client.

An action-specific folder is created to contain downloads as they are pre-fetched. Existing files that match the sha1 values don't need to be downloaded again. Any other files will be pre-fetched from the parent relay. When all the downloads are available on the client, the files will be moved from the action-specific folder (this is a change from pre 6.0 client behavior) to the __Download folder of the action site and the action will be started.

When the action completes, any files left in the __Download folder that were pre-fetched with sha1 will be candidates for utility caching. These files will have their sha1 values re-computed and any files with matching sha1 values can be moved into the utility cache.

A least-recently used scheme is used to keep the cache within its size limits. For short intervals only, the cache limit may be exceeded by single files.

Syntax

`utility <pathname>`

Example

- ```
  prefetch patch.exe sha1:92c643875dda80022b3ce3f1ad580f62704b754f__size:813160
  http://www.download.windowsupdate.com/msdownload/update/v3-
  __19990518/cabpool/q307869_f323efa52f460ea1e5f4201b011c071ea5b95110.exe
  utility __Download\patch.exe
  wait __Download\patch.exe
  ```

This example prefetches a file, saves the file to the utility cache as patch.exe and waits for its completion to continue the action.

BES 6.0 and above -- Windows Only
  ```

appendfile

The appendfile command creates a text file named **__appendfile** in the site directory (by default C:\Program Files\BigFix__Data\<site name>). Each time you invoke the command, it appends the specified text to the end of the file. This command may be useful for creating diagnostic files or dynamically building files that incorporate attributes of the end-user's machine. This file is automatically deleted when the Action Shell Commands begin.

Syntax

```
appendfile <text>
```

Where **text** represents information to be placed in the file.

Examples

- `appendfile This file will contain details about your computer`
- `appendfile Operating System={name of operating system}`
- `appendfile Windows is installed on the {location of windows folder} drive`

  The above commands record the OS and Windows location in the append file

- `appendfile {("Disk " & name of it & ", free space=" & free space of it as string) of drives}`

  The above example records the name and the free space available for all the drives on the client PC.

Note

Use the **appendfile** command as part of an action that builds a script which is subsequently passed to a script interpreter. For example, you can use the following syntax to create an .ini file using BigFix Action commands:

- ```
  appendfile [HKR]
  appendfile HostBasedModemData\Parameters\Driver,ModemOn,1,00,00
  delete {location of system folder}\smcfg.ini
  copy __appendfile {location of system folder}\smcfg.ini
  run smcfg
  ```

  This same technique can be used to build .bat files, .cmd files, visual basic scripts, bash shell scripts, etc.

BES 5.1 and above
  ```

createfile until

This command creates a text file named **__createfile** in the site directory. It allows you to fill a file with a series of statements up to a terminating string. The form of the command is as follows:

```
createfile until <end-delim-string>
 line 1
 line 2
 ...
end-delim-string
```

NOTE: make sure that the lines labeled 'line 1, line 2, …' do not unintentionally contain the end-delim-string. If they do, the action parser will begin looking for action commands after the first instance of the end-delim-string.

Syntax

```
createfile until <delimiter>
  statements
delimiter
```

Examples

```
parameter "config" = "{pathname of folder (value of variable "tmp" of
environment)}\config.txt"
createfile until end
 Operating system = {name of operating system}
 Processor count = {number of processors}
end
delete "{parameter "config"}"
copy __createfile "{parameter "config"}"
```

Defines a parameter named "config" that contains the pathname of a config file in the tmp folder, creates a new name=value file containing the operating system and processor count, deletes the config file from the tmp folder and replaces it with the new file.

BES 6.0 and above -- Windows Only

archive now

This command invokes the Archive Manager. If the Archive Operating Mode is set to manual, this command will trigger archiving and uploading of the configured set of files. To set the appropriate archive mode to manual, use this setting:

_BESClient_ArchiveManager_OperatingMode = 2

The **archive now** command will return a status of Failed if the operating mode is not set to manual. It will also return Failed if an existing archive is currently being uploaded.

Syntax

```
archive now
```

Examples

- `archive now`

  This command initiates archiving and uploading of the configured set of files.

BES 5.1 and above

extract

Extracts files from the specified archive in the download folder (__Download) and leaves the results in the same folder.

An archive file is similar to a compressed tar file. BigFix uses a tool called Archivewriter to construct the archive. This can be useful for copying an entire directory to a computer, which is often required by installers that contain multiple files along with a setup executable. There is a wizard in the BES Console that facilitates the distribution of directories that use this kind of archive.

Syntax

```
extract <Archive File>
```

Examples

- `extract InstallMyApp`

  Extracts the constituent files of InstallMyApp in the __Download folder, places the results back in the __Download folder and deletes the original InstallMyApp file.

BES 5.1 and above

relay select

The relay select command forces the BES Client to select the nearest BES Relay if one is available. This command issues a request to the client to perform a relay selection at the next opportunity and always succeeds immediately, regardless of the success or failure of the pending relay selection.

