

Prerequisite Scanner
Version 1.2 Sprint 3 Driver

User's Guide



Prerequisite Scanner
Version 1.2 Sprint 3 Driver

User's Guide



Note

Before using this information and the product it supports, read the information in “Notices” on page 153.

This edition applies to Version 1.2 of IBM Prerequisite Scanner and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 2009, 2012.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	v
--------------------------	----------

Tables	vii
-------------------------	------------

Chapter 1. Prerequisite Scanner

overview	1
Prerequisite Scanner architecture.	1
Prerequisite properties	1
Product codes	12
Prerequisite Scanner configuration files	13
Prerequisite Scanner collectors	20
Prerequisite Scanner evaluators.	22
Output formats	23
Prerequisite Scanner Java Developer toolkit.	33
XML schema file for the XML result file	33
Scanning process	34
New in this release	36

Chapter 2. Installing Prerequisite

Scanner	39
Prerequisites	39
Installing the compressed file	40
Uninstalling Prerequisite Scanner	40

Chapter 3. Extending Prerequisite

Scanner	41
Before you run Prerequisite Scanner	41
Required checks and extension tasks for Windows systems	41
Required checks and extension tasks for UNIX systems.	42
Adding product codes.	43
Creating custom configuration files	43
Adding prerequisite properties	45
Editing prerequisite properties	47
Creating custom collectors for Windows systems	48
Creating custom VBScript collectors common to all configuration files	48
Creating custom VBScript collectors specific to a product and product version	50
Creating custom collectors for UNIX systems	52
Editing the package test script for UNIX systems.	53
Creating custom evaluators for Windows systems	55
Creating custom evaluators for UNIX systems.	59

Chapter 4. Running Prerequisite

Scanner	61
prereq_checker	61
Running Prerequisite Scanner from the command line	67
Common directory locations.	68

Chapter 5. Troubleshooting

Prerequisite Scanner	69
Troubleshooting on Windows systems	69
Troubleshooting on UNIX systems.	71
Execution problems.	73
Return codes	74

Appendix A. Product codes reference 77

Appendix B. Configuration files reference. 81

Appendix C. Prerequisite properties reference. 85

Common data properties	86
System behavior for Memory prerequisite property and Tivoli Monitoring agents	90
Autonomic Deployment Engine data properties	90
Connectivity data properties.	91
DB2 data properties	92
MS SQL Server data properties	92
Internet Explorer data properties	93
Network data properties	93
Oracle data properties.	94
Operating system data properties	95
Installed software data properties	106
User data properties	106
Windows network data properties	107
UNIX network data properties	107

Appendix D. Predefined collectors for UNIX systems 109

Appendix E. Common functions for Windows systems 115

allFiles()	116
arrayToString()	116
bigthan()	117
changeMG()	117
checkItemToString()	118
dictionaryToString()	118
exeCommand()	119
filterCommand()	119
filterFile().	120
findNewest()	120
findSuitableFile()	121
fmt()	122
formatForDisplay()	122
formatSizeForDisplay()	123
getDecimalSeparator()	123
getFirstMatch()	123
isMatch()	124
notInLatter().	124

passOrFail()	125
ppread()	126
readFile()	126
unitMGTOG()	127
varToString()	127

Appendix F. Logging utility sub routines for Windows systems 129

Appendix G. File utility sub routines for Windows systems 131

Appendix H. Other common functions and sub routines for Windows systems. 133

ffirstMatch()	133
getValue()	134
removeSpecialCharacters()	135
versionCompare()	135

Appendix I. Common functions for UNIX systems 137

changeMG()	137
AddMG()	138
compare()	138
cutdown()	139
mes4path()	140
mes4Path1()	140

findOSInfo()	141
telnetNFS()	141
NFScheck()	142

Appendix J. Other functions for UNIX systems. 143

formatSizeDisplay()	144
versionCompare()	144
checkHpux()	146
checkLinux()	146
checkSunOS()	146
getValue()	147
setValue()	147
copyValue()	147
getSystemId()	148
getClosestExistingParentDir()	148
parseDirParameter()	149
printDirSize()	149

Appendix K. Logging utility functions for UNIX systems. 151

Notices 153

Support information and feedback 157

Index 159

Figures

- | | | |
|-----|---|----|
| 1. | Output to the command-line interface on Windows systems | 25 |
| 2. | Output to the command-line interface on UNIX systems. | 26 |
| 3. | precheck.log file | 27 |
| 4. | prs.debug file on UNIX systems. | 28 |
| 5. | prs.trc file on UNIX systems | 29 |
| 6. | result.txt file on Windows systems | 30 |
| 7. | result.txt file on UNIX systems | 31 |
| 8. | result.XML file on Windows systems | 32 |
| 9. | Prerequisite Scanner architecture and scanning process | 35 |
| 10. | Running the script and setting the detail parameter on UNIX systems | 64 |
| 11. | Running the script without setting the detail parameter on Windows systems. | 65 |
| 12. | precheck.log file with the debug data | 70 |
| 13. | precheck.log file without debug data | 71 |
| 14. | prs.debug file on UNIX systems. | 72 |
| 15. | prs.trc file on UNIX systems | 73 |

Tables

1. Special characters to represent types of ranges	2	20. DB2 data properties.	92
2. Examples of prerequisite properties	3	21. MS SQL Server data properties	92
3. Basic prerequisite properties categories.	4	22. Internet Explorer data properties	93
4. Predefined subtypes	6	23. Network data properties	93
5. Predefined qualifiers	9	24. Oracle data properties	94
6. Supported data type categories and values	15	25. Operating system data properties	95
7. Scanned sections of a configuration file for Windows	18	26. Installed software data properties.	106
8. Scanned sections of a configuration file for UNIX	19	27. User data properties	106
9. New configuration files	36	28. Windows network data properties	107
10. Defects fixed in this release	37	29. UNIX network data properties.	107
11. Checks and tasks before using a configuration file for Windows systems	41	30. UNIX collectors.	109
12. Checks and tasks before using a configuration file for UNIX systems	42	31. Functions in common_function.vbs	115
13. Special characters legend for the Prerequisite Scanner script.	61	32. Called function for each variable type.	127
14. Execution problems checklist.	73	33. Logging utility sub routines.	129
15. Predefined product codes	77	34. File utility sub routines	131
16. Predefined configuration files	81	35. File utility functions	131
17. Predefined categories for prerequisite properties	85	36. Other common functions and sub routines for Windows systems	133
18. Common data prerequisite properties	86	37. Parent functions calling ffirstMatch()	133
19. Autonomic Deployment Engine data properties	91	38. Scripts using getValue()	134
		39. Parent functions calling versionCompare	135
		40. Functions in common_function.sh.	137
		41. Common functions in multiple files	143
		42. Common functions in TAD722_impl.sh	143
		43. Parent functions calling versionCompare	145
		44. Logging utility functions on UNIX systems	151

Chapter 1. Prerequisite Scanner overview

IBM® Prerequisite Scanner is a scanning tool that performs identification, checking, and verification of prerequisites for specified software before the actual deployment takes place. It scans for hardware and software prerequisites based on the values set for prerequisite properties. The Scanner displays the results of the scan in the command-line interface and also saves the results to text and optionally XML files. It also writes informational, trace, and debugging messages to log files.

Prerequisite Scanner initially checks the operating system of the machine and verifies whether it is the correct version for the specified software. If it is a supported operating system, the Scanner continues the prerequisite scan and generates the results; otherwise, it exits. If any of the individual checks for prerequisites fail, the overall scan fails.

You can run the Prerequisite Scanner after an installation or at any time to confirm your current environment. The Prerequisite Scanner does not require that you run the installation program of the software for which you want to check prerequisites.

You can extend the Prerequisite Scanner to scan for prerequisites that are not part of the core set of prerequisite checks supplied with the Scanner.

Prerequisite Scanner invokes the following types of scripts depending upon your platform:

- Windows: VBScript and batch
- UNIX: shell

Note: You cannot run the UNIX scripts on Windows systems even if you have installed a UNIX-like environment on the Windows machines, for example, Cygwin.

Prerequisite Scanner architecture

IBM Prerequisite Scanner comprises the following main components: a script to run in a command-line interface, a set of properties for the prerequisite checks, prerequisite property configuration files, prerequisite collectors, and prerequisite evaluators. The results of running the Prerequisite Scanner are available in various output formats.

Prerequisite properties

Prerequisite properties are the expected values for different software and hardware prerequisites, required by the products or solutions to be installed. Examples of prerequisite properties include the total disk space available on the machine, the set of ports that are not in use on a machine, and the current set of installed applications.

Because the values for these prerequisite properties can change with different products, the properties and their values are represented as name value pairs, with optional qualifiers. They are contained in the prerequisite properties configuration files. There is only one prerequisite property on each line.

Prerequisite properties adhere to the following format:

```
[prefix_identifier.]property_name[.suffix_identifier]=  
[[qualifier_name:qualifier_value]]property_value
```

where:

- *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
- *property_name* is the name of the prerequisite property.
- *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
- *qualifier_name* is an optional attribute for the prerequisite property. IBM Prerequisite Scanner uses it to qualify the prerequisite property or type of check to perform on the prerequisite property.

Note: You can have multiple qualifiers, each separated by a comma. The set of qualifiers must be enclosed by [] square brackets.

- *qualifier_value* is the value for the optional attribute. Each qualifier and its value must be delimited by a : colon.
- *property_value* is the value for the prerequisite property and it can be a string or integer.

A prerequisite property can have one or many values depending upon the data type and qualifier as follows:

- A single integer, for example, 8080 to represent the value of a port number.
- A range or group of integers represented by using special characters as outlined in Table 1.

Table 1. Special characters to represent types of ranges

Special character	Description
*	Identifies a placeholder for multiple values. For example, ports.* can represent a superset of ports for both a database product, ports.DB and IBM WebSphere® Application Server, ports.WAS.
+	Identifies that the actual version must at least match the value for expected version. For example, os.versionNumber=5.0+, means that the version must be 5.0 or later.
-	Identifies that the actual version must at most match the value for expected version. For example, os.versionNumber=5.0-, means that the version must be 5.0 or earlier.
.*	Identifies that the actual version can match any wildcard value for the expected version. For example, os.versionNumber=5.*, means that the version can be 5.0, 5.0.1 or 5.5.

Restriction: On Windows systems, the * wildcard is only supported if used within a regular expression in the OS Version prerequisite property.

- A string that can represent any of the following values for prerequisite types:
 - A numeric value with a unit, for example, 8GB or 10MB
 - An application, operating system, architecture, or package, for example, IBM Lotus Symphony, RedHat Enterprise Linux 5.4, 32-bit, or ftp

Note: A string might also comprise multiple values separated by a comma, for example, a list of applications.

- Either or values represented by one of the following combinations, such as, True|False, Available|Unavailable, or Enabled|Disabled

Table 2 outlines examples of prerequisite properties.

Table 2. Examples of prerequisite properties

Prerequisite property	Explanation
Disk=1GB	The amount of free disk space, where: <ul style="list-style-type: none"> • <i>property_name</i> is Disk • <i>property_value</i> is 1GB
user.isAdmin=True	Whether the logged on user belongs to an Administrator group, where: <ul style="list-style-type: none"> • <i>prefix_identifier</i> is user, for user prerequisite properties • <i>property_name</i> is isAdmin • <i>property_value</i> is True
network.availablePorts.DB=60000-60005 network.availablePorts.WAS=8080 network.availablePorts.FTP=21	Checks whether ports 60000-60005 are available for the database server, port 8080 is available for the WebSphere Application Server, and port 21 for FTP, where: <ul style="list-style-type: none"> • <i>prefix_identifier</i> is network, for general prerequisite properties • <i>property_name</i> is availablePorts • <i>suffix_identifier</i> are DB for available database ports, WAS for the available WebSphere Application Server port, and FTP for the available FTP port • <i>property_value</i> is 60000-60005, 8080, or 21
os.dir.home=[dir:/home,type:permission]755+	Checks whether the home directory has drwxr-xr-x permissions, where: <ul style="list-style-type: none"> • <i>prefix_identifier</i> is os, for operating system prerequisite properties • <i>property_name</i> is dir • <i>suffix_identifier</i> is home for the directory to check • <i>qualifier_name</i> are dir and type that qualify the prerequisite property and type of check • <i>qualifier_value</i> are home and permission, the values for the qualifiers • <i>property_value</i> is 755+, that is, the octal digit representation of the access permissions for the home directory

You can add or edit predefined prerequisite properties for each product for which you want to run the Prerequisite Scanner. You can also create custom prerequisite properties and use Prerequisite Scanner collectors and evaluators as required to scan for and compare the prerequisite properties.

Related concepts:

“Predefined qualifiers for prerequisite properties” on page 8
IBM Prerequisite Scanner provides a set of basic qualifiers for some prerequisite properties in a predefined category. Qualifiers represent attributes of the prerequisite property that Prerequisite Scanner uses to qualify the prerequisite property or type of check to perform on that prerequisite property.

Predefined categories of prerequisite properties

IBM Prerequisite Scanner provides a set of basic prerequisite properties for different categories of data: common, installed software, operating system, user, connectivity, Internet Explorer, database server, environment variables, and network including platform-specific properties for Windows and UNIX.

`<prefix_identifier>` is an identifier for a predefined category of prerequisite properties.

Table 3 outlines the predefined categories of hardware and software prerequisites.

Table 3. Basic prerequisite properties categories

Data category	Description	Required prefix identifier
Common	This category checks common prerequisites such as processor speed, RAM, disk, and temporary space. This example is the prerequisite property for checking the operating system: OS Version=RedHat Enterprise Linux 5.4	None
Installed software	This category checks installed software prerequisites such as the programs registered in the Windows registry and whether cygwin and gskit are installed. This example is the prerequisite property for scanning the operating system registry for installed programs with locations: installedSoftware=list_of_installed_programs	None
User	This category checks user prerequisites such as whether the logged on user had administrative rights or is the root user. This example is the prerequisite property for checking whether the logged on user is a member of the Administrator group: user.isAdmin=True	user
Operating system	This category checks operating system prerequisites such as version, architecture, total memory, available memory, and total physical memory. This example is the prerequisite property for checking whether the remote registry service is running: os.isServiceRunning.remoteRegistry=True	os
Connectivity	This category checks connectivity prerequisites such as whether Telnet is running and to which IP addresses and ports the Scanner can connect.	None
Network	This category checks network prerequisites that can be common across all platforms such as whether there are ports available. This example is the prerequisite property for checking whether the 8080 port is available for IBM WebSphere Application Server: network.availablePorts.was=8080	network
Windows network	This category checks Windows network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties. This example is the prerequisite property for checking whether at least one adapter with a valid IP address has NetBIOS enabled as a protocol: network.netBIOSEnabled=True	network

Table 3. Basic prerequisite properties categories (continued)

Data category	Description	Required prefix identifier
UNIX network	This category checks UNIX network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties. This example is the prerequisite property for checking whether the local host responds to the ping protocol: network.pingLocalhost=True	network
Internet Explorer	This category checks Microsoft Internet Explorer prerequisites such as the version. This example is the prerequisite property for checking whether the Internet Explorer version is 7.0: internetExplorer.version=7.0	internetExplorer
Database server, DB2®	This category checks DB2 prerequisites such as the version. This example is the prerequisite property for whether the DB2 version is at least 9.5: DB2 Version=9.5.*	DB2
Database server, Oracle	This category checks Oracle prerequisites such as the version. This example is the prerequisite property for checking whether Oracle client version is at least 9.2.0.8: oracle.Client=9.2.0.8+	Oracle
Environment variables	This category checks environment variable prerequisites such as whether the environment variable has been set. This example is the prerequisite property for checking whether the class path contains the Derby JAR file: env.classpath.derbyJAR=False	env
Autonomic Deployment Engine	This category checks Autonomic Deployment Engine prerequisites such as whether the Autonomic Deployment Engine is installed or the installation unit for Tivoli® Integrated Portal. This example is the prerequisite property for checking whether the installation unit for Tivoli Integrated Portal Version 2.1.1.0 or 2.1.1.1 is installed on a Windows system: de.installationUnit=regex{.*C37109911C8A11D98E1700061BDE7AEA.* .*TIP 2.1.1.0.* .*TIP 2.1.1.1.*}	de
Database server, MS SQL	This category checks MS SQL prerequisites such as the version. This example is the prerequisite property for checking whether MS SQL Server version is SQL Server 2008 R2 Developer Edition: mssql.Server=10.50.1600.1	mssql

Predefined subtypes for prerequisite properties

IBM Prerequisite Scanner provides a set of basic subtypes for some prerequisite properties in a predefined category. Subtypes further categorize a prerequisite property such as categorization by application, utility, or service subtype.

For example, you can have a prerequisite property for available network ports. You can further categorize that prerequisite property to check available ports for a database server, application server, or protocol.

<suffix_identifier> is an optional identifier for a subtype in the prerequisite property name.

Table 4 outlines the predefined subtypes for different categories of prerequisite properties including the <suffix_identifier>.

Table 4. Predefined subtypes

Prerequisite property subtype	Suffix identifier	Platform	Description	Valid values for the subtype
Platform independent network category				
network.availablePorts. <i>app_type</i>	<i>app_type</i>	All	Use this naming convention for checking whether the port or range of ports is not being listened to or is available for the <i>app_type</i> application type.	String to represent <i>app_type</i> , for example: <ul style="list-style-type: none"> • DB2 checks ports for DB2 database server • WAS checks ports for WebSphere Application Server • ftp checks the FTP port
network.portsInUse. <i>app_type</i>	<i>app_type</i>	All	Use this naming convention for checking whether the port or range of ports is being listened to or is in use for the <i>app_type</i> application type.	String to represent <i>app_type</i> , for example: <ul style="list-style-type: none"> • DB2 checks ports for DB2 database server • WAS checks ports for WebSphere Application Server • ftp checks the FTP port
Operating system category				
os.dir. <i>dir_name</i>	<i>dir_name</i>	UNIX	Use this naming convention for checking the <i>dir_name</i> file system. The value for the prerequisite property uses predefined qualifiers.	String to represent <i>dir_name</i> , for example: <ul style="list-style-type: none"> • tmp • home
os.file. <i>script_name</i>	<i>script_name</i>	UNIX	Use this naming convention for checking whether the <i>script_name</i> script is available on the machine.	String to represent <i>script_name</i> , for example: <ul style="list-style-type: none"> • bash • expect • gzip • tar
os.file. isServiceRunning. <i>service_name</i>	<i>service_name</i>	Windows	Use this naming convention for checking whether the <i>service_name</i> service is running on the machine.	String to represent <i>service_name</i> , for example: <ul style="list-style-type: none"> • remoteRegistry • DNSClient • terminalServices

Table 4. Predefined subtypes (continued)

Prerequisite property subtype	Suffix identifier	Platform	Description	Valid values for the subtype
os.lib. <i>lib_name_version</i>	<i>lib_name</i> <i>_version</i>	UNIX	Use this naming convention for checking that the supported version of the <i>lib_name_version</i> library is installed on the machine.	<p>String to represent <i>lib_name_version</i>, for example, in bold:</p> <ul style="list-style-type: none"> • 32-bit libstdc++.so.# library • 64-bit libstdc++.so.# library • 32-bit libXft.so.# library • 32-bit libXtst.so.# library • 64-bit libaio.so.# library • 32-bit xlC.rte XLC runtime level • 32-bit xlC.aix50.rte XLC run time for AIX Version 5.3 • 32-bit xlC.aix61.rte XLC run time for AIX Version 6.1 • AIX® IOCP bos.iocp.rte library • bos.loc.iso.en_us, the ISO code fileset for the AIX base operating system <p>regex <i>{str}</i>, a regular expression with the input parameter <i>str</i>, representing the search pattern for the library name, for example: regex <i>{.*libgcc.*}</i></p> <p>Checks whether a version of the GCC low-level runtime library, <i>libgcc</i>, for that operating system exists.</p>

Table 4. Predefined subtypes (continued)

Prerequisite property subtype	Suffix identifier	Platform	Description	Valid values for the subtype
os.package. <i>package_name</i>	<i>package_name</i>	UNIX	Use this naming convention for checking that the supported version of the <i>package_name</i> package is installed on the machine.	String to represent <i>package_name</i> , for example, in bold: <ul style="list-style-type: none"> • bash shell • expect for the TCL extension package • libgcc for GCC low-level runtime package • openssh for the Open Source secure shell • openssl for the Open Source toolkit for SSL/TLS • perl for the Perl scripting package • rpm for the RPM or RPM Build packages • telnet for the Telnet package • wget for the GNU file retrieval package
os.space. <i>dir_name</i>	<i>dir_name</i>	UNIX	Use this naming convention for checking the available disk space for the specified <i>dir_name</i> file system. The value for the prerequisite property uses predefined qualifiers.	String to represent <i>dir_name</i> , for example: <ul style="list-style-type: none"> • usr • home • tmp • var

Predefined qualifiers for prerequisite properties

IBM Prerequisite Scanner provides a set of basic qualifiers for some prerequisite properties in a predefined category. Qualifiers represent attributes of the prerequisite property that Prerequisite Scanner uses to qualify the prerequisite property or type of check to perform on that prerequisite property.

For example, you can have a prerequisite property for a file system. You can qualify which check to perform for that prerequisite property based on its file system name and access permissions attributes. You can also qualify which type of units to use when checking the available disk space based on the file system path and unit attributes.

Qualifiers support customization to meet the needs of your environment and prevent the Scanner having to make implicit assumptions about the attributes of multidimensional prerequisites such as the default path and access permissions. You can change the values for the predefined qualifiers, but you cannot add new qualifiers to the existing set of predefined qualifiers for a predefined prerequisite property.

Qualifiers must adhere to the following format:

```
[qualifier_name:qualifier_value, qualifier_name:qualifier_value]
property_value
```

where:

- *qualifier_name* is an optional attribute for the prerequisite property that IBM Prerequisite Scanner uses to qualify the prerequisite property or type of check to perform on the prerequisite property.
- *qualifier_value* is the value for the optional attribute.
The value for the qualifier can also be a name value pair to support multiple valid values depending upon user type. For example, different paths for the home directory depending up whether it is a root or non-root user.
- *property_value* is the value for the prerequisite property and it can be a string or integer.

Each qualifier and its value must be delimited by a : colon. You can have multiple qualifiers, each separated by a comma. The set of qualifiers must be enclosed by [] square brackets.

Table 5 outlines the predefined qualifiers for different categories of prerequisite properties. Some prerequisite properties also use predefined subtypes to further categorize a prerequisite property.

Important: You cannot use the predefined qualifiers with other predefined prerequisite properties.

Table 5. Predefined qualifiers

Prerequisite property	Platform	Description	Valid qualifiers and values
Operating system category with predefined subtype			
os.dir.dir_name	UNIX	<p>Checks the <i>dir_name</i> file system based on the following qualification attributes:</p> <ul style="list-style-type: none">• <i>dir</i> attribute, to determine which file system to check• <i>type</i> attribute, to determine which attribute of the file system to check, for example, the <octal_digits> octal digit representation for the access permissions to that file system <p><<i>dir_name</i>> can represent for example:</p> <ul style="list-style-type: none">• tmp• home	<p>String with the following qualifier format:</p> <p>[<i>dir:dir_name</i>, type:permission] <i>octal_digits</i>+</p> <p>For example, to check whether the home directory has drwxr-xr-x permissions:</p> <p>os.dir.home=[<i>dir:/home</i>, type:permission]755+</p>

Table 5. Predefined qualifiers (continued)

Prerequisite property	Platform	Description	Valid qualifiers and values
os.space. <i>dir_name</i>	UNIX	<p>Checks the available disk space for the specified <i>dir_name</i> file system based on one or more of the following qualification attributes:</p> <ul style="list-style-type: none"> • <i>dir</i> attribute, to determine which path to the file system to check • <i>unit</i> attribute, to determine which units for disk space to use <p>The value for <i>dir</i> attribute is dependant on the logged on user; thus, the value is a name value pair to represent the user type, that is, root or non-root, and the associated path.</p> <p><i>dir_name</i> can represent for example:</p> <ul style="list-style-type: none"> • <i>usr</i> • <i>home</i> • <i>tmp</i> • <i>var</i> 	<p>String with the following qualifier format for the file system of a root user:</p> <pre>[dir:root=<i>dir_path</i>, unit:<i>unit_name</i>] disk_space</pre> <p>For example:</p> <pre>os.space.usr= [dir:root=/usr/ibm/common/acsi, unit:GB]200</pre> <p>String with the following qualifier format for the file system of a non-root user:</p> <pre>[dir:non_root=<i>dir_path</i>, unit:<i>unit_name</i>] disk_space</pre> <p>For example:</p> <pre>os.space.home= [dir:non_root=USERHOME/.acsi_HOST, unit:MB]200</pre> <p>String with the following qualifier format, using only one qualifier:</p> <pre>[dir:<i>dir_path</i>] disk_space MB</pre> <p>For example:</p> <pre>os.space.home=[dir:/home/sat]250MB</pre>
Operating system category without predefined subtype			
os.mountcheck	UNIX	<p>Checks whether the file system is mounted based on the following qualification attributes:</p> <ul style="list-style-type: none"> • <i>drive</i> attribute, to determine which directory is the mounted file system • <i>nosuid</i> attribute, to determine whether the mount option is set if the file system is mounted 	<p>String with the following qualifier format:</p> <pre>[drive:<i>dir_name</i>, mount_option: false true] True False</pre> <p>For example, to check whether /home directory is mounted and the nosuid option is not set:</p> <pre>os.mountcheck=[drive:/home, nosuid:false]True</pre>

Table 5. Predefined qualifiers (continued)

Prerequisite property	Platform	Description	Valid qualifiers and values
os.SELinux	Linux	<p>Checks the enforcement status of the Security-Enhancement Linux feature based on the following qualification attributes:</p> <ul style="list-style-type: none"> source attribute, to determine the command to use for the relevant operating system 	<ul style="list-style-type: none"> String with the following qualifier format: [source:Command] Disabled Enabled For example, to check whether the feature is disabled or has a permissive status on either Red Hat or SUSE operating system: os.SELinux=[source:Command]Disabled String without a qualifier, where the operating system is a generic Linux variant: os.SELinux=Disabled
os.ulimit	UNIX	<p>Use this naming convention for checking whether an unlimited number of processes can be run based on the following qualification attributes:</p> <ul style="list-style-type: none"> type attribute, to determine which additional limit to check, for example, the filedescrptorlimit checks the limit for the number of file descriptors that processes can open 	<p>String with the following qualifier format: [type:limit_name]limit_value, limited unlimited</p> <p>For example, to check whether the file descriptor limit is greater than 8192, with unlimited number of processes: os.ulimit=[type:filedescriptorlimit]8192+,unlimited</p> <p>Valid types of limits to check, where <i>limit_name</i> represents the type of limit are as follows:</p> <ul style="list-style-type: none"> ALL, checks all limits corefilesizelimit datasegmentlimit filedescriptorlimit filesizelimit hardlimit processlimit maxmemorysizelimit maxprocesseslimit stacksizelimit threadlimit
Common category without predefined subtype			

Table 5. Predefined qualifiers (continued)

Prerequisite property	Platform	Description	Valid qualifiers and values
Disk	Windows	<p>The amount of free disk space, with the following optional qualification attributes:</p> <ul style="list-style-type: none"> • dir attribute, to determine which path to the directory to check • unit attribute, to determine which units for disk space to use 	<p>String with the following qualifier format:</p> <pre>[dir:dir_path, unit:unit_name] disk_space</pre> <p>For example:</p> <pre>Disk= [dir:C:\Program Files\IBM\SQLLIB, unit:MB] 1431</pre> <p>Numeric format in MBs or GBs:</p> <pre><disk_space>MB GB</pre> <p>For example:</p> <pre>Disk=250MB</pre>

Product codes

IBM Prerequisite Scanner uses multicharacter codes in file names and parameter names to identify products and components and determine which type of configuration file to use.

product_code

It is the variable to represent a product code on either Windows or UNIX systems. Product codes identify the product, an individual platform such as Windows, AIX, HP-UX, Linux, and Solaris, and optionally the version of the operating system that is supported by that product. They are stored in the codename.cfg file. Any product that supports multiple platforms has multiple product codes, with each one identifying a product, platform, and version of the operating system as required.

For example, the COD, COK and COX product codes identify some of the supported operating systems and versions for IBM Tivoli Provisioning Manager:

```
COD=Tivoli Provisioning Manager for AIX 6.1
COK=Tivoli Provisioning Manager for HP-UX
COX=Tivoli Provisioning Manager for Windows 2008
```

When you run the Prerequisite Scanner, you pass the product code and optionally the product version as input parameters. The Scanner automatically detects the operating system of the machine. It then checks the operating system against the operating system listed for that product code in the codename.cfg file.

If the operating system of the machine is supported, the Scanner then uses the input parameters to find the configuration file in the *ips_root/Windows|UNIX_Linux* directory. The file name contains the same product code and product version as the input parameters. If you do not pass the optional product version parameter, the Scanner uses the latest version of the configuration file that it finds in this directory. Prerequisite Scanner then begins the scan.

Important: If the platform is not supported, Prerequisite Scanner exits and displays a message to state that the platform is not supported.

Prerequisite Scanner configuration files

The IBM Prerequisite Scanner configuration files for individual platforms contain the prerequisite properties and their expected values for each platform that is supported by the product. Prerequisite Scanner provides a predefined set of configuration files that you can edit. You must create configuration files for new products and platforms to be supported.

When the operating system of the machine is supported, Prerequisite Scanner checks the prerequisite properties in the configuration files. It can also check in only the sections associated with that operating system.

Configuration files have a .cfg file extension. You store them in the *ips_root*/*<OS>* directory, where *<OS>* is the name of the operating system type, for example, Windows or UNIX_Linux.

Configuration files must adhere to the following rules:

- File extension must be .cfg
- Naming convention for the file name:

product_code[_*<version>*].cfg

where:

– *product_code*

It is the variable to represent a product code on either Windows or UNIX systems. Product codes identify the product, an individual platform such as Windows, AIX, HP-UX, Linux, and Solaris, and optionally the version of the operating system that is supported by that product. They are stored in the *codename.cfg* file. Any product that supports multiple platforms has multiple product codes, with each one identifying a product, platform, and version of the operating system as required.

– *<version>* is the 8-digit code to represent the version, release, modification, and level, with two digits for each part of the code; for example, 7.3.21 is 07032100.

- Group prerequisite properties under sections that must follow a naming convention for the section titles.
- Standard format for each prerequisite property is a name value pair with optional qualifiers, and only one property on each line:

```
[<prefix_identifier>].<property_name>[.<suffix_identifier>]=  
[[<qualifier_name>:<qualifier_value>]]<property_value>
```

Example of a configuration file without sections

This example checks for prerequisite properties but does not differentiate between different prerequisite properties for the required operating system versions.

```
os.space.var=[dir:root=/var/ibm/common/acs,unit:MB]1.0  
os.space usr=[dir:root=/usr/ibm/common/acs,unit:MB]200  
os.space.home=[dir:non_root=USERHOME/.acsi_HOST,unit:MB]200  
os.space.tmp=30MB  
env.classpath.derbyJAR=False  
network.pingSelf=True  
network.pingLocalhost=True  
network.availablePorts.Derby=4130  
OS Version=RedHat Enterprise Linux 4.*,RedHat Enterprise Linux 5.*  
os.package.compat-libstdc++-33=compat_libstdc++_33  
os.package.libgcc=libgcc-3.4.3-9
```

Related concepts:

“Sections in configuration files”

Prerequisite properties can be grouped under a set of sections in configuration files, with each section representing a data type category. Sections are optional in configuration files.

Sections in configuration files

Prerequisite properties can be grouped under a set of sections in configuration files, with each section representing a data type category. Sections are optional in configuration files.

The naming convention for the section title is:

`[category_name:category_value]`

where:

- *category_name* is the multicharacter code that represents the data type category
- *category_value* is the multicharacter code that represents an allowed value for the category

Note: The values can use the special characters as outlined in Table 1 on page 2.

Each category name and its value must be delimited by a : colon and enclosed by [] square brackets.

You can have multiple data type categories by combining section titles, thereby restricting prerequisite properties to that set of specified categories only.

