Tivoli. *Endpoint Manager*
Version 8.1

Action Guide

IBM®

Note: Before using this information and the product it supports, read the information in Notices.

Contents

Part One

Introduction

After a Fixlet® message identifies a potential problem on a computer, it offers to fix it with a Tivoli Endpoint Manager Shell Command, called an Action script. Although there are other ways to create scripts, the most powerful method is to use the **Tivoli Endpoint Manager Action Language**. This is a guide to that language.

Many Action Commands allow or require parameters. Those parameters can either be hard-coded (static) values or expressions that are evaluated and substituted by the Tivoli Endpoint Manager Relevance language. These are called **substitution variables** and they let you create scripts that are finely targeted and highly flexible. All commands may perform substitution on their arguments before processing them, with a few noted exceptions.

This help file describes all the Tivoli Endpoint Manager Action Commands, with specific examples. At the bottom of each Action topic is a version number, such as Version7.2 and above. This represents the first version that is compatible with the given command. Some actions are marked "Windows Only," and will fail on Unix or Macintosh systems.

Audience

This guide is for Tivoli Endpoint Manager content authors who are interested in creating Fixlet Actions.

Versions

The document includes the functionality introduced in the Tivoli Endpoint Manager Version 8.1.

System Requirements

Tivoli Endpoint Manager Actions can run on computers that meet the following minimum requirements:

- **Hardware:** x86-based computers, Mac or SPARC with 32 MB RAM and 20 MB free hard disk space. Extra temporary disk space may be required for some downloads.
- **Platforms:**
  Windows 2000, XP, Vista, 7, 2, XP-64bit, 2003, 2008, 2008R2
  Red Hat Linux 8,9 x86
  RedHat Enterprise Linux 3/4/5 x86 and x64, RHEL 4 on s390z
  Oracle Enterprise Linux 4,5 x86 and x64
  CentOS 4,5 x86 and x64
  Fedora 1,2,3,4,5,6,7 x86 and x64
  SuSE Linux Enterprise 9,10,11 x86 and x64
  Solaris 8,9,10 SPARC
  Solaris 10 x86
  Mac OSX 10.4/10.5/10.6
  HPUX 11.0 and up
  AIX 5.1 through 6.1 officially
  ESX 3.x and 4.x
  Debian 5 x86/x64
  Ubuntu 8/10 x86/x64

Part Two

Getting Started

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 Tivoli Endpoint Manager 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:** Check this box to 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 ***Tivoli Endpoint Manager Relevance Language Reference*** and the ***Tivoli Endpoint Manager 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 have specified.

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

Introducing the Prefetch Block

Prior to version 7.2, Action scripts containing download commands were processed by scanning the action to deduce which downloads corresponded to which sha1's and sizes. This was error prone because the download commands had to be correlated with **continue if** commands and the information was spread out throughout the action, comingled with other action logic. This made Actions harder to understand than was necessary. The prefetch block resolves these issues.

The prefetch block must be the first entry in the Action script (other than comments or blank lines). It contains all the download prefetch logic needed to prepare for subsequent Action execution. As a consequence, Actions written with a prefetch block are easier to understand. Some of the methods that can be used in a prefetch block include:

- **Literal downloads.** These are ordinary static downloads, which are still available.
- **Conditional downloads.** Only those commands inside TRUE condition pathways are performed.
- **Variable Substitution.** This includes downloads that use relevance substitution to determine which files to collect.
- **Custom logic.** This takes advantage of a plug-in to create download manifests.

Unlike the pre-parsing algorithm used in the traditional downloading actions, prefetch block downloads can be viewed as a top-down approach: the prefetch block comes first and must successfully complete before the rest of the Action can continue. This provides greater control, flexibility and power.

Notes:

Only one prefetch block is allowed per Action. When it is used, the **begin prefetch block** command must be the first executable in the script. Only blank lines and comments are allowed to precede it. An **end prefetch block** command is required for termination.