Syntax

```
relay select
```

Examples

- `relay select`

  This command instructs the BES Client to search for and connect to the nearest relay.

BES 5.1 and above

Setting Commands

setting

Settings are named values that can be applied to individual Fixlet sites or to client computers. Each setting has a time associated with it. This time stamp is used to establish priority -- the latest setting will trump any earlier ones.

Settings can be created and propagated by BES Console Operators. Settings issued by the BES Console will be tagged with the current time and date. Settings are separated into groups, including one for each site and one for the client. Each group of settings is independent of the others and is persistent on the client.

Settings can also be created by Actions in Fixlet messages, and typically use the substitution {now}, which is evaluated when the action is executed. You can examine these settings and their time stamps with Inspectors such as "effective date of <setting>" (see the Inspector Guides for more information).

Syntax

```
setting "<name>"="value" on "<date>" for client
```

```
setting "<name>"="value" on "<date>" for site "<sitename>"
```

Where **name=value** describes the setting, and **date** is a time-stamp used to establish priority. These can be set for the **client** computers or for a named **site**.

Examples

- `setting "name"="Bob" on "31 Jan 2007 21:09:36 gmt" for client`

  Sets the name variable to Bob on the client machine with a MIME date/time stamp provided by the BES Console when this setting was created. It will supersede any other name setting with an earlier date.

- `setting "preference"="red" on "{now}" for site "color_site"`

  Upon execution of the Action containing this command, {now} is evaluated as a MIME date/time and substituted into the string. This command sets the "preference" variable to "red" for the specified Fixlet site. Note that unless there are multiple sites with the same name, you can specify the site without the full gather URL. You may have a different "preference" setting on each site.

- `setting "time"="{now}" on "{now}" for current site`

  Immediately sets the time variable to the current time on the current site.

- `setting "division"="%22design group%22" on "15 Mar 2007 17:05:46 gmt" for client`

  This example uses %xx to indicate special characters by their hexadecimal equivalent. In this case, %22 encloses the value of the variable in double quotes.

Note

When a client is reset, the effective dates of the settings are removed and any subsequent setting commands will overwrite them. There are several ways that clients can be reset, including computer-ID collisions (most often caused by accidentally including the computer ID in an image that gets copied to multiple systems), changing an Action site masthead to a new server, or instructing the client to collect a new ID.

The Actions that run next will establish a new effective date, but the setting *values* will be the same as before the reset. The values are retained because they contain information such as relay selections. That way, when a deployment reset occurs, you don't have to issue new actions to reset your network relay structure.

BES 5.1 and above

---

setting delete

This action deletes a named setting variable on the client computer. It includes a time stamp which will be compared to the time stamp on the original setting. If the delete date is later than the setting date, the setting will be deleted. Otherwise, the delete command will be ignored.

Syntax

```
setting delete "<name>" on "<date>" for client

setting "<name>" on "<date>" for site "<site_url>"
```

Where **name** describes the setting to delete, and **date** is when the setting will be deleted. Settings can be deleted on the **client** computers or on a named **site**.

Examples

- `setting delete "name" on "14 Apr 2007 21:09:36 gmt" for client`

  Deletes the "name" variable on the client machine if the time-stamp is later than the corresponding setting time. Otherwise, the delete command is ignored.

- `setting delete "abc" on "{now}" for site "siteurl"`

  Immediately deletes the "abc" variable on the specified site.

- `setting delete "abc" on "{now}" for current site`

  Immediately deletes the "abc" variable on the current site.

BES 5.1 and above

Registry Commands

regset

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

```
regset "<registry key>" "<value name>"=<value>
```

Where **registry key** is the key of interest and **value name** is the key value to set to **value**. These values are entered just as they are in a REGEDIT4 registry file, in keeping with the rules for Regedit, the Windows program that edits the registry. String values are delimited by quotes, and the standard 4-byte integer (dword) is identified using dword: followed by the numeric value entered in hexadecimal (with leading zeroes) as shown below.

Examples

- ```
  regset "[HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security]"
  "Level"=dword:00000002
  ```

  This example sets the Level value of the specified registry to a double-word 2.

- ```
  regset "[HKEY_CURRENT_USER\Software\BigFix Inc.]" "testString"="bob"
  ```

  This example sets the testString value of the specified registry key to bob.

- ```
  regset "[HKEY_CLASSES_ROOT\ShellScrap]" "AlwaysShowExt"=""
  ```

  This example clears the data of the specified registry value.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

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 REGEDIT4 registry 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 **value** field 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"
  ```

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

  The last example uses the **escape** relevance clause to automatically convert backslashes to double backslashes.

  In situations where you need to issue many regset commands, you might consider using the **appendfile** or **createfile until** commands to build a properly formatted regedit file, and then run regedit silently:

- ```
  Createfile until end-reg-edit-commands
  REGEDIT4

  [HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion]
  "SourcePath1"="c:\\I386"

  "SourcePath2"="{escapes of pathname of windows folder}"

  end-reg-edit-commands

  move __createfile setup.reg

  wait regedit /s setup.reg
  ```

  If the specified key doesn't already exist, it will be created by this command.

BES 5.1 and above -- Windows Only

---

regdelete

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

`regdelete "<registry key>" "<value name>"`

Where **registry key** is the name of the key and **value name** is the value in the registry key you wish to delete.

Example

- `regdelete "[HKEY_CLASSES_ROOT\ShellScrap]" "NeverShowExt"`

  Deletes the NeverShowExt value from the specified registry key.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.
  ```

In order to delete a non-empty registry key and all its sub-keys, you need to create a file, say del.reg, that looks like this:

```
REGEDIT4
[-HKEY_CURRENT_USER\keep\removethisandbelow]

```

There should be three lines in this file: the last line must be a blank. Note the dash (-) in front of the registry path. Now you can execute an action like this:

- `regedit /s del.reg`

When this action is executed, the key named removethisandbelow, along with all its sub-keys, is deleted. You can use the **appendfile** command to build this .reg file.

If the specified key doesn't already exist, it will be created by this command.

BES 5.1 and above -- Windows Only

Execution Commands

dos

Issues a standard DOS command. If the DOS command fails, the action script that contains it is terminated.

Syntax

dos <DOS command line>

Example

- `dos rmdir /Q /S "{pathname of windows folder & "\temp\BigFixQ"}"`

  This example deletes an empty directory from a temporary folder in the windows directory.

- `dos scandisk.exe e:`

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

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

On a Windows system, this has the same effect as issuing a system (Dos command line syntax) statement from the Windows API. It is also the same as typing the DOS command line to a DOS prompt. The DOS command uses the PATH environment variable to try to locate the command on the user's hard drive. As with any other DOS command, for other locations you must specify a complete pathname.

Be sure to use quotes if you have spaces in the filenames.

BES 5.1 and above -- Windows Only

run

Executes the indicated program. If the process can't be created, the action script is terminated. Run does not wait for the process to terminate before executing the next line of the action script. The command line contains the name of the executable and may optionally contain parameters. If you wish to wait for one program to finish before starting another one, use the **wait command**.

Syntax

```
run <command line>
```

Examples

- `run "{pathname of regapp "wordpad.exe"}"`
- `run "c:\winnt\ftp.exe" ftp.mycorp.net`
- `run wscript /e:vbs x.vbs arg1 arg2`

These examples show how you might run a script and pass it some arguments. Quotes around the command line are recommended, and necessary if there are spaces in file names.

Note

On a Windows computer, this command has the same effect as calling the CreateProcess API with <command line>. This is also the same as using <command line> in the Windows RUN dialog. See the Windows documentation on CreateProcess for a discussion of the method used to locate the executable from a <command line>.

BES 5.1 and above

rundetached

Rundetached is used to prevent pop-up DOS windows when you execute a program. It's the same as the **run** command, but the process created doesn't access the parent's console, which inhibits the distracting DOS window. Rundetached should not be used for running interactive programs. If this is done, the interactive program will not be able to show its user interface and may appear to be hung. This command is provided strictly for running programs that do not display a user interface.

Syntax

```
rundetached <command line>
```

Examples

- `rundetached "{pathname of regapp "background_app.exe"}"`
- `rundetached "c:\winnt\ftp.exe" ftp.filesite.net`

These examples show how you might run a program and pass it some arguments. Quotes around the command line are recommended, and necessary if there are spaces in file names.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

On a Windows computer, this command has the same effect as issuing a CreateProcess (CommandLine) statement from the 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.

BES 5.1 and above -- Windows Only

---

runhidden

This command uses CreateProcess to launch a command in a hidden window. It hides the window by setting the STARTUPINFO dwFlag to STARTF_USESHOWWINDOW and setting wShowWindow to SW_HIDE. After launching, the next action command line is executed. To wait for the launch to complete before continuing the action, use the **waithidden** command.

Syntax

`runhidden <command line>`

Examples

- `runhidden "{pathname of regapp "wordpad.exe"}"`
- `runhidden "c:\winnt\ftp.exe" ftp.mycorp.net`
- `runhidden wscript /e:vbs x.vbs arg1 arg2`

These examples show how you might run a script in a hidden window and pass it some arguments. Quotes around the command line are recommended, and necessary if there are spaces in the file names.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

If the launched process requires user input, it will wait for it with its window hidden, unless the command explicitly shows its window.

On a Windows computer, this command has the same effect as calling the CreateProcess API with <command line> and setting the flags to hide the window. See the Windows documentation on CreateProcess for a discussion of the method used to locate the executable from a <command line>.