`[category_name:category_value][category_name:category_value]`

For example, to specify prerequisite properties that apply to a machine running 32-bit, SUSE Linux Enterprise Server version 11, Itanium operating system:

`[OSType:SUSELinuxEnterpriseServer11][OSArch:64-bit][CPU:Itanium]`

For all platforms, you can use the | logical OR symbol to use either or data type categories. For example, to have any of the environment variables set to True, the combination of section titles is:

- **UNIX systems**

`[@TPAE_DB_FEATURE:True|@TPAE_DIR_FEATURE:True|@TPAE_J2EE_FEATURE:True]`

- **Windows systems**

`[@TPAE_DB_FEATURE:True] | [@TPAE_DIR_FEATURE:True] | [@TPAE_J2EE_FEATURE:True]`

Important: The position of the | logical OR symbol is different between Windows and UNIX systems. For UNIX systems, the set of section titles are enclosed by one set of [] square brackets only with each section title separated by the symbol. For Windows systems, the symbol delimits each complete section title with associated [] square brackets.

For Windows systems only, you can use the ! logical NOT symbol to exclude a data type category. For example, to exclude Windows Server 2003 R2 variant, the combination of section titles is:

`[OSType:Windows Server 2003][!OSType:Windows Server 2003 R2]`

Table 6 on page 15 outlines the supported data type categories and associated allowed values.

Table 6. Supported data type categories and values

Data type category	Description	Allowed values
OSType	The operating system type	<ul style="list-style-type: none"> • UNIX Indicates that all properties in this category are common to all UNIX platforms, including, AIX, HP-UX, Linux, and Solaris, for example: [OSType:UNIX] • AIX Indicates that all properties in this category are common to all AIX operating system variants, for example: [OSType:AIX] • HP-UX Indicates that all properties in this category are common to all HP-UX operating system variants, for example: [OSType:HP-UX] • LINUX Indicates that all properties in this category are common to all Linux operating system variants, for example: [OSType:LINUX] • RedHat Indicates that all properties in this category are common to all RedHat Linux operating system variants, for example: [OSType:RedHat] • RedHatEnterpriseLinuxServer Indicates that all properties in this category are common to all RedHat Enterprise Linux Server operating system variants, for example: [OSType:RedHatEnterpriseLinuxServer] • SUSE Indicates that all properties in this category are common to all SUSE Linux operating system variants, for example: [OSType:SUSE] • SUSELinuxEnterpriseServer Indicates that all properties in this category are common to all SUSE Linux Enterprise Server operating system variants, for example: [OSType:SUSELinuxEnterpriseServer] • Solaris Indicates that all properties in this category are common to all Solaris operating system variants, for example: [OSType:Solaris]

Table 6. Supported data type categories and values (continued)

Data type category	Description	Allowed values
		<ul style="list-style-type: none"> Windows Indicates that all properties in this category are common to all Windows operating systems, for example: [OSType:Windows] Windows 2000 Workstation (Version 5.0.*) Indicates that all properties in this category are common to all Windows 2000 operating system variants, for example: [OSType:Windows 2000] Windows XP Workstation (Version 5.1.*) Indicates that all properties in this category are common to all Windows XP Professional 32-bit operating system variants, for example: [OSType:Windows XP] Windows XP Workstation (Version 5.2.*) Indicates that all properties in this category are common to all Windows XP Professional 64-bit operating system variants, for example: [OSType:Windows XP] Windows Vista Workstation (Version 6.0.*) Indicates that all properties in this category are common to all Windows Vista operating system variants, for example: [OSType:Windows Vista] Windows 7 Workstation (Version 6.1.*) Indicates that all properties in this category are common to all Windows 7 operating system variants, for example: [OSType:Windows 7] Windows 2000 Server (Version 5.0.*) Indicates that all properties in this category are common to all Windows 2000 Server operating system variants, for example: [OSType:Windows 2000] Windows Server 2003 (Version 5.2.*) Indicates that all properties in this category are common to Windows Server 2003 operating system variants, for example: [OSType:Windows Server 2003] Windows Server 2003 R2 (Version 5.2.* and other type of OS description is R2) Indicates that all properties in this category are common to the Windows Server 2003 R2 operating system variant only, for example: [OSType:Windows Server 2003 R2]

Table 6. Supported data type categories and values (continued)

Data type category	Description	Allowed values
		<ul style="list-style-type: none"> Windows Server 2008 (Version 6.0.*) Indicates that all properties in this category are common to Windows Server 2008 operating system variants, for example: [OSType:Windows Server 2008] Windows Server 2008 R2 (Version 6.1.*) Indicates that all properties in this category are common to the Windows Server 2008 R2 operating system variant only, for example: [OSType:Windows Server 2008 R2] <OS_Name_Version> Indicates that all properties in this category are common to that version of the operating system, for example: [OSType:RedHatEnterpriseLinuxServer4.2] <p>Note: The special wildcard character, *, is allowed to specify multiple versions.</p>
OSArch	The architecture for the operating system	<ul style="list-style-type: none"> 32-bit, for example: [OSArch:32-bit] 64-bit, for example: [OSArch:64-bit]
CPU	The generic processor family name	Itanium, for example: [CPU:Itanium]
CPUArch	The architecture for the processor	Architecture for 64-bit PowerPC® and Power Architecture processors, that is: <ul style="list-style-type: none"> ppc4 POWER4 POWER5 POWER6 POWER7 For example: [CPUArch:ppc4]
@<EnvVar_Name>	The environment variable for a product	Adheres to the rules of that product, for example: [@TPAE_DB_SERVER:True]

Example of a configuration file for Windows that uses sections

This example uses sections to categorize prerequisite properties for any Windows machine and then machines running specific versions of Windows.

```
#Properties for all Windows operating systems, that is, Windows XP and above
[OSType:Windows]
os.versionNumber=5.1+
network.pingSelf=True
network.pingLocalhost=True
network.availablePorts.Derby=4130
env.CIT.homeExists=True
env.classpath.derbyJAR=False
# Disk space properties
commonPath=10MB
installPath=200MB
```

```
tempPath=30MB
```

```
[OSType:Windows Vista]  
os.servicePack=2+
```

When you run Prerequisite Scanner, it scans and checks for different prerequisite properties depending upon the operating system and version that is installed on the machine.

For example, Table 7 outlines the different sections containing the prerequisite properties that are checked based on the example.

Table 7. Scanned sections of a configuration file for Windows

Platform or operating system	Sections with prerequisite properties
Machine with Windows XP and above	[OSType:Windows]
Machine with Windows Vista only	[OSType:Windows] [OSType:Windows Vista]

Example of a configuration file for UNIX that uses sections

This example contains prerequisite properties for all platforms, individual platforms, and versions of operating systems for a specific product.

```
# Properties common to all UNIX platforms  
[OSType:UNIX]  
os.space.var=[dir:root=/var/ibm/common/acsi,unit:MB]1.0  
os.space.usr=[dir:root=/usr/ibm/common/acsi,unit:MB]200  
os.space.home=[dir:non_root=USERHOME/.acsi_HOST,unit:MB]200  
os.space.tmp=30MB  
env.classpath.derbyJAR=False  
network.pingSelf=True  
  
# Properties common to all Linux platforms  
[OSType:Linux]  
os.shell.default=bash  
os.SELinux=[source:Command]Disabled  
os.package.rpm=rpm  
  
# Properties common to Linux platforms with the ppc64 CPU architecture  
[OSType:Linux][CPUArch:ppc64]  
os.package.vacpp.rte=vacpp.rte-9.0.0-5+  
  
# Properties common to all RedHat OS  
[OSType:RedHat]  
env.classpath.derbyJAR=False  
  
# Properties common to all versions of Red Hat Enterprise  
# Linux Server OS  
[OSType: RedHatEnterpriseLinuxServer]  
network.pingLocalhost=True  
  
# Properties common to all Red Hat Enterprise Linux Server  
# OS Version 6.x(6.1,6.2...) OS  
[OSType: RedHatEnterpriseLinuxServer6.*]  
os.package.compat-libstdc++-33=compat_libstdc++_33-3.2.3-68  
  
[OSType:RedHatEnterpriseLinuxServer5.*]  
os.package.compat-libstdc++-33=compat_libstdc++_33  
  
# Properties common to all Red Hat Enterprise Linux Server  
# Version 4.x(6.1,6.2...) OS and for Itanium family CPU  
[OSType:RedHatEnterpriseLinuxServer4.*][CPU:Itanium]  
os.package.ia32el=ia32el-1.1-20
```

```

# Properties common to all Red Hat Enterprise Linux Server
# Version 4.x(6.1,6.2...) OS and for a 64-bit OS architecture
[OSType:RedHatEnterpriseLinuxServer4.*] [OSArch:64-bit]
os.package.libgcc=libgcc-3.4.3-9

# Properties specific to RedHatEnterpriseLinuxServer5.2 OS
[OSType:RedHatEnterpriseLinuxServer5.2]
network.availablePorts.Derby=4130

# Properties specific to a 64 bit SUSE Linux Enterprise Server 11 OS
[OSType:SUSELinuxEnterpriseServer11] [OSArch:64-bit]
os.package.libstdc++33-32bit=libstdc++33_32bit-3.3.3-11.9

# Properties specific to a 64 bit SUSE Linux Enterprise Server 11 OS
# and if the environment variable TPAE_DB_Server is set to 'True'
[OSType:SUSELinuxEnterpriseServer11] [@TPAE_DB_Server:True]
os.package.libstdc++31-32bit=libstdc++31_32bit

# Properties specific to a 64 bit SUSE Linux Enterprise Server 11 OS
# and if the environment variables TPAE_DB_Server and TPAE_DIR_Server
# are set to 'True'
[OSType:SUSELinuxEnterpriseServer11] [@TPAE_DB_Server:True]
[@TPAE_DIR_Server:True]
os.package.libstdc++34-32bit=libstdc++34_32bit

# Properties common to all AIX platforms
os.ulimit=[type:filesize:limit]unlimited
os.ulimit=[type:filedescriptor:limit]8192+,unlimited
os.FreePagingSpace=4GB+

# Properties specific to AIX 5.3.0.0 and
# if the environment variables TPAE_DB_FEATURE or TPAE_DIR_FEATURE
# are set to 'True'
[OSType:AIX5.3.0.0] [@TPAE_DB_FEATURE:True] [@TPAE_DIR_FEATURE:True]
os.lib.xlC.aix50.rte=xlC.aix50.rte.9.0.0.8+

```

When you run Prerequisite Scanner, it scans and checks for different prerequisite properties depending upon the operating system and version that is installed on the machine.

For example, Table 7 on page 18 outlines the different sections containing the prerequisite properties that are checked based on the example.

Table 8. Scanned sections of a configuration file for UNIX

Operating systems and versions	Sections with prerequisite properties
Machine with 64-bit SUSE Linux Enterprise Server 11	[OSType:UNIX] [OSType:LINUX] [OSType:LINUX] [CPUArch:ppc64] [OSType:SUSE Linux Enterprise Server 11] [OSArch:64-bit]
Machine with Red Hat Enterprise Linux Server 6.3	[OSType:UNIX] [OSType:LINUX] [OSType:RedHat] [OSType:RedHatEnterpriseLinuxServer] [OSType:RedHatEnterpriseLinuxServer6.*]
Machine with SUSE Linux Enterprise Server 11 and the environment variable @TPAE_DB_Server set to true	[OSType:UNIX] [OSType:LINUX] [OSType:SUSELinuxEnterpriseServer11] [@TPAE_DB_Server:True]
Machine with AIX 5.3.0.0 and the environment variables @TPAE_DB_FEATURE or @TPAE_DIR_FEATURE set to True	[OSType:UNIX] [OSType:AIX] [OSType:AIX5.3.0.0] [@TPAE_DB_FEATURE:True] [@TPAE_DIR_FEATURE:True]

Prerequisite Scanner collectors

IBM Prerequisite Scanner collectors collect actual data about the current environment based on the prerequisite properties set for the products to be installed. The collectors obtain the data through native code. Data can be common data, such as processor speed and RAM, installed software data, operating system data, user data, network, and connectivity data. Collectors are also extensible, so you can create custom collectors to obtain actual values for custom prerequisite properties.

Prerequisite Scanner uses collectors in the following languages depending upon your platform:

- Windows: VBScript with .vbs extension
- UNIX: Shell with .sh or no extension

Note: You cannot run the UNIX scripts on Windows systems even if you have installed UNIX-like environments on the Windows machines, for example, Cygwin.

Collectors for Windows systems

VBScript collectors for Windows systems are run in the Windows Script Host environment. They use the Component Object Model to access elements of the Windows environment, for example, FileSystemObject and TextStream.

Prerequisite Scanner runs the VBScript collectors to obtain the actual values for prerequisite properties for the Windows environment. Each collector can obtain data for one or many prerequisite properties.

For each prerequisite property in a VBScript collector, the collector writes the name of the prerequisite property and its actual value as standard output. Prerequisite Scanner writes this standard output to a temporary text file, that is, `localhost_hw.txt`.

You can create custom common VBScript collectors to collect data for prerequisite properties that apply to any product and product version. You can also create custom product-specific ones to collect data that apply to a specific product and product version.

When you run the Prerequisite Scanner, it runs the collectors in the following order: predefined VBScript collectors; the custom common VBScript collectors in the *ips_root/lib* directory; and the custom product-specific VBScript collectors by searching for the *product_code[_<version>].vbs* file in the *ips_root/Windows* directory.

For example, the *env.tcrhome.vbs* file is a custom collector that checks the home directory environment variable for Tivoli Common Reporting. It is stored in the *ips_root/lib* directory.

VBScript collectors must adhere to the following rules:

- Naming convention for the custom common VBScript collector file
It contains a prerequisite property to be made available to any product and product version, that is, all configuration files:
prefix_identifier.]property_name.vbs

where:

- *prefix_identifier* is the prefix identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories, for example, *env*.

- *property_name* is the prerequisite property name, for example, *tcrhome*.

Store this type of VBScript collector in the *ips_root/lib* directory.

- Naming convention for the custom product-specific VBScript collector file

It contains properties to be made available to a specific product and product versions, that is, one configuration file:

product_code[_<version>].vbs

where:

- *product_code*

It is the variable to represent a product code on either Windows or UNIX systems. Product codes identify the product, an individual platform such as Windows, AIX, HP-UX, Linux, and Solaris, and optionally the version of the operating system that is supported by that product. They are stored in the *codename.cfg* file. Any product that supports multiple platforms has multiple product codes, with each one identifying a product, platform, and version of the operating system as required.

- *<version>* is the 8-digit code to represent the version, release, modification, and level, with two digits for each part of the code; for example, 7.3.21 is 07032100.

Store this type of VBScript collector in the *ips_root/Windows* directory.

- Standard output for each prerequisite property is as follows:

```
WScript.Echo "property_name=" & <var_for_value>
```

- *property_name* that represents the prerequisite property as written in the configuration file, for example, *env.tcrhome*.

- *var_for_value*, that is, the VBScript variable for the actual value that the collector obtains for the prerequisite property.

For example, the following standard output writes the prerequisite property for the Tivoli Common Reporting home environment variable and its actual value:

```
WScript.Echo "env.tcrhome=" & tcr_home
```

Collectors for UNIX systems

Collectors for UNIX systems are run in the relevant Shell host environment for either AIX, HP-UX, Linux, or Solaris. They use the commands and options specific to that platform to access elements of the host environment.

Each UNIX collector obtains data for a prerequisite property or a prerequisite property with predefined subtypes. The collector writes the result of the check for the prerequisite property as standard output. Prerequisite Scanner writes this standard output to a temporary text file.

You can create custom UNIX collectors to collect data for custom prerequisite properties. Each collector, predefined or custom, is called in the *ips_root/UNIX_Linux/packageTest.sh* file.

When you run the Prerequisite Scanner, it runs the collectors in the following order: predefined collectors with *_plug* in the file name in the *ips_root/lib* directory; predefined collectors in the *ips_root/UNIX_Linux* directory; and the custom UNIX collectors in the *ips_root/UNIX_Linux* directory.

For example, the `installedSoftware.TCR.version` file is a custom collector that obtains the version of Tivoli Common Reporting that is installed on the machine. It is stored in the `ips_root/UNIX_Linux` directory.

UNIX collectors must adhere to the following rules:

- Naming convention for the custom UNIX collector file with no file extension:

`[prefix_identifier.]property_name`

where:

- `prefix_identifier` is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories, for example, `installedSoftware`.
- `property_name` is the name of the prerequisite property, for example, `TCR.version`.

Store the collector in the `ips_root/UNIX_Linux` directory. Ensure that it does not have a file extension.

- Standard output for a prerequisite property that returns the actual value for the prerequisite property if it is an integer or string; for example, the software version or the amount of available disk space for a mounted file system. Alternatively, it can return "Unavailable" if it does not.

```
echo "True"|"False" 'If the scan checks for the existence of the prerequisite
property
echo $res 'If the scan checks returns the value, for example, product version,
of the prerequisite property
echo "Unavailable" 'If the scan returns no value for the prerequisite property
echo "Available" 'If the scan returns a valid check for the prerequisite property
```

- Code to call and run the collector in the `ips_root/UNIX_Linux/packageTest.sh` script.

```
res=`echo $line | grep installedSoftware.TCR.version`
if [ $res ]; then
ExpValue=`echo $res | cut -d "=" -f2`

echo "\`wrTrace "Starting" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
echo "\`wrTrace "Executing" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
echo "\`wrDebug "Starting" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
echo "\`wrDebug "Expected" "ExpValue" \`" >>/tmp/prs.check

echo "ss=`./installedSoftware.TCR.version`\`" >>/tmp/prs.check
echo "\`wrTrace "Finished" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
echo "echo \"os.userLimits=\$ss\" >>/tmp/prs.check
echo "\`wrDebug "Finished" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
echo "\`wrDebug "OutPutValueIs" \$ss\`" >>/tmp/prs.check
echo "\`wrTrace "Done" "installedSoftware.TCR.version"\`" >>/tmp/prs.check
fi
```

Prerequisite Scanner evaluators

IBM Prerequisite Scanner evaluators are scripts that compare the actual data from the collectors and the expected data for the same properties that are in the configuration files. Evaluations can be: platform-specific; based on simple operators, such as, less than, equal to, or greater than; and based on whether a property is installed, present, or enabled. They can also verify whether ports are in use or available and connectivity status of the machine. You can create or edit evaluators.

Prerequisite Scanner uses evaluators in the following languages depending upon your platform:

- Windows: VBScript with `.vbs` extension

- UNIX: shell with .sh extension

Note: You cannot run the UNIX scripts on Windows systems even if you have installed a UNIX-like environment on the Windows machines, for example, Cygwin.

You store evaluators in *ips_root/OS*, where *OS* is the name of the operating system, for example, Windows or UNIX_Linux.

Evaluator files must adhere to the following rules:

- Naming convention for the file name:

`[prefix_identifier.]property_name[.suffix_identifier]_compare.vbs|sh`

where:

- *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
- *property_name* is the name of the prerequisite property.
- *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
- Optionally pass two input parameters to the script for complex evaluations:
 - *expected_value*, that is, the expected value for the prerequisite property that is set in the configuration file.
 - *actual_value*, that is, the actual value that the collector discovers for the prerequisite property on the machine.
- Standard output is as follows:
 - "PASS" when the expected value for the prerequisite property is equal to or greater than the actual value for the prerequisite property.
 - "FAIL" when the expected value for the prerequisite property does not equal the actual value for the prerequisite property.

Output formats

IBM Prerequisite Scanner produces output for the following screen and human-readable file formats: output to the command-line interface, debugging and trace log files, and text and XML files for the results.

Prerequisite Scanner saves the scan results and log files to the *ips_output_dir* directory. You can set this directory by using the **outputDir** input parameter when you run the Scanner. If you do not set this parameter, the default output location is *ips_root*.

Note: Prerequisite Scanner creates temporary files during its execution, but these files are deleted before the Scanner completes its execution. These temporary files are located in the *ips_output_dir/temp* subdirectory. The Scanner also deletes the *ips_output_dir/temp* subdirectory, unless the subdirectory contains the debug and trace files that are generated on UNIX systems only.

You can also use the parameter to specify a location, if you choose to run Prerequisite Scanner from a CD, DVD, or a read-only network drive.

Important: If the output directory does not exist, Prerequisite Scanner creates the directory. You must have write permissions to create or write to the output directory in which Prerequisite Scanner saves the files.

Command-line interface output

When you run the Prerequisite Scanner script and set the optional **detail** parameter, Prerequisite Scanner displays detailed results of the scan in the command-line interface. The detailed results contain:

- The version of Prerequisite Scanner
- The version of the operating system on which the Scanner was run
- The name of the products or components for which the prerequisites checks were run
- For each prerequisite property: the name of the prerequisite property checked, the PASS or FAIL result, the actual value, and expected value
- For all components: the name of the general prerequisite property checked, the PASS or FAIL result, the actual value, and the expected value
- The overall PASS or FAIL result, with any failure of an individual check resulting in the failure of the overall scan

```

C:\>prerequisites\prerequisites_checker.bat DMO detail

IBM Prerequisite Scanner
Version : 1.1.1.8
Build : 20110927
OS Name : Microsoft Windows XP Professional Service Pack 3
User Name: <User Name>

Machine Info
Machine name : <Machine name>
Serial Number: <Serial number>
OS Serial : <OS serial number>

DMO - Prerequisite Scanner Demo [version 01000000]:

Property          Result Found Expected
=====
OS Version        PASS Microsoft Win... regex<Windows...
Memory            PASS 645MB 128MB
Disk#1 (C:\ibm\ITM) PASS 1.38GB 1.00GB
os.versionNumber  PASS 5.1.2600 5.1.*
os.servicePack    PASS 3.0 2+
os.architecture   PASS 32-bit 32-bit
os.totalPhysicalMemory PASS 3.00GB 2.00GB
os.is8dot3FileFormatEnabled PASS True True
os.isServiceRunning.terminalServices PASS True True
os.isServiceRunning.remoteRegistry FAIL True False
os.isServiceRunning.DNSClient PASS True True
user.isAdmin      PASS True True
network.availablePorts.DB PASS 135,445,523,1... 60000-60005
network.availablePorts.WAS PASS 135,445,523,1... 8080
network.availablePorts.FTP PASS 135,445,523,1... 21
network.netBIOSEnabled PASS True True
network.pingSelf  PASS True True
network.DHCPEnabled FAIL True False
cygwinVersion     FAIL 0.0 1.5+

ALL COMPONENTS :
Property          Result Found Expected
=====
Memory            PASS 645MB 128MB
C:                PASS 1.38GB 1.00GB

Prereq Scanner Overall Result: FAIL

Details also available in C:\prerequisites\prerequisites_checker\result.txt

C:\prerequisites\prerequisites_checker>_

```

Figure 1. Output to the command-line interface on Windows systems

If you do not set the **detail** parameter, the Scanner displays only the PASS or FAIL result on the screen.

```

root@aqlinux15:~/prs/20110927-0849
File Edit View Terminal Tabs Help
[root@aqlinux15 20110927-0849]# ./prereq_checker.sh DM0
IBM Prerequisite Scanner
  Version: 1.1.1.8
  Build   : 20110927
  OS Name: Linux

Machine Info
Machine Name : <Machine name>
Serial Number: <Serial number>

TPS detected : Red Hat Enterprise Linux Server release 5.5 {32-bit}
Using the DM0 config file
Using config file - /root/prs/20110927-0849/UNIX_Linux/DM0_0750000.cfg for DM0
FAIL
[root@aqlinux15 20110927-0849]#

```

Figure 2. Output to the command-line interface on UNIX systems

Prerequisite Scanner generates returns codes dependent on the results of the scan and whether it must exit because of errors. These return codes are written to the log files. For example, if the Prerequisite Scanner fails to run the scan because cannot read the configuration file, it generates return code of 2.

Aggregating memory and disk space prerequisite properties

You can run Prerequisite Scanner to simultaneously check for the prerequisites of one or many products or components, when you specify multiple product codes as input parameters. Prerequisite Scanner aggregates the results of memory and disk space prerequisite checks in the following aggregated sections of the output, if the associated prerequisite properties have been specified in any of the configuration files:

- On UNIX systems, in the TOTAL ALL SPECIFIED COMPONENTS section
- On Windows systems, in the ALL COMPONENTS section

The general prerequisite properties are as follows:

- Total amount of physical memory that is currently available in the target environment, that is:
 - Memory
- Disk space of the file systems for the following prerequisite properties:

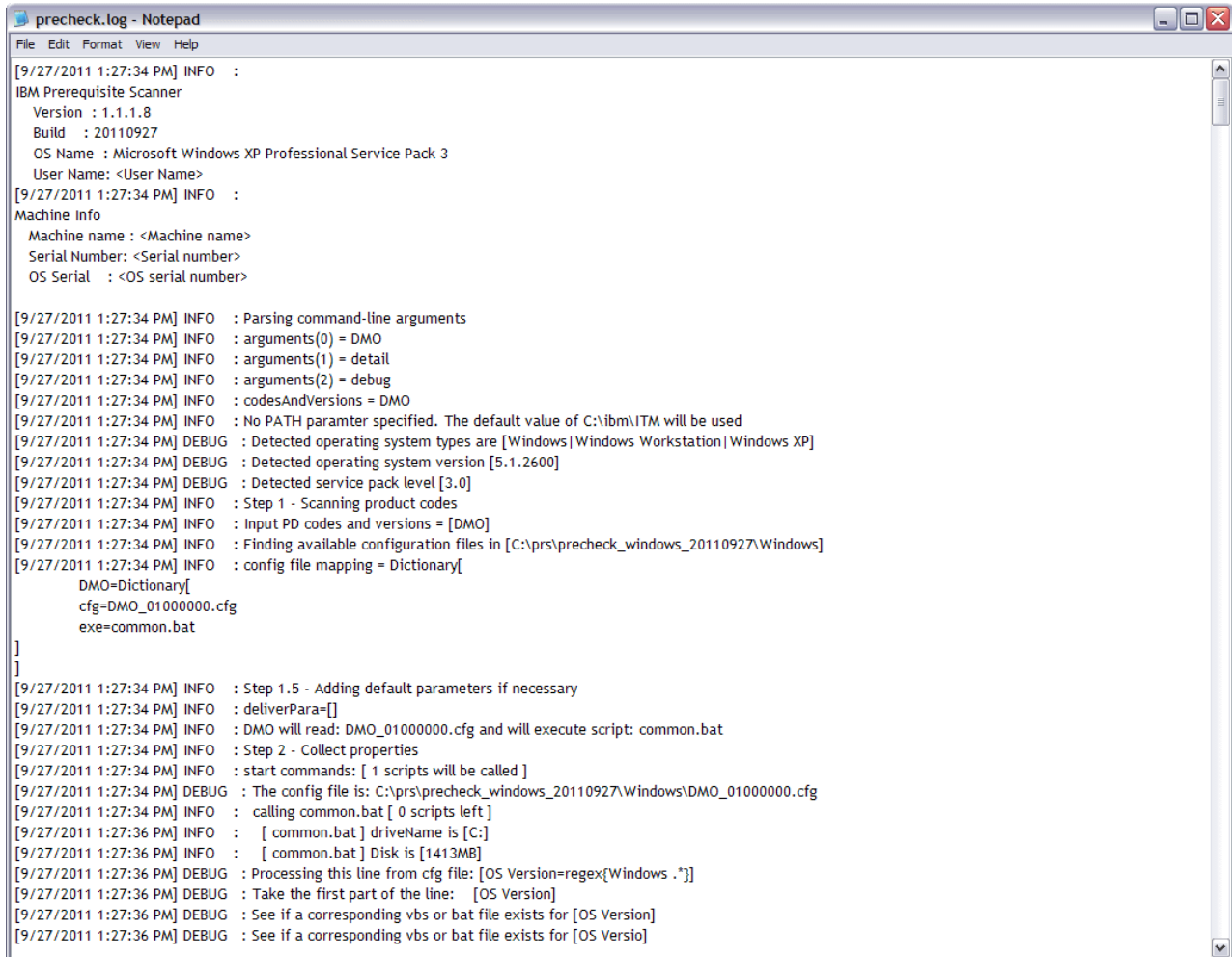
Platform	Prerequisite properties
UNIX and Linux	<ul style="list-style-type: none"> os.space.home os.space.opt os.space.tmp os.space usr os.space.var
Windows	Disk

Note: If opt, usr, and var have not been defined as file systems on the target computer, Prerequisite Scanner does not display the expected values and returned results for these prerequisite properties in the collated section.

Prerequisite Scanner does not display the aggregated section, if neither memory nor disk space prerequisite properties exist in the configuration files.

