PTC.com

PTC

Windchill® Client Inspector Deployment and Administration Guide

Windchill 9.0 & Windchill 9.1

Windchill PDMLink®
Windchill ProjectLink™
Pro/INTRALINK® 9.0 & Pro/INTRALINK 9.1
Pro/ENGINEER® Wildfire® 4.0 & Pro/ENGINEER Wildfire 5.0

October 2010

Contents

1 Overview

This chapter provides an overview of the Windchill Client Inspector.

What the Windchill Client Inspector Does

The Windchill Client Inspector inspects relevant Pro/ENGINEER Wildfire and Windchill settings on client machines and stores the values in a results XML file. After generating results files from client machines, you can use the extraction tool that is included with the Windchill Client Inspector installation to present the results in either an XLS or CSV formatted file.

The reference XML file included with the inspector defines what is inspected.

Note: In addition to checks for the operating system, processor, memory, storage, mouse, network, video and display configuration, and installed software, the installed reference XML file contains group checks related to issues that have been discovered and documented using Windchill TPIs and TANs. As issues are discovered, PTC may update the definition XML file to add check groups for checking additional settings. Ensure that you are using the latest reference file that is available by downloading the latest WCClientInspector.zip file that is available on the PTC web site.

You can use the client inspector is the following ways:

- Install and run the inspector on individual machines. This allows you to quickly record existing settings from that machine. After the results are collected, you can either view the results in the XML results file or extract the collected information into one row of a spreadsheet or comma-delimited text file.
- Install and run the inspector from a common location. Running the inspector from a common location allows you to set up an environment to record the results collected from a group of machines in a common location. After the results are collected from all machines, you can extract the collected information from the results files and insert the results into a spreadsheet or text file.

- Configure client machines to run the client inspector once upon login. Work with your IT department to determine how to best set up and execute the client inspector automatically on clients. One way of setting this up is to set the following variable on clients to the command that runs the inspector:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce
```

The client inspector does not make any changes to client settings.

After reviewing the results that are extracted, direct individual users to make recommended changes to settings on their client machines.

Audience

This document is written for the CAD system administrator and assumes you have the following knowledge and skills:

- Use of operating system command shell utilities that allow you to edit configuration files, view log files, and so on.
- Use of the DOS batch files and VBS files.

Windchill Client Inspector System Requirements

To run the client inspector and associated extraction tool for compiling the inspection results, the following items must be true:

- Java version installed on the machine running the extraction tool must be 1.6.0_13 or later and the Java bin directory must be in the PATH environment variable. On some platforms, Java 1.6 is called Java 6.
- The version of Pro/ENGINEER Wildfire that is installed on client machines must be 4.0 or higher.

Supported Releases

The Windchill Client Inspector provides useful client information for clients interacting with the following Windchill releases:

- Windchill 9.0 M080
- Windchill 9.1 M060 and later 9.1 maintenance releases

The Windchill Client Inspector provides useful client information for clients running Pro/ENGINEER Wildfire at the following release levels:

- Pro/ENGINEER Wildfire 4.0
- Pro/ENGINEER Wildfire 5.0

Affected Files

The client inspector does not modify any files on the client machines that it inspects. It only reports the current settings that are in use on each machine by generating a result file containing the settings.

Each result file is an XML file and is uniquely named to indicate the client on which it was run, the user ID under which the check was run, and a timestamp that identifies when the check was run.

2

Installing the Windchill Client Inspector

This chapter describes the Windchill Client Inspector contents, the installation steps, and the steps used to uninstall the inspector.

Media Content

The Windchill Client Inspector is deployed as a standard Visual Basic script with a corresponding DOS batch file. Additional files provide the definition and DTD for the Visual Basic script. Also included is a second DOS batch file and accompanying support files that can be used to collect results from individual clients.

You download the media as the WCClientInspector.zip file. When unzipped, the ZIP file creates a set of directories that establishes a default environment for collecting results. The environment consists of the following set of directories and files:

- WCClientInspector.bat

  Use the WCClientInspector.bat for running the client inspector unattended.

  You should not need to manually alter the WCClientInspector.bat file.

- readme.txt

  The readme.txt file describes the contents of the ZIP file.

- doc\WCClientInspectorDeployAdmin.pdf

  The WCClientInspectorDeployAdmin.pdf file is this document.

- logs

  The logs directory is the default directory for storing inspector log files.

- META-INF\MANIFEST.MF

  The MANIFEST.MF file contains the version information for the manifest, Ant, and Java used to create the ZIP content.

- ResultExractor

  The ResultExractor directory contains the files that are used when extracting data from the inspection results files. This directory contains the following:

  – lib\poi.jar

    Apache POI package used for formatting the output for Excel.

  – extract-software.xml

    Sample XML file for defining regular expressions that are used to extract specific Windows programs/applications (for example, web browsers) and add as additional CSV/Excel columns.

  – run.bat

    The DOS batch file for running the extraction tool.

  – xpath.jar

    Application package file.

- results

  The results directory is the default directory for storing inspection result files.

- vbs\WCClientInspector.vbs

  The WCClientInspector.bat calls the WCClientInspector.vbs file to perform the actual checks. If you run the Visual Basic script directly from a command line, you will see the messages on screen.

  The script is designed in such a way that it extracts relevant information from the machine but does not alter the existing settings.

  You should not need to manually alter the WCClientInspector.vbs file.

- xml\WCClientInspectorReference.xml

  The WCClientInspectorReference.xml file is an ant XML build file. The definition XML file acts as a load map for the client-side inspection script (WCClientInspector.vbs). For reference file details, see Modifying Your Client Inspector Definition File.

- xml\WCClientInspectorReference.dtd

  The WCClientInspectorReference.dtd file defines the elements that can be in the definition XML file. For DTD details, see Appendix A.

Download Instructions

You can download the WCClientInspector.zip file containing the media content from the **PTC Software Downloads** page. Start by accessing the Technical Support page using the following URL:

http://www.ptc.com/support/index.htm

From the support page, you can navigate to the page where you can download the Windchill Client Inspector. For example:

1. Click **Order or Download Software Updates** and then click the **Order or Download Software Updates** link.
2. Enter your customer number and click **Continue**.
3. Navigate to the Windchill Client Inspector and download the ZIP file that has the content.

Installation Steps

Complete the following steps to install the Windchill Client Inspector:

1. Download the ZIP file containing the installation media from the PTC web site as described in Download Instructions.
2. Extract the contents of the ZIP file into a directory on the system where you can access the Pro/ENGINEER Wildfire clients that you are checking. The content of the ZIP file is described in the Media Content section.

   Tip: Although you can unzip the contents on any client machine that meets the client inspector requirements, the location you choose is usually on a common server that can be set up to access all client machines.

Uninstalling the Inspector

The client inspector does not register itself or make any other changes to the server on which it is installed.

You can uninstall the inspector by simply deleting the top-level client inspector installation directory and its contents. If you created additional directories for storing logs and results, also delete those directories.

3

Running the Windchill Client Inspector

This chapter describes how to run the Windchill Client Inspector on a local machine using a DOS batch file and how to deploy the client inspector for checking all clients that use Pro/ENGINEER Wildfire across your environment. Included in the chapter are the available command line options.

Execution Options

You can run the Windchill Client Inspector as follows:

- Run the inspector in unattended mode where all screen outputs (normal, warning, and error messages) are redirected into log file.

  Use the WCClientInspector.bat file to run the inspector in unattended mode.

- Run the inspector so that you see the messages on the screen.

  Use the WCClientInspector.vbs file to view messages while the inspector is running.

All command line options are available when executing either the batch file or the Visual Basic script.

Running Client Inspector in Unattended Mode

To run the client inspector in unattended mode using the established defaults, either double-click the WCClientInspector.bat file located in the top-level directory where you unzipped the downloaded ZIP file or, from a command line, navigate to the top-level directory and enter:

```
WCClientInspector.bat
```

When using the BAT file, the outputs from the client inspector are captured in an indexed log file.

You can also include options on the WCClientInspector.bat command:

```
WCClientInspector.bat [/option1 ... /optionn]
```

The command line options available when using the BAT file are described in Command Line Options for Running the Client Inspector.

Running Client Inspector and Viewing Output

To run the client inspector and view output on the screen, navigate to the directory containing the WCClientInspector.vbs file and enter the following command:

```
cscript WCClientInspector.vbs [/option1 ... /optionn]
```

When running directly from the VBS script, the client inspector does not capture the output in a log file.

The command line options available when using the VBS file are described in next section, Command Line Options for Running the Client Inspector.

Command Line Options for Running the Client Inspector

The client inspector command line options can be included when running the inspector from either the BAT file or the VBS file.

When no command line options are supplied, the following defaults are used:

- Log files generated when using the BAT file are saved in the logs directory that is under the client inspector installation directory.
- The settings retrieved from running the client inspector are stored under results directory that is under the client inspector installation directory.
- Checks the machine where client inspector is running.
- The client inspector can run for a maximum of five minutes. If the execution time reaches five minutes, the inspector stops running and logs the issue.
- The client inspector uses the Definition XML file that is installed with the inspector (xml\WCClientInspectorReference.xml).
- The result file generated by the client inspector is stored in the results directory that is under the client inspector installation directory. No copies of the result file are made.

Command line options are formatted as follows:

```
/<name>:<value>
```

You can display the list and format of options on your screen by entering:

```
cscript WCClientInspector.vbs /?
```

Options can be specified in any order. The following sections provide details about the options.

/timeout:<*seconds*>

Limits the length of time the client inspector runs. Replace <*seconds*> with the maximum number of seconds you want the client inspector to run.

By default, the timeout is set to 300 (5 minutes).

Timeout is used to prevent the client inspector from running indefinitely under unexpected cases. For example, storage device check may take excessively long time if user’s machine had an unresponsive network disk drive mapped to one of drive letters.

Note: If the client inspector execution time reaches the timeout value, the execution stops. This fact is indicated in log file and the resulting XML file is not created.

/exec:<*check_level*>

Identifies which levels of checking are done for this run.

Replace <*check_level*> with the sum of the levels as defined in the following table:

Check Group	**Level**
Basic hardware and operating system check	1
Installed software and versions	2
TPI and TAN check	4
Reserved for future use	8, 16, and beyond

By default, the client inspector performs check levels 1 trough 4. This is equivalent to including the following command line option:

```
/exec:7    [1 + 2 + 4 = 7]
```

To run only the hardware and operating system check and the TPI and TAN check, include the following command line option:

```
/exec:5    [1 + 4 = 5]
```

/verbose:<*level_sum*>

Identifies which levels of verbosity are used for this run.

Replace <*level_sum*> with the sum of the levels as defined in the following table:

Verbosity	**Level**
Minimum checking; logs host name and user name information.	1
Normal checking; logs basic information related to the WMI service and registry.	2
Group checking; logs information about each check group that is run.	4
Check item checking; logs information about each check element that is run.	8
Reserved for future use	16 and beyond

By default, the client inspector uses a verbosity that includes levels 1 through 4. This is equivalent to including the following command line option:

```
/verbose:7  (1 + 2 + 4 = 7)
```

The default verbosity captures the status of CheckGroup actions that is similar to the following:

```
[4/14/2009 14:16:35 @ localhost] CheckGroup + OS start
[4/14/2009 14:16:35 @ localhost] CheckGroup - OS Done
```

To debug issues, you can change the verbosity to also capture the status of CheckItem actions as follows:

```
/verbose:15
```

The status messages captured are similar to the following:

```
[4/14/2009 14:25:19 @ localhost] CheckGroup + OS start
[4/14/2009 14:25:19 @ localhost] CheckItem Name
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckItem Version
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckItem Build type
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckItem Service pack
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckItem Organization
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckItem Hotfixes
[4/14/2009 14:25:19 @ localhost] CheckItem Done
[4/14/2009 14:25:19 @ localhost] CheckGroup - OS Done
```

/host:<*host_name*>

Identifies a remote machine on which to check settings. Replace <*host_name*> with the host name defined for the remote client.

Note: The client inspector cannot remotely obtain the information that is specific to individual login user accounts. For instance, you cannot obtain HKEY_CURRENT_USER registry information of a specific user on the machine unless you have logged in as that user, with the user's proper credentials.

By default, the client inspector performs the check on the machine from where you are executing it. This is equivalent to including /host:localhost command line option.

Note: Remote queries require that your local machine and user account must have adequate access permission. For instance, your login user account must have an administer privilege on the remote machines. Furthermore, your network topology, security, and firewall settings must allow remote WMI execution. This document does not describe details on required network and security settings. For additional details, see:

http://msdn.microsoft.com/en-us/library/aa389290(VS.85).aspx

/defintion:<*file_path_or_url*>

Specifies the location of the definition XML file that you want used by the client inspector.

By default, the definition XML file used is the installed WCClientInspectorReference.xml file that resides in the xml directory where the client inspector is installed.

To include this option, replace <*file_path_or_url*> with the either of the following:

- If the file is located on an accessible hard disk, use the complete file path to the file.

  For example, if you want to use an updated reference file named MyClientInspectorReference.xml that resides on a mapped Q drive in the inspector\xml directory, include the following:

  ```
  /definition:Q:\inspector\xml\MyClientInspectorReference.xml
  ```

- If you have placed the file in a web server, use a URL to identify where the file resides.

  For example, if you want to use an updated reference file named MyClientInspectorReference.xml that resides in the web server available from www.mycompany.com in the inspector\xml directory, include the following:

```
/definition:http://www.mycompany.com/inspector/xml/MyClientInspectorReference.xml
```

/copy:*<result_copy_dir>*

When executing the client inspector locally, use this option to specify a location where the inspector stores a copy of the result file. The copy action is done after all the processing steps have completed.

By default, the inspector does not make any copies of the result file.

To include this option, replace <*result_copy_dir*> with the either of the following:

- If you have mapped a drive to a common storage location, include the drive letter and file path.

  For example, if results\april file path is on the mapped Q drive, include the following:

  ```
  /copy:Q:\results\april
  ```

- If the common storage location is available using a UNC path, include the machine name and the shared file path.

  For example, if the machine name is server1 and the file path is results\april file path, include the following:

  ```
  /copy:\\server1\results\april
  ```

/ftp:*<ftp_host>*,*<ftp_user>*,*<ftp_passwd>*,*<ftp_dir>*

When executing the client inspector locally, use this option to send a copy of the result file using FTP. The send action is done after all the processing steps have completed.

By default, the inspector does not send any copies of the result file using FTP.

To include this option, replace *<ftp_host>,<ftp_user>,<ftp_passwd>,<ftp_dir>* with FTP hostname, FTP user account name, FTP password for the user account, and FTP directory into which the result files is copied.

Note: The format of the FTP information must have no spaces and is comma delimited.

For example, assume you have a FTP server “server1” with user account “user1”, password “pass1”, and want to copy the results under FTP path “/pub/results/april”. Then, use following option on your command:

```
/ftp:server1,user1,pass1,/pub/results/april
```

If you are using an FTP account that has no password, leave the *<ftp_passwd>* field blank. For example:

```
/ftp:server1,anonymous,,/pub/results/april
```

Deployment Recommendations

Although you should rely on the expertise of your IT specialist when it comes to deployment and execution of the client inspector, the following recommendations can make deployment, execution, and maintenance of the inspector a little easier.

Common Location for Storing Copies of Result Files

Set up either an FTP server or a shared disk for storing copies of the result files that are generated from running the client inspector.

Under the root directory, create a subdirectory that is used for the copied result files, such as \inspector\results\april.

You can use the /copy or /ftp option on the client inspector command to implement this.

Common Location for Client Inspector Files

Copy the client inspector files to central location so that all machines have access and execute the same client inspector instead of installing the client inspector on individual machines.

Although it is not necessary to have the client inspector in central location, there are advantages for keeping it in one central location:

- The common location guarantees every client is always using the same copy/version of the inspector.
- There is no need for deployment and it is easy to update the client script in case of new features or bug fixes.
- All machines can execute it with identical command line. The identical command line simplifies setting up of RunOnce registry implementation described in the Trial Client Inspector Runs section.

For example, install the inspector software at:

Z:\common\tools\WCClientInspector

Trial Client Inspector Runs

Before deploying the client inspector across your site, manually run the client inspector on selected machines to ensure that a result file is generated from each run. If you have set up a common location for storing a copy of results file, you can also verify that the result file is copied or sent to the common location.

For example, assume that you have done the following:

- Installed the client inspector at Z:\common\tools\WCClientInspector.
- Set up an FTP server where the server name is “ftpserver.ptc.com” and you have an FTP account with user name “ftpuser” and its password is “ftppasswd”.
- Created the /inspector/results/april directory on the FTP server.

Then the client inspector command line to enter is:

```
Z:\common\tools\WCClientInspector\WCClientInspector.bat
/ftp:ftpserver.ptc.com,ftpuser,ftppasswd,/inspector/results/april
```

If you want to have the client inspector run once when a machine boots up, you can set the following registry key to the previous command on selected machines:

HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\RunOnce

Ensure that the client inspector runs once on those machines and that the expected output is generated before deploying this approach across your site.

Location of DTD File and XML Files

All of the XML files accessed and created by the client inspector must be well-formed and valid. This includes the definition XML file (default: WCClientInspectorReference.xml) and all generated result XML files.

The location of the DTD is hardcoded in each XML file. For example:

```
<!DOCTYPE clientCheck SYSTEM "../xml/WCClientInspectorReference.dtd">
```

To ensure that the DTD is available to the XML files, you can do either of the following:

- Create directory named "xml" adjacent to the directory holding the result XML files and place the reference XML file and the DTD file under the “xml” directory. For example:

```
results/ +- ptccc.091210085022.myhost.user123.xml
         |
         +- ptccc.091210085346.myhost.user567.xml
         |
    xml/ -+- WCClientInspectorReference.dtd
         |
         +- WCClientInspectorReference.xml
```

  This is the default directory structure set up by extracting the files from the WCClientInspector.zip file using the folder names in the ZIP file.

- Create directory named “xml” and place all of your XML files and the DTD file under the same xml directory. For example:

  ```
  xml/ -+- WCClientInspectorReference.dtd
         |
         +- WCClientInspectorReference.xml
         |
         +- ptccc.091210085022.myhost.user123.xml
         |
         +- ptccc.091210085346.myhost.user567.xml
  ```

  Although valid, this configuration is not recommended because the reference file is in the same directory as the result XML files.

Viewing Result File Information

Each result file that is generated is an XML file and can be viewed in an Internet Explorer browser or in any XML editing tool. However, viewing individual files in this manner requires that you have an understanding of the elements in the file.

Instead, you can use the extraction tool that is provided with the client inspector to compile the results from one or more client inspector runs into a spreadsheet or a text file. For details, see Running the Extraction Tool.

4
Running the Extraction Tool

This chapter describes how to run the extraction tool that you can use to extract useful information from the client inspector result files and save that information in a CSV text or Excel (.xls) file.

The extraction tool is a Java application that is run by executing the supplied run.bat file.

The extraction tool files are stored in the ResultExractor directory as described in Media Content.

Command Line Options for run.bat

To view the help for run.bat, navigate to the directory containing run.bat and enter the following command:

```
run.bat /?
```

When entering the run.bat command to extract information from existing result files, use the following syntax:

```
run.bat <xml_files_dir> <output_filename>
```

The arguments are as follows:

<xml_files_dir>	is the directory containing all result XML files that were generated by the client inspector. See Extractor Input Files.
<output_filename>	is the file path of the output file created by the extraction tool, including the file name and extension. See Extractor Output Files.

For example, assume the following:

- Process all the result files under D:\WCClientInspector\results.
- Save the output to D:\temp\results.xls.

From a command prompt, navigate to the directory containing the run.bat file and enter either of the following commands:

```
run.bat D:\WCClientInspector\results D:\temp\results.xls
```

To save the output as a CSV text file named D:\temp\results.csv, enter the following command:

```
run.bat D:\WCClientInspector\results D:\temp\results.csv
```

Extractor Input Files

The extraction tool uses the following files as input:

- The XML files that reside in the directory specified on the run.bat command.
- The WCClientInspectorReference.dtd file.

  Because the result XML files contain references to a DTD file (which is "../xml/WCClientInspectorReference.dtd"), the DTD file must be present at the location relative to the XML files. For additional information, see Location of DTD File and XML Files.

  For example, assume the input *<xml_files_dir>* is:

  ```
  D:\WCClientInspector\results
  ```

  Then, the DTD file must be at:

  ```
  D:\WCClientInspector\xml\WCClientInspectorReference.dtd
  ```

- The extract-software.xml property file which contains regular expressions that are used by extraction tool.

  For information about updating this file, see the next section, Updating extract-software.xml Property File.

Updating extract-software.xml Property File

PTC provides a sample extract-software.xml property file in the ResultExtractor directory. Use this file to reduce the output of the installed software check to only the software that is of interest.

The initial content of this file is used by the extraction tool to extract specific Windows programs/application information. The following lines in the file define the output for two columns in the output file:

```
<entry key='Java Application'> (java\s*.*ptc|java.*sun) </entry>

<entry key='Windows Application'> Windows.*Microsoft </entry>
```

The key attribute string specifies the column title and the value in each entry tag is the regular expression used to find matches. For example, matches to the Windows.*Microsoft regular expression would include the following:

"Microsoft Windows SDK .NET Framework Tools 6.1"

"Microsoft Windows SDK for Windows Server 2008"

The matches will not include "Security Update for Microsoft Office Excel"

You can edit the extract-software.xml file as needed. For example, if you would like to know which web browser applications are installed on a client, you can add the following to the property file. Add the entry element as one line in the file:

```
<entry key='Web Browser'> (Windows\s+Internet\s+Explorer|
Google\s+Chrome|Mozilla\s+Firefox|Safari) </entry>
```

This entry tells the extraction tool to search the list of all installed programs/applications, add anything containing string "Windows Internet Explorer" or "Google Chrome" or "Mozilla Firefox" or "Safari" to the corresponding CSV/Excel cell of the additional Web Browser column.

Extractor Output Files

The extraction tool generates the following output files:

- The generated CSV or XLS file as specified on the run.bat command. If the full path is not specified, the file is created in the directory where the run.bat file resides.

  To generate an Excel spreadsheet, include the XLS extension (XLSX is not supported).

  To generate a file that has comma-separated content, include the any extension but XLS (for example, CSV).

  For the details of what is generated, see the next section, Generated CSV/XLS File Content.

- The installed_software.txt file contains the list of installed Windows programs/applications. This file is stored in the same directory where the generated CSV or XLS file is stored.

- Text files containing cell content of any cells that exceed the Excel cell size limit of 30720 characters.

  For the details on the files generated, see Generated Text Files for Large Data Cells.

Generated CSV/XLS File Content

The generated CSV or XLS file has the following characteristics:

- Each row represents a client. If the client inspector has been executed multiple times on the same client, there is one row per execution. You can differentiate between multiple runs by the Timestamp column.
- Each column generated lists the value of an item inspected by the client inspector. These item checks are defined in the reference XML file.

  By default, column headers in the generated file are the following:

  – *<Check Group ID>*:*<Check ID>* values generated from the id elements in the reference file.

  – Any key attribute values defined in the extract-software.xml property file.

  Additionally, the column headers for TPI and TAN check groups are hyperlinks to the TANs or TPIs on www.ptc.com.

  For details on ID values and the checkLevel values, see Managing Check Group Execution.
- Any cell containing characters in red font indicates that the required value is either missing or the correct value has not been set.

The XLS file has three sheets: one for TPI and TAN checks, one for hardware and operating system checks, and one for software checks.

Generated Text Files for Large Data Cells

When a data cell in the generated XLS file exceeds the Excel cell size limit of 30720 characters, the extraction tool puts the data into a text file. Generated text files are stored in the same directory where the generated CSV or XLS file is stored.

The files are named using the following format:

```
<Check Group ID>_<Check ID>.txt
```

The *<Check Group ID>* and *<Check ID>* values are taken from the column title.

5 Modifying Your Client Inspector Definition File

The Windchill Client Inspector installation provides the WCClientInspectorReference.xml that you can use as your definition file. It is an ant XML build file and acts as a load map for the client-side inspection script (WCClientInspector.vbs). The inspector performs a generalized check action based on the definition XML file. The checks are performed sequentially in the exact order that they are defined in the XML file.

You can use the installed definition file without modifications or modify the installed file to create a definition file that meets the needs at your site. To modify the definition file, use an XML editor.

Note: In addition to checks for the operating system, processor, memory, storage, mouse, network, video and display configuration, and installed software, the installed reference XML file contains group checks related to issues that have been discovered and documented using Windchill TPIs and TANs. As issues are discovered, PTC may update the definition XML file to add check groups for checking additional settings. Ensure that you are using the latest reference file that is available by downloading the latest WCClientInspector.zip file that is available on the PTC web site.

If you create your own definition file and maintain it as a separate file, you must identify the file using the /definition option on the client inspector command. See /defintion:<file_path_or_url>.

Note: The definition XML file is used as a template for check result outputs (which are also in XML format). If you compare one of your result XML files with the XML definition, you can see that the two files are very similar. In fact, the result XML files are built on top of the definition file. In another words, each result XML has exactly the same contents as the definition XML file with additional check results information added. Think of the definition XML as a questionnaire and the inspector utility fills in the blanks to the questions. The questions and answers are the resulting XML file. For this reason, both the definition XML and result XML files use the same DTD file for their validation.

For the client inspector and extraction tool to work properly, both definition and result files must be well-formed and valid XML file. Be sure to validate your modifications against the WCClientInspectorReference.dtd file using the editor, a standalone application, or a web browser add on.

To learn more about the elements you can use in your definition file, review the element defined in the DTD. For details, see the WCClientInspectorReference.dtd appendix.

WCClientInspectorReference.xml Content

The out-of-the-box WCClientInspectorReference.xml file contains the following groups of checks:

- Operating system
- Processor
- Memory and storage
- Mouse
- Network
- Video and display configuration
- Installed software
- TPI and TAN groups as described in the comments at the beginning of the reference XML file. Open the reference file to view comments about the TPI and TAN groups.

Understanding Definition and Result XML Files

Before modifying the out-of-the -box WCClientInspectorReference.xml file, review the following sections to learn about the definition XML file.

Relationship between Definition and Result Files

The following XML snippet shows the beginning lines in the WCClientInspectorReference.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clientCheck SYSTEM "../xml/WCClientInspectorReference.dtd">

<clientCheck>
  <definitionVersion>2</definitionVersion>
  <outScriptVersion></outScriptVersion>
  <outRetrievingHostname></outRetrievingHostname>
  <outRetrievingUser></outRetrievingUser>
  <outCheckedHostname></outCheckedHostname>
  <definitionDescription>Client check file</definitionDescription>
```

The two XML elements with values are:

- definitionVersion

  The version of the out-of-the box definition XML file is set to "2". For details on when to change this number, see Definition File Version.

- definitionDescription

  The description is set to "Client check file".

All other elements in this snippet are blank. This is because they are "out" elements whose values are filled in when the client inspector runs and are stored in the result file that is generated. For example the following XML snippet is from a result file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clientCheck SYSTEM "../xml/WCClientInspectorReference.dtd">
<clientCheck>
  <definitionVersion>2</definitionVersion>
  <outScriptVersion>3</outScriptVersion>
  <outRetrievingHostname>JUPITER</outRetrievingHostname>
  <outRetrievingUser>MYDOMAIN\jsmith</outRetrievingUser>
  <outCheckedHostname>JUPITER</outCheckedHostname>
  <definitionDescription>Client check file</definitionDescription>
```

The snippet shows the following values that were retrieved by the client inspector:

- outScriptVersion

  Shows the version of client-side inspection script used to obtain the result XML. In this example, inspection script is at version 3.

  Note: The version and contents of the client inspection script are controlled by PTC. PTC recommends that you do not to alter the version number.

- outRetrievingHostname

  Shows the host name of the machine from where you executed the inspection script. In this example, the host name is "JUPITER".

  Note: When you run the inspection script remotely, there is a distinction between host name in this element and host name in the outCheckedHostname element.

- outRetrievingUser

  Shows the domain name and user name of the person who executed the inspection script. In this example, it was MYDOMAIN\jsmith.

- outCheckedHostname

  Shows the host name of the machine that is being examined. In this example, the script was executed locally so the host name "JUPITER" is the same host name shown in both the outCheckedHostname and outRetrievingHostname elements.

Throughout the WCClientInspectorReference.xml file, there are many elements that start with "out". Values for all elements starting with "out" are generated when the client inspector runs and are save in the result file.

Definition File Version

The version specified in definition file does not affect the client inspection script behavior. Use the version number to manage any changes you make to your definition file. By knowing what is in each version of the definition file, you can ensure that the checks are performed according to intended definition file. The version can be especially useful when you need to perform client checks on large numbers of client machines.

Tip: Each time you make a set of changes to your definition file, increase the version number stored in the defintionVersion element.

Managing Check Group Execution

The execution of checks within a checkGroup element is determined by the level of checking included when the client inspector is run and the values supplied in following subelements:

- minInspectorVer sets the minimum version of the client inspection script that supports the execution of the group. If the group execution requires a specific version of the inspection script, use this element to set the version.

  If the version specified is greater than the version of the inspection script, then the checks within the group are not executed.

  If the minInspectorVer element is not included, the checks in the group are always executed.

- checkLevel sets the execution of a group according to the level of checking that is done by the client inspector. PTC has defined the following levels:

Check Group	**Level**
Basic hardware and operating system check	1
Installed software and versions	2
TPI and TAN check	4
Reserved for future use	8, 16, and beyond

  For example, set the checkLevel element to 2 for a new check group that checks installed software. If the check group checks for items related to a TAN or TPI, set the checkLevel element to 4.

To show how the checkLevel element can be used, add the following simple check group named “Sample group” that consists of one registry check action to the end of the WCClientInspectorReference.xml file (before </clientCheck>) as follows:

```
  </checkGroup>
  <checkGroup>
    <id>Sample group</id>
    <description>Sample group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>Example</id>
      <description>Sample check for demonstration</description>
      <registry>
        <path>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control</path>
        <name>ComputerName</name>
        <getValue />
      </registry>
      <outResult />
    </check>
  </checkGroup>
</clientCheck>
```

In the example, minInspectorVer is 1 and checkLevel of the group is 32.

Save your changes and run the client inspector with modified definition XML file from a command shell as follows:

```
cscript vbs\WCClientInspector.vbs
```

The output written to the screen is similar to the following:

```
                          :
[4/27/2009 9:26:48 @ localhost] CheckGroup + TAN138710 start
[4/27/2009 9:26:49 @ localhost] CheckGroup - TAN138710 Done
[4/27/2009 9:26:49 @ localhost] CheckGroup + TAN120097 start
[4/27/2009 9:26:49 @ localhost] CheckGroup - TAN120097 Done
[4/27/2009 9:26:49 @ localhost] CheckGroup + TPI130854 start
[4/27/2009 9:26:49 @ localhost] CheckGroup - TPI130854 Done
[4/27/2009 9:26:49 @ localhost] CheckGroup * Sample group skipped
[4/27/2009 9:26:49 @ localhost] End
```

From the output, you can see that the Sample group was skipped. It was skipped because the checkLevel value was 32 and the default execution check level is 7 (1+2+4)

Save your changes and run the client inspector again, specifying an execution level of 255:

```
cscript vbs\WCClientInspector.vbs/exec 255
```

The output written to the screen is similar to the following:

```
                          :
[4/27/2009 9:47:32 @ localhost] CheckGroup + TAN138710 start
[4/27/2009 9:47:34 @ localhost] CheckGroup - TAN138710 Done
[4/27/2009 9:47:34 @ localhost] CheckGroup + TAN120097 start
[4/27/2009 9:47:34 @ localhost] CheckGroup - TAN120097 Done
[4/27/2009 9:47:34 @ localhost] CheckGroup + TPI130854 start
[4/27/2009 9:47:34 @ localhost] CheckGroup - TPI130854 Done
[4/27/2009 9:47:34 @ localhost] CheckGroup + Sample group start
[4/27/2009 9:47:34 @ localhost] CheckGroup - Sample group Done
[4/27/2009 9:47:34 @ localhost] End
```

From the output, you can see that the Sample group was run along with the other check groups.

Save your changes and run the client inspector again, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The output written to the screen is similar to the following:

```
                          :
[4/27/2009 9:56:5 @ localhost] CheckGroup * TPI133412 skipped
[4/27/2009 9:56:5 @ localhost] CheckGroup * TAN138710 skipped
[4/27/2009 9:56:5 @ localhost] CheckGroup * TAN120097 skipped
[4/27/2009 9:56:5 @ localhost] CheckGroup * TPI130854 skipped
[4/27/2009 9:56:5 @ localhost] CheckGroup + Sample group start
[4/27/2009 9:56:5 @ localhost] CheckGroup - Sample group Done
[4/27/2009 9:56:5 @ localhost] End
```

From the output, you can see that the Sample group was the only check group run.

To show how the minInspectorVer element can be used, change the value in the simple check group named “Sample group” to the following:

```
    <minInspectorVer>9</minInspectorVer>
```

Save your changes and rerun the client inspector, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The output written to the screen is similar to the following:

```
                                    :
[4/27/2009 10:49:25 @ localhost] CheckGroup * TAN138710 skipped
[4/27/2009 10:49:25 @ localhost] CheckGroup * TAN120097 skipped
[4/27/2009 10:49:25 @ localhost] CheckGroup * TPI130854 skipped
[4/27/2009 10:49:25 @ localhost] WARNING: Insufficient script version 1 (minimum 9)
[4/27/2009 10:49:25 @ localhost] WARNING: CheckGroup Sample group will be skipped
[4/27/2009 10:49:25 @ localhost] End
```

The warring messages are returned because the new check group does not meet minimum script version of “9” that is specified in the definition.

Before moving on to the next example, set minimum script version back to “1”.

Creating a Registry Check

The following example demonstrates how to add a new registry check. The example creates a check that retrieves the time zone setting for the client machine. The information is extracted in from the following registry path:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation

The information is in the value named:

StandardName

It is a string vale REG_SZ.

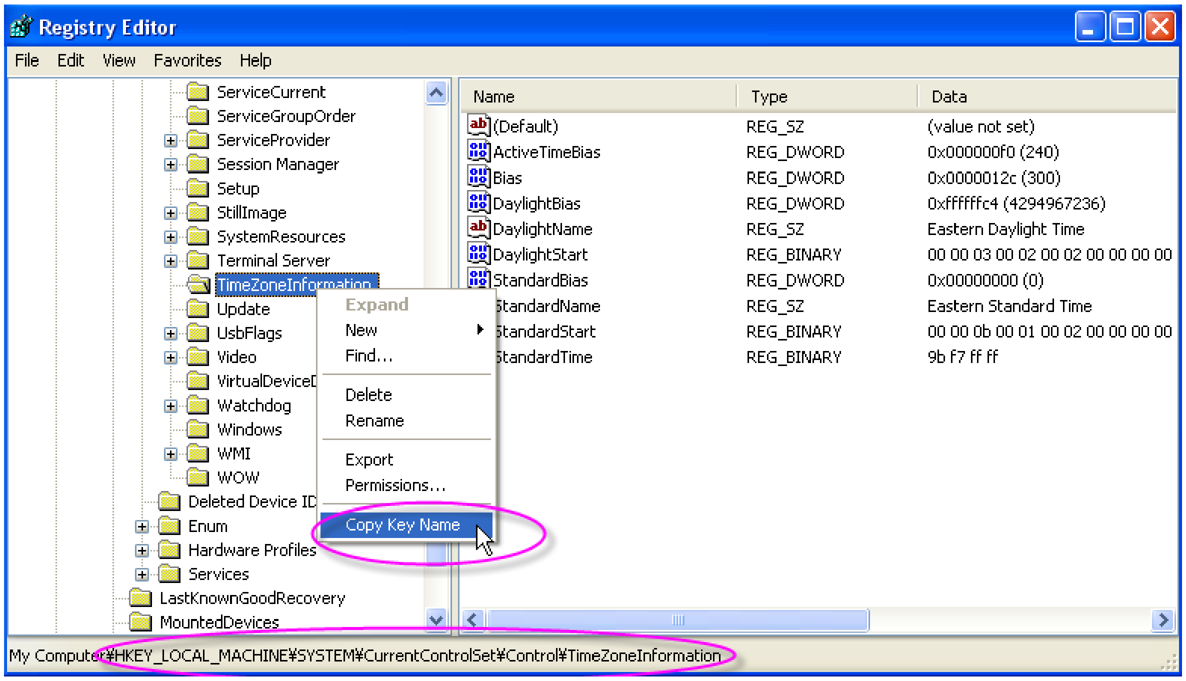
Add the following new check group to the existing definition XML before </clientCheck>:

```
  <checkGroup>
    <id>Timezone</id>
    <description>Timezone check group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>Timezone</id>
      <description>Timezone check for demonstration</description>
      <registry>
        <path></path>
        <name></name>
        <getValue />
      </registry>
      <outResult />
    </check>
  </checkGroup>
</clientCheck>
```

In the example, minInspectorVer is 1 since the existing the inspection script can execute this check. To keep all of the examples in this chapter under a common check level, checkLevel is set to 32.

The id and description elements for the check group and check item can be any strings of your choosing. PTC recommends that descriptions clearly state purpose of the check. Also the id should be a short string that is unique within the definition XML file. The ID is intended to be used as label when results are summarized in a spreadsheet file. For extraction details, see Running the Extraction Tool.

To reduce amount of typing and chances of making any mistake, open regedit and copy the registry key name as shown in the following screen capture:



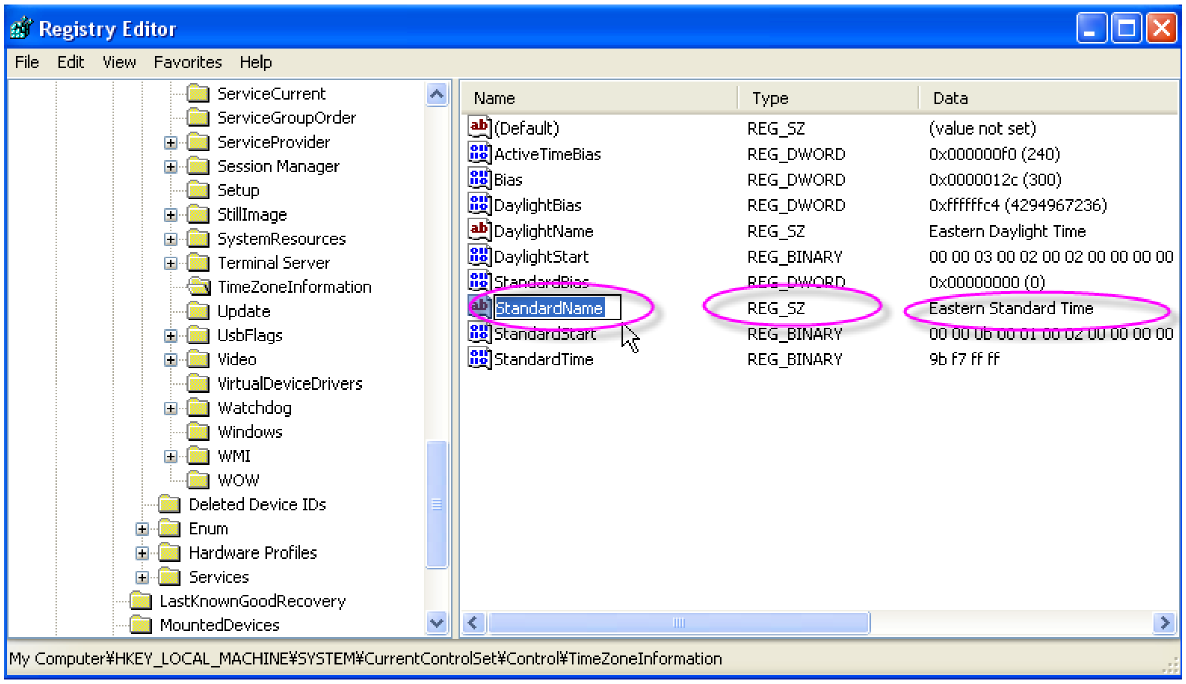
Name	Type	Data
(Default)	REG_SZ	(value not set)
ActiveTimeBias	REG_DWORD	0x000000f0 (240)
Bias	REG_DWORD	0x0000012c (300)
DaylightBias	REG_DWORD	0xffffffc4 (4294967236)
DaylightName	REG_SZ	Eastern Daylight Time
DaylightStart	REG_BINARY	00 00 03 00 02 00 02 00 00 00 00
StandardBias	REG_DWORD	0x00000000 (0)
StandardName	REG_SZ	Eastern Standard Time
StandardStart	REG_BINARY	00 00 0b 00 01 00 02 00 00 00 00
StandardTime	REG_BINARY	9b f7 ff ff

My Computer¥HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Control¥TimeZoneInformation

Return to the XML editor and paste the key between <path and </path>. The path element is similar to the following:

```
<path>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation</path>
```

Similarly, copy the registry value name from regedit and paste it into XML editor in the name element. For example, copy the name from the registry value name:



ServiceCurrent
ServiceGroupOrder
ServiceProvider
Session Manager
Setup
StillImage
SystemResources
Terminal Server
TimeZoneInformation
Update
UsbFlags
Video
VirtualDeviceDrivers
Watchdog
Windows
WMI
WOW
Deleted Device IDs
Enum
Hardware Profiles
Services
LastKnownGoodRecovery
MountedDevices

Name	Type	Data
(Default)	REG_SZ	(value not set)
ActiveTimeBias	REG_DWORD	0x000000f0 (240)
Bias	REG_DWORD	0x0000012c (300)
DaylightBias	REG_DWORD	0xffffffc4 (4294967236)
DaylightName	REG_SZ	Eastern Daylight Time
DaylightStart	REG_BINARY	00 00 03 00 02 00 02 00 00 00 00
StandardBias	REG_DWORD	0x00000000 (0)
StandardName	REG_SZ	Eastern Standard Time
StandardStart	REG_BINARY	00 00 0b 00 01 00 02 00 00 00 00
StandardTime	REG_BINARY	9b f7 ff ff

My Computer¥HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Control¥TimeZoneInformation

Then paste the name between <name> and </name>:

```
<name>StandardName</name>
```

Save your changes and run the client inspector again, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The output written to the screen is similar to the following:

```
                              :
[4/27/2009 12:7:23 @ localhost] CheckGroup * TAN138710 skipped
[4/27/2009 12:7:23 @ localhost] CheckGroup * TAN120097 skipped
[4/27/2009 12:7:23 @ localhost] CheckGroup * TPI130854 skipped
[4/27/2009 12:7:23 @ localhost] CheckGroup + Sample group start
[4/27/2009 12:7:23 @ localhost] CheckGroup - Sample group Done
[4/27/2009 12:7:23 @ localhost] CheckGroup + Timezone start
[4/27/2009 12:7:23 @ localhost] CheckGroup - Timezone Done
```

The resulting XML file contains content similar to the following:

```
  <checkGroup>
    <id>Timezone</id>
    <description>Timezone check group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>Timezone</id>
      <description>Timezone check for demonstration</description>
      <registry>
<path>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\TimeZoneInformation</path>
        <name>StandardName</name>
        <getValue/>
      </registry>
    <outResult>
      <outTimestamp>4/27/2009 12:7:23</outTimestamp>
      <outStatus>Success</outStatus>
      <outDataType>REG_SZ</outDataType>
      <outData>Eastern Standard Time</outData>
    </outResult>
    </check>
  </checkGroup>
```

In this example, the machine has been set to time zone “Eastern Standard Time” as REG_SZ (string).

Creating a File Attribute Check

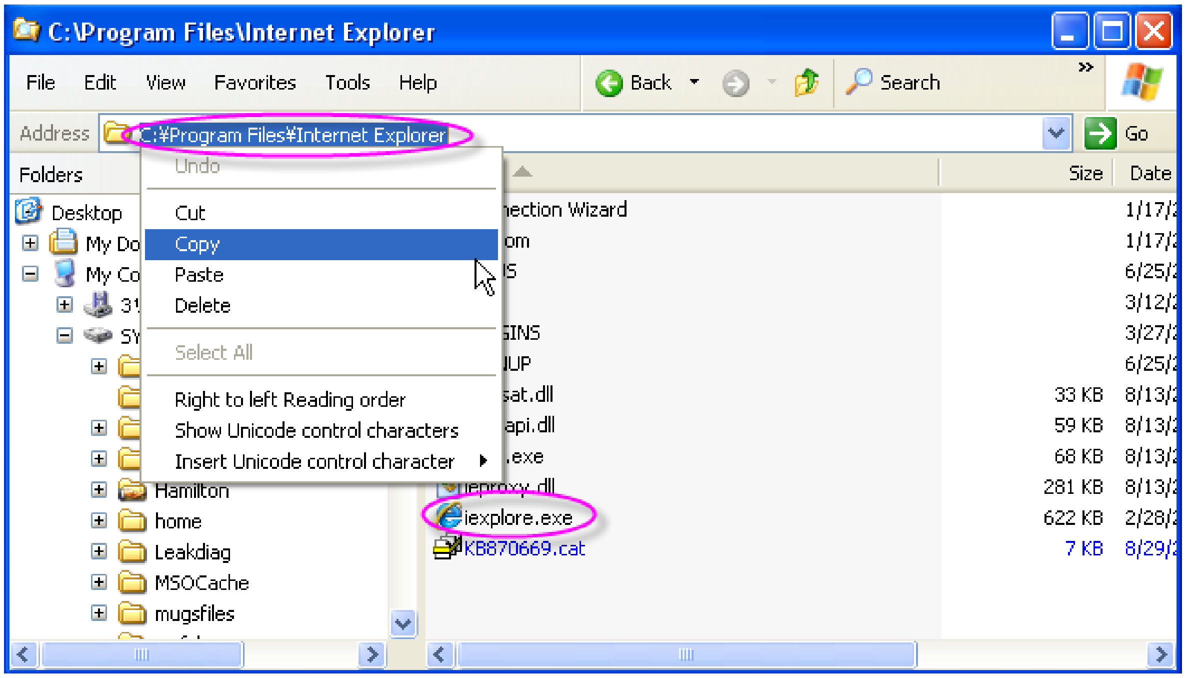
The following example demonstrates how to create a check that extracts information about, Internet Explorer executable file. The example assumes that Internet Explorer 7, under your Program Files folder.

Tip: For the following example, you can use any file. You can change the file to one that is in your environment.

Add the following new check group to the existing definition XML before </clientCheck>:

```
  <checkGroup>
    <id>IE executable file</id>
    <description>IE executable file check group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>IE executable file</id>
      <description>IE executable file check for demonstration</description>
      <file>
        <path></path>
      </file>
      <outResult />
    </check>
  </checkGroup>
</clientCheck>
```

Open the Internet Explorer installation folder in Windows Explorer and copy the path and name of “iexplore.exe”. For example:


C:\Program Files\Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Address
C:¥Program Files¥Internet Explorer
Go
Folders
Undo
Cut
Copy
Paste
Delete
Select All
Right to left Reading order
Show Unicode control characters
Insert Unicode control character
Size
Date
iexplore.exe
622 KB
KB870669.cat
7 KB

Return to the XML editor and paste the file path between <path and </path>. The path element is similar to the following:

```
<path>C:\Program Files\Internet Explorer\iexplore.exe</path>
```

This check only works if the Internet Explorer installation folder is located on the C drive; it will not execute successfully if the folder is on the D drive. To remove drive dependencies such as this, you can replace the drive letter with the %SystemDrive% pseudo variable:

```
<path>%SystemDrive%\Program Files\Internet Explorer\iexplore.exe</path>
```

Save your changes and run the client inspector again, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The resulting XML file contains content similar to the following:

```
  <checkGroup>
    <id>IE executable file</id>
    <description>IE executable file check group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>IE executable file</id>
      <description>IE executable file check for demonstration</description>
      <file>
        <path>%SystemDrive%\Program Files\Internet Explorer\iexplore.exe</path>
      </file>
      <outResult>
        <outTimestamp>4/27/2009 13:2:24</outTimestamp>
        <outStatus>Success</outStatus>
        <outDataType>NA</outDataType>
        <outData>CreationDate=20070117150624.514750-300||
LastModified=20090227235441.000000-300||LastAccessed=20090427122553.437198-
240||FileSize=636072 byte||Version=7.00.6000.16827 (vista_gdr.090226-1506)</outData>
        </outResult>
      </check>
    </checkGroup>
```

In the result file, the executable “iexplore.exe” on this machine is file size 636,072 byte and version “7.00.6000.16827 (vista_gdr.090226-1506)”.

Note: If you have a global environment that consists of machines with different languages, you will have to either prepare a localized definition XML file for each language or consider other workaround (use of DOS file short name, for example).

The definition XML is UTF-8 format and it should be able to handle non-ASCII strings (for example, “Program Files” in German, French, Japanese, and so on.) However, the definition file that in this example is only specific to an English operating system.

Creating an Environment Variable Check

There are two types of environment variables in Windows:

- User variable
- System variable

The following example demonstrates how to create a check for each type of environment variable and how the inspector extracts the values.

To set up your environment for this example:

- Create a new User variable named "USER_VAR_TEST" with a value of "Hello".
- Create a System variable named “SYSTEM_VAR_TEST” with a value of “World".

After the variables have been created, add the following new check group to the existing definition XML before </clientCheck>:

```
<checkGroup>
  <id>Variable checks</id>
  <description>Variable check group for demonstration</description>
  <minInspectorVer>1</minInspectorVer>
  <checkLevel>32</checkLevel>
  <check>
    <id>Variable check A</id>
    <description>Variable check demonstration</description>
    <variable>
      <name>USER_VAR_TEST</name>
    </variable>
    <outResult />
  </check>
  <check>
    <id>Variable check B</id>
    <description>Another variable check demonstration</description>
    <variable>
      <name>SYSTEM_VAR_TEST</name>
    </variable>
    <outResult />
  </check>
</checkGroup>
```

Save your changes and run the client inspector again, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The resulting XML file contains content similar to the following:

```
<checkGroup>
  <id>Variable checks</id>
  <description>Variable check group for demonstration</description>
  <minInspectorVer>1</minInspectorVer>
  <checkLevel>32</checkLevel>
  <check>
   <id>Variable check A</id>
    < description>Variable check demonstration</description>
    <variable>
    <name>USER_VAR_TEST</name>
    </variable>
    <outResult>
      <outTimestamp>4/27/2009 13:58:1</outTimestamp>
      <outStatus>Success</outStatus>
      <outDataType>NA</outDataType>
      <outData>User=PTCNET\yonizuka||Value=Hello</outData>
    </outResult>
  </check>
  <check>
    <id>Variable check B</id>
    <description>Another variable check demonstration</description>
    <variable>
    <name>SYSTEM_VAR_TEST</name>
    </variable>
    <outResult>
      <outTimestamp>4/27/2009 13:58:2</outTimestamp>
      <outStatus>Success</outStatus>
      <outDataType>NA</outDataType>
      <outData>User=SYSTEM||Value=World</outData>
    </outResult>
  </check>
</checkGroup>
```

Note: Both variables are extracted along with information that "USER_VAR_TEST" is set for user "yonizuka" in domain "PTCNET" and "SYSTEM_VAR_TEST" is a system-wide variable.

After you are satisfied with the results of this example, you can remove the two test variables from your environment variables and delete the Variable checks checkGroup from the definition file.

Creating a WMI Check

There are many WMI queries that can be done. This topic does not document all of types of WMI queries that you can do. For WMI information and query examples, see the Microsoft Web site at:

http://msdn.microsoft.com/en-us/library/aa394582(VS.85).aspx

The inspector client allows any valid WMI query statement against information stored in "root\CMIV2" name space.

The required query statement is very similar to a database SQL statement and is called WQL. The major difference is that only "select" statements are allowed. There also is variation of the "where" clause that is available for complex queries.

The following example demonstrates how to create a check to extract information about your motherboard BIOS. The WQL used to retrieve BIOS name is:

Select Caption from Win32_BIOS

(This query was found using the Microsoft reference page or another source.)

The query is the exact query used in scripting, command line utilities, or high level language programming.

Add the following new check group to the existing definition XML before </clientCheck>:

```
  <checkGroup>
    <id>BIOS</id>
    <description>BIOS check group for demonstration</description>
    <minInspectorVer>1</minInspectorVer>
    <checkLevel>32</checkLevel>
    <check>
      <id>BIOS</id>
      <description>BIOS check group for demonstration</description>
      <wmi>
        <wql>Select Caption from Win32_BIOS</wql>
        <name>Caption</name>
      </wmi>
      <outResult />
    </check>
  </checkGroup>
```

Save your changes and run the client inspector again, specifying an execution level of only 32:

```
cscript vbs\WCClientInspector.vbs/exec 32
```

The resulting XML file contains content similar to the following:

```
<outResult>
  <outTimestamp>4/27/2009 14:46:31</outTimestamp>
  <outStatus>Success</outStatus>
  <outDataType>NA</outDataType>
  <outData>Phoenix ROM BIOS PLUS Version 1.10 A01</outData>
</outResult>
```

This output shows that the machine has BIOS made by company “Phoenix” and its version is “Version 1.10 A01”.

For additional WMI information, see wmi Check Action.

A

WCClientInspectorReference.dtd

The appendix has the contents of the DTD and describes the elements in the DTD.

By design, the same WCClientInspectorReference.dtd file is used for both the definition XML file and result XML file. All definition and result files must be well-formed and valid XML files.

DTD Content

Note: Although the DTD file allows nested “check” elements (meaning “check” inside of “check”), this functionality is intended to be used in a future extension and is not supported in the current client inspection script.

The following DTD content has been modified to fit within the page margins:

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT clientCheck (definitionVersion, outScriptVersion, outRetrievingHostname,
        outRetrievingUser, outCheckedHostname, definitionDescription, checkGroup*)>
<!ELEMENT definitionVersion (#PCDATA)>
<!ELEMENT outScriptVersion (#PCDATA)>
<!ELEMENT outRetrievingHostname (#PCDATA)>
<!ELEMENT outRetrievingUser (#PCDATA)>
<!ELEMENT outCheckedHostname (#PCDATA)>
<!ELEMENT definitionDescription (#PCDATA)>
<!ELEMENT checkGroup (id, description, minInspectorVer, checkLevel, check+)>
<!ELEMENT id (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT minInspectorVer (#PCDATA)>
<!ELEMENT checkLevel (#PCDATA)>
<!ELEMENT check (id, description, (registry|wmi|storage|software|security|file|
          variable|driver|network), check*, outResult+)>
<!ELEMENT outResult (outTimestamp?, outStatus?, outDataType?, outData?)>
<!ELEMENT outTimestamp (#PCDATA)>
<!ELEMENT outStatus (#PCDATA)>
<!ELEMENT outData (#PCDATA)>
<!ELEMENT outDataType (#PCDATA)>
<!ELEMENT registry (path, name, (getKey|getValue))>
<!ELEMENT file (path)>
<!ELEMENT variable (name)>
<!ELEMENT wmi (wql, name)>
<!ELEMENT security (wql, name)>
<!ELEMENT path (#PCDATA)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT getKey EMPTY>
<!ELEMENT getValue EMPTY>
<!ELEMENT wql (#PCDATA)>
<!ELEMENT storage EMPTY>
<!ELEMENT software EMPTY>
<!ELEMENT network EMPTY>
<!ELEMENT driver (deviceClassMOUSE|deviceClassKEYBOARD|deviceClassNET|
          deviceClassDISPLAY|deviceClassMONITOR|deviceClassUSB)>
<!ELEMENT deviceClassMOUSE EMPTY>
<!ELEMENT deviceClassKEYBOARD EMPTY>
<!ELEMENT deviceClassNET EMPTY>
<!ELEMENT deviceClassDISPLAY EMPTY>
<!ELEMENT deviceClassMONITOR EMPTY>
<!ELEMENT deviceClassUSB EMPTY>
```

DTD Element Descriptions

The following sections, describe many of the DTD elements.

As it is mentioned earlier, some part of the definition XML is used to define check actions and some part of XML is kept blank so that the check script can fill in values at the run time.

The DTD uses a naming convention for the elements that contain blanks so that they are easily recognized. These elements start with “out”. This indicates that the elements are expecting output from the inspector software. For example, the outScriptVersion element must be exist, but should be kept as an empty in XML definition. Upon the execution of the inspector, the resulting XML file has element value for the outScriptVersion element, which is the version of VBScript used to the check the client machine.

definitionVersion Element

This element is used to keep track of definition XML file version.

Note: Use this is information to ensure that the inspector is using intended version of definition XML file. PTC recommends that you use an integer for the value.

The software does not check level specified in this element value.

outScriptVersion Element

This element is filled in at the inspector run time.

The element value is the version of VBScript "WCClientInspector.vbs" that is being used for checking the client machine.

Note: Use this is information to ensure that the inspector is using intended version of the client-side script.

Furthermore, the same script version is used before individual check group execution to see the script is sufficiently new for the specific check. This feature for version check per check group is described in detail in

outRetrievingHostname Element

This element is filled in at the inspector run time.

The element value is the host name of the machine from where you have executed the inspection script.

For example, if you are running the inspector with remote execution "/host" option, the host name is not the remote client machine from which the inspector is examining and extracting the information, but rather is the machine name where the inspector is running. If you are checking the local machine, this outRetrievingHostname element value is the same as the machine being checked (see outCheckedHostname Element).

outRetrievingUser Element

This element is filled in at the inspector run time.