Version 7.2 and above

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 Tivoli Endpoint Manager Client 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
```

You can also use substitution with **add prefetch item** commands in prefetch blocks:

```
begin prefetch block
  parameter "manifest"="{pathname of file "manifest.spec" of client folder
  of site "AV"}"
  add prefetch item {concatenation " ; " of lines of file (parameter
  "manifest")}
end prefetch block
```

Introducing Dynamic Downloads

A new feature called dynamic downloading has been available since version 7.2 of the Tivoli Endpoint Manager to extend the flexibility of Action scripts. To understand how it works, it is helpful to understand the existing static download method.

Static Downloading

Before it runs an Action, the Tivoli Endpoint Manager Client parses it, looking for download or prefetch commands. Static downloads include the URL, sha1 and size for each item as literal values in the action script. The literal values allows an operator to observe exactly what the action script is going to do. These literals are used to construct a numbered list of downloads associated with the action that is then stored on the Tivoli Endpoint Manager Server. This stage of Action processing is called **prefetch processing**.

As a consequence of prefetch processing, the Client will notify the nearest Tivoli Endpoint Manager Relay of the need for downloads by requesting a URL ending in <actionid>/0, which in turn triggers the Relay to download all the items corresponding to that specified Action. When they are ready, the Relay pings the clients back with the Action ID. All the Tivoli Endpoint Manager Clients running that Action will then collect the files by asking for them one at a time as <actionid>/1, <actionid>/2, etc.

However, because the download information is represented by literal expressions, only those URLs already known when the Action is authored can be represented. This means that static downloads can't be used for those instances where the downloads change, but the Action script remains the same.

Dynamic Downloading

Dynamic downloads add the ability to use relevance clauses to specify downloads. These new commands must be embedded in a special segment of Action code called a **prefetch block**. For instance, if you created a file in the AV Fixlet site named download.spec containing a named variable in the first line such as:

```
name=update.exe sha1=123 size=456 url=http://site.com/download/patch.exe
```

You could then access this patch using relevance substitution in a prefetch block:

```
begin prefetch block
  parameter "downloadFile"="{pathname of file "download.spec" of client
  folder of site "AV"}"
  add prefetch item {line 1 of file (parameter "downloadFile")}
end prefetch block
```

This code block creates a variable named downloadFile that points to a file in the AV site. It then adds this file to the prefetch queue for subsequent downloading. In this way, a Fixlet message in the AV site could offer to keep something automatically updated and the download.spec file would be refreshed whenever a new version became available. Deploying the action from the Fixlet as a Policy Action would then execute the update whenever the download.spec file was changed.

Note that this code block terminates with an **end prefetch block** command, which ensures that the file is successfully downloaded before execution of the Action script. A prefetch block must be at the top of the Action script, and it must be closed with the end prefetch block statement before the script can continue.

Another popular technique is to use a data file, or manifest, containing a list of multiple downloads, each with its own URL, sha1 and size. This manifest can change as often as necessary, making it easy to update spy ware or anti-virus definitions. One way to implement this is to create a file named manifest.spec with a list of downloads such as

```
name=patch1.exe sha1=123 size=456 url=http://site.com/download/patch1.exe
name=patch2.exe sha1=234 size=567 url=http://site.com/download/patch2.exe
name=patch3.exe sha1=345 size=678 url=http://site.com/download/patch3.exe
```

You can then download these patches with a prefetch block that pulls these files from the manifest:

```
begin prefetch block
  parameter "manifest"="{pathname of file "manifest.spec" of client folder
  of site "AV"}"
  add prefetch item {concatenation " ; " of lines of file (parameter
  "manifest")}
end prefetch block
```

You can also use small executables to process files into a fresh manifest. This is accomplished with the **execute prefetch plug-in** command, as the following example illustrates:

```
begin prefetch block
  add prefetch item name=myPlugIn.exe sha1=123 size=456
  url=http://mysite/plugin.exe
  // collect the plug-in before continuing:
  collect prefetch items
  parameter "ini"="{file "prepass.ini" of site (value of setting
  "CustomSite") of client}"
  execute prefetch plug-in "{download path "myPlugIn.exe"}" /downloads
  "{parameter "ini"}" "{download path "manifest"}"
  add prefetch item {concatenation " ; " of lines of download file
  "manifest"}
end prefetch block
```

This prefetch block first adds the plug-in to the prefetch queue and then executes the **collect prefetch items** command. This causes prefetch processing to delay until the items added to the prefetch queue are downloaded before prefetch processing continues. Once successfully downloaded, the plug-in is executed with arguments including the path for the data file and the manifest to be produced from it. The final **add prefetch item** command queues up the downloads specified in the freshly created manifest. A technique like this might also be used to decrypt a secure file into a plain-text manifest.

Dynamic downloads must specify files with the confirmation of a size or sha1. But the URL, size, and sha1 are allowed to come from a source outside of the Action script. This flexibility entails extra scrutiny. Since any client can use dynamic downloading to request a file, it creates an opportunity for people to use your server to host files indiscriminately. To prevent this, dynamic downloading uses a white-list. Any request to download from a URL (that isn't explicitly authorized by use of a literal URL in the action script) must meet one of the criteria specified in a white-list of URLs on the Tivoli Endpoint Manager Server, located at **<BES Server Install Path>\Mirror Server\Config\DownloadWhitelist.txt**. This file contains a newline-separated list of regular expressions using a Perl regex format, such as the following:

```
http://.*\.sitename\.com/.*
http://software\.sitename\.com/.*
http://download\.sitename\.com/patches/JustThisOneFile\.qfx
```

The first line is the least restrictive, allowing any file at the sitename domain to be downloaded. The second line requires a specific domain host and the third is the most restrictive, limiting the URL to a single file named "JustThisOneFile.qfx". If a requested URL fails to match an entry in the white-list, the download immediately fails with status **NotAvailable**. A note is made in the Relay log containing the URL that failed to pass. An empty or non-existent white-list will cause all dynamic downloads to fail. A white-list entry of ".*" (dot star) will allow any URL to be downloaded.

Prefetch blocks allow conditional statements:

```
begin prefetch block
  if {name of operating system = "Windows 2000"}
    add prefetch item name=up.exe sha1=123 size=456
    url=http://site.com/patch2k.exe
  else
    add prefetch item name=up.exe sha1=123 size=456
    url=http://site.com/patch.exe
  endif
end prefetch block
wait "{download path "up.exe"}"
```

This Action script branches on the existence of Win2K, but the downloads in this example are described statically (as literal text). Although the clients will only download the particular items they need, all the static files are downloaded to servers and relays as soon as they are requested. Dynamic downloads can improve on this situation because only those files actually needed by clients are fetched to the server and relay in the first place. Here's an example using dynamic downloading:

```
begin prefetch block
  if {name of operating system = "Windows 2000"}
    add prefetch item {"name=up.exe sha1=123 size=456
    url=http://site.com/patch2k.exe"}
  else
    add prefetch item {"name=up.exe sha1=123 size=456
    url=http://site.com/patch.exe"}
  endif
end prefetch block
wait "{download path "up.exe"}"
```

By using relevance substitution in the prefetch block, with a properly configured white list file on the server, this code only fetches the necessary file, potentially improving bandwidth requirements and efficiency.

You can also branch execution based on the contents of a file, allowing you to automate updates. This can be especially useful for dealing with changing version numbers. For instance, you could create a file named 'manifest.txt' containing two named variables such as:

```
version=1234
download=name=update.exe sha1=123 size=456
url=http://site.com/download/patch.exe
```

Note that the download variable contains the name, sha1, size and URL of the patch file. You can then use relevance substitution to extract these variables with an expression such as:

```
parameter "ver"="{key "version" of file "{download path
"manifest.txt"}"}"
```

```
parameter "filename"={key "download" of file "{download path
"manifest.txt"}"}
```

By comparing the extracted version against some stored value, you can determine if and when you need to download the specified file. This technique can be expanded to include multiple versions and can even be used to distinguish between patches and full replacement updates.

No matter which technique is used, once the files have been downloaded, they can be examined with various Inspectors which may have different interpretations, depending on whether or not the Action is active (or in prefetch processing). Before execution, these files are collected in a prefetch folder. During Action execution, they reside in the __Download folder. There are new Inspectors that can be used to locate the files before or during action execution:

- **download folder:** During the prefetch parsing, this Inspector returns a folder object from the __Global\<sitename>\<actionid>\named folder. Once the Action is active and the download has completed, this Inspector returns the expected folder object from the __Download directory.
- **download path "pathname":** This Inspector returns a string containing the full pathname to the specified file, whether it exists or not. The download filename is equivalent to "(pathname of download folder) & <pathseparator> & filename".
- **download file "filename":** This Inspector returns a file object from the download folder or another named folder. The download filename is equivalent to "file 'filename' of download folder". If the file isn't yet in the download folder, the Inspector returns 'does not exist'.

It is up to the action script author to protect users of these actions and ensure that downloads and their checksums have not been compromised. An end-to-end authentication mechanism resistant to man-in-the-middle attacks is the best defense. When authoring a dynamic download action it is critical to craft the action so that it authenticates information before using it, typically by using a plug-in as described above. It is also wise to explicitly identify those steps in the action script that perform this authentication so that users of your action can audit the mechanism before deciding to trust it.

Notes:

Only one prefetch block is allowed per Action. When it is used, the **begin prefetch block** command must be the first command in the script. Only blank lines and comments are allowed to precede it. An **end prefetch block** command is required to separate the prefetch block from the remainder of the Action.

Version 7.2 and above

Part Three

The Action Commands

Execution Commands

action 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

action launch preference low-priority

Examples

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

Version 6.0 and above -- Windows Only

action 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 **action launch preference low-priority** command.

Syntax

action launch preference normal-priority

Examples

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

Version 6.0 and above -- Windows Only

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"}"
  ```

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.