Debugging log file output on Windows systems

Prerequisite Scanner outputs processing information, warning and error messages, and the scan results in the *ips_output_dir/precheck.log* file. When you run the Prerequisite Scanner script and set the optional **debug** parameter, Prerequisite Scanner outputs additional debugging messages in this file.



```
precheck.log - Notepad
File Edit Format View Help

[9/27/2011 1:27:34 PM] INFO :
IBM Prerequisite Scanner
  Version : 1.1.1.8
  Build : 20110927
  OS Name : Microsoft Windows XP Professional Service Pack 3
  User Name: <User Name>
[9/27/2011 1:27:34 PM] INFO :
Machine Info
  Machine name : <Machine name>
  Serial Number: <Serial number>
  OS Serial : <OS serial number>

[9/27/2011 1:27:34 PM] INFO : Parsing command-line arguments
[9/27/2011 1:27:34 PM] INFO : arguments(0) = DMO
[9/27/2011 1:27:34 PM] INFO : arguments(1) = detail
[9/27/2011 1:27:34 PM] INFO : arguments(2) = debug
[9/27/2011 1:27:34 PM] INFO : codesAndVersions = DMO
[9/27/2011 1:27:34 PM] INFO : No PATH paramter specified. The default value of C:\ibm\ITM will be used
[9/27/2011 1:27:34 PM] DEBUG : Detected operating system types are [Windows|Windows Workstation|Windows XP]
[9/27/2011 1:27:34 PM] DEBUG : Detected operating system version [5.1.2600]
[9/27/2011 1:27:34 PM] DEBUG : Detected service pack level [3.0]
[9/27/2011 1:27:34 PM] INFO : Step 1 - Scanning product codes
[9/27/2011 1:27:34 PM] INFO : Input PD codes and versions = [DMO]
[9/27/2011 1:27:34 PM] INFO : Finding available configuration files in [C:\prs\precheck_windows_20110927\Windows]
[9/27/2011 1:27:34 PM] INFO : config file mapping = Dictionary[
  DMO=Dictionary[
    cfg=DMO_01000000.cfg
    exe=common.bat
  ]
]
[9/27/2011 1:27:34 PM] INFO : Step 1.5 - Adding default parameters if necessary
[9/27/2011 1:27:34 PM] INFO : deliverPara=[]
[9/27/2011 1:27:34 PM] INFO : DMO will read: DMO_01000000.cfg and will execute script: common.bat
[9/27/2011 1:27:34 PM] INFO : Step 2 - Collect properties
[9/27/2011 1:27:34 PM] INFO : start commands: [ 1 scripts will be called ]
[9/27/2011 1:27:34 PM] DEBUG : The config file is: C:\prs\precheck_windows_20110927\Windows\DMO_01000000.cfg
[9/27/2011 1:27:34 PM] INFO : calling common.bat [ 0 scripts left ]
[9/27/2011 1:27:36 PM] INFO : [ common.bat ] driveName is [C:]
[9/27/2011 1:27:36 PM] INFO : [ common.bat ] Disk is [1413MB]
[9/27/2011 1:27:36 PM] DEBUG : Processing this line from cfg file: [OS Version=regex{Windows .*}]
[9/27/2011 1:27:36 PM] DEBUG : Take the first part of the line: [OS Version]
[9/27/2011 1:27:36 PM] DEBUG : See if a corresponding vbs or bat file exists for [OS Version]
[9/27/2011 1:27:36 PM] DEBUG : See if a corresponding vbs or bat file exists for [OS Versio]
```

Figure 3. precheck.log file

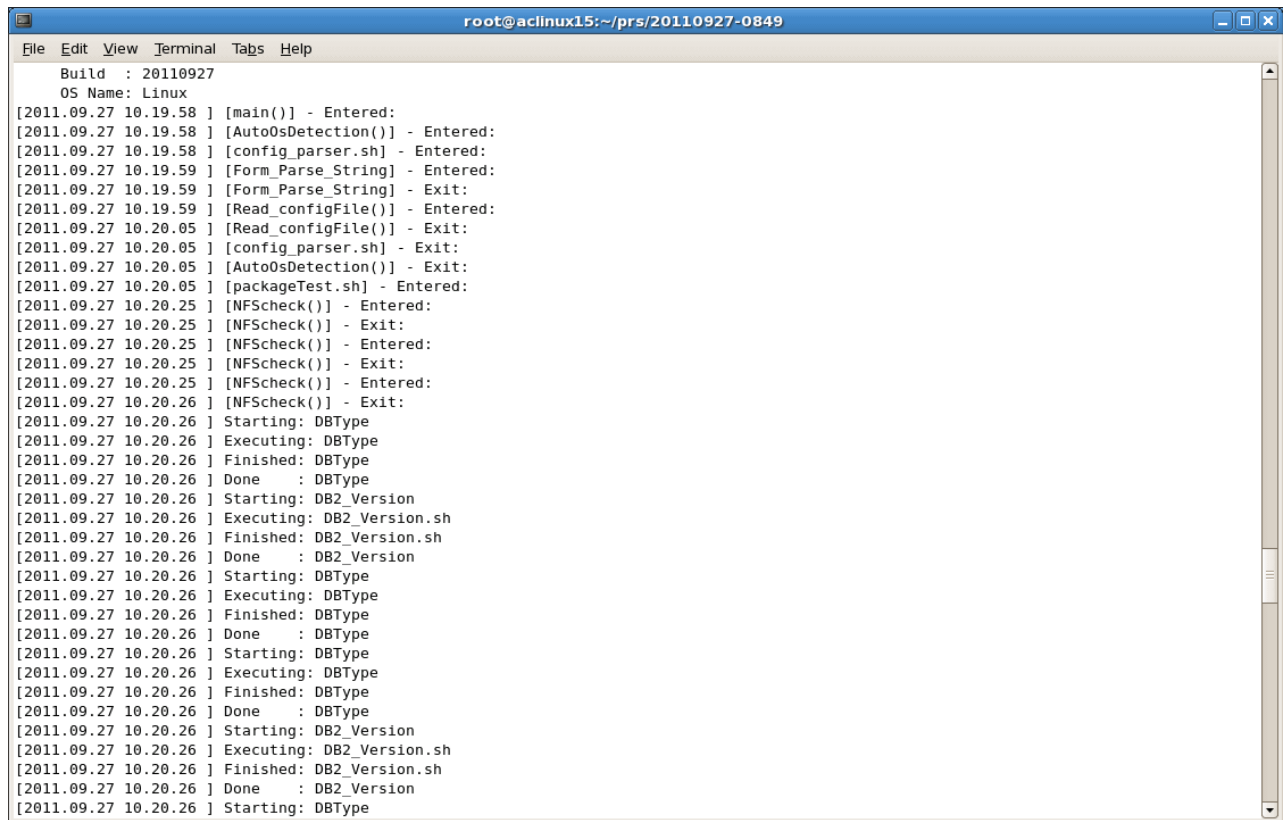
Trace and debugging log file output on UNIX systems

When you run the Prerequisite Scanner script and set the optional **debug** parameter, Prerequisite Scanner outputs detailed processing information, warning and error messages, and the scan results in the *ips_output_dir/temp/prs.debug* file.

```
root@aclinux15:~/prs/20110927-0849
File Edit View Terminal Tabs Help
Build : 20110927
OS Name: Linux
[2011.09.27 10.12.15 ] [main()] - Entered
[2011.09.27 10.12.15 ] ==== Step 1: Detecting OS...
[2011.09.27 10.12.15 ] OS Detected: Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] product_version: DMO
[2011.09.27 10.12.15 ] [AutoOsDetection()] - Entered
[2011.09.27 10.12.15 ] [Param] ProductInfo:DMO
[2011.09.27 10.12.15 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] [Param] CPU Arch:Kernel=i686
[2011.09.27 10.12.15 ] Finding product code in product.cfg
[2011.09.27 10.12.15 ] product code found :
[2011.09.27 10.12.15 ] Found DMO code in product.cfg
[2011.09.27 10.12.15 ] Finding OS Arch and CPU Type
[2011.09.27 10.12.15 ] Found OS Arch = 32-bit, CPU Type=
[2011.09.27 10.12.15 ] Calling config_parser.sh...
[2011.09.27 10.12.15 ] [config_parser.sh] - Entered
[2011.09.27 10.12.15 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] [Param] ProductCode:DMO
[2011.09.27 10.12.15 ] [Param] OSArch:Arch=32-bit
[2011.09.27 10.12.16 ] [Param] CPUArch:CPU=
[2011.09.27 10.12.16 ] [Param] Version:version=
[2011.09.27 10.12.16 ] [Param] XXX:Kernel=i686
[2011.09.27 10.12.16 ] Forming parse array...
[2011.09.27 10.12.16 ] [Form_Parse_String] - Entered
[2011.09.27 10.12.16 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.16 ] [Param] ProductCode:DMO
[2011.09.27 10.12.16 ] [Param] OSArch:Arch=32-bit
[2011.09.27 10.12.16 ] [Param] CPU:CPU=
[2011.09.27 10.12.16 ] [Param] CPUArch:Kernel=i686
[2011.09.27 10.12.16 ] Form_Parse_String - ParseArray: [OSType:UNIX][OSType:Linux][OSType:RedHat][OSType:RedHatEnterpriseLinuxServer][OS
Type:RedHatEnterpriseLinuxServer5.*][OSType:RedHatEnterpriseLinuxServer5.5][OSArch:32-bit][CPUArch:i686]
[2011.09.27 10.12.16 ] [Form_Parse_String] - Exit
[2011.09.27 10.12.16 ] Reading config file and parsing using parse array...
[2011.09.27 10.12.16 ] [Read_configFile()] - Entered
[2011.09.27 10.12.16 ] [Param] ConfigFile:/root/prs/20110927-0849/UNIX_Linux/DMO_0750000.cfg-Master
[2011.09.27 10.12.16 ] [Param] Product:DMO
[2011.09.27 10.12.17 ] Writing DBType=Oracle to DMO_0750000.cfg
[2011.09.27 10.12.21 ] Found Env Var - TPAE_DB_Server
```

Figure 4. prs.debug file on UNIX systems

When you run the Prerequisite Scanner script and set the optional **trace** parameter, Prerequisite Scanner outputs trace information in the *ips_output_dir*temp/prs.trc file.



```
Build : 20110927
OS Name: Linux
[2011.09.27 10.19.58 ] [main()] - Entered:
[2011.09.27 10.19.58 ] [AutoOsDetection()] - Entered:
[2011.09.27 10.19.58 ] [config_parser.sh] - Entered:
[2011.09.27 10.19.59 ] [Form_Parse_String] - Entered:
[2011.09.27 10.19.59 ] [Form_Parse_String] - Exit:
[2011.09.27 10.19.59 ] [Read_configFile()] - Entered:
[2011.09.27 10.20.05 ] [Read_configFile()] - Exit:
[2011.09.27 10.20.05 ] [config_parser.sh] - Exit:
[2011.09.27 10.20.05 ] [AutoOsDetection()] - Exit:
[2011.09.27 10.20.05 ] [packageTest.sh] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Exit:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Exit:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.26 ] [NFScheck()] - Exit:
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DB2_Version
[2011.09.27 10.20.26 ] Executing: DB2_Version.sh
[2011.09.27 10.20.26 ] Finished: DB2_Version.sh
[2011.09.27 10.20.26 ] Done : DB2_Version
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DB2_Version
[2011.09.27 10.20.26 ] Executing: DB2_Version.sh
[2011.09.27 10.20.26 ] Finished: DB2_Version.sh
[2011.09.27 10.20.26 ] Done : DB2_Version
[2011.09.27 10.20.26 ] Starting: DBType
```

Figure 5. prs.trc file on UNIX systems

Text file output

Prerequisite Scanner outputs detailed scan results in the *ips_output_dir/result.txt* file. It saves the results to the text file regardless of whether you set the **detail** parameter.

```

result.txt - Notepad
File Edit Format View Help

IBM Prerequisite Scanner
Version : 1.1.1.8
Build : 20110927
OS Name : Microsoft Windows XP Professional Service Pack 3
User Name: <User Name>
Machine Info
Machine name : <Machine name>
Serial Number: <Serial number>
OS Serial : <OS serial number>

DMO - Prerequisite Scanner Demo [version 01000000]:

Property          Result Found          Expected
=====
OS Version         PASS  Microsoft Windows XP Professional Service Pack 3  regex[Windows .*]
Memory            PASS  645MB  128MB
Disk#1 (C:\ibm\ITM) PASS  1.38GB  1.00GB
os.versionNumber   PASS  5.1.2600  5.1.*
os.servicePack     PASS  3.0  2+
os.architecture    PASS  32-bit  32-bit
os.totalPhysicalMemory PASS  3.00GB  2.00GB
os.is8dot3FileFormatEnabled PASS  True  True
os.isServiceRunning.terminalServices PASS  True  True
os.isServiceRunning.remoteRegistry FAIL  True  False
os.isServiceRunning.DNSClient PASS  True  True
user.isAdmin       PASS  True  True
network.availablePorts.DB PASS  135,445,523,1035,1067,1099,1527,2967,3389,5157,16310,16311,16312,16313,16315... 60000-60005
network.availablePorts.WAS PASS  135,445,523,1035,1067,1099,1527,2967,3389,5157,16310,16311,16312,16313,16315... 8080
network.availablePorts.FTP PASS  135,445,523,1035,1067,1099,1527,2967,3389,5157,16310,16311,16312,16313,16315... 21
network.netBIOSEnabled PASS  True  True
network.pingSelf   PASS  True  True
network.DHCPEnabled FAIL  True  False
cygwinVersion      FAIL  0.0  1.5+

ALL COMPONENTS :
Property          Result Found          Expected
=====
Memory            PASS  645MB  128MB
C:                PASS  1415MB  1024MB

Prereq Scanner Overall Result: FAIL

```

Figure 6. result.txt file on Windows systems


```
[root@aqlinux15 20110927-0849]# cat result.txt
IBM Prerequisite Scanner
  Version: 1.1.1.8
  Build   : 20110927
  OS Name : Linux

Machine Info
Machine Name : <Machine name>
Serial Number: <Serial number>

DMO - Prerequisite Scanner Demo [0750000]:
Evaluation      PASS/FAIL      Result      Expected Result
DBType          FAIL          Unknown     Oracle
DBType          FAIL          Unknown     DB2
DBType          FAIL          Unknown     regex{.*Oracle.*}
DBType          FAIL          Unknown     regex{.*DB2.*}
DBTypeDetails  FAIL          Unknown     oracle
DBTypeDetails  FAIL          Unknown     DB2
DBTypeDetails  FAIL          Unknown     regex{.*Oracle.*}
DBTypeDetails  FAIL          Unknown     regex{.*DB2.*}
OS Version      PASS          "Red Hat Enterprise Linux Server release 5.5 (Tikanga)" "regex{Red Hat.*Tikanga.*}"

os.lib.libstdc++      PASS          /usr/lib/gcc/i386-redhat-linux/4.1.1/libstdc++.so
libstdc++.so.1

os.lib.libgcc         PASS          /usr/lib/gcc/i386-redhat-linux/3.4.6/libgcc_s.so
libgcc_s.so.1

os.lib.libXp          PASS          /usr/lib/libXmu.so.6
libXmu.so.6

os.space.var          PASS          "38GB"
common/acsi"

os.space.usr          PASS          "38GB"
common/acsi"

os.space.tmp          PASS          36GB
env.classpath.derbyJAR PASS          False
network.pingSelf      PASS          True
env.classpath.derbyJAR PASS          False
```

Figure 7. result.txt file on UNIX systems

XML file output

Prerequisite Scanner outputs detailed scan results in the `ips_output_dir/result.xml` file when you specify the optional **xmlResult** input parameter. You can use it to indicate to the tool to output the results to the XML result file in addition to the plain test result file. It saves the results to the XML file regardless of whether you set the **detail** parameter.

```

<PRSInfo>

<MachineInfo>
  <MachineName>my_machine_name</MachineName>
  <MachineSerialNumber>serial_number</MachineSerialNumber>
  <MachineOSSerial>os_serial_number</MachineOSSerial>
  <MachineOSName>Microsoft Windows XP Professional Service Pack 3</MachineOSName>
</MachineInfo>

<UserInfo>

<ProductInfo>
  <ProductElement>
    <ProductCode>DMO</ProductCode>
    <ProductName>Prerequisite Scanner Demo</ProductName>
    <ProductVersion>01000000</ProductVersion>
  </ProductElement>
</ProductInfo>

<DetailedResults>
  <DetailedProductResultsElement>
    <ProductCode>DMO</ProductCode>
    <ResultElement>
      <PropertyName>OS Version</PropertyName>
      <Result>FAIL</Result>
      <Found>Microsoft Windows XP Professional Service Pack 3</Found>
      <Expected>Windows 7 Ultimate</Expected>
    </ResultElement>
    <ResultElement>
      <PropertyName>Memory</PropertyName>
      <Result>PASS</Result>
      <Found>960MB</Found>
      <Expected>128MB</Expected>
    </ResultElement>
    <ResultElement>
      <PropertyName>Disk#1 (C:\ibm\ITM)</PropertyName>
      <Result>PASS</Result>
      <Found>22072MB</Found>
      <Expected>1GB</Expected>
    </ResultElement>
    <ResultElement>
      <PropertyName>os.versionNumber</PropertyName>
      <Result>FAIL</Result>
      <Found>5.1.2600</Found>
      <Expected>5.2.*</Expected>
    </ResultElement>
  </DetailedProductResultsElement>
</DetailedResults>

```

Figure 8. result.XML file on Windows systems

Developers can use the Prerequisite Scanner Java Developer toolkit to parse and read the XML file.

Related concepts:

The Prerequisite Scanner Java Developer toolkit is a set of APIs to allow you, as a developer, to programmatically parse and read the contents of the results XML file for your needs; for example, parse the results of the scan to use in your installation program.

Prerequisite Scanner generates return codes dependent on the results of the scan and whether it must exit because of errors. These return codes are written to the log files.

You can use the execution problems checklist to troubleshoot errors you might encounter when you run Prerequisite Scanner.

Related reference:

The `prereq_checker` script runs the IBM Prerequisite Scanner and checks for prerequisites based on the set of parameters that you specify when you run the script.

Prerequisite Scanner Java Developer toolkit

The Prerequisite Scanner Java Developer toolkit is a set of APIs to allow you, as a developer, to programmatically parse and read the contents of the results XML file for your needs; for example, parse the results of the scan to use in your installation program.

The toolkit provides the following packages:

- `com.ibm.prs.common.exception`
It contains the `PRSApiException` class that provides methods to throw exceptions for the XML query API.
- `com.ibm.prs.common.reports.api`
It contains the `PRSXmlResultReader` interface that defines the XML query API for the XML result file.
- `com.ibm.prs.common.reports.api.impl`
It contains the `PRSXmlResultReaderImpl` class that implements `PRSXmlResultReader`.

Prerequisite Scanner can validate the formatting and structure against the `ips_root/PRSResults.xsd` XML schema file.

Javadoc is available for the toolkit in the `ips_root/api/javadoc` directory.

XML schema file for the XML result file

Prerequisite Scanner provides an XML schema file against which the result XML file can be validated.

The XML schema file contains the following elements representing sections:

- `PRInfo` to manage Prerequisite Scanner details
- `MachineInfo` to manage information about the target environment on which the scan is run
- `UserInfo` to manage information about the logged in user running the scan
- `ProductInfo` to manage information about the product or component and its configuration file
- `DetailedResults` to manage the scan results for each set of prerequisite properties for a product or component as grouped by `DetailedProductResultsElement`

- `AggregateResults` to manage the aggregated scan results for disk space and memory
- `OverallResult` to manage the overall PASS or FAIL result of the scan

The name and location of the XML schema is: `ips_root/PRSResults.xsd`

As a developer or deployer, you can invoke methods from the query XML API to validate the result XML file. The Javadoc is available for the toolkit in the `ips_root/api/javadoc` directory.

Scanning process

When you run IBM Prerequisite Scanner, it performs a set of tasks in each stage of the scanning process. The user opens a command-line interface and runs the Prerequisite Scanner script with the set of input parameters including a product code.

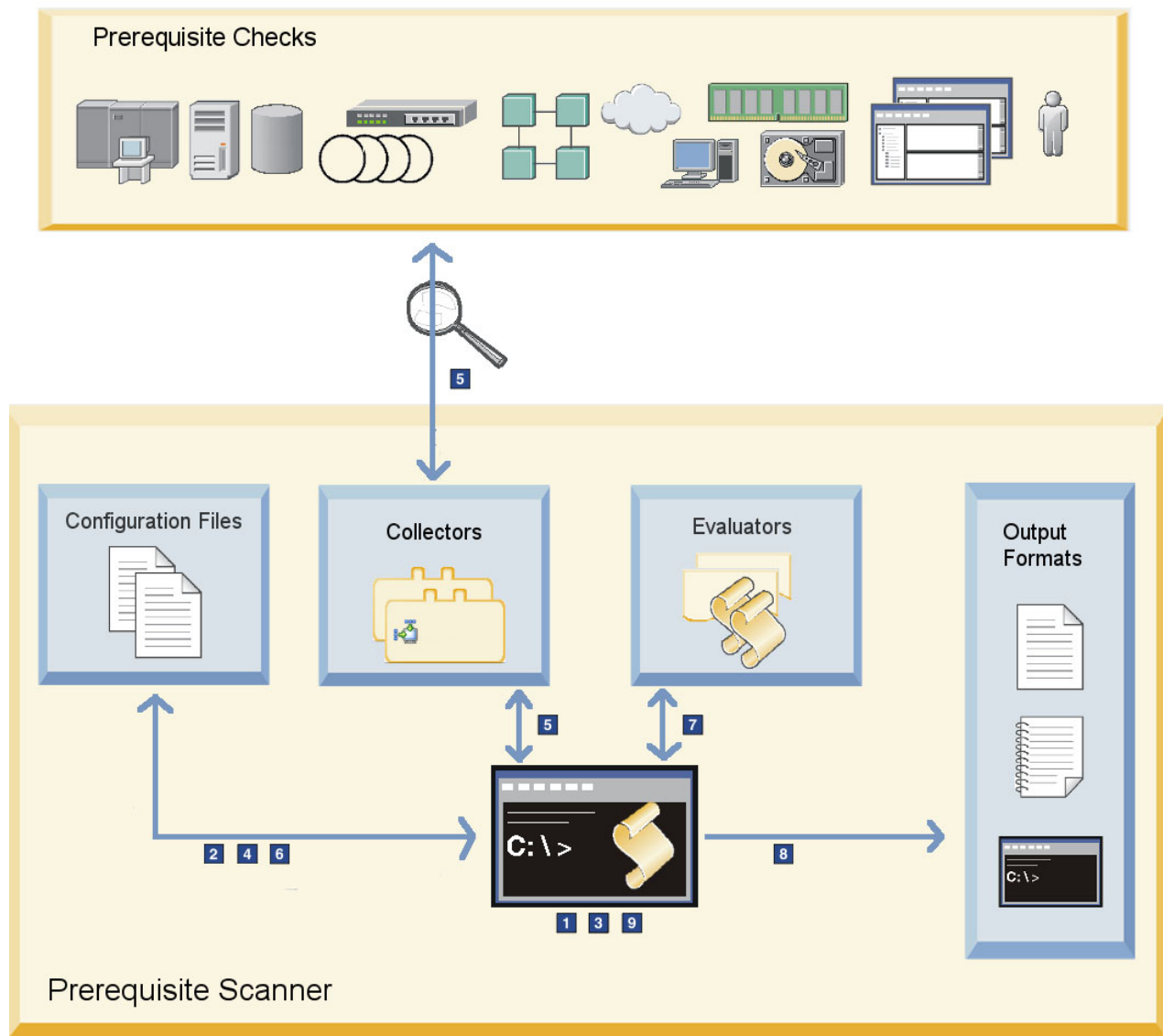


Figure 9. Prerequisite Scanner architecture and scanning process

The scanning process in Figure 9 is summarized as follows:

1. Prerequisite Scanner verifies the format of the input parameters that are passed to the Scanner.
2. The Scanner determines whether the product code that is passed as one of the input parameters is a valid product code in the `codename.cfg` file.
3. The Scanner automatically detects the operating system of the machine.
4. The Scanner verifies whether the actual operating system of the machine is a supported operating system. It uses one of the following methods:
 - Product code file
 - Section title in the configuration file

The Scanner checks the actual operating system against the expected supported operating system listed for the product code in the `codename.cfg` file.

The Scanner checks the actual operating system against the expected supported operating system in the section titles in the configuration file, whose file name contains the same product code and the product version as the input parameters. If the optional product version parameter was not passed, the Scanner uses the latest version of the configuration file that it finds in the *ips_root/Windows|UNIX_Linux* directory.

If the platform is not supported, Prerequisite Scanner exits and displays a message to state that the platform is not supported.

5. The Scanner collects the actual prerequisite properties for the prerequisite checks by using the Prerequisite Scanner collectors.
6. If the actual operating system is supported, the Scanner checks the prerequisite properties in the configuration file that is associated with the product code and product version.

The Scanner checks the actual operating system against the expected supported operating system in the section titles in the configuration file, whose file name contains the same product code and the product version as the input parameters. If the optional product version parameter was not passed, the Scanner uses the latest version of the configuration file that it finds in the *ips_root/Windows|UNIX_Linux* directory.

7. The Scanner reads the prerequisite properties from the configuration file and analyzes the actual and expected values of the prerequisite properties for the prerequisite checks. It uses the Prerequisite Scanner evaluators where necessary.
8. The Scanner outputs the results of the scan to the command-line interface, results text file, and the human-readable log files.
9. The Scanner cleans up and removes temporary files and directories.

New in this release

IBM Prerequisite Scanner Version 1.2 Sprint 3 driver provides new prerequisite properties and enhancements. It also contains fixes for defects.

New features in this modification

Ability to parse and read the new scan results XML file.

The Prerequisite Scanner Java Developer toolkit is a set of APIs to allow developers to programmatically parse and read the contents of the results XML file for their needs; for example, parse the results of the scan to use in an installation program. See “Prerequisite Scanner Java Developer toolkit” on page 33.

New configuration files in this modification

Table 9 outlines the new configuration files and product codes shipped with Prerequisite Scanner Version 1.2 Sprint 3 driver

Table 9. New configuration files

Product or component	Product code	Configuration file
Tivoli Composite Application Manager Agent for WebSphere MQ	KMQ	<i>ips_root/Windows UNIX_Linux/KMQ_07010000.cfg</i>

New prerequisite properties in this modification

None

Enhancements in this modification

Ability to write scan results to an XML file.

ips_output_dir/result.xml is the new scan results file in XML format. By default the tool outputs the results to the plain text result file only. See “Output formats” on page 23.

xmlResult is a new, optional input parameter for the Prerequisite Scanner script in Prerequisite Scanner Version 1.2 Sprint 3 driver. You can use it to indicate to the tool to output the results to the XML result file in addition to the plain test result file. See “prereq_checker” on page 61.

Removal of the aggregated section in the results if neither memory nor disk space prerequisite properties exist in the configuration files. Prerequisite Scanner no longer displays the aggregated sections in the result file when none of the memory or disk space prerequisite properties exist in the configuration files. See “Output formats” on page 23.

Deprecated features in this modification

None

Defects fixed with this modification

Table 10 outlines the defects fixed in this modification.

Table 10. Defects fixed in this release

Defect number	Description
25890	Previous versions of the documentation did not document how Prerequisite Scanner handles the expected value for the Memory prerequisite property, when running a scan on a computer that already has an existing version of the Tivoli Monitoring agent installed. It has been added to the documentation for the Prerequisite Scanner Version 1.2 Sprint 3 driver. See “System behavior for Memory prerequisite property and Tivoli Monitoring agents” on page 90.
26489	network.ValidateHostsFile prerequisite property was missing from the previous versions of the documentation. It has been added to the documentation for the Prerequisite Scanner Version 1.2 Sprint 3 driver. See “Windows network data properties” on page 107.

Chapter 2. Installing Prerequisite Scanner

There is no installation program for IBM Prerequisite Scanner. When you extract the contents of the compressed file, the core files are in the root directory, with the following subdirectories: /api for the Prerequisite Scanner Java Developer toolkit to support query XML API, /lib for the collectors and common scripts; /Windows for the evaluators and configuration files on Windows; /UNIX_Linux for the evaluators and configuration files on UNIX platforms; and /licenses for the license files.

Prerequisites

IBM Prerequisite Scanner can run on Windows systems, Windows XP or above, 32-bit, or 64-bit. It can also run on variants of AIX, HP-UX, Linux, and Solaris operating systems.

Ensure that you have the following utilities installed or available in the target environments:

Target system	Prerequisites
Windows	The Telnet client is enabled, so that the connectivity checks in the predefined Connectivity collector can function correctly.
UNIX	<ul style="list-style-type: none">Bash is installed, so that the Prerequisite Scanner UNIX collectors can function correctly.For non-root users, the location of the mount, swapinfo, and psrinfo commands must be set in the PATH environment variable, so that the commands are available to Prerequisite Scanner. The commands are in the /usr/sbin directory; for example, set the PATH environment variable as follows: export PATH=\$PATH:/usr/sbin/Ensure the correct access permissions are assigned to the lscfg command, including any specific permissions that are set by the access-right flags such as setuid bit. The correct access permissions mean that Prerequisite Scanner can run the command and retrieve system information. The command is in the /usr/sbin directory; for example to set the setuid bit for lscfg, run chmod command as follows: chmod 4777 /usr/sbin/lscfg

Prerequisite Scanner supports all hardware and operating systems of the specified product or IBM solution for which you are running Prerequisite Scanner.

Installing the compressed file

You can extract the contents of the compressed file for IBM Prerequisite Scanner. You must have write permissions to the root directory in which you extract the contents of the compressed file.

Procedure

1. Open your web browser and enter the URL to IBM Fix Central. Ensure that you sign into IBM.com or IBM Support Portal.
2. From the **Product Group** list, select **Tivoli**.
3. From the **Product** list, select IBM Prerequisite Scanner.
4. From the **Installed Version** list, select the version that you want to download.
5. From the **Platform** list, select the platform on which you want to install Prerequisite Scanner.
6. Click **Continue**. The Identify Fixes page opens.
7. Use the default option, **Browse for fixes**, and click **Continue**.
8. On the Select fixes page, select the package, and click **Continue**.
9. On the Download option page, select the download option, and click **Download now**.
10. Extract the contents of the compressed files to your preferred location as specified by *ips_root*.

What to do next

Ensure that you check your product's installation documentation or Technotes for any additional steps that must be performed before running Prerequisite Scanner. For example, you might need to set the environment variable that indicates to Prerequisite Scanner which components or features are being installed on the target computer and consequently, which prerequisites to check.

Uninstalling Prerequisite Scanner

Remove IBM Prerequisite Scanner if you want to install a newer version, move it to another environment, or it is a version that you no longer need.

Procedure

1. Open the *ips_root* directory.
2. Delete the directory and its contents.

Chapter 3. Extending Prerequisite Scanner

IBM Prerequisite Scanner provides a basic set of collectors, evaluators, and configurations that you can use to run the tool and scan for prerequisites. If the basic set of files, the prerequisite properties and values, and the prerequisite checks do not meet your requirements, you can extend Prerequisite Scanner.

Before you run Prerequisite Scanner

Before running IBM Prerequisite Scanner, determine whether the predefined prerequisite properties, their expected values, and configuration files meet your requirements for prerequisite scanning. If any of them do not meet your needs, you can perform a set of prerequisite tasks to configure or extend Prerequisite Scanner. The set of prerequisite checks and tasks is dependent upon the platform and the number of prerequisite checks.

Required checks and extension tasks for Windows systems

You should perform a set of checks and tasks before you run the IBM Prerequisite Scanner. These checks determine whether you can edit and use existing configuration files or you must extend Prerequisite Scanner.

Table 11 provides a list of checks and tasks to perform.

Table 11. Checks and tasks before using a configuration file for Windows systems

	Check	Task
<input type="checkbox"/>	Check whether the product, its supported operating systems, and versions of the operating system are listed in the <code>codename.cfg</code> file.	<ul style="list-style-type: none">• If yes, perform the next check.• If no, add a product code for the product, individual operating system and the optional operating system version to the file. For more information, see “Adding product codes” on page 43.
<input type="checkbox"/>	Check whether a configuration file exists for the product code associated with the product version.	<ul style="list-style-type: none">• If yes, perform the next check.• If no, create a configuration file to contain the prerequisite properties for that operating system and version of the operating system. For more information, see “Creating custom configuration files” on page 43.
<input type="checkbox"/>	Open the configuration file and check whether it contains the correct prerequisite properties.	<ul style="list-style-type: none">• If yes, perform the next check.• If no, add prerequisite properties. For more information, see “Adding prerequisite properties” on page 45.
<input type="checkbox"/>	Check whether the prerequisite properties have the expected values.	<ul style="list-style-type: none">• If yes, run the Prerequisite Scanner. For more information, see Chapter 4, “Running Prerequisite Scanner,” on page 61.• If no, edit the prerequisite properties. For more information, see “Editing prerequisite properties” on page 47.

Table 11. Checks and tasks before using a configuration file for Windows systems (continued)

	Check	Task
<input type="checkbox"/>	For any new prerequisite properties, check whether predefined collectors can collect the actual values for the prerequisite properties.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, create custom collectors. For more information, see “Creating custom collectors for Windows systems” on page 48.
<input type="checkbox"/>	For any new or edited prerequisite properties, check whether predefined evaluators can compare the expected and actual values for the prerequisite property.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, create custom evaluators. For more information, see “Creating custom evaluators for Windows systems” on page 55.
<input type="checkbox"/>	Ensure all files have been saved in the correct directories: <ul style="list-style-type: none"> • Configuration files, any custom product-specific collectors and associated batch files, and any custom evaluator files in the <i>ips_root/Windows</i> directory • Custom common collectors in the <i>ips_root/lib</i> directory 	Run the Prerequisite Scanner. For more information, see Chapter 4, “Running Prerequisite Scanner,” on page 61.

Required checks and extension tasks for UNIX systems

You should perform a set of prerequisite checks and tasks before you run the IBM Prerequisite Scanner. These checks determine whether you can edit and use existing configuration files or you must extend Prerequisite Scanner.

Table 12 provides a list of required checks and tasks to perform.

Table 12. Checks and tasks before using a configuration file for UNIX systems

	Check	Task
<input type="checkbox"/>	Check whether the product is listed in the <i>codename.cfg</i> file.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, add a product code to the <i>codename.cfg</i> file. For more information, see “Adding product codes” on page 43.
<input type="checkbox"/>	Check whether a configuration file exists for the product code associated with the product.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, create a configuration file to contain the prerequisite properties for all supported platforms of the product. For more information, see “Creating custom configuration files” on page 43.
<input type="checkbox"/>	Open the configuration file and check whether it contains the correct prerequisite properties.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, add prerequisite properties. For more information, see “Adding prerequisite properties” on page 45.
<input type="checkbox"/>	Check whether the prerequisite properties have the expected values.	<ul style="list-style-type: none"> • If yes, run the Prerequisite Scanner. For more information, see Chapter 4, “Running Prerequisite Scanner,” on page 61. • If no, edit the prerequisite properties. For more information, see “Editing prerequisite properties” on page 47.

Table 12. Checks and tasks before using a configuration file for UNIX systems (continued)

	Check	Task
<input type="checkbox"/>	For any new prerequisite properties, check whether predefined collectors can collect the actual values for the prerequisite properties.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, create custom collectors. For more information, see “Creating custom collectors for UNIX systems” on page 52.
<input type="checkbox"/>	For any new or edited prerequisite properties, check whether evaluators can compare the expected and actual values for the prerequisite property.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, create custom evaluators. For more information, see “Creating custom evaluators for UNIX systems” on page 59.
<input type="checkbox"/>	For any new or edited prerequisite properties, check whether the code to call and run the collectors is in the <i>ips_root/UNIX_Linux/packageTest.sh</i> script.	<ul style="list-style-type: none"> • If yes, perform the next check. • If no, edit the master package test script. For more information, see “Editing the package test script for UNIX systems” on page 53.
<input type="checkbox"/>	Ensure all files have been saved in the correct directories: <ul style="list-style-type: none"> • Configuration files, any custom collectors files, and any custom evaluator files in the <i>ips_root/UNIX_Linux</i> directory 	Run the Prerequisite Scanner. For more information, see Chapter 4, “Running Prerequisite Scanner,” on page 61.