BES 6.0 and above -- Windows Only

---

wait

The wait command behaves the same as the **run** command, except that it waits for the completion of the process or program before continuing.

Syntax

`wait <command line>`

Example

- `wait "scandskw.exe"`

  Runs the scandskw program and waits for the program to complete before continuing with the Action script. The use of quotes is recommended practice, and necessary if there are spaces in the file name.

Note

On a Windows computer, this has the same effect as issuing a CreateProcess <command line> statement from the Windows API, and then waiting for completion.

BES 5.1 and above

waitdetached

Waitdetached is used to prevent pop-up DOS windows when waiting for a program to complete. It's the same as the **wait** command, but the process created doesn't access the parent's console, inhibiting the distracting DOS window. Rundetached should not be used for running interactive programs. If this is done, the interactive program will not be able to show its user interface and may appear to be hung. This command is provided strictly for running programs that do not display a user interface.

Syntax

```
waitdetached <command line>
```

Example

- `waitdetached "scandskw.exe"`
- `waitdetached wscript /e:vbs x.vbs arg1 arg2`

  This example shows how you might run a script, pass it some arguments and then wait for its completion before continuing the Action script.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

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

BES 5.1 and above -- Windows Only

waithidden

This command is similar to the **runhidden** command and uses CreateProcess to execute a command in a hidden window. It hides the window by setting the STARTUPINFO dwFlag to STARTF_USESHOWWINDOW and setting wShowWindow to SW_HIDE. This action waits for the completion of the process before continuing with subsequent action commands.

Syntax

```
waithidden <command line>
```

Examples

- `waithidden "{pathname of regapp "notepad.exe"}"`
- `waithidden "c:\winnt\ftp.exe" ftp.myurl.net`
- `waithidden wscript /e:vbs x.vbs arg1 arg2`

These examples show how you might run a script in a hidden window and pass it some arguments. Quotes around the command line are recommended, and necessary if there are spaces in the file names.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

If the launched process requires user input, it will wait for it with its window hidden, unless the command explicitly shows its window.

On a Windows computer, this command has the same effect as calling the CreateProcess API with <command line> and setting the flags to hide the window. See the Windows documentation on CreateProcess for a discussion of the method used to locate the executable from a <command line>.

BES 6.0 and above -- Windows Only

script

Not to be confused with an action script, the script keyword executes an external script (created for a scripting language like JavaScript or Visual Basic) with the given name. The action script containing the script keyword will terminate if the appropriate scripting engine is not installed or if the script cannot be executed. The next line of the Action Shell Command is not executed until the specified script terminates.

Syntax

```
script <script name>
```

Example

- `script attrib.vbs`

  Runs the Visual BASIC script attrib.vbs.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

On a Windows computer, this command has the same effect as issuing a wscript "scriptName" statement from Windows, and then waiting for completion. This is also the same as using scriptName from the Windows RUN dialog. If you need to pass parameters to your script, use the **run** command instead.

BES 5.1 and above -- Windows Only

notify client ForceRefresh

This command is equivalent to right clicking on a Client computer in the BES Console and selecting **Send Refresh**. This command may be necessary if the UDP connection to the BES Client is blocked.

Syntax

```
notify client ForceRefresh
```

BES 6.0.14 and above

application launch preference low-priority

When this command is executed, subsequent Action commands that launch programs will do so with lower priority than normal. This will help to mitigate the impact of large patches or service pack upgrades.

low-priority preference only effects the launch priority of applications launched from the current action. This preference is maintained until the action completes or the client executes the **action launch preference normal-priority** command.

Syntax

```
application launch preference low-priority
```

Examples

- `application launch preference low-priority`
- `run "{pathname of regapp "background_app.exe"}"`
- `application launch preference normal-priority`

This example lowers the launch priority before running background_app so that it will not dominate the system when it executes. It then sets the priority level back to normal.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

BES 6.0 and above -- Windows Only

application launch preference normal-priority

When this command is executed, subsequent Action commands that launch programs will do so with normal-priority. This statement is only needed to return the priority to normal after an **application launch preference low-priority** command.

Syntax

```
application launch preference normal-priority
```

Examples

- `application launch preference low-priority`
- `run "{pathname of regapp "background_app.exe"}"`
- `application launch preference normal-priority`

This example lowers the launch priority before running background_app, then returns the priority to normal for subsequent launch statements.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

BES 6.0 and above -- Windows Only

Comments

double forwardslash

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

Syntax

```
//
```

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"
  ```

  The double slashes allow you to comment your action scripts.

BES 5.1 and above
  ```

Flow Control Commands

if, elseif, else, endif

The if, elseif, else and endif commands allow conditional execution of your Action commands. These conditional statements operate on expressions in curly brackets as in the following schematic:

```
if {EXPR1}
    statements to execute on EXPR1 = TRUE
elseif {EXPR2}
    statements to execute on EXPR1 != TRUE and EXPR2 = TRUE
else
    statements to execute when EXPR1 != TRUE and EXPR2 != TRUE
endif
```

In the action schematic above, if the expression in curly brackets following the **if** statement is true, the following statements (up to the endif statement) are evaluated. **If** blocks can be nested any number of levels deep.