Version 5.1 and above -- Windows Only

notify client ForceRefresh

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

Syntax

notify client ForceRefresh

Version 6.0.14 and above

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>.

Version 5.1 and above

rundetached

Rundetached modifies the **run** command by setting the DETACHED_PROCESS flag when calling CreateProcess() on Windows machines. By default, a created process inherits its parent's console. When detached, this behavior is inhibited. This gives the new process some more control over how it may interact with the user.

Among other things, this can be 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.

Version 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. The process that is created may modify that flag to subsequently show the window again.

After launching, the following Action command line is immediately 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>.

Version 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.

Version 5.1 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.

Version 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.

Version 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>.

Version 6.0 and above -- Windows Only

Flow Control Commands

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`

Version 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 Tivoli Endpoint Manager Console before the action is sent out, allowing the Fixlet author to ask the operator for deployment-specific parameters needed to run the action.

Version 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 Tivoli Endpoint Manager Unix agents.

Version 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 Tivoli Endpoint Manager Console will report 'Pending Restart' until the affected machine is restarted.

Syntax

action requires restart

Example

- `action requires restart`

Version 5.1 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 the Tivoli Endpoint Manager 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 = 325 and sha1 of it = "013e48a5") 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.

Version 5.1 and above

exit

The exit command terminates the action and returns an integer value to the caller.

```
exit {integer exit code}
```

This command can employ relevance substitution. When it is executed, the value of the integer is transferred to the exit code inspector, and the action is terminated at that line. Exit codes can affect the 'fixed' status of run-to-completion actions by exiting from the script before the last line. If there are no executable lines after the exit command, the action will complete successfully. However, if there are other commands after the exit command, the run-to-completion action will fail.

In the Tivoli Endpoint Manager Console, the value of the last exit command is displayed in the View Action Information dialog, along with other status information.

In a Relevance expression, this value can be evaluated using the 'exit code of <action>' Inspector.

Syntax

exit <{expression}>

Where **expression** is the integer value to be passed back to the caller. This is limited to 32-bit signed numbers, however on Unix, the limit is 8 or 16 bits, which can be determined by running the WIFEXITED macro.

Example

```
wait 'foo'
parameter "error" = "{exit code of action}"
if {parameter "error" != "0"}
  exit {parameter "error"}
endif
// continue processing
```

This example represents a a script that reports errors as non-zero exit codes. It allows you to terminate the script early and report the exit code to the caller. This is one of four script commands (wait, waithidden, waitdetached and exit) that can change the exit code inspector

value. The three wait commands set the exit code according to the executable, with the OS limiting the size of the number. The exit command sets the exit code according to the number passed to it as a signed 32 bit number, regardless of the OS.

You may use the exit command in conjunction with DOS, however DOS can't set the exit code itself because of a limitation of the system command API.

Note for Unix shell scripts: For actions of type 'x-sh', the exit code of the script is collected into the inspector value when the client finishes processing a shell script. Exit codes from UNIX shell scripts are written to the client log.

Version 8.0

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
  sstatements 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 Tivoli Endpoint Manager 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 work:

```
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.

Version 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 Tivoli Endpoint Manager 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. Note that both the name of the parameter and the expression must be inside quotes.

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.

Version 6.0 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"}`

Version 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. Also, when the Client UI is showing, there is a 60 second minimum delay before restarting.

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.

Version 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`

Version 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.

CAUTION

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.

Version 5.1 and above

File System Commands

add nohash prefetch item

Adds a download item to the prefetch queue. This command must reside between a **begin prefetch block** and an **end prefetch block** command. This is a singular command -- it can only specify a single download at a time. Relevance substitution is not allowed, permitting the Tivoli Endpoint Manager Server to cache the download when the Action is created. If the Client requests any ordinal files, the Relay will collect them all. The Client will only download the item if this command is in a TRUE condition block.

Syntax

add nohash prefetch item [name=<n>] [size=<s>] url=<u>

Where:

<n> is an optional name, limited to 32 characters (including alphanumerics, dashes, underlines and non-leading periods). If no name is explicitly specified, the name will be derived from the final component of the URL (following the final slash). Only one download item can be specified per command.

<s> is an optional file size. Although it is not required, when it is known and specified, the program can provide meaningful progress information.

<u> is a required url. If the name is not specified, then it will be derived from final component of the supplied URL.

Relevance substitution is *not* allowed with the arguments of this command. The arguments may be in any order desired, but unrecognized commands will generate a syntax error.

Clients and Relays will collect these files by Action ID and ordinal number. An sha1 is *not allowed* with this prefetch command. To use sha1, use the **add prefetch item** command. This technique is preferred because it identifies the exact file to use, rather than any file provided by the URL.

Example