Adding product codes

IBM Prerequisite Scanner provides a set of predefined product version codes in the *codename.cfg* file. You can add product codes if the file does not contain them for the product version, its supported platforms, and versions of the operating systems.

Procedure

1. Open the *ips_root/codename.cfg* file.
2. Check whether the file already contains name value pairs for the product versions.
3. If product code does not exist, add one and ensure that you use the correct format as follows:

```
product_code=code_value
```

Restriction: IBM Tivoli Monitoring and Tivoli Composite Application Manager have predefined product codes that Prerequisite Scanner considers as reserved. These codes must not be used as Prerequisite Scanner product codes unless they refer to their associated IBM Tivoli Monitoring and Tivoli Composite Application Manager agents. For more information about the product codes, see the ITM 6.X Product Codes Technote.

For example, to add a product code for IBM Tivoli Monitoring for Energy Management on all Windows platforms, add the following line to the file:

```
MEA=IBM Tivoli Monitoring for Energy Management
```

Creating custom configuration files

You can create custom configuration files from the sample configuration file if the predefined configuration files do not meet your requirements for prerequisite properties. Before you create the custom configuration file, ensure that you know the prerequisite properties that you want to add and their expected values.

About this task

Important: You must adhere to the naming conventions and formatting rules that govern the creation and editing of a custom configuration file. If you do not, Prerequisite Scanner cannot successfully run a scan by using this file.

Procedure

1. If necessary add product codes for the product to the `codename.cfg` file.
 2. Create the configuration file by using a text editor in the `ips_root/OS` directory. Ensure that you use the following naming convention for the file name:
`product_code_version.cfg`
where:
 - *product_code*
It is the variable to represent a product code on either Windows or UNIX systems. Product codes identify the product, an individual platform such as Windows, AIX, HP-UX, Linux, and Solaris, and optionally the version of the operating system that is supported by that product. They are stored in the `codename.cfg` file. Any product that supports multiple platforms has multiple product codes, with each one identifying a product, platform, and version of the operating system as required.
 - *version* is the 8-digit code to represent the version, release, modification, and level, with two digits for each part of the code; for example, 7.3.21 is 07032100.
 3. Review the basic prerequisite properties outlined in Appendix C, “Prerequisite properties reference,” on page 85 and determine which prerequisite properties that you want to check.
 4. Optional: Add a section and ensure that you use the following naming convention for the section title:
 - **Single, predefined data-type category**
`[category_name:category_value]`

For example, to create a section for prerequisite properties common to all Windows platforms, add the following section title:
`[OSType:Windows]`

For example, to create a section for prerequisite properties common to all RedHat Linux OS variants, add the following section title:
`[OSType:RedHat]`
 - **Combined, predefined data-type categories**
`[category_name:category_value]`
`[category_name:category_value]`

For example, to create a section for prerequisite properties for Windows Server 2003 variants excluding Windows Server 2003 R2 variant, add the following combination section title:
`[OSType:Windows Server 2003][!OSType:Windows Server 2003 R2]`

For example, create a section for prerequisite properties for SUSE Linux Enterprise Server 11 OS and whether the environment variable `@TPAE_DB_SERVER` is set to true. Add the following combination section title:
`[OSType=SUSELinuxEnterpriseServer][@TPAE_DB_SERVER:true]`
- where:

category_name is the multicharacter code that represents the data type category as outlined in Table 6 on page 15

category_value is the multicharacter code that represents an allowed value for the category as outlined in Table 6 on page 15

5. Optional: For each section, review the basic prerequisite properties outlined in Appendix C, “Prerequisite properties reference,” on page 85 and determine which prerequisite properties that you want to check.
6. For each prerequisite property that you want to add, enter a name value pair with optional qualifiers as required. Ensure that you use the following format, with only one prerequisite property on each line:

```
[prefix_identifier.]property_name[.suffix_identifier]=  
[qualifier_name:qualifier_value]property_value
```

where:

- *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
- *property_name* is the name of the prerequisite property.
- *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
- *qualifier_name* is an optional attribute for the prerequisite property. IBM Prerequisite Scanner uses it to qualify the prerequisite property or type of check to perform on the prerequisite property, as outlined in “Predefined qualifiers for prerequisite properties” on page 8.

Note: You can have multiple qualifiers, each separated by a comma. The set of qualifiers must be enclosed by [] square brackets.

- *qualifier_value* is the value for the optional attribute. Each qualifier and its value must be delimited by a : colon.
- *property_value* is the value for the prerequisite property and it can be a string or integer.

For example, the user predefined category of prerequisite properties has the user prefix identifier. The prerequisite property for checking whether the logged on user belongs to the Administrator user group is:

```
user.isAdmin=True
```

7. If a prerequisite property does not exist in the predefined categories, add the name for the custom prerequisite property, its value, and optional qualifiers. You must then create the following files to check for and compare the custom prerequisite property as required: a custom collector to collect the actual value for the prerequisite property and a custom evaluator if the standard compare functions cannot compare the actual and expected values.

Adding prerequisite properties

You can add basic prerequisite properties from the predefined categories for prerequisite properties to configuration files. Alternatively, you can add custom prerequisite properties.

About this task

Important: You must adhere to the formatting rules that govern the addition and editing of prerequisite properties to a configuration file. If you do not, Prerequisite Scanner cannot successfully run a scan for that prerequisite property.

Procedure

1. Open the configuration file.
2. Review the basic prerequisite properties outlined in Appendix C, “Prerequisite properties reference,” on page 85 and determine which prerequisite properties that you want to check.
3. For each prerequisite property that you want to add, enter a name value pair with optional qualifiers as required.

For example, to add prerequisite properties from the common predefined category, enter the property name and expected value only. Add the following prerequisite properties to the file:

```
Disk=1GB
OS Version=regex{Windows 200[3-8]}
```

For example, the network predefined category of prerequisite properties has the network prefix identifier and the prerequisite property name for checking available ports is `availablePorts`. You can further categorize the available ports by application subtypes, DB2 for DB2 database server, WAS for WebSphere Application Server, FTP for FTP protocol. Add the following prerequisite properties to the file:

```
network.availablePorts.DB2=5000-5005
network.availablePorts.WAS=9080
network.availablePorts.FTP=21
```

For example, the operating system predefined category of prerequisite properties has the `os` prefix identifier and the prerequisite property name for checking available disk space for file systems is `space`. You can further categorize the check by file system subtypes, `usr` and `home`. You can specify values for the `dir` and `unit` qualifiers.

Add the following prerequisite properties to the file:

```
os.space.usr=[dir:root=/usr/ibm/common/acsi,unit:GB]2
os.space.home=[dir:non_root=USERHOME/.acsi_HOST,unit:MB]200
```

Important: You can use the predefined qualifiers only with specific predefined prerequisite properties, as outlined in Table 5 on page 9.

4. If a prerequisite property does not exist in the predefined categories of prerequisite properties, add the name value pair with optional qualifier for the custom prerequisite property and value. Ensure that you use the following format, with only prerequisite property on each line.

```
[prefix_identifier.]property_name[.suffix_identifier]=
[[qualifier_name:qualifier_value]]property_value
```

where:

- *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
- *property_name* is the name of the prerequisite property.
- *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
- *qualifier_name* is an optional attribute for the prerequisite property. IBM Prerequisite Scanner uses it to qualify the prerequisite property or type of check to perform on the prerequisite property, as outlined in “Predefined qualifiers for prerequisite properties” on page 8.

Note: You can have multiple qualifiers, each separated by a comma. The set of qualifiers must be enclosed by [] square brackets.

- *qualifier_value* is the value for the optional attribute. Each qualifier and its value must be delimited by a : colon.
- *property_value* is the value for the prerequisite property and it can be a string or integer.

For example, `env.tcrhome` is a custom prerequisite property that checks the home directory environment variable for Tivoli Common Reporting, and the expected value should be True:

```
env.tcrhome=True
```

`env.path.jar` is a custom prerequisite property that checks whether the JRE is set in the PATH environment variable, and the expected value should be False:

```
env.path.jar=False
```

Note: You must then create the following files to check for and compare the custom prerequisite property as required: a custom collector to collect the actual value for the prerequisite property and a custom evaluator only if the standard compare functions cannot compare the actual and expected values.

Editing prerequisite properties

You can edit prerequisite properties, change the expected values for those prerequisite properties, or change the associated values of the qualifiers.

Before you begin

Check whether the new value is a valid value that is supported by the prerequisite property. For example, the Disk prerequisite property expects a numeric format with either the MB or GB unit. If you want to check the available disk space in terabytes (TB), you must extend the compare API to support TB comparisons. You must also edit the Disk prerequisite property in the relevant configuration files.

Check the predefined qualifiers and valid values for the prerequisite property, as outlined in “Predefined qualifiers for prerequisite properties” on page 8.

Procedure

1. Open the configuration file.
2. For each prerequisite property that you want to edit, enter the new expected value or change the value for the qualifier. For example, a new system administrator is the root user, so the value for the `user.userID` prerequisite property must change. Change the value to the new name:

```
user.userID=smithj
```

For example, the type qualifier for the `os.ulimit` prerequisite property currently has a value of `filedescriptorlimit` to check the limit for the file descriptors. You might want to check another limit such as the stack size. Change the following qualifier's value for the prerequisite property from:

```
os.ulimit=[type:filedescriptorlimit]8192+,unlimited
```

to:

```
os.ulimit=[type:stacksize]512+,unlimited
```

Important: You can use the predefined qualifiers only with specific predefined prerequisite properties, as outlined in Table 5 on page 9.

Creating custom collectors for Windows systems

You can create custom connectors if the basic set collectors do not collect values for the prerequisite properties required for the product to be installed. You can create custom common VBScript collectors to collect data for prerequisite properties that apply to any product and product version. Alternatively, you can create custom product-specific ones to collect data that apply to a specific product and product version. While each type of custom VBScript collector collects data by using the same methods, the rules for creation, storage, and execution are slightly different.

Creating custom VBScript collectors common to all configuration files

When you create custom common VBScript collectors, the file name must contain the name of the prerequisite property and stored in the `/lib` subdirectory. The collector contains code to obtain the actual value for a prerequisite property. It can also use the common functions and sub routines to obtain the value if required.

Before you begin

Ensure that you review the set of predefined functions and sub routines in the following appendixes before you create the collectors. Determine whether you can use any of them to obtain the actual values:

- Appendix E, “Common functions for Windows systems,” on page 115
- Appendix G, “File utility sub routines for Windows systems,” on page 131
- Appendix F, “Logging utility sub routines for Windows systems,” on page 129
- Appendix H, “Other common functions and sub routines for Windows systems,” on page 133

Determine whether the collector must check that the prerequisite property exists and if it does, what other information must be gathered. Each check must return a value, whether one exists or not. For example:

- Check whether an environment variable exists, such as the home directory of a product, for example `TCR_HOME` for Tivoli Common Reporting.
- Check whether the environment variable contains a JAR file, binary, or path, such as the path to the JRE in the `PATH` environment variable.
- Check the actual value of an environment variable, such as the home directory of a product, for example `TCR_HOME` for Tivoli Common Reporting.
- Check whether a product is installed.
- Check what version of the product is installed.

Procedure

1. Create a VBScript file. Save the file in the `ips_root/lib` directory, with a variant of the following file naming convention:

`[prefix_identifier.]property_name.vbs`

where:

- *prefix_identifier* is the prefix identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4.
- *property_name* is the prerequisite property name and is used in the collector name.

For example, `mssqlVersion.vbs` contains the code to obtain the actual value for the MS SQL server prerequisite property on the Windows machine.

- Using a VBScript editor, add the code to obtain the value for the prerequisite property. Use VBScript COM and functions to access elements of the Windows environment and run in the Windows Script Host environment. Ensure the check returns standard output as follows:

```
WScript.Echo "property_name=" &#38; var_for_value
```

- property_name* that represents the prerequisite property as written in the configuration file, for example, `env.tcrhome`.
- var_for_value*, that is, the VBScript variable for the actual value that the collector obtains for the prerequisite property.

To check whether the `TCR_HOME` environment exists and return the actual value, where the prerequisite property name is `env.tcrhome`:

```
set wshShell = WScript.CreateObject("WScript.Shell")
tcr_home=WshShell.ExpandEnvironmentStrings("%TCR_HOME%")
WScript.Echo "env.tcrhome=" &#38; tcr_home
```

To check whether the JRE is set in the `PATH` variable, where the prerequisite property name is `env.path.jre`:

```
Set wshShell = WScript.CreateObject("WScript.Shell")
path = WshShell.ExpandEnvironmentStrings("%PATH%")
Set objRegExp = new RegExp
objRegExp.Pattern = "(^|([:;\\\/]))(C:\Program Files\IBM\Java60\jre\bin)($|[:;])"
objRegExp.IgnoreCase = True
objRegExp.Global = True
Set matches = objRegExp.Execute(path)
WScript.Echo "env.path.jre=" &#38; (matches.Count > 0)
```

To check the version of Tivoli Directory Integrator installed, where the prerequisite property name is `installedSoftware.TDI.version`:

```
strComputer = "."
strKeyPath = "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"
regDisName = "DisplayName"
regDisVer = "DisplayVersion"

Set oReg = GetObject("winmgmts:{impersonationLevel=Impersonate}!\\" &#38; strComputer &#38; "\root\default:StdRegProv")

Set sftReg = new RegExp
sftReg.pattern = "Tivoli Directory Integrator"
sftReg.Global=False
oReg.EnumKey HKEY_LOCAL_MACHINE, strKeyPath, arrSubKeys
For Each subkey In arrSubKeys
    searchkey = strKeyPath & "&" &#38; subkey
    oReg.GetStringValue HKEY_LOCAL_MACHINE, searchkey, regDisName, strName
    oReg.GetStringValue HKEY_LOCAL_MACHINE, searchkey, regDisVer, strVersion
    If Not IsNull(strName) Then
        Set matches = sftReg.Execute(strName)
        If matches.Count > 0 Then
            Wscript.Echo "installedSoftware.TDI.version=" &#38; strVersion
        End If
    End If
Next
```

- Run the VBScript collector to ensure that there are no runtime errors and debug as necessary.
- Create a custom evaluator only if the standard compare functions cannot compare the actual and expected values.

Creating custom VBScript collectors specific to a product and product version

When you create custom product-specific VBScript collectors, the file name must be the same product code as the configuration file and stored in the/Windows subdirectory. The collector can contain code to gather actual values for one or many prerequisite properties. It can also use the common functions and sub routines to gather those values if required.

Before you begin

Ensure that you review the set of functions and sub routines in the following appendixes before you create the collectors. Determine whether you can use any of them to obtain the actual values:

- Appendix E, "Common functions for Windows systems," on page 115
- Appendix G, "File utility sub routines for Windows systems," on page 131
- Appendix F, "Logging utility sub routines for Windows systems," on page 129
- Appendix H, "Other common functions and sub routines for Windows systems," on page 133

Determine whether the collector must check that the prerequisite property exists and if it does, what other information must be gathered. Each check must return a value, whether one exists. For example:

- Check whether the directory exists.
- Check the available disk space for a directory.
- Check whether a product is installed.
- Check what version of the product is installed.

Procedure

1. Create a VBScript file. Save the file in the *ips_root/Windows* directory, with a variant of the following file naming convention:

product_code[_version].vbs

where:

- *product_code*

It is the variable to represent a product code on either Windows or UNIX systems. Product codes identify the product, an individual platform such as Windows, AIX, HP-UX, Linux, and Solaris, and optionally the version of the operating system that is supported by that product. They are stored in the *codename.cfg* file. Any product that supports multiple platforms has multiple product codes, with each one identifying a product, platform, and version of the operating system as required.

- *version* is the 8-digit code to represent the version, release, modification, and level, with two digits for each part of the code; for example, 7.3.21 is 07032100.

2. Using a VBScript editor, open the file and include the path to the *common_function.vbs* if you must use common functions, as follows:
`Include("../lib/common_function.vbs")`
3. If you must use the values of the *PATH* and *-p* flag passed from the Prerequisite Scanner, then use `Wscript.Arguments()` where `Wscript.Arguments(0)` is the value for *PATH*. `Wscript.Arguments(1)` is the *-p* flag and its values.

4. Add the code to obtain the value for the prerequisite property by using VBScript COM and functions to access elements of the Windows environment. Run in the Windows Script Host environment. Ensure the check returns standard output as follows:

```
WScript.Echo "property_name=" &#38; var_for_value
```

- *property_name* that represents the prerequisite property as written in the configuration file, for example, *env.tcrhome*.
- *var_for_value*, that is, the VBScript variable for the actual value that the collector obtains for the prerequisite property.

To check the available disk space for the installation directory for a product. For example, to check for Tivoli Monitoring for Energy Management Reporting and Optimization by using the “getValue()” on page 134 sub routine, where the prerequisite property is InstallDir:

```
Set wshShell = WScript.CreateObject("WScript.Shell")
'Check the disk space for the installation path that is passed as
'the value for the PATH argument
installPath = Wscript.Arguments(0)
sInstallPath= "InstallDir="
Wscript.Echo "installation path      : " &#38; installPath
set fso = CreateObject("Scripting.FileSystemObject")

getValue fso, sInstallPath, installPath

'Common sub routine
Sub getValue(fso, sKey, drvPath)
    Wscript.Echo "getValue(" &#38; skey &#38; ", " &#38; drvPath &#38; ")"
    If fso.driveExists(fso.getDriveName(drvPath)) then
        Set disk = fso.GetDrive(fso.getDriveName(drvPath))
        'Value returned is in bytes. Convert to MB
        cSize = CLng((disk.FreeSpace/1024)/1024) &#38; "MB"
        WScript.Echo sKey &#38; cSize
    Else
        Wscript.Echo " Disk for " &#38; sKey &#38; " -> " &#38; drvPath &#38; " does NOT exist"
    End If
End Sub
```

5. Run the VBScript collector to ensure that there are no runtime errors and debug as necessary.
6. Create a batch file to call the VBScript collector. The batch file must have the same name as the configuration file and a .bat extension, *product_code[_version].bat*, as follows:

```
@echo off

set CMD_LINE_ARGS=
:setArgs
if "%1"=="%" goto doneSetArgs
set CMD_LINE_ARGS=%CMD_LINE_ARGS% %1
shift
goto setArgs
:doneSetArgs

cscript.exe //nologo collector_file_name.vbs %CMD_LINE_ARGS%
```

7. Create a custom evaluator only if the standard compare functions cannot compare the actual and expected values.

Creating custom collectors for UNIX systems

You can create custom connectors if the basic set collectors do not collect values for the prerequisite properties required for the product to be installed. When you create custom collectors, the file name must be the same as the prerequisite property though without the subtype in its name. The collector is stored in the `/UNIX_Linux` subdirectory. The collector can contain code to obtain actual values for one or many prerequisite properties. It can also use the common functions to obtain those values if required.

Before you begin

Ensure that you review the set of functions in the following appendixes before you create the collectors. Determine whether you can use any of them to obtain the actual values:

- Appendix I, “Common functions for UNIX systems,” on page 137
- Appendix J, “Other functions for UNIX systems,” on page 143
- Appendix K, “Logging utility functions for UNIX systems,” on page 151

Determine whether the collector must check that the prerequisite property exists and if it does, what other information must be gathered. Each check must return a value, whether one exists. For example:

- Check whether a product is installed, for example, a package installed with RPM.
- Check what version of the product is installed.
- Check whether the available disk space for a mounted file system

If you want to use subtypes, *suffix_identifier*, and further categorize a prerequisite property by application, utility, or service subtype, you can create a common collector. Pass the differentiator for the *suffix_identifier* subtype, that is, *differentiator_suffix_identifier* to its collector. For example, `os.package` is the common collector to check for the existence of packages. To check for the existence of `openssh`, pass the name of the package when invoking the `os.package` collector in the `packageTest.sh` script file, as follows:

```
./os.package openssh
```

Where `openssh` is the name of the package, that is, the *suffix_identifier* subtype and the *differentiator_suffix_identifier* differentiator.

Procedure

1. Create a shell script file. Save the file in the `ips_root/Unix_Linux` directory, with a variant of the following file naming convention but without a file extension:
`[prefix_identifier.]property_name`
where:
 - *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories, for example, `env`.
 - *property_name* is the name of the prerequisite property, for example, `path.jre`
2. Using an editor, open the file and include the path to the `common_function.sh` if you must use common functions, as follows:

```
../lib/common_function.sh
```

3. Add the code to obtain the value for the prerequisite property by using commands and options specific to that platform to access elements of the host environment. For example, the custom `env.path.jar` prerequisite property needs to check whether the JRE is set in the `PATH` variable. The following code runs the `env` command, searches the output for `PATH` variable, and then searches its value for the JRE path.

```
envJRE=`env | grep "PATH" | grep -w "/opt/IBM/Java60/jre/bin"``
```

4. Ensure the check returns standard output:

```
echo "True"|"False" 'If the scan checks for the existence of the prerequisite
property
echo $res 'If the scan checks returns the value, for example, product version,
'of the prerequisite property
echo "Unavailable" 'If the scan returns no value for the prerequisite property
echo "Available" 'If the scan returns a valid check for the prerequisite property
```

In the example, based on the value of the `$envJRE` variable, the check either returns `True` or `False`:

```
if [ $envJRE ]; then
    echo "True"
else
    echo "False"
fi
```

5. Run the custom collector to ensure that there are no runtime errors and debug as necessary.
6. Edit the `ips_root/UNIX_Linux/packageTest.sh` script to call and run the custom collector.
7. Create a custom evaluator only if the custom collector returns values other than boolean values.

Editing the package test script for UNIX systems

You can update the `packageTest.sh` script file to call custom collectors on UNIX systems.

Before you begin

Ensure that you know the names of the collectors associated with predefined prerequisite properties, as outlined in Appendix D, “Predefined collectors for UNIX systems,” on page 109. If the prerequisite property is further categorized by application, utility, or service subtype, pass the differentiator for the *suffix_identifier* subtype, that is, *differentiator_suffix_identifier* to its collector.

For example, `os.package` is the common collector to check for the existence of packages. To check for the existence of `openssh`, pass the name of the package when invoking the `os.package` collector in the `packageTest.sh` script file, as follows:

```
./os.package openssh
```

Where `openssh` is the name of the package, that is, the *suffix_identifier* subtype and *differentiator_suffix_identifier* differentiator.

Procedure

1. Using an editor, open the `ips_root/UNIX_Linux/packageTest.sh` script.
2. Add the code to read the custom prerequisite property from the configuration file and parse its value.


```
res=`echo $line | grep [prefix_identifier.]property_name[.suffix_identifier]`
if [ $res ]; then
ExpValue=`echo $res | cut -d "=" -f2`
```

For example, to read the custom `env.path.jar` prerequisite property and check whether the JRE is set in the `PATH` variable:

```
res=`echo $line | grep env.path.jar`
if [ $res ]; then
ExpValue=`echo $res | cut -d "=" -f2`
```

In the example:

```
echo "\`wrlTrace "Starting" "env.path.jar"\`" >>/tmp/prs.check
echo "\`wrlTrace "Executing" "env.path.jar"\`" >>/tmp/prs.check
echo "\`wrlDebug "Starting" "env.path.jar"\`" >>/tmp/prs.check
echo "\`wrlDebug "Expected" "ExpValue" "\`" >>/tmp/prs.check
```

3. Call logging functions for trace and debug data before calling the custom collector.

```
echo "\`wrlTrace "Starting" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >>/tmp/prs.check
echo "\`wrlTrace "Executing" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >>/tmp/prs.check
echo "\`wrlDebug "Starting" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >>/tmp/prs.check
echo "\`wrlDebug "Expected" "ExpValue" "\`" >>/tmp/prs.check
```

4. Call the custom collector.

Note: If the custom collector has subtypes, that is, `[suffix_identifier]` in the file name and needs additional checks based on subtype, pass the `[differentiator_suffix_identifier]` differentiator for the subtype to the custom collector.

```
echo "ss=\`./[prefix_identifier.]property_name[.suffix_identifier]
[differentiator_suffix_identifier]"\" >>/tmp/prs.check
```

In the example:

```
echo "ss=\`./env.path.jar\" >>/tmp/prs.check
```

Note: Examples of differentiators for the `script_name` subtype for the `os.file.script_name` prerequisite properties are the paths to the scripts that are passed to the `os.filepath` collector:

```
echo "ss=\`./os.filepath /usr/bin/expect\" >>/tmp/prs.check #os.file.expect
echo "ss=\`./os.filepath /usr/bin/tar\" >>/tmp/prs.check #os.file.tar
echo "ss=\`./os.filepath /usr/bin/gzip\" >>/tmp/prs.check #os.file.gzip
```

5. Call the logging functions for trace and debug data upon exiting the custom collector.

```
echo "\`wrlTrace "Finished" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >/tmp/prs.check
echo "echo \"[prefix_identifier.]property_name
[.suffix_identifier]=\`ss\"\" >>/tmp/prs.check
echo "\`wrlDebug "Finished" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >>/tmp/prs.check
echo "\`wrlDebug "OutPutValueIs" \`ss\"\" >/tmp/prs.check
echo "\`wrlTrace "Done" "[prefix_identifier.]property_name
[.suffix_identifier]"\" >>/tmp/prs.check
fi
```

In the example:

```
echo "ss=\`./env.path.jar\" >>/tmp/prs.check
echo "\`wrlTrace "Finished" "env.path.jar\" >>/tmp/prs.check
echo "echo \"env.path.jar=\`ss\"\" >>/tmp/prs.check
```



```

echo "\`wrldDebug "Finished" "env.path.jar"\`" >>/tmp/prs.check
echo "\`wrldDebug "OutPutValueIs" \$ss\`" >>/tmp/prs.check
echo "\`wrldTrace "Done" "env.path.jar"\`" >>/tmp/prs.check
fi

```

6. Repeat steps 2 to 5 for each custom prerequisite property.

Creating custom evaluators for Windows systems

You can create VBScript evaluators if the basic evaluators do not compare the expected and actual values for the prerequisite properties by using the correct evaluation criteria. When you create custom evaluators, the file name must end with `_compare` and stored in the `/Windows` subdirectory. The custom evaluator can use the common functions and sub routines to compare the values if required.

Before you begin

Ensure that you review the set of functions and sub routines in the following appendixes before you create the evaluator. Determine whether you can use any of them to compare the values:

- Appendix E, “Common functions for Windows systems,” on page 115
- Appendix G, “File utility sub routines for Windows systems,” on page 131
- Appendix F, “Logging utility sub routines for Windows systems,” on page 129
- Appendix H, “Other common functions and sub routines for Windows systems,” on page 133

Note: The common function, “`passOrFail()`” on page 125, can compare the actual and expected values for the following data types: a generic number; size in MBs or GBs; processor speed in MHz or GHz; boolean value; or a string. Create only a custom evaluator if the `passOrFail` function cannot be used.

Procedure

1. Create a VBScript file. Save the file in the `ips_root/Windows` directory, with a variant of the following file naming convention:

```
[prefix_identifier.]property_name[.suffix_identifier]_compare.vbs
```

where:

- *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
 - *property_name* is the name of the prerequisite property.
 - *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
2. Add the code to compare the actual and expected values that are passed to the evaluator as arguments by using VBScript COM and associated functions. Ensure the comparison returns standard output as follows:
 - "PASS" when the expected value for the prerequisite property is equal to or greater than the actual value for the prerequisite property
 - "FAIL" when the expected value for the prerequisite property does not equal the actual value for the prerequisite property
 3. Run the custom evaluator to ensure that there are no runtime errors and debug as necessary.

Example

This custom evaluator checks the actual and expected values for the version of Tivoli Directory Integrator. It uses the common function, "versionCompare()" on page 135.

```
wscript.echo "expect: " &#38; wscript.arguments(0)
wscript.echo "real value: " &#38; wscript.arguments(1)
wscript.echo tdiVersionCompare(wscript.arguments(0), wscript.arguments(1))

function tdiVersionCompare(expect, real)
    if len(real) = 0 then
        tdiVersionCompare = "FAIL"
        exit function
    end if

    expect = Trim(expect)
    real = Trim(real)

    Dim expectedVersion
    'if (StrComp(Right(expect,1),"+")=0 or StrComp(Right(expect,1),"-")=0) Then
    if (Right(expect,1)="/" or Right(expect,1)="-") Then
        expectedVersion = Left(expect,len(expect)-1)
    else
        expectedVersion = expect
    end if

    Dim cmp
    cmp = versionCompare(expectedVersion,real)

    if (StrComp(Right(expect,1),"+")=0) Then
        ' Version must be at least expected value
        if (cmp=0 or cmp=-1) Then
            tdiVersionCompare = "PASS"
        else
            tdiVersionCompare = "FAIL"
        end if
    elseif (StrComp(Right(expect,1),"-")=0) Then
        ' Version must be less than or equal to expected value
        if (cmp=0 or cmp=1) Then
            tdiVersionCompare = "PASS"
        else
            tdiVersionCompare = "FAIL"
        end if
    elseif cmp=0 then
        tdiVersionCompare = "PASS"
    else
        tdiVersionCompare = "FAIL"
    end if
end function

' Generic function for comparing 2 version strings
'
' Parameters
'     ver1 The first version string
'     ver2 The second version string
'
' ver1 and ver2 are expected to be dot-separated version strings
' (e.g. 1.0.0.4, 2.3, 3.40.26.7800, 2.3.a) Version strings can have any
' number of parts. When comparing versions with different numbers of
' parts, missing parts of the shorter version string will be treated
' as if there was a zero there. If any non-numeric characters are
' included in a version part, those corresponding parts will be compared
' as strings and not parsed into numeric form
'
' Returns
'     1 version1 > version2
```

```

'      -1 version1 < version2
'      0 version1 = version2
'
' Special cases:
' RESULT   version 1   version 2
' 0         empty      empty
' 1         validString empty
' -1        empty      validString
'
' NOTE: This function should eventually move to common_functions.vbs

function versionCompare(ver1, ver2)
    WScript.echo "Comparing [" & ver1 & "]" to [" & ver2 & "]"

    Const UNASSIGNED = "*UNASSIGNED*"
    Dim v1Default, v2Default

    ' Handle special cases:
    if (IsEmpty(ver1) and IsEmpty(ver2)) Then
        versionCompare = 0
        exit function
    end if
    if (IsEmpty(ver1) and not IsEmpty(ver2)) Then
        versionCompare = -1
        exit function
    end if
    if (not IsEmpty(ver1) and IsEmpty(ver2)) Then
        versionCompare = 1
        exit function
    end if

    Dim ver1Parts, ver2Parts

    ' Versions are not empty. Break into parts and compare numbers
    ver1Parts = Split(ver1, ".")
    ver2Parts = Split(ver2, ".")

    Dim v1Size, v2Size
    v1Size = ubound(ver1Parts)
    v2Size = ubound(ver2Parts)

    ' If last version part is "*", treat all missing parts as "*"
    '(so 2.* matches 2.1.3, for example)
    if (v1Size > v2Size) Then
        Redim Preserve ver2Parts(v1Size)
        if (ver2Parts(v2Size) = "*") Then
            for i = v2Size to v1Size
                ver2Parts(i) = "*"
            next
        end if
    elseif (v2Size > v1Size) Then
        Redim Preserve ver1Parts(v2Size)
        if (ver1Parts(v1Size) = "*") Then
            for i = v1Size to v2Size
                ver1Parts(i) = "*"
            next
        end if
    end if

    Dim i
    i = 0

    Do While (i <= ubound(ver1Parts) or i <= ubound(ver2Parts))
        Dim v1, v2, v1Str, v2Str

        v1Str = UNASSIGNED
        v2Str = UNASSIGNED
    
```

```

if (i<ubound(ver1Parts)) Then
    on error resume next
    v1 = Int(ver1Parts(i))
    if not Err=0 Then
        v1Str = ver1Parts(i)
        if (i<ubound(ver2Parts)) Then
            v2Str = ver2Parts(i)
        else
            v2Str = "0"
        end if
    end if
else
    v1 = 0
end if

if (i<ubound(ver2Parts)) Then
    on error resume next
    v2 = Int(ver2Parts(i))
    if not Err=0 Then
        if (i<ubound(ver1Parts)) Then
            v1Str = ver1Parts(i)
        else
            v1Str = "0"
        end if
        v2Str = ver2Parts(i)
    end if
else
    v2 = 0
end if

if (not v1Str=UNASSIGNED or not v2Str=UNASSIGNED) Then
    if (IsEmpty(v1Str)) Then
        v1Str = "0"
    end if
    if (IsEmpty(v2Str)) Then
        v2Str = "0"
    end if
    End if

    'WScript.echo "Comparing as strings: " &#38; v1Str &#38; " : " &#38; v2Str
    ' Compare as Strings if either part could not be converted to a number
    if (not v1Str="" and not v2Str="") Then
        if (not v1Str=v2Str) Then
            versionCompare = StrComp(v1Str,v2Str)
            exit function
        end if
    end if
else
    'WScript.echo "Comparing as numbers: " &#38; v1 &#38; " : " &#38; v2

    if (v1 > v2) Then
        versionCompare = 1
        exit function
    end if
    if (v2 > v1) Then
        versionCompare = -1
        exit function
    end if
end if

i = i + 1
Loop

' If we got here, versions must be equal
versionCompare = 0

end function

```

Creating custom evaluators for UNIX systems

You can create custom evaluators if the custom collector does not return boolean values, that is, True or False. When you create custom evaluators, the file name must end with `_compare` and stored in the `/UNIX_Linux` subdirectory. The custom evaluator can use the common functions to compare the values if required.