The element value is the login user name who is executing the inspection script.

Note: Some check items are sensitive to the login user. For instance, registry value reading for HKEY_CURRENT_USER is specific to the login user. In another words, the result taken as one user from a machine is different from result taken as another user from exact the same machine.

outCheckedHostname Element

This element is filled in at the inspector run time.

The element value is the host name of the machine that is being examined. For example, if you are running the inspector with remote execution "/host" option, the value is the host name of the remote machine. If you are checking the local machine, this outCheckedHostname element value is same as the machine name in outRetrievingHostname.

definitionDescription Element

This element is for informational purposes only.

You can enter any descriptive information about this definition XML file in this element.

checkGroup Element

The chcekGroup element can be used to group set of check items into one logical group. The chcekGroup consists of one or many check elements.

This element also contains a several supporting subelements such as "id", "description", "minInspectorVer", and "checkLevel".

id Element (Subelement of checkGroup)

This element identifies the check group.

Tip: Although there is no restriction in the inspection script, PTC recommends that the ID is unique across the definition XML file.

Use a descriptive but one or two word phrase because the ID is intended to be used as label when results are summarized in a spreadsheet file. For extraction details, see Running the Extraction Tool.

description Element (Subelement of checkGroup)

This element is for informational purposes only.

Enter descriptive information about the purpose of this check group.

minInspectorVer Element (Subelement of checkGroup)

The inspection script has a version number (an integer) which is stored in the script.

Use the minInspectorVer element value to identify which version of the inspection script must be used to execute this specific check group:

- If the version specified in minInspectorVer is less than or equal to the inspection script version, then the check group is performed.
- If the version specified in minInspectorVer is greater than the inspection script version, then the check group is skipped.
- If the minInspectorVer element is omitted, then the check group is performed.

For an example, if one of your end user clients has an outdated inspection script, such as version 5, and there is a check group with the minInspectorVer element set to 6, then this specific check group is skipped.

checkLevel Element (Subelement of checkGroup)

Identifies at which check level the check group should execute.

Setting check levels for each check group allows you to selectively execute certain check groups. The checkLevel value is examined by the inspection script before the execution of each check group and the group is skipped if the checkLevel specified for the check group is not one of the check levels specified for the specific client inspector run.

The level is processed against bitmask and, therefore, it must be an integer that is power of two (1, 2, 4, 8, 16, …). The following table describes which level to select for a check group:

Check Group	**Level**
Basic hardware and operating system check	1
Installed software and versions	2
TPI and TAN check	4
Reserved for future use	8, 16, and beyond

By default, check groups with check levels of 1, 2, and 4 execute during a client inspector run. You can use the /exec:<check_level> option to change the client inspector check level.

For examples of the check level, see Modifying Your Client Inspector Definition File.

check Element (Subelement of checkGroup)

Each check group must contain at least one check element and can contain many check elements.

Each check element holds exactly one atomic check action. An atomic check action is the minimum unit of generalized and specialized check item. There are a several types of atomic check actions available that are discussed later in this section.

Along with the check action, the check element also contains id, description, and outResult (which is a placeholder for the extracted result).

id Element (Subelement of check)

This element identifies the check item.

Tip: Although there is no restriction in the inspection script, PTC recommends that the ID is unique across the definition XML file.

Use a descriptive but one or two word phrase because the ID is intended to be used as label when results are summarized in a spreadsheet file. For extraction details, see Running the Extraction Tool.

description Element (Subelement of check)

This element is for informational purposes only.

Enter descriptive information about the purpose of this check item.

Atomic Check Actions (Subelements of check)

An atomic check action is the minimum unit of a generalized or specialized check item. Each atomic check action is an act performed within a check that is independent from other check actions.

The following table describes the supported check actions:

Check Action	**Description**
wmi	Returns the result of any WMI query against the CIMV2 class.
registry	Returns specific registry key or value information.
file	Returns specific file information (size, timestamps, and so on).
variable	Returns specific environment variable value.
storage	Returns storage device information.
software	Returns installed software list.
security	Returns security center information.
driver	Returns registered device driver information.
network	Returns network adapter information.

For check actions details, see Atomic Check Actions.

outResult Element (Subelement of check)

This element is placeholder for the extracted result.

In your definition XML, you must have one outResult element with its value empty. For example:

```
<outResult />
```

Upon the completion of the each atomic check item, the inspector appends the outResult element with extracted check result.

Note: There can be multiple outResult elements created under one check element. For example, if you perform CPU clock speed check on a machine with four CPUs, there would be four outResult elements created under the check element.

The outResult element consists of four subelements:

outTimestamp
outStatus
outDataType
outData

outTimestamp Element (Subelement of outResult)

This element is filled in at the inspector run time.

This element contains the timestamp of when the check was performed.

The time captured is based on system clock of each machine. Issues can arise if system clock of a specific machine is set to incorrect time. Also be aware that you can have result XMLs from multiple time zones.

Note: The format of the timestamp is always the same; it is not affected by each machines locale settings. The format is:

mm/dd/yyyy hh:mm:ss

For example, 4/20/2009 8:57:18.

outStatus Element (Subelement of outResult)

This element is filled in at the inspector run time.

Typically, the element value is the word "Success" or other informative descriptions in case when the inspector failed to obtain the asked value.

Note: A value other than "Success" does not necessary mean there is a problem.

For example, assume your purpose is to make sure the client machine does not have certain registry key created in specific registry path. The only way the inspector can verify the non-existence of the registry key is to try to retrieve the value and let it fail to retrieve the result. The status in this case is the value "Query no match".

Another case that you would legitimately expect not to get "Success" is due to the fact that the inspector software does not have dynamic conditional flow change based on obtained result (for instance, there is no if-else-else-if…). For this reason, the definition XML must be designed to cover all possible scenarios. The inspector will perform the check even if the check may not be applicable to the certain machine.

For example, suppose there are two sets of different check items one for the machines with Internet Explorer 7 and the other for the machines with Internet Explorer 8. The inspector has to perform both Internet Explorer 7 and Internet Explorer 8 checks regardless of what version of Internet Explorer that the client actually has in the system.

All the interpretation of the check results is done in post processing of the resulting XML and not during the run time.

The main reasons for not implements dynamic conditional flow change include the following:

- Ease of administration -- it would be very difficult to create and maintain the check definitions XML that is a faithful translation of decision tree that covers all possible configuration scenarios in the real life.
- Simplified design -- it would make the inspector client script overly complex. Since it is important that the client side script be able to run on any machine without preconditions, it is not desirable to implement complex logic on the client side.

Therefore, by design, all necessary complexities of the result interpretations will have to reside in post processing of the results and not in the data collection software.

outDataType Element (Subelement of outResult)

This element is filled in at the inspector run time.

Note: The outDataType element is used specifically for registry value retrieval. All other atomic check types set the value to "NA".

In case of a registry check, the value contains the data type of the registry so that you can differentiate between the number 1 and the string "1".

Currently supported registry data types are:

REG_SZ
REG_EXPAND_SZ
REG_DWORD
REG_MULTI_SZ
REG_BINARY

For the REG_BINARY registry type, the inspector can detect its data type but the outData element value is set to an empty string.

For the REG_MULTI_SZ registry type, the inspector concatenates multiple strings into a multiline string (delimited by "carriage return and new line").

For all other registry data types, the inspector returns the Microsoft internal enumeration integer for the data type and outStatus element value is the "Query type not supported" string.

outData Element (Subelement of outResult)

This element is filled in at the inspector run time.

The outData element contains the extracted value from each atomic check action.

By design, the outData element value has to be a string that uses UTF-8 encoding. If the extracted value is not valid UTF-8, the outData element value is overridden and replaced with the "Omitted due to invalid UTF-8 string" string.

Atomic Check Actions

Atomic check actions are subelements of the check element. The check action is the minimum unit of a check performed by the inspector.

The inspector software intentionally restricts itself so that all atomic check actions must be performed through WMI (Windows Management Instrumentation) queries. The inspector allows any WMI query on each client and there are vast amount of information available through the WMI data class. Most check needs should be covered despite this restriction.

There are no shell command actions used for the check actions. There are a few exceptions outside of the check actions where the inspection script allows limited shell command execution. Shell command execution is allowed by the inspector to copy or FTP the result file to a central location.

This restriction is for security purposes; it guarantees that there will be no accidental alternation of the client machine configurations. It also greatly reduces or eliminates the chance to accidentally expose sensitive personal information about individual end users.

The supported atomic check actions are described in the following sections.

wmi Check Action

The inspector allows you to perform any WMI query against "root\CIMV2" name space using Microsoft's WQL language.

The name space "root\CIMV2" is where a majority of the hardware and software information is stored. See the Microsoft Web site for details about the vast amount of data classes that you can access via WMI query.

The wmi action element consists of the wql and name subelements:

- wql contains the valid WQL statement that you would use in any other tools, scripts, or high level programming language to perform WMI queries.
- name is used to specify the specific value name to be retrieved in the result.

For an example, following is a check element that performs a "wmi" action to obtain OS name.

```
<check>
  <id>Demo1</id>
  <description>OS name</description>
  <wmi>
    <wql>Select Caption From Win32_OperatingSystem</wql>
    <name>Caption</name>
  </wmi>
  <outResult />
</check>
```

In the previous example, the inspector queries "Win32_OperatingSystem" class and retrieves only "Caption" and the retrieved string is returned as output.