Normal **if** block semantics are enforced. All statements up to an **endif**, **elseif** or **else** constitute a block. The **elseif {EXPR}** and **else** statements are optional. Any number of **elseif** statements may be used, but only one trailing **else** block.

A note about prefetching:

The BES Client parses Actions before it actually executes them, looking for downloads to prefetch. If the prefetching process doesn't parse properly, an **action syntax** error will be returned and the Action will not run. This can be problematic if you are creating Actions that work in multiple environments where only one branch of an **if** statement may parse correctly. For instance, you might want to load files that are unique to specific platforms. A script like this would seem to do the trick:

```
if {not exists key "foo" of registry}
    prefetch windows_file ...
else if {not exists package "bar" of rpm}
    prefetch unix_file ...
endif
```

Here a Windows registry key triggers the first prefetch, while a Unix package triggers the second. The problem is that the registry Inspector will fail on Unix systems, and the package Inspector will fail on Windows, causing the prefetch parser to throw an error in both cases.

The answer here is to use cross-platform inspectors (such as "name of operating system") to make sure the wrong blocks are not evaluated:

```
if {name of operating system starts with "Win"}
    if {not exists key "foo" of registry}
        prefetch windows_file ...
    endif
else if {name of operating system starts with "Redhat"}
    if {not exists package "bar" of rpm}
        prefetch unix_file ...
    endif
Endif
```

By checking first for the proper operating system, you can avoid this type of prefetch parse error. However, sometimes there may be no way to avoid a potential error. For instance, an Action may create and access a file that doesn't yet exist in the prefetch phase:

```
wait chkntfs c: > c:\output.txt
if {line 2 of file "c:\output.txt" as lowercase contains "not dirty"}
    regset "HKLM\Software\MyCompany\" "Last NTFS Check"="OK"
else
    regset "HKLM\Software\MyCompany\" "Last NTFS Check"="FAIL"
endif
```

In this Windows example, the output file doesn't exist until the script is actually executed. The prefetch parser will notice that the file doesn't exist when it checks for its contents. It will then throw an error and terminate the Action. However, you can adjust the if-condition to allow the prefetch pass to succeed. One technique is to use the “not active of action” expression which always returns TRUE during the prefetch pass. You can use this to avoid the problematic block during the pre-parse:

```
wait chkntfs c: > c:\output.txt
if {not active of action OR (line 2 of file "c:\output.txt" as lowercase contains "not
dirty")}
    regset "HKLM\Software\MyCompany\" "Last NTFS Check"="OK"
else
    regset "HKLM\Software\MyCompany\" "Last NTFS Check"="FAIL"
endif
```

By checking first to see whether the Action is being pre-parsed or executed, you get a successful prefetch pass and the desired behavior when the action is running.

Syntax

if {<expression>}

<statements>

endif

Example

```
if {name of operating system = "WinME"}
    prefetch patch1.exe sha1:e6dd60e1e2d4d25354b339ea893f6511581276fd size:4389760
http://download.microsoft.com/download/whistler/Install/310994/WIN98MeXP/EN-
US/WinXP_EN_PRO_BF.EXE
    wait __Download\patch1.exe
elseif {name of operating system = "WinXP"}
    prefetch patch2.exe sha1:92c643875dda80022b3ce3f1ad580f62704b754f size:813160
http://www.download.windowsupdate.com/msdownload/update/v3-
19990518/cabpool/q307869_f323efa52f460ea1e5f4201b011c071ea5b95110.exe
    wait __Download\patch2.exe
else
    prefetch patch3.exe sha1:c964d4fd345b6e5fd73c2235ec75079b34e9b3d2 size:845416
http://www.download.windowsupdate.com/msdownload/update/v3-
19990518/cabpool/q310507_2f3c5854999b7c58272a661d30743abca15caf5c.exe
    wait __Download\patch3.exe
endif
```

This code snippet prefetches, renames and downloads a file, based on the operating system.

BES 6.0 and above

parameter

The parameter command can be used to create new action variables during the execution of the action. It takes the form:

- `parameter x = {expression}`

This command allows you to access the parameter using the inspector **parameter "x"**. The parameter is only inspectable within the current action. Parameters are initialized just prior to the startup of the action from headers added to the action by the BES Console.

You can't reset a parameter that already has a value. When this occurs, the client will abort the action at the line that is attempting to reset the parameter. Any errors that result from evaluating the expression will be handled by making the named parameter become undefined.