Before you begin

Ensure that you review the set of functions in the following appendixes before you create the custom evaluators. Determine whether you can use any of them to compare the actual and expected values:

- Appendix I, “Common functions for UNIX systems,” on page 137
- Appendix J, “Other functions for UNIX systems,” on page 143
- Appendix K, “Logging utility functions for UNIX systems,” on page 151

There are two script files that you can use as a starting point, that is, `._compare.sh` and `_compare.sh` in the `/Unix_Linux` subdirectory.

Important: Do not create custom evaluators if your custom collectors return True or False. IBM Prerequisite Scanner uses predefined evaluators for any collector that returns boolean values.

Procedure

1. Create a shell file. Save the file in the `ips_root/UNIX_Linux` directory, with a variant of the following file naming convention:
`[prefix_identifier.]property_name[.suffix_identifier]_compare.sh`
where:
 - *prefix_identifier* is an identifier for a predefined category of prerequisite properties as outlined in Table 3 on page 4. This prefix identifier is required by some of the predefined categories.
 - *property_name* is the name of the prerequisite property.
 - *suffix_identifier* is an optional identifier for a subtype of prerequisite properties as outlined in Table 4 on page 6.
2. Add the code to compare the actual and expected values that are passed to the evaluator as arguments and associated functions. Ensure the comparison returns standard output as follows:
 - "PASS" when the expected value for the prerequisite property is equal to or greater than the actual value for the prerequisite property
 - "FAIL" when the expected value for the prerequisite property does not equal the actual value for the prerequisite property
3. Run the custom evaluator to ensure that there are no runtime errors and debug as necessary.

Chapter 4. Running Prerequisite Scanner

You can use a command-line interface to run the IBM Prerequisite Scanner. The Prerequisite Scanner script, `prereq_checker`, takes a set of required and optional parameters and a command flag for additional optional parameters.

Table 13 explains the special characters that are used in the syntax of the Prerequisite Scanner script.

Table 13. Special characters legend for the Prerequisite Scanner script

Special character	Description
<>	Identifies a placeholder name.
[]	Identifies an optional parameter. Parameters not enclosed in brackets are required.
...	Indicates that you can specify multiple values for a parameter.
	Indicates mutually exclusive parameters. Specify either the parameter to the left of the separator or the parameter to the right of the separator, but not both.
{ }	Encloses a set of mutually exclusive parameter separated by .

`prereq_checker`

The `prereq_checker` script runs the IBM Prerequisite Scanner and checks for prerequisites based on the set of parameters that you specify when you run the script.

Syntax

```
prereq_checker.bat|sh
  "Product_Code [Product_Version][,Product_CodeN [Product_VerN]]..."
  [detail]
  [outputDir="ips_output_dir"]
  [xmlResult]
  [PATH="product_root"]
  [-p Product_Code.instance.parameter=value,...]
  [debug]
  [trace]
```

The `prereq_checker` script has one required parameter and several optional parameters.

""Product_Code [Product_Version][,Product_CodeN [Product_VerN]]..." on page 62

Required parameter

"[detail]" on page 62

Optional parameter

"[outputDir="ips_output_dir]" on page 65

Optional parameter

"[xmlResult]" on page 65

Optional parameter

"[PATH="product_root]" on page 65

"[-p **Product_Code**.instance.parameter=value,...]" on page 66
Optional flag

"[debug]" on page 66
Optional parameter

"[trace]" on page 66
Optional parameter

"*Product_Code* [*Product_Version*][,*Product_CodeN*
[*Product_VerN*]..."

You must set at least one **Product_Code** parameter to identify the product or component for which to run the prerequisite check and the associated configuration file. **Product_Code** is the product code that you set in the *ips_root/codename.cfg* file.

For example, KMS is the product code for the Tivoli Enterprise Monitoring Server in the *product.cfg* file. To run the Scanner, enter the following script with the product code:

```
./prereq_checker.sh KMS
```

If you set a **Product_Code** parameter that does not have a corresponding configuration file, the Prerequisite Scanner ignores it without error. The log file contains a message that no configuration file was found.

The **Product_Version** parameter for the associated **Product_Code** parameter indicates the version of the product. It is the 8-digit code to represent the version, release, modification, and level, with two digits for each part of the code; for example, 7.3.21 is 07032100. **Product_Version** is an optional parameter. If you do not set it, the Prerequisite Scanner checks the latest available version.

You can set one or many **Product_Code** parameters with the optional **Product_Version** parameter, each separated by a comma.

Important: When you set more than one **<Product_Code>** parameter with the optional **<Product_Version>** parameter, enclose the parameters in quotation marks. If you do not, the Scanner fails.

This example checks prerequisites for the latest version of Tivoli Monitoring Operating System Agent for Windows and Version 6.2.1 of Tivoli Monitoring Agent for DB2.

```
prereq_checker.bat "KNT,KUD 06210000"
```

[detail]

This optional parameter indicates whether to display detailed results of the scan in the command-line interface.

Important: Do not enclose this parameter in quotation marks.

When you set the **detail** parameter, the detailed results contain:

- The version of the Prerequisite Scanner
- The version of the operating system on which the Scanner was run
- The name of the products or components for which the prerequisites checks were run

- For each prerequisite property: the name of the prerequisite property checked, the PASS or FAIL result, the actual value, and expected value
- For all components: the name of the general prerequisite property checked, the PASS or FAIL result, the actual value, and expected value
- The overall PASS or FAIL result

Prerequisite Scanner also saves these results to the *ips_output_dir/result.txt* file. It saves the results to the text file regardless of whether you set the **detail** parameter.

```

root@aclinux15:~/prs/20110927-0849
File Edit View Terminal Tabs Help
[root@aclinux15 20110927-0849]# ./prereq_checker.sh DMO detail
IBM Prerequisite Scanner
  Version: 1.1.1.8
  Build : 20110927
  OS Name: Linux

Machine Info
Machine Name : <Machine name>
Serial Number: <Serial number>

TPS detected : Red Hat Enterprise Linux Server release 5.5 {32-bit}
Using the DMO config file
Using config file - /root/prs/20110927-0849/UNIX_Linux/DMO_0750000.cfg for DMO
DMO - Prerequisite Scanner Demo [0750000]:
Evaluation                                PASS/FAIL Result                                Expected Result

DBType                                    FAIL      Unknown                                Oracle
DBType                                    FAIL      Unknown                                DB2
DBType                                    FAIL      Unknown                                regex{.*Oracle.*}
DBType                                    FAIL      Unknown                                regex{.*DB2.*}
DBTypeDetails                            FAIL      Unknown                                oracle
DBTypeDetails                            FAIL      Unknown                                DB2
DBTypeDetails                            FAIL      Unknown                                regex{.*Oracle.*}
DBTypeDetails                            FAIL      Unknown                                regex{.*DB2.*}
OS Version                                PASS      "Red Hat Enterprise Linux Server release 5.5 (Tikanga)"
" regex{Red Hat.*Tikanga.*}"
                                           regex{AIX.*}
                                           regex{Solaris.*}
}
os.lib.libstdc++                          PASS      /usr/lib/gcc/i386-redhat-linux/4.1.1/libstdc++.so libstdc++
os.lib.libgcc                             PASS      /usr/lib/gcc/i386-redhat-linux/3.4.6/libgcc_s.so [Check
Package:True]regex{libgcc.*}
os.lib.libXp                              PASS      /usr/lib/libXmu.so.6                                regex{libX.*}
os.space.var                              PASS      "38GB"
r/libm/common/acsi"
                                           " [dir:root=/va
unit:MB]1.0
os.space.usr                              PASS      "38GB"
r/libm/common/acsi"
                                           " [dir:root=/us
unit:MB]200
os.space.tmp                              PASS      36GB
env.classpath.derbyJAR                    PASS      False
network.pingSelf                          PASS      True
env.classpath.derbyJAR                    PASS      False
network.pingLocalhost                     PASS      True
os.package.compat-libstdc++-33            PASS      compat-libstdc++-33-3.2.3-61                        compat-libstdc++-33
TOTAL ALL SPECIFIED COMPONENTS:
Evaluation                                PASS/FAIL Result                                Expected Result
/                                           PASS      38.00GB
                                           201MB
/tmp                                        PASS      36.00GB
                                           30MB

Prereq Check Overall Result: FAIL
[root@aclinux15 20110927-0849]#

```

Figure 10. Running the script and setting the detail parameter on UNIX systems

If you do not set the **detail** parameter, the Scanner displays only the PASS or FAIL result in the command-line interface.

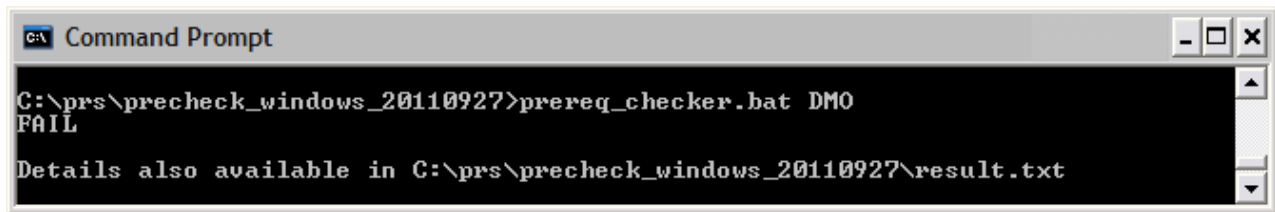


Figure 11. Running the script without setting the detail parameter on Windows systems

[outputDir="ips_output_dir"]

This optional parameter indicates that you want to set the output directory for the scan results and log files for Prerequisite Scanner.

When you run the Prerequisite Scanner script and set the optional **outputDir** parameter, Prerequisite Scanner outputs the results text, XML, and log files to the directory specified by the parameter's value. This value is known as *ips_output_dir* throughout the documentation.

If you do not set this parameter, the default output location is *ips_root*.

You must use the parameter to specify a location, if you choose to run Prerequisite Scanner from a CD, DVD, or read-only network drive. You must have write permissions to write to *ips_output_dir*; otherwise, Prerequisite Scanner fails.

Important: If the output directory does not exist, Prerequisite Scanner creates the directory. You must have write permissions to create or write to the output directory in which Prerequisite Scanner saves the files.

[xmlResult]

This optional parameter indicates that you want to output the results to the XML result file in addition to the plain test result file.

When you run the Prerequisite Scanner script and set the optional **xmlResult** parameter, Prerequisite Scanner outputs the results to the *ips_output_dir/result.xml* file.

If you do not set this parameter, the results are output to the plain text file only.

[PATH="product_root"]

This optional parameter indicates the installation directories for the products.

Important: On Windows, do not set the path to a drive letter only, that is, C:. Ensure that you set a valid path.

If you do not set the **path** parameter, the Scanner checks the default installation directories for IBM Tivoli products:

- On **UNIX systems:** /opt/ibm/itm
- On **Windows systems:** C:\IBM\itm

`[-p Product_Code.instance.parameter=value,...]`

The optional **-p** flag indicates that the proceeding parameters must be passed to a script file for additional prerequisite checking. **<Product_Code>** is the product code. Only each set of *instance.parameter=value* is passed to the script. You can pass multiple sets of parameters, separated by a comma.

The script to which the parameters are passed is determined by the following options:

- With a **Product_Code** prefix, the parameters are passed to the script with the associated **Product_Code**
- Without the **Product_Code** prefix, the parameters are passed to the common collectors.

Example 1

```
-p KUD.inst1.DB2_INST_OWNER=db2inst1, KUD.inst2.DB2_INST_OWNER=db2inst2
```

This flag with parameters passes db2inst1.DB2_INST_OWNER=db2inst1 and db2inst2.DB2_INST_OWNER=db2inst2 to the KUD.**Product_Version**.bat script file.

Example 2

```
-p SERVER=IP.PIPE://mymachine:1918
```

This flag with parameters passes SERVER=IP.PIPE://mymachine:1918 to the common collector to check the ports.

Note: This script accepts the parameters in **-p** as tacmd createNode. You can set the SERVER, PROTOCOL, PORT, BACKUP, and BSERVER parameters in *ips_root/lib/common_configuration*. Prerequisite Scanner prioritizes the parameters passed from the command-line interface above those parameters set in the *common_configuration* file.

[debug]

This optional parameter indicates that you want to turn on debugging while running the Prerequisite Scanner.

When you run the Prerequisite Scanner script and set the optional **debug** parameter, Prerequisite Scanner outputs detailed processing information, warning and error messages, and the scan results in the log file. It is the *ips_output_dir/prs.debug* file on UNIX systems and the *ips_output_dir/precheck.log* file on Windows systems.

Important: Debugging the Scanner is turned off by default.

[trace]

(UNIX systems only) This optional parameter indicates that you want to turn on trace logging while running the Prerequisite Scanner.

When you run the Prerequisite Scanner script and set the optional **trace** parameter, Prerequisite Scanner outputs trace information in the *ips_output_dir/prs.trc* file.

Important: Trace logging for the Scanner is turned off by default.

Running Prerequisite Scanner from the command line

You can run IBM Prerequisite Scanner from the command-line interface and enter the relevant input parameters for the script.

Before you begin

Ensure that you check your product's installation documentation or Technotes for any additional steps that must be performed before running Prerequisite Scanner. For example, you might need to set the environment variable that indicates to Prerequisite Scanner which components or features are being installed on the target computer and consequently, which prerequisites to check.

Procedure

1. Open the command-line interface and open the *ips_root* directory.
2. Run the Prerequisite Scanner script file, **prereq_checker**, as follows:

UNIX

```
./prereq_checker.sh
"Product_Code [Product_Version][,Product_CodeN [Product_VerN]]..."
[detail]
[outputDir="ips_output_dir"]
[xmlResult]
[PATH="product_root"]
[-p Product_Code.instance.parameter=value,...]
```

The following example runs Prerequisite Scanner for Autonomic Deployment Engine using a configuration file and its associated product code, ADE:

```
./prereq_checker.sh
ADE 072000
detail
PATH=/opt/ibm/tivoli
```

Windows

```
prereq_checker.bat
"Product_Code [Product_Version][,Product_CodeN [Product_VerN]]..."
[detail]
[outputDir="ips_output_dir"]
[xmlResult]
[PATH="product_root"]
[-p Product_Code.instance.parameter=value,...]
```

The following example runs Prerequisite Scanner for Tivoli Provisioning Manager for Windows 2003 and 2008 using product codes, COX and COY.

```
prereq_checker.bat
"COX, COY 07200000"
detail
PATH="D:\ibm\tivoli"
-p SERVER=IP.PIPE://mytems:1234
```

The following example runs Prerequisite Scanner for Tivoli zEnterprise™ Monitoring Agent using product code KZE. It also sets the location of the results and log files to *ips_output_dir* by using the optional **outputDir** parameter.

Important: You must use the **outputDir** parameter to specify a location, if you choose to run Prerequisite Scanner from a CD, DVD, or read-only network drive. You must have write permissions to write to *ips_output_dir*; otherwise, Prerequisite Scanner fails.

```
Windows  
prereq_checker.bat  
"KZE 06230000"  
outputDir="%TEMP%\ips"  
UNIX  
./prereq_checker.sh  
"KZE 06230000"  
outputDir="/tmp/ips"
```

The Scanner outputs the `result.txt` file and `precheck.log` files to the following locations:

- On Windows systems: `D:\temp\ips` where `TEMP` is environment variable for the temporary folder.
- On UNIX systems: `/tmp/ips`

Important: If the output directory does not exist, Prerequisite Scanner creates the directory. You must have write permissions to create or write to the output directory in which Prerequisite Scanner saves the files.

Common directory locations

There are path name variables for common directories.

IBM Prerequisite Scanner installation directory

ips_root describes the location where Prerequisite Scanner is installed. This location can be specified during installation.

Prerequisite Scanner output directory

ips_output_dir describes the location where the scan results and log files for Prerequisite Scanner are saved. This location can be specified by using the **outputDir** input parameter when you run the Scanner. If you do not set this parameter, the default output location is *ips_root*.

Note: Prerequisite Scanner creates temporary files during its execution, but these files are deleted before the Scanner completes its execution. These temporary files are located in the *ips_output_dir/temp* subdirectory. The Scanner also deletes the *ips_output_dir/temp* subdirectory, unless the subdirectory contains the debug and trace files that are generated on UNIX systems only.

Chapter 5. Troubleshooting Prerequisite Scanner

You can troubleshoot issues in IBM Prerequisite Scanner by using log files and logging functions when you create custom prerequisite checks.

Prerequisite Scanner generates return codes dependent on the results of the scan and whether it must exit because of errors. These return codes are written to the log files. For example, if the Prerequisite Scanner fails to run the scan because it cannot read the configuration file, it generates return code of 2.

Troubleshooting on Windows systems

When you run IBM Prerequisite Scanner, it creates a log file by default. It contains detailed information with each step and function that the Scanner performs in sequence. The file also contains timestamps, including start and end times of each function and step. You can debug and review the log file to determine where and when the error occurred.

Prerequisite Scanner outputs processing information, warning and error messages, and the scan results in the *ips_output_dir/precheck.log* file. When you run the Prerequisite Scanner script and set the optional **debug** parameter, Prerequisite Scanner outputs additional debugging messages in this file.

Figure 12 on page 70 shows an example of the log file when the optional **debug** parameter is set and Figure 13 on page 71 shows the log file when the parameter is not set.

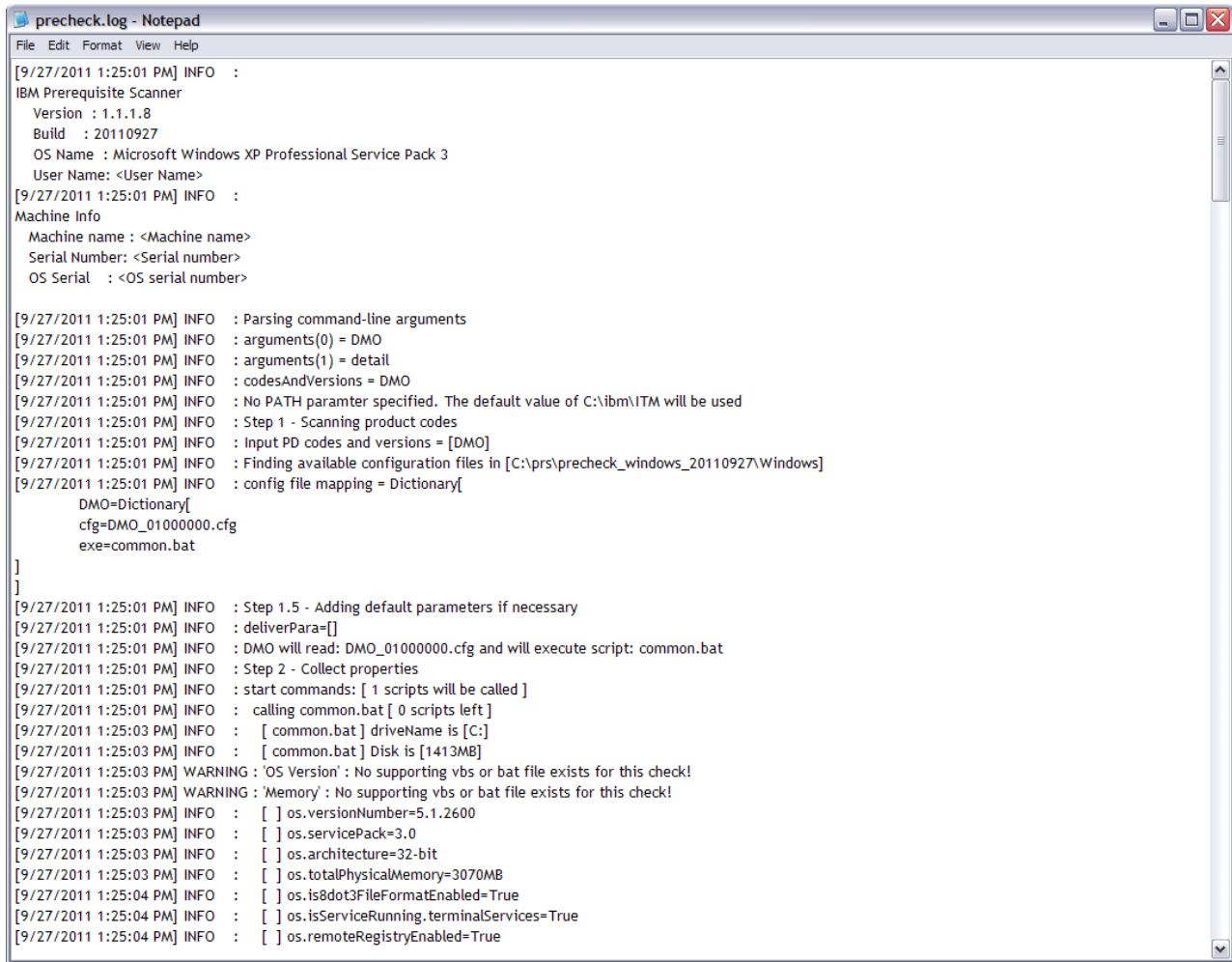
```
precheck.log - Notepad
File Edit Format View Help

[9/27/2011 1:27:34 PM] INFO :
IBM Prerequisite Scanner
  Version : 1.1.1.8
  Build : 20110927
  OS Name : Microsoft Windows XP Professional Service Pack 3
  User Name: <User Name>
[9/27/2011 1:27:34 PM] INFO :
Machine Info
  Machine name : <Machine name>
  Serial Number: <Serial number>
  OS Serial : <OS serial number>

[9/27/2011 1:27:34 PM] INFO : Parsing command-line arguments
[9/27/2011 1:27:34 PM] INFO : arguments(0) = DMO
[9/27/2011 1:27:34 PM] INFO : arguments(1) = detail
[9/27/2011 1:27:34 PM] INFO : arguments(2) = debug
[9/27/2011 1:27:34 PM] INFO : codesAndVersions = DMO
[9/27/2011 1:27:34 PM] INFO : No PATH paramter specified. The default value of C:\ibm\ITM will be used
[9/27/2011 1:27:34 PM] DEBUG : Detected operating system types are [Windows|Windows Workstation|Windows XP]
[9/27/2011 1:27:34 PM] DEBUG : Detected operating system version [5.1.2600]
[9/27/2011 1:27:34 PM] DEBUG : Detected service pack level [3.0]
[9/27/2011 1:27:34 PM] INFO : Step 1 - Scanning product codes
[9/27/2011 1:27:34 PM] INFO : Input PD codes and versions = [DMO]
[9/27/2011 1:27:34 PM] INFO : Finding available configuration files in [C:\prs\precheck_windows_20110927\Windows]
[9/27/2011 1:27:34 PM] INFO : config file mapping = Dictionary[
  DMO=Dictionary[
    cfg=DMO_01000000.cfg
    exe=common.bat
  ]
]

[9/27/2011 1:27:34 PM] INFO : Step 1.5 - Adding default parameters if necessary
[9/27/2011 1:27:34 PM] INFO : deliverPara=[]
[9/27/2011 1:27:34 PM] INFO : DMO will read: DMO_01000000.cfg and will execute script: common.bat
[9/27/2011 1:27:34 PM] INFO : Step 2 - Collect properties
[9/27/2011 1:27:34 PM] INFO : start commands: [ 1 scripts will be called ]
[9/27/2011 1:27:34 PM] DEBUG : The config file is: C:\prs\precheck_windows_20110927\Windows\DMO_01000000.cfg
[9/27/2011 1:27:34 PM] INFO : calling common.bat [ 0 scripts left ]
[9/27/2011 1:27:36 PM] INFO : [ common.bat ] driveName is [C:]
[9/27/2011 1:27:36 PM] INFO : [ common.bat ] Disk is [1413MB]
[9/27/2011 1:27:36 PM] DEBUG : Processing this line from cfg file: [OS Version=regex{Windows .*}]
[9/27/2011 1:27:36 PM] DEBUG : Take the first part of the line: [OS Version]
[9/27/2011 1:27:36 PM] DEBUG : See if a corresponding vbs or bat file exists for [OS Version]
[9/27/2011 1:27:36 PM] DEBUG : See if a corresponding vbs or bat file exists for [OS Versio]
```

Figure 12. precheck.log file with the debug data



```
[9/27/2011 1:25:01 PM] INFO :
IBM Prerequisite Scanner
Version : 1.1.1.8
Build : 20110927
OS Name : Microsoft Windows XP Professional Service Pack 3
User Name: <User Name>
[9/27/2011 1:25:01 PM] INFO :
Machine Info
Machine name : <Machine name>
Serial Number: <Serial number>
OS Serial : <OS serial number>

[9/27/2011 1:25:01 PM] INFO : Parsing command-line arguments
[9/27/2011 1:25:01 PM] INFO : arguments(0) = DMO
[9/27/2011 1:25:01 PM] INFO : arguments(1) = detail
[9/27/2011 1:25:01 PM] INFO : codesAndVersions = DMO
[9/27/2011 1:25:01 PM] INFO : No PATH paramter specified. The default value of C:\ibm\ITM will be used
[9/27/2011 1:25:01 PM] INFO : Step 1 - Scanning product codes
[9/27/2011 1:25:01 PM] INFO : Input PD codes and versions = [DMO]
[9/27/2011 1:25:01 PM] INFO : Finding available configuration files in [C:\prs\precheck_windows_20110927\Windows]
[9/27/2011 1:25:01 PM] INFO : config file mapping = Dictionary[
    DMO=Dictionary[
        cfg=DMO_01000000.cfg
        exe=common.bat
    ]
]
[9/27/2011 1:25:01 PM] INFO : Step 1.5 - Adding default parameters if necessary
[9/27/2011 1:25:01 PM] INFO : deliverPara=[]
[9/27/2011 1:25:01 PM] INFO : DMO will read: DMO_01000000.cfg and will execute script: common.bat
[9/27/2011 1:25:01 PM] INFO : Step 2 - Collect properties
[9/27/2011 1:25:01 PM] INFO : start commands: [ 1 scripts will be called ]
[9/27/2011 1:25:01 PM] INFO : calling common.bat [ 0 scripts left ]
[9/27/2011 1:25:03 PM] INFO : [ common.bat ] driveName is [C:]
[9/27/2011 1:25:03 PM] INFO : [ common.bat ] Disk is [1413MB]
[9/27/2011 1:25:03 PM] WARNING : 'OS Version': No supporting vbs or bat file exists for this check!
[9/27/2011 1:25:03 PM] WARNING : 'Memory': No supporting vbs or bat file exists for this check!
[9/27/2011 1:25:03 PM] INFO : [ ] os.versionNumber=5.1.2600
[9/27/2011 1:25:03 PM] INFO : [ ] os.servicePack=3.0
[9/27/2011 1:25:03 PM] INFO : [ ] os.architecture=32-bit
[9/27/2011 1:25:03 PM] INFO : [ ] os.totalPhysicalMemory=3070MB
[9/27/2011 1:25:04 PM] INFO : [ ] os.is8dot3FileFormatEnabled=True
[9/27/2011 1:25:04 PM] INFO : [ ] os.isServiceRunning_terminalServices=True
[9/27/2011 1:25:04 PM] INFO : [ ] os.remoteRegistryEnabled=True
```

Figure 13. *precheck.log* file without debug data

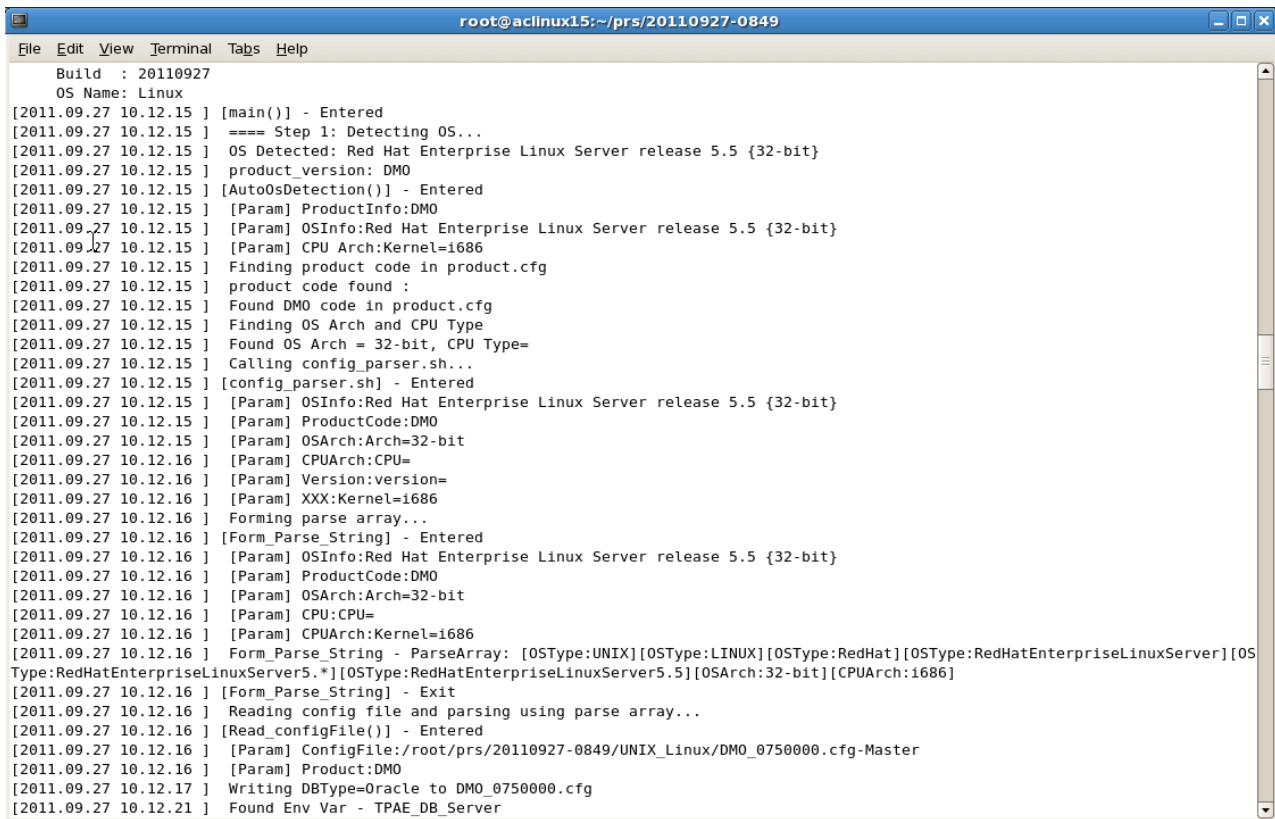
Troubleshooting on UNIX systems

Writing messages to log files is disabled by default on UNIX systems. You can enable debugging or tracing functions by using the **debug** and **trace** input parameters. The Scanner writes the debug and trace data to different log files and uses timestamps to flag the start and end times of steps or functions. You can use both files to correlate and troubleshoot a specific issue, function, or prerequisite check.