The Demo1 check result is similar to the following result XML:

```
<check>
  <id>Demo1</id>
  <description>OS name</description>
  <wmi>
    <wql>Select Caption From Win32_OperatingSystem</wql>
    <name>Caption</name>
  </wmi>
  <outResult>
    <outTimestamp>4/20/2009 8:57:18</outTimestamp>
    <outStatus>Success</outStatus>
    <outDataType>NA</outDataType>
    <outData>Microsoft Windows XP Professional</outData>
  </outResult>
</check>
```

Since this is the first time that there is an actual example of result XML that has "outResult" element added by the inspector, notice that the result XML looks very similar to the definition XML file. In fact, the result XML will contains exactly the same contents as in definition XML along with added "result" information. In this example, there is only one matching result that consists of the time when check was executed, status of the check (successfully obtained in the example), data type of the obtained data (not relevant for this check action), and the obtained value (outData is set to "Windows XP Professional" in the example).

Tip: The definition XML specifies value name in two places in the previous example; the word "Caption" is in both the wql element and name element). This is done so that the wql element value accepts any valid WQL select statement. Otherwise you would need to force an artificial limitation or parameterize a part of the statement and then manipulate it internally in the inspection script. Internally, the code performs the WQL select statement in the wql element exactly as-is and returns only the value that has the name specified in the name element if that exists. Therefore, the following subelements also result in the same result output (select everything from "Win32_OperatingSystem" and return only name "Caption"):

```
<wql>Select * From Win32_OperatingSystem</wql>
<name>Caption</name>
```

However for performance purposes, the use of "select *" is not encouraged.

registry Check Action

The registry action element extracts registry key or value information and can be used in two ways:

- Extract a value under specific registry key that you know the exact registry full path. This is achieved by specifying the "<getValue />" empty element as an instruction for the inspection script. For example:

```
<check>
 <id>Demo2</id>
 <description>Demo for registry check</description>
 <registry>
  <path>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters</path>
  <name>Hostname</name>
  <getValue />
 </registry>
 <outResult />
</check>
```

  In this example, registry value of name "Hostname" under registry key path "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters" is extracted. This should result in retrieving hostname of the machine being checked.

- Retrieve all value names under a specific registry full path. This is achieved by specifying the "<getKey />" empty element as an instruction for the inspection script. For example:

```
<check>
 <id>Demo3</id>
 <description>Demo for registry check</description>
 <registry>
  <path>HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters</path>
  <name>Hostname</name>
  <getKey />
 </registry>
 <outResult />
</check>
```

  In this example, registry keys under the path "HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters" are retrieved. Notice the distinction between registry key and registry value. The registry keys are ones represented as the folder-like icons that you see in "regedit.exe". By specifying "<getKey />", the inspector extracts those immediate sub-registry keys (the folder-like items) and not the names of values under the specified path. Therefore, even though the two examples use the same exact full path, the second example does not show value name Hostname (seen the first example) because "Hostname" is name of a value and not a registry key.

In the registry check, following registry hives (the top-most key in the registry path) are supported:

HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
HKEY_USERS
HKEY_CURRENT_CONFIG

It is important to realize that contents of certain registry hives can be different depending on user login context. For instance, contents of registry hive "HKEY_CURRENT_USER" will differ depending on who is currently logging into the system. This is because all login accounts have personalized settings in "HKEY_CURRENT_USER". For this reason, the inspector must be run as each login user of interest if you need to check values under "HKEY_CURRENT_USER". Also, the contents of the registry hive "HKEY_USERS" can be loaded and unloaded dynamically by OS depending on user operations.

file Check Action

The file atomic check action extracts file attribute information from specific file for which you know its full path. For example:

```
<check>
  <id>Demo4</id>
  <description>Demo for file attribute check</description>
  <file>
    <path>D:\Temp\test.exe</path>
  </file>
  <outResult />
</check>
```

Upon successful extraction, the inspector returns the following file attributes in a single string line that is delimited by two pipe characters (||):

CreationDate
LastModified
LastAccessed
FileSize
Version

Version information is returned only when it as available (for example, for ".exe", ".dll" files.). Otherwise, the version is shown as "NA". If specified path is a folder, outResult is shown as "File is a directory". If no file or folder exist in specified path, outResult is shown as "File not found".

Note: The following pseudo environment variables can be used as a part of a path for the file attribute check:

Pseudo Variable	Description
%SystemDrive%	Use in place of "C:", "D:", and so on, where the operating system is installed.
%WindowsDirectory%	Use in place of "C:\WINNT", "D:\WINNT", and so on.
%UserProfileBase%	Use in place of "D:\User Profiles", "E:\User Profiles", and so on.

By using pseudo variables, you can create flexible file attribute checks regardless of specific operating system installation situation of each client machine. The following example shows a series of three checks:

```
<check>
  <id>Demo5-1</id>
  <description>File attribute check demo</description>
  <file>
    <path>%SystemDrive%\Program Files\Java\jdk1.6.0_10\bin\java.exe</path>
  </file>
  <outResult />
</check>
<check>
  <id>Demo5-2</id>
  <description>File attribute check demo</description>
  <file>
    <path>%WindowsDirectory%\NOTEPAD.EXE</path>
  </file>
  <outResult />
</check>
<check>
  <id>Demo5-3</id>
  <description>File attribute check demo</description>
  <file>
    <path>%UserProfileBase%\jsmith.mydomain\My Documents</path>
  </file>
  <outResult />
</check>
```

variable Check Action

The variable atomic check action extracts a system environment variable that has a name that you know in advance. For example:

```
<check>
  <id>Demo6</id>
  <description>Variable check demo</description>
  <variable>
    <name>JAVA_HOME</name>
  </variable>
  <outResult />
</check>
```

This example looks for environment variable name "JAVA_HOME". Since there are two types of variables, "user variables" and "system variables", the inspector returns following information in a single string line delimited by two pipe characters (||):

- If the variable is set as user variable, a string "User=" followed by domain name and username and then followed by the actual value.
- If the variable is set as system variable, a string "User=SYSTEM" followed by the actual value.

If variable is not set as either user variable or system variable, "outStatus" contains the string "Query no match".

storage Check Action

The storage atomic check action extracts information about attached hard disk and any other storage devices. This check is a part of the basic hardware and OS check (in check group 1) and there is no need to add this check action into another check group.

The check can be used in the definition XML as follows:

```
<check>
  <id>Attached storage</id>
  <description>Attached storage devices</description>
  <storage/>
  <outResult/>
</check>
```

This atomic check action is a predefined collection of checks and there is no need for user input in the definition XML. In another words, the element that indicates the check type as "storage" does not take any element value; it is an empty element "<storage />".

Upon successful execution, the inspector returns the result as multiple "outResult" elements (one device per one "outResult"). Each "outData" element consists of following information in a single string line delimited by two pipe characters (||):

- Drive letter that the device is mapped
- Device type description
- Free space in byte (if applicable)

software Check Action

The software atomic check action extracts information about installed system-wide software, as well as software installed by the login user account. This check action is predefined collection of checks and there is no place for user input in the definition XML.

The check can be used in the definition XML as follows:

```
<check>
  <id>Installed software</id>
  <description>Installed software</description>
  <software/>
  <outResult/>
</check>
```

This check is a part of the default check group 2 and there is no need to add this check action into another check group.

Upon successful execution, the inspector returns the result as multiple outResult elements (one software product per "outResult"). Each outData element consists of following information in a single string line delimited by two pipe characters (||):

- DisplayName
- Publisher
- DisplayVersion

security Check Action

The security atomic check action extracts information managed by Windows control panel "Security Center".

The following example shows how to extract company name of the AntiVirus software product:

```
<check>
  <id>AntiVirus company</id>
  <description>AntiVirus company name</description>
  <security>
    <wql>Select companyName from AntiVirusProduct</wql>
    <name>companyName</name>
  </security>
  <outResult />
</check>
```

This atomic check is created specifically because "Security Center" information resides outside of typical "root\CIMV2" WMI name space (otherwise, the same checks could have been achieved by administrator using "wmi" check action without needs for dedicated atomic action).

This check is a part of the basic hardware and OS check (in check group 1) and there is no need to add this check action into another check group.

Note: If virus scan and local firewall software is not managed by "Security Center", no information is obtained by this check.

If there are multiple company names returned, each name has one dedicated "outResult" and each "outData" element contains the information without alteration (unlike other checks where each "outData" consists multiple pieces of information concatenated into one string delimited by "||").

driver Check Action

The driver atomic check action extracts information about digitally signed device drivers. The device drivers are grouped into a several device classes and the check action expects one instructional element to specify the device class of interest. For example, by having an empty element "<deviceClassMONITOR />" in the following check, the inspector retrieves information about monitor device drivers:

```
<check>
  <id>Monitor driver</id>
  <description>Monitor driver</description>
  <driver>
    <deviceClassMONITOR />
  </driver>
  <outResult />
</check>
```

Following instructional elements are available:

Element	**Description**
deviceClassMOUSE	Mouse and pointing device
deviceClassKEYBOARD	Keyboard

Element	Description
deviceClassNET	Network card
deviceClassDISPLAY	Display device
deviceClassMONITOR	Monitor
deviceClassUSB	USB driver

Some device classes (but not all) checks are parts of the basic hardware and OS check (in check group 1).

Upon successful extraction, the inspector returns the following file attributes in a single string line delimited by two pipe characters (||):

- Description
- .inf file name
- Driver provider
- Driver date
- Driver version

network Check Action

The network atomic check action extracts information about the network adaptor that is enabled. It can be used in the definition XML as follows:

```
<check>
  <id>Network card</id>
  <description>Network card and settings</description>
  <network />
  <outResult />
</check>
```

This check is a part of the basic hardware and OS check (in check group 1) and there is no need to add this check action into another check group.

Upon successful extraction, the inspector returns the following file attributes in a single string line delimited by two pipe characters (||):

Attribute	Description
Description	Description of the adapter.
DefaultTOS	Type of Service value.
DHCPEnabled	Determines if DHCP is used to obtain IP address.