The rules of the parameter command are:

- Parameter expressions will be coerced into strings.
- Plural expressions that result in no values will result in an empty parameter value.
- Plural expressions that result in a single value that can be coerced into a string will assign the value.
- Plural expressions that result in more than one value will result in a failure of the action.

Syntax

`parameter <x> = <{expression}>`

Where **x** is the name of the parameter and **expression** is the value.

Example

```
parameter "loc" = "{pathname of folder (value of variable "tmp" of environment)}"
createfile until end
   Operating system = {name of operating system}
   Processor count = {number of processors}
end
delete "{parameter "loc"}\config.txt"
copy __createfile "{parameter "loc"}\config.txt"
```

Defines a parameter named "loc" that contains the pathname of the tmp folder, creates a new name=value file containing the operating system and processor count, deletes the config file from the tmp folder and replaces it with new file.

BES 6.0 and above

continue if

This command allows the next line in the script to be executed if the value provided as a parameter evaluates to true. It will stop without error if the specified value evaluates to false. You can use relevance substitution to compute the value. This command is useful in making sure that certain conditions are met to run the remainder of an action. The line number where the action script exited is reported to the console. Users of BES can use this line number to identify why an action is failing if you insert a **continue if** statement that identifies an invariant required by your action.

Syntax

`continue if <true condition>`

Where **true condition** represents a relevance expression to evaluate.

Examples

- ```
  continue if {name of operating system = "Win2k"}
  download now http://www.real-time.com/downloads/win98/dun40.exe
  ```

  This example will download the dun40.exe file only if the operating system is Win 2000.

- ```
  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
  ```

  This example will run the dun40.exe file only if the size and sha1 value are as specified.

BES 5.1 and above

pause while

The action will not continue to the next command while the relevance expression specified evaluates to true. It will continue and execute the next command of the Action as soon as the value evaluates to false or the value fails to evaluate. Use relevance substitution syntax to define the condition.

Syntax

`pause while <true condition>`

Where **true condition** represents a relevance expression to evaluate.

Examples

- `pause while {exists running application "updater.exe"}`
- `pause while {not exists file "C:\70sp3\result.log"}`
- `pause while {not exists section "ResponseResult" of file "C:\70sp3\result.log"}`

BES 5.1 and above

---

action requires restart

This 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 machine, 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 machine is restarted.

Syntax

```
action requires restart
```

Example

- `action requires restart`

BES 5.1 and above

---

action may require restart

When this command is executed, the client looks at the system for telltale signs that a restart is needed. If so, it sets the action completion status such that the action will appear as 'Pending Restart' in the console, until a restart occurs. Once the restart is completed, the action completion status of the action will take on the value of ‘success’ if the relevance of the action is no longer relevant, or ‘failed’ if it is still relevant.

If the telltale signs of restart are not present, the action completion status of the action will take on the value of ‘success’ if the relevance of the action is no longer relevant, or ‘failed’ if it is still relevant.

Syntax

```
action may require restart
```

Example

- `action may require restart`

BES 5.1 and above

action requires login

This command informs the client that the current action will not be completed until the computer is restarted and an administrator logs in. Once this action has been completed on a machine, the inspector **pending login** will return true.

Syntax

```
action requires login
```

Example

- `action requires login`

Note

This Action is ignored by BES Unix agents.

BES 5.1 and above

action parameter query

This 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 (“). Once entered, the user input becomes the default in subsequent invocations (for BES, the user is the console operator approving the action for deployment).

Syntax

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

Where **parameter name** is the name of the relevance parameter and the **with description** option lets you present a prompt to the user. The **and with default** option lets you specify a default value for the parameter.

Examples

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

- `action parameter query "tips" with description "Enter ‘on’ or ‘off’ to control Fixlet tips." With default "on" regset "[HKEY_CURRENT_USER\Software\BigFix]" "tips"="{parameter "tips" of action}"`

Note

The parameter values input by the user may include %xx where xx stands for a two-digit hexadecimal number to specify the character you want to embed. To embed a percent sign, use %25. To embed a double quote, use %22.

While the action is executing, you can retrieve the action parameter value entered by the console operator. For example, in your action you could use relevance substitution: {parameter “parameter name” of action}.

Relevance substitution is **NOT** performed on the **action parameter query** command line itself. This is because the command is interpreted in the BES console before the action is sent out, allowing the Fixlet author to ask the operator for deployment-specific parameters needed to run the action.

BES 5.1 and above

---

set clock

Causes the client to re-register with the registration server, and to sets 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 BES-only command is not available when the client is operating under an evaluation license.

Syntax

```
set clock
```

Example

- `set clock`

BES 5.1 and above

restart

The restart command will restart the computer. If the optional <delay seconds> parameter is provided, the shutdown will happen automatically after the specified delay.

If a user is logged in, a dialog will be displayed that shows the delay counting down. In this case, the interface will have a **Restart Now** button instead of a **Cancel** button.