```
begin prefetch block
  add nohash prefetch item url=http://www.mysite/downloads/download25.exe
end prefetch block
wait {download path "download25.exe"}
```

This example uses a static download in a prefetch block and retrieves it without a hash. This technique is intrinsically insecure, but it uses a white-list on the Tivoli Endpoint Manager Server to validate the URL.

Version 7.2 and above

add prefetch item

Adds a download item to the prefetch queue. This command must reside between a **begin prefetch block** and an **end prefetch block** command. This is a plural command -- it can specify multiple downloads separated by semicolons.

The Tivoli Endpoint Manager Server will cache the download when the Action is created, unless relevance substitution is employed. Tivoli Endpoint Manager Relays only collect those files that the Clients request, and the Clients will only request a file if the command is inside a TRUE condition block.

Instead of listing the download items in the command line, you can put them in a file (one item per line) and then use a relevance substitution like the following:

{concatenation ";" of lines of file <your file>}

This is a common usage when specifying a file in a Fixlet site that contains the download information.

add prefetch item [name=<n>] sha1=<h> size=<s> url=<u> [; ...]

Where:

<n> is an optional name, limited to alphanumerics, dashes, underlines and non-leading period characters. If no name is explicitly specified, the name will be derived from the final component of the URL (following the final slash).

<h> is the required sha1 for the specified file.

<s> is the required file size.

<u> is a required url. If the name is not specified, then it will be derived from final component of the supplied URL.

[; ...] denotes that the command is plural; extra files can be specified, with each separated by a semicolon.

Relevance substitution is allowed with the arguments of this command; however when substitution is used, the Tivoli Endpoint Manager Server can't cache the download item at action creation time.

The arguments may be in any order desired, and unrecognized arguments will be ignored.

When used *without* relevance substitution, Tivoli Endpoint Manager Clients and relays will collect these files by ActionID and ordinal. When used *with* relevance substitution, clients and relays will collect these files by URL and sha1. To specify a download without specifying the sha1, use the **add nohash prefetch item** command.

Example

```
begin prefetch block
  if {name of operating system = "Windows 2000"}
    add prefetch item {"name=up.exe sha1=12 size=45
    url=http://ms.com/hot2k.exe"}
  else
    add prefetch item {"name=up.exe sha1=12 size=45
    url=http://ms.com/hot.exe"}
  endif
end prefetch block
wait {download path "up.exe"}
```

This example demonstrates a conditional download in a prefetch block. By checking the OS first, only the proper file will be prefetched, potentially saving considerable time and bandwidth.

Version 7.2 and above

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 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.

Version 5.1 and above
  ```

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.

Version 5.1 and above

begin prefetch block

Starts a set of commands to download files. Normally, when you download a file using Tivoli Endpoint Manager Actions, the checksum is evaluated to guarantee authenticity. However, if the target of your download is in flux, such as an anti-virus definition, that requirement may be too restricting. To handle a case like this, the Tivoli Endpoint Manager provides dynamic download commands, which are bracketed by **begin prefetch block** and **end prefetch block**. For more information, see **Introducing the Prefetch Block** and **Introducing Dynamic Downloads**.

This feature is tightly integrated with your Tivoli Endpoint Manager Relay structure, optimizing download speeds and bandwidth. When an Action requests a file, the relay checks its cache, and immediately forwards the file if available. Otherwise, the request is passed up the line until it reaches the Tivoli Endpoint Manager Server.

Dynamic downloading uses a white-list located at **<BES Server Install Path>\Mirror Server\Config\DownloadWhitelist.txt** to ensure that only trusted sites are accessed. This file contains a list of URLs formatted as regular expressions, such as http://.*\.mysite\.com/.*. The URL you provide in a prefetch statement must match an entry in the white-list before it can be downloaded. If the URL isn't found in the list, the command fails with **NotAvailable** error.

These existing commands are allowed within the prefetch block:

```
// comment lines and blank lines
if/elseif/else/endif
parameter
action parameter query (treated as a comment by the client)
```

The following new commands are allowed within the prefetch block, but are not allowed outside of it:

```
add prefetch item
add nohash prefetch item
collect prefetch items
execute prefetch plug-in
```

Syntax

begin prefetch block

Only one prefetch command block can be used in an Action script and it must be closed with an **end prefetch block** command.