Debugging log file

When you run the Prerequisite Scanner script and set the optional **debug** parameter, Prerequisite Scanner outputs detailed processing information, warning and error messages, and the scan results in the *ips_output_dir/temp/prs.debug* file. It contains detailed information with each step and function that the Scanner performs in sequence. The file also contains timestamps, including start and end times of each function and step. The *ips_output_dir/temp* subdirectory also contains the interim *result1.txt* and *result2.txt* files that provide the input to the final *ips_output_dir/result.txt* file. You can use these interim files to determine issues

with results for specific prerequisite checks.



```
Build : 20110927
OS Name: Linux
[2011.09.27 10.12.15 ] [main()] - Entered
[2011.09.27 10.12.15 ] ==== Step 1: Detecting OS...
[2011.09.27 10.12.15 ] OS Detected: Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] product_version: DMO
[2011.09.27 10.12.15 ] [AutoOsDetection()] - Entered
[2011.09.27 10.12.15 ] [Param] ProductInfo:DMO
[2011.09.27 10.12.15 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] [Param] CPU Arch:Kernel=i686
[2011.09.27 10.12.15 ] Finding product code in product.cfg
[2011.09.27 10.12.15 ] product code found :
[2011.09.27 10.12.15 ] Found DMO code in product.cfg
[2011.09.27 10.12.15 ] Finding OS Arch and CPU Type
[2011.09.27 10.12.15 ] Found OS Arch = 32-bit, CPU Type=
[2011.09.27 10.12.15 ] Calling config_parser.sh...
[2011.09.27 10.12.15 ] [config_parser.sh] - Entered
[2011.09.27 10.12.15 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.15 ] [Param] ProductCode:DMO
[2011.09.27 10.12.15 ] [Param] OSArch:Arch=32-bit
[2011.09.27 10.12.16 ] [Param] CPUArch:CPU=
[2011.09.27 10.12.16 ] [Param] Version:version=
[2011.09.27 10.12.16 ] [Param] XXX:Kernel=i686
[2011.09.27 10.12.16 ] Forming parse array...
[2011.09.27 10.12.16 ] [Form_Parse_String] - Entered
[2011.09.27 10.12.16 ] [Param] OSInfo:Red Hat Enterprise Linux Server release 5.5 {32-bit}
[2011.09.27 10.12.16 ] [Param] ProductCode:DMO
[2011.09.27 10.12.16 ] [Param] OSArch:Arch=32-bit
[2011.09.27 10.12.16 ] [Param] CPU:CPU=
[2011.09.27 10.12.16 ] [Param] CPUArch:Kernel=i686
[2011.09.27 10.12.16 ] Form_Parse_String - ParseArray: [OSType:UNIX][OSType:Linux][OSType:RedHat][OSType:RedHatEnterpriseLinuxServer][OS
Type:RedHatEnterpriseLinuxServer5.*][OSType:RedHatEnterpriseLinuxServer5.5][OSArch:32-bit][CPUArch:i686]
[2011.09.27 10.12.16 ] [Form_Parse_String] - Exit
[2011.09.27 10.12.16 ] Reading config file and parsing using parse array...
[2011.09.27 10.12.16 ] [Read_configFile()] - Entered
[2011.09.27 10.12.16 ] [Param] ConfigFile:/root/prs/20110927-0849/UNIX_Linux/DMO_0750000.cfg-Master
[2011.09.27 10.12.16 ] [Param] Product:DMO
[2011.09.27 10.12.17 ] Writing DBType=Oracle to DMO_0750000.cfg
[2011.09.27 10.12.21 ] Found Env Var - TPAE_DB_Server
```

Figure 14. prs.debug file on UNIX systems

Trace log file

When you run the Prerequisite Scanner script and set the optional **trace** parameter, Prerequisite Scanner outputs trace information in the *ips_output_dir/temp/prs.trc* file. It contains information with each function that the Scanner performs in sequence. The file also contains timestamps, including start and end times of each function.

```

root@acliunix15:~/prs/20110927-0849
File Edit View Terminal Tabs Help
Build : 20110927
OS Name: Linux
[2011.09.27 10.19.58 ] [main()] - Entered:
[2011.09.27 10.19.58 ] [AutoOsDetection()] - Entered:
[2011.09.27 10.19.58 ] [config_parser.sh] - Entered:
[2011.09.27 10.19.59 ] [Form_Parse_String] - Entered:
[2011.09.27 10.19.59 ] [Form_Parse_String] - Exit:
[2011.09.27 10.19.59 ] [Read_configFile()] - Entered:
[2011.09.27 10.20.05 ] [Read_configFile()] - Exit:
[2011.09.27 10.20.05 ] [config_parser.sh] - Exit:
[2011.09.27 10.20.05 ] [AutoOsDetection()] - Exit:
[2011.09.27 10.20.05 ] [packageTest.sh] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Exit:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.25 ] [NFScheck()] - Exit:
[2011.09.27 10.20.25 ] [NFScheck()] - Entered:
[2011.09.27 10.20.26 ] [NFScheck()] - Exit:
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DB2_Version
[2011.09.27 10.20.26 ] Executing: DB2_Version.sh
[2011.09.27 10.20.26 ] Finished: DB2_Version.sh
[2011.09.27 10.20.26 ] Done : DB2_Version
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DBType
[2011.09.27 10.20.26 ] Executing: DBType
[2011.09.27 10.20.26 ] Finished: DBType
[2011.09.27 10.20.26 ] Done : DBType
[2011.09.27 10.20.26 ] Starting: DB2_Version
[2011.09.27 10.20.26 ] Executing: DB2_Version.sh
[2011.09.27 10.20.26 ] Finished: DB2_Version.sh
[2011.09.27 10.20.26 ] Done : DB2_Version
[2011.09.27 10.20.26 ] Starting: DBType

```

Figure 15. prs.trc file on UNIX systems

Execution problems

You can use the execution problems checklist to troubleshoot errors you might encounter when you run Prerequisite Scanner.

Run the Prerequisite Scanner script with the optional **debug** and **trace** input parameters to assist in debugging the issues.

Table 14. Execution problems checklist

Check	Item
<input type="checkbox"/>	When you set the optional outputDir parameter on the command line and the output directory does not exist, Prerequisite Scanner creates the directory. You must have write permissions to create or write to the output directory in which Prerequisite Scanner saves the files. If you do not have write permissions, the following error message is written to the command-line interface: ERROR: Cannot create files in output directory <i>ips_output_dir</i> . Exit.
<input type="checkbox"/>	Before you run Prerequisite Scanner, ensure that disk to which you want to run Prerequisite Scanner and save the results to the output directory is not full; otherwise, the following error message is written to the command-line interface: ERROR: Cannot create files in output directory <i>ips_output_dir</i> . Exit.
<input type="checkbox"/>	If Prerequisite Scanner generates a return code of 2, a script usage or collector error might have occurred. Review the causes associated with this error code. If a script usage error occurred, rerun Prerequisite Scanner by using the correct syntax.

Related concepts:

Writing messages to log files is disabled by default on UNIX systems. You can enable debugging or tracing functions by using the **debug** and **trace** input parameters. The Scanner writes the debug and trace data to different log files and uses timestamps to flag the start and end times of steps or functions. You can use both files to correlate and troubleshoot a specific issue, function, or prerequisite check.

Prerequisite Scanner generates return codes dependent on the results of the scan and whether it must exit because of errors. These return codes are written to the log files.

The `prereq_checker` script runs the IBM Prerequisite Scanner and checks for prerequisites based on the set of parameters that you specify when you run the script.

Return codes

Prerequisite Scanner generates return codes dependent on the results of the scan and whether it must exit because of errors. These return codes are written to the log files.

Prerequisite Scanner generates return codes based on a set of defined outcomes as follows:

Return code	Description
0	Returns 0 when Prerequisite Scanner runs successfully and all scan results are PASS.
1	Returns 1 when Prerequisite Scanner runs successfully, but one or many prerequisite checks return FAIL.
2	Returns 2 when Prerequisite Scanner does not run successfully, and must exit because of an error categorized as follows: <ul style="list-style-type: none">• Script usage errors• Collector errors• Other errors

Script usage errors

Prerequisite Scanner can exit because of any of the following usage errors when running the script:

- The **Product_Code** input parameter is not valid; for example, it was not found or is not a supported format.
- The pattern for the **Product_Code** and **Product_Version** input parameters is not valid; for example, more than just code and version are provided within quotation marks, or the pattern is not enclosed by quotation marks.
- The **Product_Version** input parameters is not valid; for example, the product version is not all numeric characters.
- No input parameters were entered in the command-line interface.
- The syntax was incorrect when entered in the command-line interface; for example, a non-supported command-line argument was entered.
- No required **Product_Code** input parameter was entered.

Collector errors

Prerequisite Scanner can exit because of any of the following collector errors:

- The collector's temporary result file was not found in the *ips_output_dir/temp* directory.
- The collector's script file did not execute correctly.

Other errors

Prerequisite Scanner can exit, because the user does not have write permission to the *ips_output_dir* output directory.

Related concepts:

IBM Prerequisite Scanner produces output for the following screen and human-readable file formats: output to the command-line interface, debugging and trace log files, and text and XML files for the results.

Appendix A. Product codes reference

The IBM Prerequisite Scanner uses a multicharacter code, *product_code*, to identify the product, individual supported platform, and version of operating system. The *ips_root/codename.cfg* file contains the name value pairs to represent the product code for the product, its supported platform, and the version of the operating system.

Table 15 outlines the current set of predefined product codes.

Restriction: IBM Tivoli Monitoring and Tivoli Composite Application Manager have predefined product codes that Prerequisite Scanner considers as reserved. These codes must not be used as Prerequisite Scanner product codes unless they refer to their associated IBM Tivoli Monitoring and Tivoli Composite Application Manager agents. For more information about the product codes, see the ITM 6.X Product Codes Technote.

Table 15. Predefined product codes

Predefined product code	Platform	Product version, platform, operating system
ADE	All	Autonomic Deployment Engine
BSM	All	Tivoli Business Service Manager
CDB	All	Tivoli Composite Application Manager (ITCAM) for Applications: DB2
COA	UNIX	Tivoli Provisioning Manager for UNIX
COB	AIX	Tivoli Provisioning Manager for AIX
COC	AIX	Tivoli Provisioning Manager for AIX V5.3.0.0 {64 bit}
COD	AIX	Tivoli Provisioning Manager for AIX 6.1
COE	Linux	Tivoli Provisioning Manager for Linux
COF	Linux	Tivoli Provisioning Manager for Red Hat Linux
COG	Linux	Tivoli Provisioning Manager Version 7.2 for Red Hat Enterprise Linux 5 x86 64 bit
COH	Linux	Tivoli Provisioning Manager for Red Hat Enterprise Linux 5 System z® 64 bit
COI	Linux	Tivoli Provisioning Manager for SUSE 10
COJ	Solaris	Tivoli Provisioning Manager Version 7.2 for Solaris
COK	HP-UX	Tivoli Provisioning Manager Version 7.2 for HP-UX
COL	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE zSeries® 10
COM	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE 11
CON	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE zSeries 11
COX	Windows	Tivoli Provisioning Manager Version 7.2 for Windows 2008
COY	Windows	Tivoli Provisioning Manager Version 7.2 for Windows 2003

Table 15. Predefined product codes (continued)

Predefined product code	Platform	Product version, platform, operating system
COZ	Windows	Tivoli Provisioning Manager Version 7.2 for Windows
DMO	All	Prerequisite Scanner demo
GYM	UNIX	IBM Tivoli Netcool® Performance Manager
KCJ	Windows	Tivoli Enterprise Portal Client
	UNIX	Tivoli Enterprise Portal Client for UNIX
KCQ	Windows	Tivoli Enterprise Portal Server
	UNIX	Tivoli Enterprise Portal Server for UNIX
KHD	All	Warehouse Proxy Agent
KHE	UNIX	Warehouse Proxy Agent for UNIX
KIS	UNIX	Tivoli Composite Application Manager (ITCAM) for Transactions: Internet Service Monitoring
KLZ	UNIX	Tivoli Monitoring Operating System Agent for Linux
KM6	Windows	IBM Tivoli Composite Application Manager Agent for WebSphere MQ File Transfer Edition
KMQ	All	IBM Tivoli Composite Application Manager Agent for WebSphere WebSphere MQ
KMS	Windows	Tivoli Enterprise Monitoring Server
	UNIX	Tivoli Enterprise Monitoring Server for UNIX
KNT	Windows	Tivoli Monitoring Operating System Agent for Windows
	UNIX	Windows OS monitoring Agent for UNIX
KOR	Windows	Tivoli Monitoring Agent for Oracle
KSY	Windows	Summarization and Pruning Agent
	UNIX	Summarization and Pruning Agent for UNIX
KUD	Windows	Tivoli Monitoring Agent for DB2
	UNIX	Tivoli Monitoring Agent for DB2
KTO	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Reporter
KTU	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Collector
KT3	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Application Management Console
KT4	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Client Response Time
KT5	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Web Response Time
KT6	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Robotic Response Time
KZE	All	Tivoli zEnterprise Monitoring Agent
LCM	Windows	Tivoli License Compliance Manager
	UNIX	Tivoli License Compliance Manager for UNIX
NCI	All	Tivoli Netcool/Impact

Table 15. Predefined product codes (continued)

Predefined product code	Platform	Product version, platform, operating system
NOC	All	Tivoli Netcool/OMNIBus server components and desktop component
NOD	All	Tivoli Netcool/OMNIBus desktop component
NOS	All	Tivoli Netcool/OMNIBus server components
PAE	All	Tivoli Process Automation Engine
TAD	Windows	Tivoli Asset Discovery for Distributed
	UNIX	Tivoli Asset Discovery for Distributed for UNIX
TCR	All	Tivoli Common Reporting
TPM	All	Tivoli Provisioning Manager

Appendix B. Configuration files reference

The IBM Prerequisite Scanner provides a predefined set of configuration files that you can edit. These files are in either *ips_root/UNIX_Linux* or *ips_root/Windows*. The files have a .cfg extension.

Table 16 lists the currently supported predefined configuration files.

Table 16. Predefined configuration files

Configuration file	Platform	Product version, platform, operating system
ADE_01040000.cfg	All	Autonomic Deployment Engine Version 1.4
BSM_04210000.cfg	All	Tivoli Business Service Manager Version 4.2.1
BSM_06100000.cfg	All	Tivoli Business Service Manager Version 6.1
CDB_06220000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Applications: DB2 Version 6.2.2
COA_07200000.cfg	UNIX	Tivoli Provisioning Manager Version 7.2 for UNIX
COB_07200000.cfg	AIX	Tivoli Provisioning Manager Version 7.2 for AIX
COC_07200000.cfg	AIX	Tivoli Provisioning Manager Version 7.2 for AIX V5.3.0.0 {64 bit}
COD_07200000.cfg	AIX	Tivoli Provisioning Manager Version 7.2 for AIX 6.1
COE_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for Linux
COF_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for Red Hat Linux
COG_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for Red Hat Enterprise Linux 5 x86 64 bit
COH_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for Red Hat Enterprise Linux 5 System z 64 bit
COI_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE 10
COJ_07200000.cfg	Solaris	Tivoli Provisioning Manager Version 7.2 for Solaris
COK_07200000.cfg	HP-UX	Tivoli Provisioning Manager Version 7.2 for HP-UX
COL_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE zSeries 10
COM_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE 11
CON_07200000.cfg	Linux	Tivoli Provisioning Manager Version 7.2 for SUSE zSeries 11
COX_07200000.cfg	Windows	Tivoli Provisioning Manager Version 7.2 for Windows 2008
COY_07200000.cfg	Windows	Tivoli Provisioning Manager Version 7.2 for Windows 2003
COZ_07200000.cfg	Windows	Tivoli Provisioning Manager Version 7.2 for Windows
DMO_00000000.cfg	All	Prerequisite Scanner demo
DMO_01000000.cfg	All	Prerequisite Scanner Version 1.0 demo
GYM_01030200.cfg	UNIX	IBM Tivoli Netcool Performance Manager Version 1.3.2
KCJ_06200000.cfg	Windows	Tivoli Enterprise Portal Client Version 6.2
KCJ_06210000.cfg	UNIX	Tivoli Enterprise Portal Client Version 6.2.1
KCJ_06220000.cfg	All	Tivoli Enterprise Portal Client Version 6.2.2
KCQ_06200000.cfg	Windows	Tivoli Enterprise Portal Server Version 6.2
KCQ_06210000.cfg	UNIX	Tivoli Enterprise Portal Server Version 6.2.2
KCQ_06220000.cfg	All	Tivoli Enterprise Portal Server Version 6.2.2

Table 16. Predefined configuration files (continued)

Configuration file	Platform	Product version, platform, operating system
KHD_06200000.cfg	Windows	Warehouse Proxy Agent Version 6.2
KHD_06210000.cfg	All	Warehouse Proxy Agent Version 6.2.1
KHD_06220000.cfg	All	Warehouse Proxy Agent Version 6.2.2
KHE_06220000.cfg	UNIX	Warehouse Proxy Agent Version 6.2.2
KIS_07200000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Internet Service Monitoring Version 7.2
KIS_07300000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Internet Service Monitoring Version 7.3
KLZ_06210000.cfg	UNIX	Tivoli Monitoring Operating System Agent for Linux Version 6.2.1
KLZ_06220000.cfg	UNIX	Tivoli Monitoring Operating System Agent for Linux Version 6.2.2
KM6_070100000.cfg	Windows	Tivoli Composite Application Manager Agent for WebSphere MQ File Transfer Edition Version 7.1
KMQ_070100000.cfg	All	Tivoli Composite Application Manager Agent for WebSphere MQ Version 7.1
KMS_06200000.cfg	Windows	Tivoli Enterprise Monitoring Server Version 6.2
KMS_06210000.cfg	All	Tivoli Enterprise Monitoring Server Version 6.2.1
KMS_06220000.cfg	All	Tivoli Enterprise Monitoring Server Version 6.2.2
KNT_06200000.cfg	Windows	Tivoli Monitoring Operating System Agent for Windows Version 6.2
KNT_06210000.cfg	Windows	Tivoli Monitoring Operating System Agent for Windows Version 6.2.1
KNT_06220000.cfg	Windows	Tivoli Monitoring Operating System Agent for Windows Version 6.2.2
KOR_06220000.cfg	Windows	Tivoli Monitoring Agent for Oracle Version 6.2.2
KSY_06200000.cfg	Windows	Summarization and Pruning Agent Version 6.2
KSY_06210000.cfg	All	Summarization and Pruning Agent Version 6.2.1
KSY_06220000.cfg	All	Summarization and Pruning Agent Version 6.2.2
KTO_07200000.cfg	UNIX	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Reporter Version 7.2
KTO_07200200.cfg	Windows	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Reporter Version 7.2.2
KTO_07300000.cfg	UNIX	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Reporter Version 7.3
KTU_07200000.cfg	UNIX	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Collector Version 7.2
KTU_07200200.cfg	Windows	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Collector Version 7.2.2
KTU_07300000.cfg	UNIX	Tivoli Composite Application Manager (ITCAM) for Transactions: Transaction Collector Version 7.3
KT3_07300000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Application Management Console Version 7.3
KT4_07300000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Client Response Time Version 7.3
KT5_07300000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Web Response Time Version 7.3
KT6_07300000.cfg	All	Tivoli Composite Application Manager (ITCAM) for Transactions: Robotic Response Time Version 7.3

Table 16. Predefined configuration files (continued)

Configuration file	Platform	Product version, platform, operating system
KUD_06100000.cfg	Windows	Tivoli Monitoring Agent for DB2 Version 6.1
KUD_06200000.cfg	All	Tivoli Monitoring Agent for DB2 Version 6.2
KUD_06210000.cfg	All	Tivoli Monitoring Agent for DB2 Version 6.2.1
KUD_06220000.cfg	All	Tivoli Monitoring Agent for DB2 Version 6.2.2
KZE_06020300.cfg	All	Tivoli zEnterprise Monitoring Agent Version 6.2.3
LCM_01000000.cfg	All	Tivoli License Compliance Manager Version 1.0
LCM_02300000.cfg	All	Tivoli License Compliance Manager Version 2.3
NCI_06100000.cfg	All	Tivoli Netcool/Impact Version 6.1
NOC_07310000.cfg	All	Tivoli Netcool/OMNIbus server components and desktop component Version 7.3.1
NOD_07310000.cfg	All	Tivoli Netcool/OMNIbus desktop component Version 7.3.1
NOS_07310000.cfg	All	Tivoli Netcool/OMNIbus server components Version 7.3.1
PAE_07500000.cfg	All	Tivoli Process Automation Engine
TAD_07200000.cfg	All	Tivoli Asset Discovery for Distributed Version 7.2
TAD_07220000.cfg	All	Tivoli Asset Discovery for Distributed Version 7.2.2
TCR_02010100.cfg	All	Tivoli Common Reporting
TPM_07210000.cfg	All	Tivoli Provisioning Manager Version 7.2.1

Appendix C. Prerequisite properties reference

This reference outlines the basic prerequisite properties for each predefined category of hardware and software prerequisites.

Table 17 outlines the predefined categories of hardware and software prerequisites.

Table 17. Predefined categories for prerequisite properties

Data category	Description	Required prefix identifier	Reference
Common	The common data properties check common prerequisites such as processor speed, RAM, disk, and temporary space.	None	"Common data properties" on page 86
Autonomic Deployment Engine	The Autonomic Deployment Engine data properties check Autonomic Deployment Engine prerequisites such as the installation unit.	de	"Autonomic Deployment Engine data properties" on page 90
Installed software	The installed software data properties check installed software prerequisites such as the programs registered in the Windows registry and whether cygwin and gskit are installed.	None	"Installed software data properties" on page 106
User	The user-data properties check user prerequisites such as whether the logged on user had administrative rights or is the root user.	user	"User data properties" on page 106
Operating system	The operating system data properties check operating system prerequisites such as version, architecture, total memory, available memory, and total physical memory.	os	"Operating system data properties" on page 95
Connectivity	The connectivity data properties check connectivity prerequisites such as whether Telnet is running and to which IP addresses and ports the scanner can connect.	None	"Connectivity data properties" on page 91
Network	The network data properties check network prerequisites that can be common across all platforms such as whether there are ports available.	network	"Network data properties" on page 93
Windows network	The Windows network data properties check network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties.	network	"Windows network data properties" on page 107
UNIX network	The UNIX network data properties check network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties.	network	"UNIX network data properties" on page 107
Internet Explorer	The Microsoft Internet Explorer data properties check Internet Explorer prerequisites such as the version.	internetExplorer	"Internet Explorer data properties" on page 93

Table 17. Predefined categories for prerequisite properties (continued)

Data category	Description	Required prefix identifier	Reference
Database server, DB2	The DB2 data properties check DB2 prerequisites such as the version.	DB2	"DB2 data properties" on page 92
Database server, MS SQL	The MS SQL Server data properties check the MS SQL server prerequisites such as the version.	mssql	"MS SQL Server data properties" on page 92
Database server, Oracle	The Oracle data properties check Oracle prerequisites such as the version.	Oracle	"Oracle data properties" on page 94
Environment variables	The environment variables check environment variable prerequisites such as whether the environment variable has been set.	env	

Common data properties

The common data properties check common prerequisites such as CPU speed, RAM, disk, and temporary space. For Windows systems, it uses the primary IBM Prerequisite Scanner script. For UNIX systems, it uses the primary Prerequisite Scanner script and the common collector, *ips_root/Unix_Linux/common.sh*.

Table 18 outlines the common data prerequisite properties. This category of prerequisite properties does not require a prefix identifier.

Table 18. Common data prerequisite properties

Prerequisite property	Platforms	Description	Valid values
CPU Name	All	The name of the CPU and used for display purposes only in the results	Not applicable
CpuArchitecture	UNIX	The architecture of the operating system	String, with multiple supported values separated by a comma, for example: x86_64,s390x,ppc64,AMD64

Table 18. Common data prerequisite properties (continued)

Prerequisite property	Platforms	Description	Valid values
DBType	All	<p>Checks the types of database server installed on the machine.</p> <p>For Oracle on UNIX systems only: The collector expects the ORACLE_BASE and ORACLE_HOME environment variables to be set in the \$HOME/.profile file, for example:</p> <pre>export ORACLE_BASE=/home/oracle/ app/oracle/product/11.2.0/ export ORACLE_HOME=/home/oracle/ app/oracle/product/11.2.0/dbhome_1</pre> <p>where \$HOME must be /home/oracle, the home directory for the Oracle user.</p>	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String representing any type of the database server, for example: any String representing the type of the database server, for example: Oracle regex{<i>str</i>}, a regular expression with the input parameter, <i>str</i>, representing the search pattern for the database server type, for example: regex{.*MSSQL.* DB2.*} <p>Checks whether the database server type is MS SQL or DB2 on Windows systems.</p> <ul style="list-style-type: none"> String representing no type of the database server, for example: unknown
DBTypeDetails	All	<p>The types of database server installed on the machine.</p> <p>For Oracle on UNIX systems only: The collector expects the ORACLE_BASE and ORACLE_HOME environment variables to be set in the \$HOME/.profile file, for example:</p> <pre>export ORACLE_BASE=/home/oracle/ app/oracle/product/11.2.0/ export ORACLE_HOME=/home/oracle/ app/oracle/product/11.2.0/dbhome_1</pre> <p>where \$HOME must be /home/oracle, the home directory for the Oracle user.</p> <p>The prerequisite property writes the details about the database server type, that is, the database server type, the installed location, and the version to the result.txt file. The details of multiple database server types are separated by semi colons</p>	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String representing any type of the database server, for example: any String representing one type of the database server, for example: DB2 regex{<i>str</i>}, a regular expression with the input parameter, <i>str</i>, representing the search pattern for the database server type, for example: regex{.*MSSQL.* DB2.*} <p>Checks whether the database server type is MS SQL or DB2 on Windows systems.</p>

Table 18. Common data prerequisite properties (continued)

Prerequisite property	Platforms	Description	Valid values
Disk	Windows	<p>The amount of free disk space, with the following optional qualification attributes:</p> <ul style="list-style-type: none"> • <code>dir</code> attribute, to determine which path to the directory to check • <code>unit</code> attribute, to determine which units for disk space to use 	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> • String with the following qualifier format: <code>[dir:dir_path, unit:unit_name] disk_space</code> For example: Disk= <code>[dir:C:\Program Files\IBM\SQLLIB, unit:MB]1431</code> • Numeric format in MBs or GBs: <code>disk_spaceMB GB</code> For example: Disk=250MB
Disk	UNIX	The amount of free disk space	Numeric format in GBs or MBs, for example: 2GB
intel.cpu	All	The CPU speed for the Intel processor	Numeric format in MHz or GHz, for example: 2GHz
Memory	All	<p>The total amount of physical memory that is currently available on the machine.</p> <p>Tip: Separately check for the amount of physical and virtual memory available by the using predefined prerequisite properties in the operating system category.</p>	Numeric format in GBs or MBs, for example: 300MB

Table 18. Common data prerequisite properties (continued)

Prerequisite property	Platforms	Description	Valid values
OS Version	All	<p>The full name and version of the operating system that is running on the machine; alternatively, you can use a regular expression to pass a string that represents the multiple variants of an operating system.</p> <p>Tip: Use this prerequisite property in conjunction with <code>os.servicePack</code> and <code>os.architecture</code> to check the current service pack and system architecture.</p>	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String that can represent multiple versions, with each version separated by a comma, for example: RedHat Enterprise Linux 6.*, SuSE Linux Enterprise Server 11, SuSE Linux Enterprise Server 10, SuSE Linux Enterprise Server 9, AIX V6.1,AIX V5.3 <p>Restriction: On Windows systems, the * wildcard is only supported within a regular expression.</p> <ul style="list-style-type: none"> <code>regex{str}</code>, a regular expression with the input parameter, <i>str</i>, representing the search pattern for the version, for example: <code>regex{Windows 200[3-8]}</code> <p>Checks whether the actual OS matches any version from Windows 2003 to Windows 2008. <code>regex{Red Hat*.*}</code></p> <p>Checks whether the actual OS matches a variant of Red Hat Linux.</p> <p>Note: The values can use the special characters as outlined in Table 1 on page 2.</p>
numCPU	Windows	The number of processors on the machine	Number, for example, 4
risc.cpu	UNIX	The CPU speed for a RISC processor	Numeric format in MHz or GHz, for example: 1.4GHz
Temp	All	The available disk space for the specified <i>Temp</i> file system	Numeric format in GBs or MBs, for example: 300MB

Related concepts:

Prerequisite Scanner handles checking the Memory prerequisite property differently depending upon whether a Tivoli Monitoring or Tivoli Composite Application Manager agent is already running on the computer.

Related reference:

The operating system data properties check operating system prerequisites such as version, architecture, total memory, available memory, and total physical memory. For Windows systems only, it uses the operating system VBScript collectors in the *ips_root/lib* directory, with the *os* prefix identifier in their file names. For UNIX systems only, it uses the UNIX operating system collectors in the *ips_root/UNIX_Linux* directory, with the *os* prefix identifier in their file names.

System behavior for Memory prerequisite property and Tivoli Monitoring agents

Prerequisite Scanner handles checking the Memory prerequisite property differently depending upon whether a Tivoli Monitoring or Tivoli Composite Application Manager agent is already running on the computer.

If an agent is already installed, Prerequisite Scanner uses an expected value for the Memory prerequisite property based on the difference between the expected value of the new and existing configuration files if the existing configuration file is still on the computer; otherwise it handles the expected value by the default behavior.

When you run Prerequisite Scanner to check prerequisites for an Tivoli Monitoring agent that is being upgraded or reinstalled, it first checks whether the agent is already running on the computer. If the agent is running, Prerequisite Scanner searches for the configuration file associated with the existing version of the running agent. The following behavior occurs depending upon the outcome of that search:

- If it cannot find the configuration file, Prerequisite Scanner assumes that the target environment has not been scanned before; therefore, Prerequisite Scanner use the expected value for the Memory prerequisite property that is specified in the new configuration file, which follows default behavior. Prerequisite Scanner writes this expected value to the result output.
- If it finds the configuration file, Prerequisite Scanner compares the expected value of the Memory prerequisite property of the existing version with the expected value in the configuration file of the new version. If there is a difference between the values, and the new value is greater the existing expected value, Prerequisite Scanner sets this difference as the expected value. Prerequisite Scanner writes the expected value differential to the result output. For example, the configuration file for agent version 1 specifies 1 GB as the expected value. The new configuration file for agent version 2, specifies 1.5 GBs as the expected value; therefore, Prerequisite Scanner uses and writes 0.5 GB as the expected value differential.

Autonomic Deployment Engine data properties

The Autonomic Deployment Engine data properties check Autonomic Deployment Engine prerequisites such as the installation unit. For Windows systems only, it uses the Autonomic Deployment Engine collectors in the *ips_root/lib/* directory, with the *de* prefix in their file names. For UNIX systems only, it uses the UNIX Autonomic Deployment Engine collectors in the *ips_root/UNIX_Linux* directory, with the *de* prefix in their file names.

Table 19 outlines the prerequisite properties. This category of prerequisite properties require the de prefix identifier.