If the delay parameter is not specified, the user is prompted to press a button to restart the computer.

Syntax

```
restart [<delay seconds>]
```

Where **delay seconds** is an optional parameter to provide a lag before restarting.

Example

- `restart 180`

Restarts the computer in three minutes.

Note

The delayed restart is a forced restart; it will not prompt the user to save changes to documents, etc. The machine will restart without further prompting.

BES 5.1 and above

shutdown

The shutdown command is similar to the **restart** command, but it simply shuts the computer down and does not reboot.

If the optional <delay seconds> parameter is provided, the shutdown will happen automatically after the specified delay.

If a user is logged in, a UI will be displayed that shows the delay counting down. In this case, the UI will have a **Shutdown Now** button instead of a **Cancel** button.

If the delay parameter is not specified, the user is prompted to press a button to shut down the computer.

Syntax

shutdown [<delay seconds>]

Where **delay seconds** is an optional parameter to provide a lag before shutting down.

Example

- `shutdown 180`

This command will shut down the computer in three minutes.

Note

The delayed shutdown is a forced shutdown; it will not prompt the user to save changes to documents, etc. The machine will shut down without further prompting.

BES 5.1 and above

Administrative Rights Commands

administrator add

This command lets you appoint specific people to administer specific BES Clients. This is accomplished by using a setting with an an effective date, passed as a parameter. The date is not optional. The effective date tests are the same as for ordinary **settings**.

Syntax

```
administrator add <administrator name> on <date>
```

Example

- `administrator add "bob" on "21 Aug 2002 17:39:14 gmt"`

  Allows the BES Console operator named bob to have administrative rights on the targeted computer(s), effective on the given date.

BES 5.1 and above

administrator delete

This command allows you to remove administrative rights for the specified administrator. This is accomplished by using a setting with an an effective date, passed as a parameter. The date is not optional. The effective date tests are the same as for ordinary **settings**.

Syntax

```
administrator delete <administrator name> on <date>
```

Example

- `administrator delete "bob" on "21 Aug 2002 17:39:14 gmt"`

  Removes the administrative rights of the BES Console operator named bob, effective on the given date.

BES 5.1 and above

Locking Commands

action lock until

Locks actions from the effective date until the expiration date occurs. The expiration date is MIME time format (as in 19 Jul 2007 12:42:51 -0700). You can use substitution with an Inspector like {now}, which will evaluate the time and insert it into the string.

Syntax

```
action lock until "<expire date>" "<effective date>"
```

Example

- `action lock until "{now + 3*days}" "{now}"`

  Locks actions immediately, unlocking them in three days.

BES 5.1 and above

action lock indefinite

Turns on the action lock, starting on the effective date, which will never expire. The date is in MIME time format (as in 15 Mar 2007 12:42:51 -0700).

Syntax

```
action lock indefinite "<effective date>"
```

Example

- `action lock indefinite "{now}"`

  Turns on the action lock immediately.

BES 5.1 and above

action unlock

Unlocks the client to act upon any actions. The effective date field is used to insure that locking and unlocking actions take place in the order in which they were created. The date is in MIME time format (as in 29 Nov 2008 12:42:51 -0700).

Syntax

```
action unlock "<effective date>"
```

Example

- `action unlock "{now}"`

  Unlocks actions immediately.

BES 5.1 and above

Site Maintenance Commands

site force evaluation

Causes the client to re-evaluate all Fixlet messages for the site. This is useful after updating files or settings, to make sure that the Fixlet's relevance is recomputed for the entire site as soon as possible.

Syntax

```
site force evaluation
```

Example

- `site force evaluation`

BES 5.1 and above

site gather schedule publisher

This command sets the schedule for gathering from the current site to that specified in the masthead for the site.

Syntax

```
site gather schedule publisher
```

Example

- `site gather schedule publisher`

BES 5.1 and above

site gather schedule manual

This command enables manual gathering from the current site. It is ineffective for action sites.

Syntax

```
site gather schedule manual
```

Example

- `site gather schedule manual`

BES 5.1 and above

site gather schedule disable

This command disables scheduled gathering from the current site. It is ineffective for action sites.

Syntax

```
site gather schedule disable
```

Example

- `site gather schedule disable`

BES 5.1 and above

site gather schedule seconds

This command sets the schedule for gathering from the origin site to the number of seconds specified.

Syntax

```
site gather schedule seconds <seconds>
```

Example

- `site gather schedule seconds 360`

  Sets the site gathering schedule to six minutes.

BES 5.1 and above

subscribe

Subscribes the client to the site identified in the masthead file. The BES console provides the **Manage Sites** dialog to automate site addition.

Syntax

```
subscribe "<masthead file name>"
```

Example

- `subscribe "__Download\Sitename.fxm"`

Note

In BES this command returns an error unless it is executed as an action in the master action site. The command is useful for subscribing clients to Enterprise Fixlet sites and for updating the action site masthead file.