Only comments or blank lines are allowed to precede this command. When processing actions with prefetch blocks, **download, download as** and **prefetch** are not allowed anywhere in the action script. The **download now as** command is allowed, but *not inside or before the prefetch block*.

Example

```
// action script to automatically update a URL manifest from a custom site
begin prefetch block
   parameter "ini"="{file "server_bf.ini" of site (value of setting
   "MyCustomSite") of client}"
   // prefetch the plug-in that provides the download list
   add prefetch item name=plugin.exe sha1=123 size=12
   url=http://www.mysite/downloads/myplugin.exe
   // collect above prefetch file (needed to create a manifest composed of
   URLs)
   collect prefetch items
   // execute the plug-in that produces a manifest from the ini data file
   execute prefetch plug-in "{download path "plugin.exe"}" /downloads
   "{parameter "ini"}" "{download path "urllist"}"
   // URL manifest formatted as lines containing: name=<n> sha1=<h> size=<s>
   url=<url>
   add prefetch item {concatenation " ; " of lines of download file
   "urllist"}
end prefetch block
// action is now active, update the files:
waithidden "{download path "plugin.exe"}" /update "{parameter
"ini"}" "{location of download folder}"
```

This example downloads a plug-in that processes another file to produce a manifest containing a list of more files to download. When the prefetch block ends, the files have been downloaded and moved to the download folder and the rest of the Action can continue.

Notes:

Older consoles and clients will reject action scripts that use the new prefetch functionality and identify them as containing syntax errors. Older relays will not process dynamic download actions even if the server and clients can handle it.

Only one prefetch block is allowed in an Action script.

Certain commands must not appear anywhere in an Action script that contains a prefetch block: **download, download as** and **prefetch**. **Download now** may appear in the script, but it must come *after* the prefetch block.

Several new inspectors have been added to allow action script to reference download files using relevance substitution. These include **download path "<name>"**, **download file "<name>"**, and **download folder**. See the Inspector Guide for more information.

Version 7.2 and above

collect prefetch items

After items have been added to the prefetch queue by commands such as **add nohash prefetch items** and **add prefetch items**, this command collects those items from the Tivoli Endpoint Manager Relay. Prefetch processing of the Action is suspended until all the specified files are collected. If the **add prefetch** command has provided the downloads with new names, they will be renamed at this point.

This command is typically used to retrieve a plug-in and/or a set of files that can be processed by a plug-in. In this case, a file is first added to the prefetch list, collected, and then processed by a subsequent **execute prefetch plug-in** command, which might create a file containing additional downloads. Each **collect prefetch items** command is treated as a synchronization point, causing the prefetch processing of the Action to wait for the files to download before proceeding. Once the files are available, the Action is reprocessed from the beginning. This allows the Action to compensate for any files that may have changed due to altered conditions on the machine. The next command in the action will be processed only after the **collect prefetch items** command is executed and all files in the prefetch list have been downloaded.

The end prefetch block command does an automatic collection, ensuring that subsequent Action commands will have the necessary files at hand.

Once the files have been collected, they can be examined with some new Inspectors which may have different interpretations, depending upon whether the Action is active during prefetch processing or not. For more information on these Inspectors, see **Introducing Dynamic Downloads**.

Syntax

collect prefetch items

Example

```
begin prefetch block
  parameter "ini"="{file "server_bf.ini" of site (value of setting
  "MyCustomSite") of client}"
  add prefetch item name=myPlugIn.exe sha1=12 size=12
  url=http://mysite/plugin.exe
  // collect the plug-in before continuing:
  collect prefetch items
  execute prefetch plug-in "{download path "myPlugIn.exe"}" /downloads
  "{parameter "ini"}" "{download path "urllist"}"
  add prefetch item {concatenation " ; " of lines of download file
  "urllist"}
end prefetch block
```

In this example, the collect statement ensures that the plug-in has been successfully downloaded before proceeding with the rest of the prefetch block.

Version 7.2 and above

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.

Version 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.

Version 6.0 and above -- Windows Only

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.

Version 5.1 and above

download

Deprecated: use **download as** or **download now as**.

Downloads the file indicated by the URL. This command is included for backward compatibility with version 2.0 of the Client Edition, and it continues to be supported to properly handle legacy Tivoli Endpoint Manager 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 Tivoli Endpoint Manager Client 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.mycompany.com/update/bfxxxx.exe`

  Prefetches the bfxxxx.exe file from the mycompany 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.mycompany.com/update/bfxxxx.exe`

  Downloads the bfxxxx.exe file from the mycompany 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 the Tivoli Endpoint Manager to prefetch downloads through a relay hierarchy to the clients.