Table 19. Autonomic Deployment Engine data properties

Prerequisite property	Platform	Description	Valid values
de.installed	All	Checks whether the is installed	Boolean, for example: true false
de.installationUnit	All	Checks whether the specified installation unit is installed by using the listIU - v command	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String to represent a single installation unit, for example, the installation unit for Tivoli Integrated Portal: C37109911C8A11D98E1700061BDE7AEA, B24209911C8A11D98E1700061BDE7AEA String to represent multiple installation units, for example: 5FFE79F918DF3BA0D67511FD3F7C358E regex {str}, a regular expression with the input parameter, str, to represent the search pattern for the installation unit, version, and installation path; for example, to check the installation unit, version of WebSphere Application Server, and the installation path for Tivoli Integrated Portal, the search pattern is as follows: regex{.*C00DA95AFD9B7E0397153CD944B5A255.*6.1.0.2100.*SIU eWAS.*C:\\IBM\\tivoli\\tip.*} <p>Note: You can also use an environment variable for the installation path; for example, by replacing the path with the TIPHOME environment variable, the search pattern is:</p> <pre>regex{.*C00DA95AFD9B7E0397153CD944B5A255.*6.1.0.2100.*SIU eWAS.*%TIPHOME%.*}</pre> <ul style="list-style-type: none"> Multiple regex {str} arguments to represent multiple checks; for example: regex{.*C37109911C8A11D98E1700061BDE7AEA.*}, regex{.*B24209911C8A11D98E1700061BDE7AEA.*}

Connectivity data properties

The connectivity data properties check connectivity prerequisites such as whether Telnet is running and to which IP addresses and ports the Scanner can connect. For Windows systems, it uses the connectivity collector, *ips_root/lib/connectivity_plug.vbs*. For UNIX systems, it uses the primary IBM Prerequisite Scanner script and the connectivity collector, *prs_root/Unix_Linux/connectivity_plug.sh*. The output is passed to the debugging log file only.

DB2 data properties

The DB2 data properties check DB2 prerequisites such as the version. For Windows systems only, it uses the DB2 collector, *ips_root/lib/db2_version_plug.bat*. For UNIX systems only, it uses the UNIX DB2 collectors in the *ips_root/UNIX_Linux* directory, with the db2 prefix in their file names.

Table 20 outlines the DB2 prerequisite properties. This category of prerequisite properties require the DB2 prefix identifier.

Table 20. DB2 data properties

Prerequisite property	Platform	Description	Valid values
DB2 Version	All	The version of DB2 that is currently installed on the machine	String, for example: v9.5.100.179FP4
db2.home.space	UNIX	The available disk space for the DB2 home directory	Numeric format in GBs, for example: 8GB

MS SQL Server data properties

The MS SQL Server data properties check MS SQL Server prerequisites such as the version and location. For Windows systems only, it uses the MS SQL Server collectors in the *ips_root/Windows* directory, with the mssql prefix in their file names.

Table 21 outlines the MS SQL Server prerequisite properties. This category of prerequisite properties require the mssql prefix identifier.

Table 21. MS SQL Server data properties

Prerequisite property	Platform	Description	Valid values
mssql.Client	Windows	Checks the version of MS SQL client that is currently installed on the machine	Expected string value can be multiple versions, separated by a comma, for example: 10.50.1600.1 Note: The values can use the special characters as outlined in Table 1 on page 2.
mssql.Server	Windows	Checks the version of MS SQL Server that is currently installed on the machine	Expected string value can be multiple versions, separated by a comma, for example: 10.50.1600.1 Note: The values can use the special characters as outlined in Table 1 on page 2.
mssql.Server.Location	Windows	Checks the home directory of the MS SQL database server	String, for example: any

Internet Explorer data properties

The Microsoft Internet Explorer data properties check Internet Explorer prerequisites such as the version. It uses the Internet Explorer collector, *ips_root/lib/internetExplorer_plug.vbs*.

Table 22 outlines the Internet Explorer prerequisite properties. This category of prerequisite properties require the *internetExplorer* prefix identifier.

Table 22. Internet Explorer data properties

Prerequisite property	Description	Valid values
internetExplorer.version	The version of the Internet Explorer installed on the machine	Numeric format, for example, 7.0+ Note: The values can use the special characters as outlined in Table 1 on page 2.

Network data properties

The network data properties check network prerequisites that can be common across all platforms such as whether there are ports available. It uses the network collectors in the *ips_root/lib* directory, with the network prefix identifier in their file names.

Table 23 outlines the network prerequisite properties that are common across all platforms. This category of prerequisite properties require the network prefix identifier.

Table 23. Network data properties

Prerequisite property	Platform	Description	Valid values
network.availablePorts. <i>app_type</i>	All	Use this naming convention for checking whether the port or range of ports is available for the <i>app_type</i> application type. Check which ports are not being listened to, for example: <ul style="list-style-type: none">network.availablePorts.DB2 checks ports for DB2 database server, where <i>app_type</i> is DB2network.availablePorts.WAS checks ports for WebSphere Application Server, where <i>app_type</i> is WAS	Positive integers, for example: <ul style="list-style-type: none">network.availablePorts.DB2 = 50000-50005network.availablePorts.WAS = 8080 Note: The values can use the special characters as outlined in Table 1 on page 2.

Table 23. Network data properties (continued)

Prerequisite property	Platform	Description	Valid values
network.portsInUse. <i>app_type</i>	All	<p>Use this naming convention for checking whether the port or range of ports is in use for the <i>app_type</i> application type. Check which ports are being listened to, for example, for example:</p> <ul style="list-style-type: none"> network.availablePorts. DB2 checks ports for DB2 database server, where <i>app_type</i> is DB2 network.availablePorts. WAS checks ports for WebSphere Application Server, where <i>app_type</i> is WAS 	<p>Positive integers, for example:</p> <ul style="list-style-type: none"> network.portsInUse. DB2 = 50900-50905 network.portsInUse. WAS = 8080 <p>Note: The values can use the special characters as outlined in Table 1 on page 2.</p>
network.validate HostsFile	Windows	<p>Checks whether all host machines that are listed in the hosts file are in the format: <i>IP_Address</i> <i>Host_Name</i> <i>Short_Name</i></p> <p>where:</p> <ul style="list-style-type: none"> <i>IP_Address</i> is the IP for the computer, for example, 127.0.0.1 <i>Host_Name</i> is the fully qualified host name of the computer, for example, localhost.localdomain <i>Short_Name</i> is the short name for the computer, for example, localhost 	Boolean value, for example, True

Oracle data properties

The Oracle data properties check Oracle prerequisites such as the version. For Windows systems only, it uses the Oracle collector. For UNIX systems only, it uses the UNIX Oracle collectors in the *ips_root/UNIX_Linux* directory, with the *oracle* prefix in their file names. For Windows systems only, it uses the Windows Oracle collectors in the *ips_root/lib* directory, with the *oracle* prefix in their file names.

Table 24 outlines the Oracle prerequisite properties. This category of prerequisite properties require the *oracle* prefix identifier.

Table 24. Oracle data properties

Prerequisite property	Platform	Description	Valid values
ORACLE Version	Windows	Checks the version of Oracle that is currently installed on the machine	Expected string value can be multiple versions, separated by a comma, for example: 9.2, 10.1, 10.2

Table 24. Oracle data properties (continued)

Prerequisite property	Platform	Description	Valid values
oracle.Client	All	Checks the version of Oracle client that is currently installed on the machine	Expected string value can be multiple versions, separated by a comma, for example: 9.2.0.8+ Note: The values can use the special characters as outlined in Table 1 on page 2.
oracle.Client.Location	All	Checks the home directory of the Oracle client	String, for example: /opt/oracle/products/10.1.0/client_1
oracle.Server	All	Checks the version of Oracle server that is currently installed on the machine	Expected string value can be multiple versions, separated by a comma, for example: 10.2.0.4g,11g R1 Note: The values can use the special characters as outlined in Table 1 on page 2.
oracle.Server.Location	All	Checks the home directory of the Oracle database server	String, for example: /opt/oracle/product/10.1.0/Db_1

Operating system data properties

The operating system data properties check operating system prerequisites such as version, architecture, total memory, available memory, and total physical memory. For Windows systems only, it uses the operating system VBScript collectors in the *ips_root/lib* directory, with the os prefix identifier in their file names. For UNIX systems only, it uses the UNIX operating system collectors in the *ips_root/UNIX_Linux* directory, with the os prefix identifier in their file names.

Table 25 outlines the operating system prerequisite properties. This category of prerequisite properties require the os prefix identifier.

Table 25. Operating system data properties

Prerequisite property	Platform	Description	Valid values
os.architecture	All	Checks the system architecture	32-bit 64-bit
os.automount	UNIX	Checks whether the automount features works	Boolean value, for example: True
os.autoUpdateEnabled	Windows	Checks whether Windows Update is automatically enabled; returns True if enabled	Boolean value, for example: True
os.availableMemory	Windows	Checks the amount of virtual memory that is currently available but unused by the operating system	Numeric format in MBs, for example: 900MB

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.dir.dir_name	UNIX	<p>Checks the <i>dir_name</i> file system based on the following qualification attributes:</p> <ul style="list-style-type: none"> • <i>dir</i> attribute, to determine which file system to check • <i>type</i> attribute, to determine which attribute of the file system to check, for example, the <i>octal_digits</i> octal digit representation for the access permissions to that file system <p><i>dir_name</i> can represent for example:</p> <ul style="list-style-type: none"> • tmp • home 	<p>String with the following qualifier format:</p> <p>[dir:<i>dir_name</i>, type:permission] <i>octal_digits</i>+</p> <p>For example, to check whether the home directory has drwxr-xr-x permissions: os.dir.home=[dir:/home, type:permission]755+</p>
os.diskquota		Checks the disk usage quota for the logged on user; returns the value for the quota in kilobytes or Unlimited	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> • Number to represent kilobytes, for example, 414000 • String to represent an unlimited disk quota, for example, Unlimited
os.expectLink	UNIX	<p>Checks whether Expect extension for TCL is available on the machine; returns Available if it has an available status</p> <p>Note: The os.file.expect prerequisite property checks whether the Expect extension is installed on the machine.</p>	Available Unavailable
os.file.script_name	UNIX	<p>Checks whether the <i>script_name</i> script is available on the machine.</p> <p><i>script_name</i> can represent for example:</p> <ul style="list-style-type: none"> • bash • expect • gzip • tar 	Boolean value, for example: True
os.Firefox	UNIX	Checks whether Mozilla Firefox is installed on the machine; returns Available if it is installed	Available Unavailable
os.FreePagingSpace	UNIX	Checks the total size of the available page cache	<p>Numeric format in MB or GBs, for example: 4GB+</p> <p>Note: The values can use the special characters as outlined in Table 1 on page 2.</p>

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.ftpusers	UNIX	Checks whether the root user is listed in the ftpusers file that determines the users for whom ftp login privileges are no allowed; returns Available if the user is not listed	Available Unavailable
os.gnu.tar	UNIX	Checks whether the GNU tar utility is available on the machine; returns Available if it is installed	Available Unavailable
os.hostformat	UNIX	Checks whether the entries in /etc/host are in the correct format	Boolean value, for example: True
os.iodevicestatus	AIX	Checks the status of the asynchronous I/O (aio0), that is, the kernel process for enhancing I/O operation performance; returns Available if it has an available status	Available Unavailable
os.is8dot3FileFormatEnabled	Windows	Checks whether 8.3 file name formats are being automatically applied; returns True if they are applied	Boolean value, for example: True
os.isServiceRunning, <i>service_name</i>	Windows	Use this naming convention for checking whether the <i>service_name</i> is running on the machine. <i>service_name</i> can represent for example: <ul style="list-style-type: none"> remoteRegistry for the Remote Registry Service DNSClient for the DNS Client Service terminalServices for Remote Desktop Services or Terminal Services 	Boolean value, for example: True
os.kernelMode	AIX	Checks the CPU architecture supporting the kernel or unrestricted mode	32-bit 64-bit
os.kernelParameters	Linux	Checks whether the kernel parameters are available for the operating system	Available Unavailable
os.kernelversion	Linux	Checks the release of the kernel for Linux operating systems	Numeric format, for example, 2.6
os.largeFile	UNIX	Checks for large file support	Boolean value, for example: True
os.ldLibPath	UNIX	Checks whether the LD_LIBRARY_PATH environment variable exists and ends with a colon, that is <code>os.ldLibPath=[endsWith=:]</code>	Available Unavailable
os.level	AIX	Checks whether the AIX operating system is greater than level 10 for AIX Version 5.3 or greater than level 3 for AIX Version 6.1	Boolean value, for example: True

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.lib.lib_name_version	UNIX	<p>Checks that the supported version of the <i>lib_name</i> library is installed on the machine. String or regular expression to represent <i>lib_name_version</i>, for example, in bold:</p> <ul style="list-style-type: none"> 32-bit libstdc++.so.# library 64-bit libstdc++.so.# library 32-bit libXft.so.# library 32-bit libXtst.so.# library 64-bit libaio.so.# library 32-bit x1C.rte XLC runtime level 32-bit x1C.aix50.rte XLC runtime for AIX Version 5.3 32-bit x1C.aix61.rte XLC runtime for AIX Version 6.1 AIX IOCP bos.iocp.rte library bos.loc.iso.en_us, the ISO code fileset for the AIX base operating system 	<p>The value can be any of the following types:</p> <p>String, for example:</p> <ul style="list-style-type: none"> /usr/lib/libstdc++.so.# as the value for the 32-bit libstdc++.so.# library /usr/lib64/libaio.so.# as the value for the 64-bit libaio.so.# library x1C.aix50.rte.9.0.0.8+ as the value for the 32-bit x1C.aix50.rte XLC runtime for AIX Version 5.3 bos.loc.iso.en_us for the ISO code fileset <p>regex {<i>str</i>}, a regular expression with the input parameter, <i>str</i>, representing the search pattern for the library name, for example: regex{.*libgcc.*}</p> <p>Checks whether a version of the GCC low-level runtime library, libgcc, for that operating system exists. Note: The values can use the special characters as outlined in Table 1 on page 2.</p>
os.loginVariable	UNIX	Checks whether the default paths for the root user are set in the PATH and SUPATH variables; returns Available if they are set	Available Unavailable
os.maximoDirectory	UNIX	Check whether /export/home/maximo directory is available	Available Unavailable
os.maximoDirOwner	UNIX	Checks the owner of the /export/home/maximo directory	maximo
os.maximumProcesses	UNIX	Checks the maximum number of processes that can run for each user	Number, for example, 2048
os.MozillaVersion	UNIX	Checks for a specific version of Mozilla Firefox on the machine unlike the os.Firefox prerequisite property	<p>Numeric format, for example, 3.0+</p> <p>Note: The values can use the special characters as outlined in Table 1 on page 2.</p>

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.mountcheck	UNIX	Checks whether the file system is mounted based on the following qualification attributes: <ul style="list-style-type: none"> • drive attribute, to determine which directory is the mounted file system • nosuid attribute, to determine whether the mount option is set if the file system is mounted 	String with the following qualifier format: [drive: <i>dir_name</i> , mount_option: false true] True False For example, to check whether /home directory is mounted and the nosuid option is not set: os.mountcheck=[drive:/home, nosuid:false]True
os.package.package_name	UNIX	Checks that the supported version of the <i>package_name</i> package is installed on the machine. String to represent <i>package_name</i> , for example, in bold: <ul style="list-style-type: none"> • bash shell • expect for the TCL extension package • libgcc for GCC low-level runtime package • openssh for the Open Source secure shell • openssl for the Open Source toolkit for SSL/TLS • perl for the Perl scripting package • rpm for the RPM or RPM Build packages • telnet for the Telnet package • wget for the GNU file retrieval package 	String, for example: <ul style="list-style-type: none"> • bash-3.2+ for bash shell • expect-1.2.0 for Expect • libgcc-3.4.3-9 for libgcc • openssh-5.0.0.5301- for openssh • openssl-4.2.0- for OpenSSL • perl-5.8.2 for Perl • rpm • telnet • wget Note: The values can use the special characters as outlined in Table 1 on page 2.
os.pagesize	UNIX	Checks the pagesize of the system.	Numeric format in KBs, for example: 4KB Note: The values can use the special characters as outlined in Table 1 on page 2.
os.RAMSize	All	Checks the RAM of the system	Numeric format in GBs, for example, 8GB

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.SELinux	Linux	<p>Checks the enforcement status of the Security-Enhancement Linux feature based on the following qualification attributes:</p> <ul style="list-style-type: none"> source attribute, to determine the command to use for the relevant operating system 	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String with the following qualifier format: [source:Command] Disabled Enabled For example, to check whether the feature is disabled or has a permissive status on either Red Hat or SUSE operating system: os.SELinux=[source:Command]Disabled String without a qualifier, where the operating system is a generic Linux variant: os.SELinux=Disabled
os.servicePack	All	Checks the current version of the service pack that is installed	<p>Numeric format, with <i>majorVersion</i>, <i>minorVersion</i> or the <i>majorVersion</i> version only</p> <p>For example, to check whether service pack 2 or later is installed, 2+</p> <p>Note: The values can use the special characters as outlined in Table 1 on page 2.</p>
os.shell.default	UNIX	Checks whether the default shell script is installed	String to represent the shell script, for example, bash

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.space.dir_name prerequisite properties			
Prerequisite Scanner has three variants of the os.space.dir_name property:			
<ul style="list-style-type: none"> os.space.dir_name that checks whether there is enough available disk space for the specified file system regardless of whether the logged in user is always root or non root. Use this prerequisite property variant when you want to check the specified path of the file system, but it does not matter whether the logged in user is always root or always non root. Note: You cannot use this variant twice for the same file system but different user types in a single configuration file; instead use a combination of the other two variants. 			
<ul style="list-style-type: none"> os.space.dir_name_nonroot that checks whether there is enough available disk space for the specified file system of the non-root user. Use this prerequisite property variant when you are logged in as a non-root user, and you want to explicitly check the specified path for the file system. Note: The non-root user should be the same user that installs the product on the target system. 			
<ul style="list-style-type: none"> os.space.dir_name_root that checks whether there is enough available disk space for the specified file system of the root user. Use this prerequisite property variant when you are logged in as a root user, and you want to explicitly check the specified path for the file system. 			
<p>You can specify os.space.dir_name_nonroot and os.space.dir_name_root variants in the same configuration file. Prerequisite Scanner outputs NOT_REQ_CHECK_ID in the actual results cell for the non-applicable variant. For example, if the logged in user is root, Prerequisite Scanner outputs NOT_REQ_CHECK_ID for the os.space.dir_name_nonroot variant.</p>			

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.space.dir_name	UNIX	<p>Checks the available disk space for the specified <i><dir_name></i> file system based on one or more of the following qualification attributes:</p> <ul style="list-style-type: none"> • dir attribute, to determine which path to the file system to check • unit attribute, to determine which units for disk space to use <p>The value for dir attribute is dependant on the logged on user; thus, the value is a name value pair to represent the user type, that is, root or non-root, and the associated path.</p> <p><i>dir_name</i> can represent for example:</p> <ul style="list-style-type: none"> • home • opt • tmp • usr • var <p>Note: You cannot use this variant twice for the same file system but different user types in a single configuration file. Use a combination of the os.space.dir_name_nonroot and os.space.dir_name_root variants.</p>	<p>String with the following qualifier format for the file system of a root user:</p> <pre>[dir:root=<dir_path>, unit:<unit_name>] <disk_space></pre> <p>For example:</p> <pre>os.space.usr= [dir:root=/usr/ibm/common/ acsi, unit:GB]200</pre> <p>String with the following qualifier format for the file system of a non-root user:</p> <pre>[dir:non_root=<dir_path>, unit:<unit_name>] <disk_space></pre> <p>For example:</p> <pre>os.space.home= [dir:non_root=USERHOME/ .acsi_HOST, unit:MB]200</pre> <p>String with the following qualifier format, using only one qualifier:</p> <pre>[dir:<dir_path>] <disk_space> MB</pre> <p>For example:</p> <pre>os.space.home= [dir:/home/sat]250MB</pre> <p>Numeric format in MB or GBs, for example:</p> <pre>os.space.opt=11GB</pre>

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.space.dir_name_nonroot	UNIX	<p>Checks the available disk space for the <i>dir_name</i> file system of the non-root user, based on one or more of the following qualification attributes:</p> <ul style="list-style-type: none"> • <i>dir</i> attribute, to determine which path to the file system to check • <i>unit</i> attribute, to determine which units for disk space to use <p><i>dir_name</i> can represent for example:</p> <ul style="list-style-type: none"> • <i>home</i> • <i>opt</i> • <i>tmp</i> • <i>usr</i> • <i>var</i> 	<p>String with the following qualifier format for the file system of a non-root user:</p> <p>[dir:non_root=<i>dir_path</i>, unit:<i>unit_name</i>] <i>disk_space</i></p> <p>For example:</p> <p>os.space.home_nonroot=[dir:non_root=USERHOME/.acsi_HOST, unit:MB]200</p> <p>String with the <i>dir</i> qualification attribute only for the file system of a non-root user:</p> <p>[dir:non_root=<i>dir_path</i>] <i>disk_space</i>GB MB</p> <p>For example:</p> <p>os.space.opt_nonroot=[dir:non_root=/opt/IBM/ITM]1024MB</p>
os.space.dir_name_root	UNIX	<p>Checks the available disk space for the <i>dir_name</i> file system of the root user, based on one or more of the following qualification attributes:</p> <ul style="list-style-type: none"> • <i>dir</i> attribute, to determine which path to the file system to check • <i>unit</i> attribute, to determine which units for disk space to use <p><i>dir_name</i> can represent for example:</p> <ul style="list-style-type: none"> • <i>home</i> • <i>opt</i> • <i>tmp</i> • <i>usr</i> • <i>var</i> 	<p>String with the following qualifier format for the file system of a root user:</p> <p>[dir:root=<i>dir_path</i>, unit:<i>unit_name</i>] <i>disk_space</i></p> <p>For example:</p> <p>os.space.usr_root=[dir:root=/usr/ibm/common/acsi, unit:GB]200</p> <p>String with the <i>dir</i> qualification attribute only for the file system of a root user:</p> <p>[dir:root=<i>dir_path</i>] <i>disk_space</i>GB MB</p> <p>For example:</p> <p>os.space.opt_root=[dir:root=/opt/IBM/ITM]1024MB</p>
os.sshdConfig	UNIX	Checks whether permitted root login is configured for SSH daemon sessions	Available Unavailable

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.swapSize	UNIX	Checks whether the swap space must be greater than the RAM size or the total amount of swap space	The value can be any of the following types: <ul style="list-style-type: none"> • Boolean value, for example: True • Numeric format in MBs or GBs, for example: 2GB
os.tmpdir	UNIX	Checks access permissions assigned to the /tmp file system, including any specific permissions that are set by access-right flags, for example, sticky, setuid, or setgid bits in the octal digits	Number to represent the octal_digits octal digits for the access permissions For example, to check whether the temp directory has drwxrwxrwt permissions with the sticky bit permission is enabled: 1777 As another example, to check whether the temp directory has drwxrwxrwx permissions excluding the sticky bit: 777
os.totalMemory	Windows	The total amount of virtual memory to which the operating system can access	Numeric format in MBs or GBs, for example: 4GB
os.totalPhysicalMemory	Windows	The total amount of physical memory that the operating system can access, but it does not indicate the true amount of physical memory on the target computer	Numeric format in MBs or GBs, for example: 2030MB

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.ulimit	UNIX	<p>Checks whether an unlimited number of processes can be run based on the following qualification attributes:</p> <ul style="list-style-type: none"> type attribute, to determine which additional limit to check, for example, the <code>filedescriptorlimit</code> checks the limit for the number of file descriptors that processes can open <p>Alternatively it checks whether the following limits have been set for the specified domains in the <code>/etc/security/limits.conf</code> file:</p> <pre>root - stack unlimited ctginst1 - stack unlimited root - nofile 8192 tioadmin - nofile 32767</pre>	<p>The value can be any of the following types:</p> <ul style="list-style-type: none"> String with the following qualifier format: <code>[type:limit_name] limit_value, limited unlimited</code> For example, to check whether the file descriptor limit is greater than 8192, with unlimited number of processes: <code>os.ulimit=[type:filedescriptorlimit] 8192+, unlimited</code> <p>Valid types of limits to check, where <i>limit_name</i> represents the type of limit are as follows:</p> <ul style="list-style-type: none"> ALL, checks all limits corefilesizelimit datasegmentlimit filedescriptorlimit filesizelimit hardlimit processlimit maxmemorysizelimit maxprocesseslimit stacksizelimit threadlimit <ul style="list-style-type: none"> Available Unavailable to specify whether the relevant domains have limit sets in the <code>/etc/security/limits.conf</code> file.
os.umask	UNIX	Checks the permissions for the file mode creation mask	Number to represent the <i>octal_digits</i> octal digits for the access permissions. For example, to check that new files are only writeable for the owner, set the octal digit to be 0022
os.userLimits	UNIX	Checks whether the maximum stack size is unlimited; returns Available if it is unlimited	Available Unavailable
os.versionNumber	Windows	Checks the current version of the operating system that is installed on the machine	Numeric format, for example, 5.0+ Note: The values can use the special characters as outlined in Table 1 on page 2.

Table 25. Operating system data properties (continued)

Prerequisite property	Platform	Description	Valid values
os.windowManager	UNIX	Checks whether GNOME or KDE is available as a graphical desktop	Available Unavailable

Installed software data properties

The installed software data properties check installed software prerequisites such as the programs registered in the Windows registry and whether cygwin and gskit are installed. For Windows systems only, it uses the installed software collectors in the *ips_root/lib* directory, with the installedSoftware, cygwin, or gskit prefix identifier in their file names.

Table 26 outlines the common data prerequisite properties. This category of prerequisite properties does not require a prefix identifier.

Table 26. Installed software data properties

Prerequisite property	Platform	Description	Valid values
installedSoftware	Windows	Scans the operating system registry for installed programs with locations	String, with multiple applications separated by a comma.
cygwinVersion	Windows	Checks the version of cygwin that is installed on the machine; returns 0.0 if no version installed	Positive integer, for example, 1.5 Note: The values can use the special characters as outlined in Table 1 on page 2.
gskit7Version	Windows	Checks whether gskit Version 7 is installed on the machine; returns 0.0 if version 7 is not installed	Positive integer, for example, 7.0
gskit8Version	Windows	Checks whether gskit Version 8 is installed on the machine; returns 0.0 if version 8 is not installed	Positive integer, for example, 8.0

User data properties

The user data properties check user prerequisites such as whether the logged on user had administrative rights or is the root user. For Windows systems only, it uses the user collector in the *ips_root/lib* directory, with the user prefix identifier in their file names. For UNIX systems only, it uses the user collector in *ips_root/lib/packageTest.sh*.

Table 27 outlines the user prerequisite properties. This category of prerequisite properties require the user prefix identifier.

Table 27. User data properties

Prerequisite property	Platform	Description	Valid values
user.userID	Windows	The ID of the currently logged on user	String, for example, smithj
user.isAdmin	All	Checks whether the logged on user is a member of the Administrator group	Boolean value, for example, True

Windows network data properties

The Windows network data properties check network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties. It uses the Windows network collectors in the *ips_root/lib* directory, with the network prefix identifier in their file names. .

Table 28 outlines the network prerequisite properties that are common across all Windows platforms. This category of prerequisite properties require the network prefix identifier.

Table 28. Windows network data properties

Prerequisite property	Description	Valid values
network.DHCPEnabled	Checks whether at least one adapter with a valid IP address has obtained that IP address by using DHCP; returns True if there is at least one.	Boolean value, for example, False
network.netBIOSEnabled	Checks whether at least one adapter with a valid IP address has NetBIOS enabled as a protocol; returns True if there is at least one.	Boolean value , for example, True
network.pingLocalhost	Checks whether the local host responds to the ping protocol; returns True if it does.	Boolean value, for example, True
network.pingSelf	Checks whether the local computer name has been resolved by using DCHP and whether it can be pinged; returns True if it can be.	Boolean value, for example, True
network.ValidateHostsFile	Checks whether the entries in C:\WINDOWS\system32\drivers\etc\hosts are in the correct format; returns True if the format is valid.	Boolean value, for example, True

UNIX network data properties

The UNIX network data properties check network prerequisites such as whether NetBIOS and DHCP are enabled on the machine, and pinging properties. It uses the network collectors in the *ips_root/UNIX_Linux* directory.

Table 29 outlines the network prerequisite properties that are common across all UNIX platforms. This category of prerequisite properties require the network prefix identifier.

Table 29. UNIX network data properties

Prerequisite property	Description	Valid values
network.DHCPEnabled	Checks whether at least one adapter with a valid IP address has obtained that IP address by using DHCP	Boolean value, for example, False
network.dns	Checks whether the DNS entry for the host machine is correct	Boolean value, for example, True
network.fqdn	Checks whether the fully qualified domain name for the host machine is set	Boolean value, for example, True
network.pingLocalhost	Checks whether the local host responds to the ping protocol	Boolean value, for example, True
network.pingSelf	Checks whether the local computer name has been resolved by using DCHP and whether it can be pinged	Boolean value, for example, True

Appendix D. Predefined collectors for UNIX systems

There are individual collectors for prerequisite properties checks on UNIX systems that are in the *ips_root/lib* directory. You can review these collectors and their input parameters before you create custom collectors.

Table 30 outlines the predefined collectors for UNIX systems.

Table 30. UNIX collectors

Collector	For prerequisite property	Inputs
DB2_Version	DB2 Version	None
DBType	DBType	None
DBTypeDetails	DBTypeDetails	None
env.classpath.derbyJAR	env.classpath.derbyJAR	None
network.DHCPEnabled	network.DHCPEnabled	None
network.dns	network.dns	None
network.fqdn	network.fqdn	None
network.pingSelf	network.pingSelf	None
network.port	network.availablePorts.* network.portsInUse.*	\$ports
oracle.Client	oracle.Client	None
oracle.Client.Location	oracle.Client.Location	None
oracle.Server	oracle.Server	None
oracle.Server.Location	oracle.Server.Location	None
os.architecture	os.architecture	32 bit 64 bit
os.automount	os.automount	None
os.cmd	os.lookup	nslookup
os.cmd	os.tar os.gnu.tar	tar gtar
os.dir	os.dir.dir_name dir_name can represent for example: <ul style="list-style-type: none">• tmp• home	String in the following format: [dir:dir_name, type:permission] octal_digits+ For example, to check whether the dir_name directory, that is, the home directory has drwxr-xr-x permissions: [dir:/home, type:permission]755+
os.diskquota	os.diskquota	None
os.expectLink	os.expectLink	None

Table 30. UNIX collectors (continued)

Collector	For prerequisite property	Inputs
os.filepath	os.file.script_name <i>script_name</i> can represent for example: <ul style="list-style-type: none"> • bash • expect • gzip • tar 	Path to script file, where the path can be: <ul style="list-style-type: none"> • /bin/bash • /usr/bin/expect • /usr/bin/gzip • /usr/bin/tar
os.Firefox	os.Firefox	None
os.FreePagingSpace	os.FreePagingSpace	None
os.ftpusers	os.ftpusers	None
os.hostformat	os.hostformat	None
os.iodevicestatus	os.iodevicestatus	None
os.kernelMode	os.kernelMode	None
os.kernelParameters	os.kernelParameters	None
os.kernelversion	os.kernelversion	None
os.largeFile	os.largeFile	None
os.ldLibPath	os.ldLibPath	Product
os.level	os.level	None
os.lib	os.lib.lib_name_version String or regular expression to represent <i>lib_name_version</i> , for example, in bold: <ul style="list-style-type: none"> • 32-bit libstdc++.so.# library • 64-bit libstdc++.so.# library • 32-bit libXft.so.# library • 32-bit libXtst.so.# library • 64-bit libaio.so.# library • 32-bit xlC.rte XLC runtime level • 32-bit xlC.aix50.rte XLC runtime for AIX Version 5.3 • 32-bit xlC.aix61.rte XLC runtime for AIX Version 6.1 • AIX IOCP bos.iocp.rte library • bos.loc.iso.en_us, the ISO code fileset for the AIX base operating system • <p>regex{str}, a regular expression with the input parameter, <i>str</i>, representing the search pattern for the library name, for example, .*libgcc.*</p>	<i>path_to_library</i> or <i>lib_name_version</i> For example, to check the actual value for the os.lib.libstdc++.so prerequisite property, the input parameters are /usr/lib/libstdc++.so.5 and libstdc++.so: os.lib /usr/lib/libstdc++.so.5 libstdc++.so For example to check whether a version of the GCC low-level runtime library, libgcc, exists on the machine, the input parameter is: regex{.*libgcc.*}
os.loginVariable	os.loginVariable	None
os.maximoDirectory	os.maximoDirectory	None

Table 30. UNIX collectors (continued)

Collector	For prerequisite property	Inputs
os.maximoDirOwner	os.maximoDirOwner	None
os.maximumProcesses	os.maximumProcesses	None
os.MozillaVersion	os.MozillaVersion	None
os.mountcheck	os.mountcheck	String with the following qualifier format: [drive:dir_name, mount_option: false true] True False For example, to check whether /home directory is mounted and the nosuid option is not set: os.mountcheck=[drive:/home, nosuid:false]True
os.package	os.package.package_name String to represent package_name, for example, in bold: <ul style="list-style-type: none"> • bash shell • expect for the TCL extension package • libgcc for GCC low-level runtime package • openssh for the Open Source secure shell • openssl for the Open Source toolkit for SSL/TLS • perl for the Perl scripting package • rpm for the RPM or RPM Build packages • telnet for the Telnet package • wget for the GNU file retrieval package 	package_name, for example, where package_name is rpm: os.package rpm
os.pagesize	os.pagesize	None
os.RAMSize	os.RAMSize	None
os.SELinux	os.SELinux	<ul style="list-style-type: none"> • String with the following qualifier format: [source:Command] Disabled Enabled For example, to check whether the feature is disabled or has a permissive status on either Red Hat or SUSE operating system: os.SELinux=[source: Command]Disabled • If there is no qualifier, no value is passed to the collector.
os.servicePack	os.servicePack	Service pack value

Table 30. UNIX collectors (continued)

Collector	For prerequisite property	Inputs
os.shell.default	os.shell.default	The expected value of the prerequisite property, for example, bash
os.space	<p>os.space.dir_name</p> <p>dir_name can represent for example:</p> <ul style="list-style-type: none"> • usr • home • tmp • var 	<p>String with the following qualifier format for the file system of a root user:</p> <p>[dir:root=dir_path, unit:unit_name] disk_space</p> <p>For example:</p> <p>os.space.usr=[dir:root=/usr/ibm/common/acsi, unit:GB]200</p> <p>String with the following qualifier format for the file system of a non-root user:</p> <p>[dir:non_root=dir_path, unit:unit_name] disk_space</p> <p>For example:</p> <p>os.space.home=[dir:non_root=USERHOME/.acsi_HOST, unit:MB]200</p> <p>String with the following qualifier format, using only one qualifier:</p> <p>[dir:dir_path] disk_space MB</p> <p>For example:</p> <p>os.space.home=[dir:/home/sat]250MB</p>
os.sshdConfig	os.sshdConfig	None
os.swapSize	os.swapSize	None
os.tmpdir	os.tmpdir	None

Table 30. UNIX collectors (continued)

Collector	For prerequisite property	Inputs
os.ulimit	os.ulimit	<p>String with the following qualifier format:</p> <pre>[type:limit_name] limit_value, limited unlimited</pre> <p>For example, to check whether the file descriptor limit is greater than 8192, with unlimited number of processes:</p> <pre>os.ulimit= [type:filedescriptorlimit] 8192+,unlimited</pre> <p>Valid types of limits to check, where <i>limit_name</i> represents the type of limit are as follows:</p> <ul style="list-style-type: none"> • ALL, checks all limits • corefilesizelimit • datasegmentlimit • filedescriptorlimit • filesizelimit • hardlimit • processlimit • maxmemorysizelimit • maxprocesseslimit • stacksizelimit • threadlimit
os.umask	os.umask	None
os.userLimits	os.userLimits	None
os.windowManager	os.windowManager	None

Appendix E. Common functions for Windows systems

Prerequisite Scanner has a set of common functions in the/lib/common_function.vbs file for running checks on Windows systems.