BES 5.1 and above

unsubscribe

Automatically unsubscribes from the current Enterprise Fixlet site.

Syntax

```
unsubscribe
```

Example

- `unsubscribe`

BES 5.1 and above

Wow64

action uses wow64 redirection

This command allows the client to get outside of the 32-bit world created for it by the **Windows On Windows64** (Wow64) facility built into the new 64-bit versions of the Windows operating system, including Windows 2003 x64 and Windows XP Pro x64.

When this command is executed in an action on a 64-bit OS with a value of **true**, the client enables Wow64 redirection in any subsequent commands that involve filenames. This state continues until the action completes or the client executes the **action uses wow64 redirection false** command.

You can use **Relevance substitution** to supply the <true|false> value. The file system redirection provided by Wow64 is disabled using the Wow64DisableWow64FsRedirection and re-enabled using the Wow64RevertWow64FsRedirection Windows API.

The commands affected by this setting include:

- dos
- run, wait, rundetached, waitdetached, runhidden, waithidden
- delete, copy, move, open

Syntax

```
action uses wow64 redirection <true|false>
```

Example

- `action uses wow64 redirection true`

  This example turns on Wow64 redirection.

- `action uses wow64 redirection false`

  This example turns off Wow64 redirection.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

BES 6.0 and above -- Windows Only

regset64

Regset64 uses the same syntax as the **regset** command, but places a call to Wow64DisableWow64FsRedirection before launching the 64-bit version of Regedit to set the registry. This allows you to use the native 64-bit registry to set a registry key to the given name and value. If the key doesn't already exist, this command creates the key with this initial value.

Syntax

```
regset64 "<registry key>" "<value name>"=<value>
```

Where **registry key** is the key of interest and **value name** is the key value to set to **value**. These values are entered just as they are in a REGEDIT4 registry file, in keeping with the rules for Regedit, the Windows program that edits the registry. String values are delimited by quotes, and the standard 4-byte integer (dword) is identified using dword: followed by the numeric value entered in hexadecimal (with leading zeroes) as shown below.

Examples

- `regset64 "[HKEY_CURRENT_USER\Software\Microsoft\Office\9.0\Word\Security]" "Level"=dword:00000002`

  This example sets the Level value of the specified registry to a double-word 2.

- `regset64 "[HKEY_CURRENT_USER\Software\BigFix Inc.]" "testString"="bob"`

  This example sets the testString value of the specified registry key to bob.

- `regset64 "[HKEY_CLASSES_ROOT\ShellScrap]" "AlwaysShowExt"=""`

  This example clears the data of the specified registry value.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

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 REGEDIT4 registry files. This syntax is necessary for the RegSet command, but not for registry Inspectors.

When you use the BigFix regset64 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 **value** field 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:

- `regset64 "[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion]" "SourcePath2"="c:\\I386"`
- `regset64 "[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion]" "SourcePath2"={escape of "c:\I386"}`

The last example uses the **escape** relevance clause to automatically convert backslashes to double backslashes.

If the specified key doesn't already exist, it will be created by this command.

BES 6.0 and above -- Windows Only

---

regdelete64

Regdelete64 uses the same syntax as the **regdelete** command, but places a call to Wow64DisableWow64FsRedirection before launching the 64-bit version of Regedit to set the registry, allowing you to use the 64-bit registry available on 64-bit machines. This 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

`regdelete64 "<registry key>" "<value name>"`

Where **registry key** is the name of the key and **value name** is the value in the registry key you wish to delete.

Example

- `regdelete64 "[HKEY_CLASSES_ROOT\ShellScrap]" "NeverShowExt"`

Deletes the NeverShowExt value from the specified registry key.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

If the specified key doesn't already exist, it will be created by this command.

BES 6.0 and above -- Windows Only

script64

Script64 uses the same syntax as the **script** command, but places a call to Wow64DisableWow64FsRedirection before executing the script. This allows you to issue a native 64-bit script command, bypassing Windows 32-bit environment built on top of 64-bit processors.

The script keyword executes an external script (created in a scripting language like JavaScript or Visual Basic) with the given name. The action script containing the script keyword will terminate if the appropriate scripting engine is not installed or if the script cannot be executed. The next line of the Action Shell Command is not executed until the specified script terminates.

Syntax

```
script64 <script name>
```

Example

- `script64 attrib.vbs`

  Runs the Visual BASIC script attrib.vbs in native 64-bit mode.

Notes

This command is Windows-only. It will cause an action script to terminate on a Unix agent.

On a Windows computer, this command has the same effect as calling Wow64DisableWow64FsRedirection and then issuing a wscript "scriptName" statement from Windows.

BES 6.0 and above -- Windows Only

Index

E

F

G

H

I

J

K

L

S

T

U

V

W