Version 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 Tivoli Endpoint Manager to prefetch downloads through a relay hierarchy to the clients.

Version 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 the Tivoli Endpoint Manager to prefetch downloads through a relay hierarchy to the clients.

Version 6.0 and above -- Windows Only

end prefetch block

Marks the end of a prefetch block (see **begin prefetch block**). This command must be present whenever the **begin prefetch block** command is specified. This command automatically performs a **collect prefetch items** command, meaning that all the files added to the prefetch list will be available when the block is ended.

Syntax

end prefetch block

Only one prefetch command block can be used in an Action script and it must be bracketed by a **begin prefetch block** command and an **end prefetch block** command.

Only comments or blank lines are allowed to precede the prefetch block. When processing actions with prefetch blocks, **download as** and **prefetch** are not allowed anywhere in the action script. The **download now** command is allowed after the prefetch block, but *not before or inside the prefetch block*.

```
begin prefetch block
  add prefetch item sha1=123 size=456
  url=http://ms.com/downloads/hotfix123.exe
end prefetch block
wait {download path "hotfix123.exe"}
```

This code demonstrates static downloading in a prefetch block. Although it doesn't take advantage of dynamic relevance substitution, this is the preferred format for downloads in versions 7.2 and later. Note that the **end prefetch block** command also collects the file (hotfix123.exe), so that the subsequent **wait** command -- which runs and waits for completion -- is guaranteed to have the file available.

Note:

Older consoles and clients will reject action scripts that use the new prefetch functionality and identify them as containing syntax errors. Older relays will not process dynamic download actions even if the server and clients can handle it.

Version 7.2 and above

execute prefetch plug-in

This command passes arguments to a named command and then executes it. It is not intended for a lengthy executable and the client will only wait 60 seconds for its completion. This command can be used to authenticate or execute downloads. It can also be used to execute custom logic that can create inspectable values for subsequent **add prefetch item** commands.

For use cases such as updating anti-virus definitions, this command can be used to run code that processes a file to produce another file containing a set of URLs to be downloaded.

Syntax

execute prefetch plug-in "executable pathname" <args>

Where:

"executable pathname" is the full pathname for the plug-in to execute. This command is designed for executables that are fast to execute and return promptly. The Tivoli Endpoint Manager Client will block out 60 seconds of time while it waits for the command to complete; only a shutdown request can interrupt this waiting period. After 60 seconds, the Client will log a message and disable the command. When it is disabled, any actions that use this command will not execute until after the client has been restarted. In general it is expected that the command will complete much faster -- if it takes longer than two seconds to execute, the client will log an appropriate message. Relevance substitution can be used to specify the pathname.

<args> are arguments passed to the executable.

The exit code of the execute prefetch plug-in application is important as it informs the client of failure or success, where 0 (zero) indicates success and all other exit codes are treated as failures and result in a failed Action attempt. For debugging purposes, the exit code is recorded in the client log.

Example

```
begin prefetch block
  parameter "ini_file"={file "server_bf.ini" of site (value of setting
  "MyCustomSite") of client}
  add prefetch item name=plugin.exe sha1=123 size=12
  url=http://mysite/myplugin.exe
  collect prefetch items
  // execute the plug-in to produce a manifest from the ini_file:
  execute prefetch plug-in "{download path "plugin.exe"}" /downloads
  "{parameter "ini_file"}" "{download path "manifest"}"
  add prefetch item {concatenation " ; " of lines of download file
  "manifest"}
end prefetch block
```

This example downloads a plug-in that processes the ini_file to produce a manifest.

Version 7.2 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. Tivoli Endpoint Manager 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 Tivoli Endpoint Manager Console that facilitates the distribution of directories that use this kind of archive.

As of version 8.2, this command also allows you to specify a target directory with an optional second argument.

Syntax

extract <Archive File><Destination_Directory>

Examples

```
extract InstallMyApp.zip
```

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

- `extract InstallMyApp.zip "d:/temp"`

Same as above, but specifying a target directory.

Note: There should be no quotes around the filename, even if there is whitespace in the name. This is to be consistent on systems that allow quotes as valid filename characters.

Version 5.1 and above

folder create

Creates the named folder. Any Action script with the create command will terminate if the folder cannot be created. This can happen due to write protection or an attempt to write to a CD-ROM, for instance. It will also terminate if the path already exists, but does not refer to a folder.

Syntax

folder create <FolderName>

Where **FolderName** is the name of the folder to create (typically enclosed in quotes). Relevance substitution is performed on the arguments of folder create command lines.

Examples

- `folder create "c:\program files\bigsoftdir"`
- `folder create "{name of drive of windows folder}\Extras"`

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

Note

It is always good practice to enclose folder names in quotes to preserve any spaces.

Version 8.0

folder delete

Deletes the named folder. This command is recursive, deleting all contained files and folders. An Action script with the delete command will terminate if the folder exists but cannot be deleted. This can happen due to write protection or an attempt to delete from a CD-ROM, for instance. This action will also fail (and leave the contents of the folder in an indeterminate condition) when the folder contents are busy, as can happen when a file inside the folder is in use by another application.

If the folder does not exist at all, however, the action script will continue to execute.

Syntax

delete folder <FolderName>

Where **FolderName** is the name of the folder to delete (typically enclosed in quotes to preserve spaces). Relevance substitution is performed on the arguments of delete folder command lines.

Examples

- `delete folder "c:\program files\bigsoftdir"`
- `delete folder "{name of drive of windows folder}\Temp"`

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

CAUTION!

This command is extremely powerful. Because it is recursive, it can delete all the files on the Client. Please use with great care!

Version 8.0

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.

Version 5.1 and above

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 the Tivoli Endpoint Manager to prefetch downloads through a relay hierarchy to the clients.

Version 6.0 and above -- Windows Only

relay select

The relay select command forces the Tivoli Endpoint Manager Client to select the nearest 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 Tivoli Endpoint Manager Client to search for and connect to the nearest relay.

Version 5.1 and above

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.

Version 6.0 and above -- Windows Only

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 Tivoli Endpoint Manager Console Operators. Settings issued by the 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 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.

Version 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.

Version 5.1 and above

Registry Commands

regdelete

Deletes a registry key value of the given name, regardless of whether it currently exists or not.

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.

Version 5.1 and above -- Windows Only

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\BigCorp 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 regset command, keep in mind that the Tivoli Endpoint Manager 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.

Version 5.1 and above -- Windows Only

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.

Version 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.

Version 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\BigCorp 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 regset64 command, keep in mind that the Tivoli Endpoint Manager 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.

Version 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.

Version 6.0 and above -- Windows Only

Administrative Rights Commands

administrator add

This command lets you appoint specific people to administer specific Tivoli Endpoint Manager Clients. This is accomplished by using a setting with 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 Console operator named bob to have administrative rights on the targeted computer(s), effective on the given date.

Version 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 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 Console operator named bob, effective on the given date.

Version 5.1 and above

BigFix Client Maintenance Commands

module add

Adds the specified inspector library file to the set of inspector libraries to be used by the client. When replacing an inspector library, you must specify it in a **module delete** command as well as specifying it in the module add command. To complete the action you must issue a **module commit** command.

Syntax

module add "<module name>"

Example

- `module add "dellinspect.dll"`

Note

For internal use only.

Version 5.1 and above

module commit

The add and delete commands set the stage for committing changes to the inspector libraries. The commit command performs the actual deletion and addition.

Syntax

module commit

Example

```
delete "dellinspect.dll"
copy "{pathname of client folder of site
"dell"}\dellinspect.dll""dellinspect.dll"
module add "dellinspect.dll"
module commit
```

Note

For internal use only.

Version 5.1 and above

module delete

Marks the specified inspector library file for deletion. To complete the action you must issue a **module commit** command.

Syntax

module delete "<module name>"

Example

```
module delete "inspectors.dll"
```

Note

For internal use only.

Version 5.1 and above

Locking Commands

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.

Version 5.1 and above

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.

- `action lock until "{apparent registration server time + 10 * minutes}" "{apparent registration server time}"`

Locks actions for 10 minutes, using the current **apparent registration server time**, which is based on the last time the Client registered with the server.

Version 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.

Version 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`

Version 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`

Version 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`

Version 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`

Version 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.

Version 5.1 and above

subscribe

Subscribes the client to the site identified in the masthead file. The Tivoli Endpoint Manager Console provides the **Manage Sites** dialog to automate site addition.

Syntax

subscribe "<masthead file name>"

Example

- `subscribe "__Download\Sitename.fxm"`

Note

In the Tivoli Endpoint Manager 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.

Version 5.1 and above

unsubscribe

Automatically unsubscribes from the current Enterprise Fixlet site.

Syntax

unsubscribe

Example

- `unsubscribe`

Version 5.1 and above

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.

Version 5.1 and above
  ```

Part Four

Notices

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

2Z4A/101

11400 Burnet Road

Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

TRADEMARKS:

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Part Five

Index

A

B

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W