Table 31. Functions in common_function.vbs

Function	Description
"allFiles()" on page 116	Reads the file names in a specified directory into an array.
"arrayToString()" on page 116	Creates a string representation for the array.
"bigthan()" on page 117	Calculates the difference between the expected and actual value of the prerequisite property if that prerequisite property is a size in MBs or GBs.
"changeMG()" on page 117	Converts the input parameter to MBs or GBs for disk space or memory prerequisite properties.
"checkItemToString()" on page 118	Creates a string representation for the CheckItem object.
"dictionaryToString()" on page 118	Creates a string representation for the scripting dictionary object.
"exeCommand()" on page 119	Executes the specified command and returns the result from that command's execution.
"filterCommand()" on page 119	Executes the specified command and returns the lines from the result of the command that match the specified pattern.
"filterFile()" on page 120	Reads and filters the contents of a file to a scripting dictionary object.
"findNewest()" on page 120	Finds the latest configuration file.
"findSuitableFile()" on page 121	Finds the relevant configuration file for a product and version.
"fmt()" on page 122	Modifies a string by adding a specified number of characters from another string to it and padding the other string with space characters if the length of the other string is too short or truncating the other string if it is too long.
"formatForDisplay()" on page 122	Formats the input parameter to make it readable.
"formatSizeForDisplay()" on page 123	Takes the input parameter and appends or trims the fractional part of the input parameter to two decimal points, for example, 123MB to 123.00MB or 12.123MBs to 12.12MBs.
"getDecimalSeparator()" on page 123	Determines the decimal separator that is used for the current locale.
"getFirstMatch()" on page 123	Gets the first match of the search string in the array.

Table 31. Functions in *common_function.vbs* (continued)

Function	Description
"isMatch()" on page 124	Checks whether the search pattern is in the string.
"notInLatter()" on page 124	Filters the first array to determine whether the content is in the second array. Depending upon the value of the <i>in_or_not</i> input parameter, the function returns the contents of first array including or excluding what matched with the second array.
"passOrFail()" on page 125	Compares the expected and actual values of the prerequisite property and determines whether the prerequisite property passes the check. The input parameters can be generic numbers, size in MBs or GBs, CPU speed in MHz or GHz, boolean, or strings.
"ppread()" on page 126	Reads the contents of a file to a scripting dictionary object, further splitting each line in the file by the specified separator input parameter if that separator exists in the line.
"readFile()" on page 126	Reads each line of a file into an index entry of an array.
"unitMGTOG()" on page 127	Adds together the contents of an array to obtain the total number of MBs.
"varToString()" on page 127	Creates a string representation of a variable. The variable to check can be a string, number, scripting dictionary object, array, or CheckItem object.

allFiles()

Reads the file names in a specified directory into an array.

Purpose

This function gets the list of files in the directory input parameter and adds them to the array. It returns the array.

Syntax

```
allFiles(filepath)
```

Input parameters

String *filepath*

The path to the directory that contains the files.

Return values

Array *fileNames*

Returns the array containing the file names in the specified directory.

arrayToString()

Creates a string representation for the array.

Purpose

This function takes the array that is passed as an input parameter and returns a string representation of the contents of that array.

Syntax

```
arrayToString(arr)
```

Input parameters

Array *arr*

Contains the array.

Return values

String *result*

Returns a string representation of the array, with each item separated by a comma.

bigthan()

Calculates the difference between the expected and actual value of the prerequisite property if that prerequisite property is a size in MBs or GBs.

Purpose

This function first calls the “changeMG()” function to change the expected and actual values of the prerequisite property to MBs if required. It then checks whether either returned value from the functions is null, and if either value is null, the returned value of the function is 0MB and the function exits. It checks for MB or GB in either value, converts to the MBs if required. It calculates the difference between the final formatted values and returns the result.

Syntax

```
bigthan(expect,real)
```

Input parameters

String *expect*

The expected value of the prerequisite property.

String *real*

The actual value of the prerequisite property.

Return values

String *bigthan*

Returns the difference in MBs or 0MBs if there is no difference.

changeMG()

Formats the input parameter to remove any additional digit grouping characters from it and returns the formatted parameter unless the input parameter contains MBs or GBs. If it does, it converts the input parameter to either GBs or MBs respectively.

Purpose

This function calls the “getDecimalSeparator()” on page 123 function to determine the decimal separator for the current locale and then removes any additional digit grouping characters for that locale from the number input parameter. It then calls the “getFirstMatch()” on page 123 function to determine whether the value is in MBs or GBs and then converts the value to GBs or MBs respectively.

Syntax

`changeMG(tochange)`

Input parameters

String *tochange*

Contains the value format and convert as necessary.

Return values

String *changeMG*

Returns either the formatted number without the digit grouping characters or the number in MBs or GBs.

checkItemToString()

Creates a string representation for the CheckItem object.

Purpose

This function takes the CheckItem object that is passed as an input parameter and returns a string representation that comprises the values of different properties for that instance of the CheckItem object.

Syntax

`checkItemToString(var)`

Input parameters

CheckItem *var*

Contains the instance of the CheckItem object.

Return values

String *result*

Returns a string representation for the properties of the CheckItem object as follows:

```
result = "CheckItem[pdCode[" & chkItem.pdCode & "],pdName[" & chkItem.pdName & "  
        "],itype[" & chkItem.itype & "],recommended[" & chkItem.recommended & "  
        "],realValue[" & chkItem.realValue & "],passOrFail[" & "  
        chkItem.passOrFail & "]"
```

dictionaryToString()

Creates a string representation for the scripting dictionary object.

Purpose

This function takes the dictionary object that is passed as an input parameter and returns a string representation of the contents of that dictionary object.

Syntax

`dictionaryToString(dic)`

Input parameters

Dictionary *dic*

Contains the dictionary object.

Return values

String *result*

Returns a string representation of the dictionary object, with each key and item separated by an equals symbol.

exeCommand()

Executes the specified command and returns the result from that command's execution.

Purpose

This function executes the command input parameter. If there are any errors, it calls the `logWarning` sub routine to display the errors; otherwise, it returns the result from that command's execution.

Syntax

`exeCommand(cmd)`

Input parameters

String *cmd*

The name of the command to execute.

Return values

String *result*

Returns a string that contains the result from that command's execution.

filterCommand()

Executes the specified command and returns the lines from the result of the command that match the specified pattern.

Purpose

This function executes the command input parameter. It parses the result from that command's execution and checks whether any line from the result matches the line pattern input parameter. If there is a match, it calls the `"getFirstMatch()"` on page 123 function to determine whether there's also match between the information line input parameter and the command's result. If there is, it then uses the `Join` function to return the contents of the dictionary object from the `getFirstMatch()` function.

Syntax

`filterCommand(cmd, line_patt, after_line, info_patt)`

Input parameters

String *cmd*

The name of the command to execute.

String *line_patt*

The line pattern for which to search in the result from that command's execution.

Number *after_line*

The number of lines after which the search for the information pattern stops.

String *info_patt*

The information pattern for which to search in each line from the result of the command.

Return values

String *filterCommand*

Returns the contents of the dictionary object as a single string.

filterFile()

Reads and filters the contents of a file to a scripting dictionary object.

Purpose

This function reads each line of the file and passes each line with the search pattern to the "getFirstMatch()" on page 123 function. If it returns a match and the line does not already exist in the dictionary object, the line is written to the dictionary object. The function loops until the end of the file is reached and then returns the dictionary object.

Syntax

`filterFile(fileName, patt)`

Input parameters

String *fileName*

The file to be filtered.

String *patt*

The pattern for which to search in each line in the file.

Return values

Dictionary *dic.keys*

Returns the *dic* dictionary object with the filtered lines from the file.

findNewest()

Finds the latest configuration file in an array.

Purpose

This function loops through the array and determines which file in the array is the latest configuration file. It returns the name of the file.

Syntax

`findNewest(arr)`

Input parameters

Array *arr*

Contains the set of configuration files to check.

Return values

String *result*

Returns the name of the latest configuration file.

findSuitableFile()

Finds the relevant configuration file for a product and version.

Purpose

This function calls the “getFirstMatch()” on page 123 function to get the set of files that has the extension input parameter as the file extension from the list of files returned by the “allFiles()” on page 116 function. It then calls the “getFirstMatch()” on page 123 function again to return the set of files that contains the product code input parameter in the file name. It calls the same function to get the set of files that contains the version input parameter in the file name. If the functions finds one or more file matches for the version, it calls the “findNewest()” on page 120 function to get the latest version of that file and returns that file name; otherwise, it returns `common.bat` file or uses the `logScreen` and `logWarning` sub routines before returning the latest version of the configuration file for the product code.

Syntax

`findSuitableFile(pd,version,suf,filepath)`

Input parameters

String *pd*

The product code associated with the file to find, as specified in the product code file, `ips_root/codename.cfg` file.

String *version*

The version of the product associated with the file to find. *<version>* is the 8-digit code to represent the version, release, modification and level, with two digits for each part of the code; for example, 7.3.21 is 07032100.

String *suf*

The extension for the type of file to find, such as `cfg` or `bat`.

String *filepath*

The path to the directory that contains the file to find.

Return values

String *findSuitableFile*

Returns one of the following file names depending upon the results of called functions:

- `pd_version.cfg`, the latest version of the file for the associated product code and version.
- `common.bat` if the value for the file extension input parameter is `bat`.

- `pd.cfg` , the latest version of the generic configuration file for the product, if no file containing the version input parameter was found.

fmt()

Modifies a string by adding a specified number of characters from another string to it and padding the other string with space characters if the length of the other string is too short or truncating the other string if it is too long.

Purpose

This function searches for the `%s` expression inside the `s` input parameter of type string. The `%s` expression determines the specified `#` number of characters from the `args` input parameter that are added to the first string at the position of that expression. If the specified `#` is greater than the length of the `args` input parameter, the difference is padded by space characters. If the specified `#` is less than the length of the `args` input parameter, the length is truncated by the difference. If the specified `#` is 0, the full length of the `args` input parameter is added to the first string at the appropriate position in the string.

Syntax

`fmt(s, args)`

Input parameters

String `s`

Contains the string to modify by the specified `#` number of characters in the `%s` expression inside that string.

Array `args`

Contains the set of characters that modify the `s` input parameter.

Return values

String `result`

Returns the modified string.

Example

```
fmt("Hello %5s!",array("Neo")) returns "Hello Neo !" padded with extra space characters
fmt("Hello %5s!",array("Mr. Anderson")) returns "Hello Mr. A!" truncated to add only "Mr. A"
fmt("Hello %0s!",array("Mr. Anderson")) returns "Hello Mr. Anderson!"
```

formatForDisplay()

Formats the input parameter to make it readable.

Purpose

This function calls the “`formatSizeForDisplay()`” on page 123 function to format the input parameter.

Syntax

`formatForDisplay(val)`

Input parameters

Variable `val`

The variable to format.

Return values

String *varToString*

Returns the result of the called “formatSizeForDisplay()” function.

formatSizeForDisplay()

Takes the input parameter and appends or trims the fractional part of the input parameter to two decimal points, for example, 123MB to 123.00MB or 12.123MBs to 12.12MBs.

Purpose

This function counts the number of characters in the input parameter, checks whether it's a number or string, and splits the input parameter into the whole and fractional parts. Depending upon the fractional part, it appends or trims it to two decimal places. It returns the result.

Syntax

`formatSizeForDisplay(size)`

Input parameters

Integer *size*

The value to round to two decimal points.

Return values

Integer *val*

Returns the value rounded to two decimal points.

getDecimalSeparator()

Determines the decimal separator that is used for the current locale.

Purpose

This function creates a fractional number and then uses the `Mid()` function to determine the decimal separator that is used in that fractional number.

Syntax

`getDecimalSeparator()`

Input parameters

None

Return values

Character *sep*

Returns the decimal separator, for example , or . for locale.

getFirstMatch()

Gets the first match of the search string in the array.

Purpose

This function uses a regular expression to search for the pattern, which is passed as an input parameter, in the array that is also passed as an input parameter. When it finds the first match of the pattern in the array, it adds the value from the array to the scripting dictionary object.

Syntax

```
getFirstMatch(patt, arr)
```

Input parameters

String *patt*

Contains the pattern for which to search.

Array *arr*

Contains the array in which to search for the search pattern.

Return values

Dictionary *keys*

Returns the keys for the scripting dictionary object.

isMatch()

Checks whether the search pattern is in the string.

Purpose

This function calls the “getFirstMatch()” on page 123 function, passing the pattern and string (contained within an array) as input parameters to this function. It invokes ubound function to check whether the returned value from the getFirstMatch() function is greater than or equal to 0. If it is, there is match; otherwise, there is no match.

Syntax

```
isMatch(patt,str)
```

Input parameters

String *patt*

Contains the pattern for which to search.

String *str*

Contains the string in which to search for the search pattern.

Return values

Boolean *True|False*

Returns True if there is a match; otherwise, it returns False.

notInLatter()

Filters the first array to determine whether the content is in the second array. Depending upon the value of the in_or_not input parameter, the function returns the contents of first array including or excluding what matched with the second array.

Purpose

Syntax

```
notInLatter(arr1, arr2, in_or_not)
```

Input parameters

Array *arr1*

copy from somewhere

Array *arr2*

to somewhere else

String *in_or_out*

Contains either "in" or "not" depending whether the function should return the contents of the first array filtered to return only the contents that matched the second array ("in") or the contents that did not match the second array ("not").

Return values

Dictionary *keys*

Returns the keys of the scripting dictionary object that contains the first array filtered to have only the contents that matched the second array (*in_or_not* = "in") or the contents that did not match the second array or (*in_or_not* = "not").

passOrFail()

Compares the expected and actual values of the prerequisite property and determines whether the prerequisite property passes the check. The input parameters can be generic numbers, size in MBs or GBs, CPU speed in MHz or GHz, boolean, or strings.

Purpose

This function first calls the "changeMG()" on page 117 function to format and if necessary convert the expected and actual values. It checks whether either value is 0, and if yes, it returns "FAIL" and exits. If the values are not 0, the function checks whether the values are boolean, numeric, size in MBs or GBs, CPU speed in MHz or GHz, or strings. It then compares the values and returns the result.

Syntax

```
passOrFail(expect,real)
```

Input parameters

String *expect*

The expected value for the prerequisite property.

String *real*

The actual value for the prerequisite property.

Return values

String *passOrFail*

Returns "PASS" or "FAIL" depending upon whether the expected value is equal to or greater than the actual value.

ppread()

Reads the contents of a file to a scripting dictionary object, further splitting each line in the file by the specified separator input parameter if that separator exists in the line.

Purpose

This function reads each line of the file, removes any leading or trailing spaces, and checks whether it contains the separator. If it contains the separator, it splits the line by the separator, adding each piece as an item to the dictionary object; otherwise, it adds the trimmed line to an item in the dictionary object. It returns an array that contains the dictionary object as the first index.

Syntax

```
ppread(fileName, sep)
```

Input parameters

String *fileName*

The name of the file to read into the dictionary object.

Character *sep*

The character that represents the separator by which to split a line in the file.

Return values

Array *array(dic)*

Returns an array with the dictionary object (dic) as its first index.

Example

Example to be provided.

readFile()

Reads each line of a file into an index entry of an array.

Purpose

This function opens the file and reads each line of the file into an index entry of the array. It returns the array.

Syntax

```
readFile(fileName)
```

Input parameters

String *fileName*

The name of the file to read into the array.

Return values

Array *fileContents*

Returns the array with the contents of the file.

unitMGTOG()

Adds together the contents of an array to obtain the total number of MBs.

Purpose

This function converts the value of each index in the array to MBs and adds them together.

Syntax

`unitMGTOG(arr)`

Input parameters

Array *arr*

Contains the array.

Return values

String *unitMGTOG*

Returns the total of the contents of the array in MBs and appends "MB" to the total.

varToString()

Creates a string representation of a variable. The variable to check can be a string, number, scripting dictionary object, array, or CheckItem object.

Purpose

This function checks the data or object type of the variable and calls the relevant function to create a string representation for that data or object type.

Table 32. Called function for each variable type.

Variable type	Called function
Array	"arrayToString()" on page 116
CheckItem object	"checkItemToString()" on page 118
Scripting dictionary object	"dictionaryToString()" on page 118

Syntax

`varToString(var)`

Input parameters

Variable *var*

The supported variables are: string, number, scripting dictionary object, array, or CheckItem object

Return values

String *vartoString*

Returns a string representation of the variable including returned values from any called functions where required.

Appendix F. Logging utility sub routines for Windows systems

IBM Prerequisite Scanner has a set of common logging sub routines in the `preq.vbs` file for displaying messages on screen or writing to the log file.

Table 33 describes the logging utilities.

Table 33. Logging utility sub routines

Sub routine	Description	Input parameters
<code>deleteLogFile</code>	Deletes the log file if it exists.	None
<code>log(level, msg)</code>	Writes the message to the log file by using the "fmt()" on page 122 function. The log also includes the current date and time.	<ul style="list-style-type: none">• <code>level</code>, a string that sets the type of message such as information or warning• <code>msg</code>, a string that represents the message to log
<code>logDebug(msg)</code>	Calls the <code>log()</code> function, passing "DEBUG" as the level input parameter.	<code>msg</code> , a string that represents the message to log
<code>logError(msg)</code>	Calls the <code>log()</code> function, passing "ERROR" as the level input parameter.	<code>msg</code> , a string that represents the message to log
<code>logInfo(msg)</code>	Calls the <code>log()</code> function, passing "INFO" as the level input parameter.	<code>msg</code> , a string that represents the message to log
<code>logScreen(msg)</code>	Writes the message to the screen.	<code>msg</code> , a string that represents the message to write to the screen
<code>logWarning(msg)</code>	Calls the <code>log()</code> function, passing "WARNING" as the level input parameter.	<code>msg</code> , a string that represents the message to log

Appendix G. File utility sub routines for Windows systems

Prerequisite Scanner has a set of common file sub routines in the/lib/common_function.vbs file to handle files. It also has a set of functions for handling files.

Table 34 describes the file utilities.

Table 34. File utility sub routines

Sub routine	Description	Input parameters
appendToFile(text, fileName)	Appends the text to the end of the specified file.	<ul style="list-style-type: none">• text, a string that contains the text to append to the file• filename, a string that represents the name of the file to modify
writeToFile(text, fileName)	Writes the text to the specified file, overwriting the existing contents if necessary.	<ul style="list-style-type: none">• text, a string that contains the text to write to the file• filename, a string that represents the name of the file to modify

Table 35 outlines the file functions that handle files.

Table 35. File utility functions

Function	Description
"allFiles()" on page 116	Reads the file names in a specified directory into an array.
"filterFile()" on page 120	Reads and filters the contents of a file to a scripting dictionary object.
"findNewest()" on page 120	Finds the latest configuration file.
"findSuitableFile()" on page 121	Finds the relevant configuration file for a product and version.
"ppread()" on page 126	Reads the contents of a file to a scripting dictionary object, further splitting each line in the file by the specified separator input parameter if that separator exists in the line.
"readFile()" on page 126	Reads each line of a file into an index entry of an array.

Appendix H. Other common functions and sub routines for Windows systems

Prerequisite Scanner has a set of other common functions and sub routines that are used in various files..

outlines the other common functions and sub routines.

Table 36. Other common functions and sub routines for Windows systems

Function or sub routine	Description
"ffirstMatch()"	Gets the first match of the search string in the array.
"getValue()" on page 134	Gets the available disk space for a specified directory.
"removeSpecialCharacters()" on page 135	Removes the trademark or other special characters to make comparisons easier.
"versionCompare()" on page 135	Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the prerequisite property passes the prerequisite check. The function expects dot-separated version strings as input parameters, for example, 1.0.0.4, 2.3, 3.40.26.7800 or 2.3.*.

ffirstMatch()

Gets the first match of the search string in the array.

Purpose

This function uses a regular expression to search for the pattern, which is passed as an input parameter, in the array that is also passed as an input parameter. When it finds the first match of the pattern in the array, it adds the value from the array to the scripting dictionary object.

Parent functions

Table 37. Parent functions calling ffirstMatch()

Parent Function, Script	Description
ud620db2level(expect, real) in DB2_Version_compare.vbs	Compares the real and expected values for the DB2 version prerequisite property.
oslevelcompare(expect, real) in OS_Version_compare.vbs	Compares the real and expected values for the OS version prerequisite property.

Syntax

ffirstmatch(patt,arr)

Input parameters

String *patt*

Contains the pattern for which to search.

Array *arr*

Contains the array in which to search for the search pattern.

Return values

Dictionary *keys*

Returns the keys for the scripting dictionary object.

getValue()

Gets the available disk space for a specified directory.

Purpose

This sub routine uses the instance of the file system object to call the `getDriveName()` function for the path input parameter and then use the `freeSpace` property to obtain the available disk space, which is then converted to MBs. The prerequisite property input parameter and its value is written to the temporary text file associated with the script file.

Scripts

Table 38. Scripts using `getValue()`

Script	Description
DEZ_01040000.vbs	Script to gather prerequisite properties and make them available to only the DEZ 01040000 configuration file
LCM_TAD_common.vbs	Script to gather prerequisite properties and make them available to only LCM 02300000 and TAD 07200000 configuration files
TAD722_impl.vbs	Script to gather prerequisite properties and make them available to only the TAD 07220000 configuration file

Syntax

`getValue fso, sKey, drvPath`

Input parameters

File system object *fso*

Instance of the file system object.

String *sKey*

Contains a string with the name of the prerequisite property and the equals symbol.

String *drvPath*

Contains the path for which to obtain the available disk space.

Return values

None

removeSpecialCharacters()

Removes the trademark or other special characters to make comparisons easier. The function is in the `/lib/common.vbs` file.

Purpose

This function calls the `Replace()` function to replace the trademark, copyright, and registered symbols with "".

Syntax

`removeSpecialCharacters(s)`

Input parameters

String s

Contains the string from which the characters must be removed

Return values

String s

Returns the string without the special characters.

versionCompare()

Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the prerequisite property passes the prerequisite check. The function expects dot-separated version strings as input parameters, for example, 1.0.0.4, 2.3, 3.40.26.7800 or 2.3.*.

Purpose

This function first handles special cases where either one or both input parameters are empty and returns return codes to represent these cases. It splits each version into several parts by the dot separator. If the last part of the version is the * wildcard character, the function considers all missing parts of the version to be the wildcard character, for example, 2.* matches 2.1 or 2.3.*. It then loops through the list of parts for each version and compares them. It then returns return codes depending upon whether the expected value is less than, equal to, or greater than the real value.

Parent functions

Table 39. Parent functions calling versionCompare

Parent Function, Script	Description
<code>cygwinVersion_compare.vbs</code>	Compares the real and expected values for the cygwin version prerequisite property.
<code>gskit7Version_compare.vbs</code>	Compares the real and expected values for the gskit Version 7 prerequisite property.
<code>gkit8Version_compare.vbs</code>	Compares the real and expected values for the gskit Version 8 prerequisite property.
<code>internetExplorer.version_compare.vbs</code>	Compares the real and expected values for Internet Explorer version prerequisite property.

Table 39. Parent functions calling versionCompare (continued)

Parent Function, Script	Description
os.servicePack_compare.vbs	Compares the real and expected values for the OS service pack prerequisite property.
os.versionNumber_compare.vbs	Compares the real and expected values for the OS version prerequisite property.

Syntax

versionCompare(ver1,ver2)

Input parameters

String ver1

Contains the expected version for a prerequisite property.

String ver2

Contains the actual version for a prerequisite property.

Return values

Integer 0

Returns the 0 return code if both input parameters are equal. The parent function returns "PASS".

Special case: Returns the 0 return code and exits if both input parameters are empty.

Integer -1

Returns the -1 return code if the first input parameter is less than the second input parameter. The parent function returns "FAIL".

Special case: Returns the -1 return code and exits if the first input parameter is empty.

Integer 1

Returns the 1 return code if the first input parameter is greater than the second input parameter. The parent function returns "PASS".

Special case: Returns the 1 return code and exits if the second input parameter is empty.

Appendix I. Common functions for UNIX systems

Prerequisite Scanner has a set of common functions in the `/lib/common_function.sh` file for running checks on UNIX-based systems.

Table 40. Functions in common_function.sh

Function	Description
"AddMG()" on page 138	Checks whether the input parameters are in MBs or GBs and adds the parameters.
"changeMG()"	Converts the input parameter to MBs or GBs for disk space or memory prerequisite properties.
"compare()" on page 138	Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the first value (real) is less than the second value (expected).
"cutdown()" on page 139	Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the first value (real) is less than the second value (expected). It then prints the difference between the two values if the first value is not less than the second value.
"findOSInfo()" on page 141	Finds the operating system version, OS release level and version, and hardware implementation data for the system.
"mes4path()" on page 140	Finds the free disk space for each mounted file system.
"mes4Path1()" on page 140	Finds the free disk space for each mounted file system on a Solaris system only.
"NFScheck()" on page 142	Checks the NFS status of mounts on a UNIX-based system.
"telnetNFS()" on page 141	Checks whether can telnet to the IP address of a mounted file system on the default port 2049.

changeMG()

Converts the input parameter to MBs or GBs for disk space or memory prerequisite properties.

Purpose

This function checks first that the function receives an input parameter. If receives an input parameter, it determines whether the value is in MBs or GBs and then converts the value to GBs or MBs respectively.

Syntax

changeMG val

Input parameters

String \$val

Contains the value for disk space or memory in MBs or GBs.

Return values

Integer 1

Returns 1 if the function does not receive an input parameter.

String *printf "%.0fM%s",mm[1]*1024,mm[2];*

Returns the value in MBs.

String *printf "%.2fG%s",mm[1],mm[2];*

Returns the value in GBs.

AddMG()

Checks whether the input parameters are in MBs or GBs and adds the parameters.

Purpose

This function checks first that the function receives input parameters. If receives an input parameters, it determines whether the value is in MBs or GBs and then adds the values.

Syntax

AddMG val1 val2

Input parameters

String \$val1

Contains the value for disk space or memory in MBs or GBs to be added to the other input parameter.

String \$val2

Contains the value for disk space or memory in MBs or GBs to be added to the other input parameter.

Return values

Integer 1

Returns 1 if the function does not receive two input parameters.

String val

Returns the added values in MBs or GBs.

compare()

Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the first value (real) is less than the second value (expected).

Purpose

This function checks first that the function receives two input parameters. If it receives them and they are not both false, it determines whether the values are in MBs or GBs and then compares the two values to check whether the first value is less than the second value. If it is, it returns a false value; otherwise it returns a pass value.

Syntax

compare real expected

Input parameters

String *\$real*

Contains the actual value for a prerequisite property.

String *\$expected*

Contains the expected value for a prerequisite property.

Return values

Integer *1*

Returns 1 if the function does not receive two input parameters.

String *"FAIL|PASS"*

Returns the string "FAIL" if the actual value is less than the expected value; otherwise, it returns the string "PASS".

cutdown()

Parses the input parameters that represent the actual and expected values for a prerequisite property and compares them to determine whether the first value (real) is less than the second value (expected). It then prints the difference between the two values if the first value is not less than the second value.

Purpose

This function checks first that the function receives two input parameters. If it receives them, it determines whether the values are in MBs or GBs and then converts them to MBs if in GBs. It then compares the two values to check whether the first value is less than the second value. If it is, it returns a "0MB" value; otherwise it returns the difference between the two values in MB.

Syntax

cutdown real expected

Input parameters

String *\$real*

Contains the actual value for a prerequisite property.

String *\$expected*

Contains the expected value for a prerequisite property.

Return values

Integer *1*

Returns 1 if the function does not receive two input parameters.

String *"FAIL|PASS"*

Returns the string "FAIL" if the actual value is less than the expected value when neither value is in MB or GB; otherwise, it returns the string "PASS".

String *"OMB|Real-ExpectedMB"*

Returns the string "OMB" if the actual value is less than the expected value; otherwise, it returns a string representation of the difference between the two converted values in MBs.

mes4path()

Finds the free disk space for each mounted file system.

Purpose

This function takes a path as input, calls the uname command to determine the operating system, then calls the NFScheck function to determine whether the system is up and the mounts. It next calls the df command to determine the free disk space for each mount on a system. It returns the value for the free disk space.

Syntax

mes4Path path

Input parameters

String *\$path*

Path to the system to check for free disk space.

Return values

Integer *1*

Returns return code 1 if the function does not receive an input parameter.

Integer *2*

Returns return code 2 if the input parameter is not a path.

String *\$NF*

Returns the free disk space for each mount.

String *"\$path Server NotAvailable Responding for \$path"*

Returns a message to state that the server for the path is unavailable.

mes4Path1()

Finds the free disk space for each mounted file system on a Solaris system only.

Purpose

This function takes a path as input, calls the uname command to determine the operating system is Solaris. It next calls the df command to determine the free disk space for each mount on the system. It returns the value for the free disk space.

Syntax

mes4Path1 path

Input parameters

String *\$path*

Path to the system to check for free disk space.

Return values

Integer *1*

Returns return code 1 if the function does not receive an input parameter.

Integer *2*

Returns return code 2 if the input parameter is not a path.

String *\$NF*

Returns the free disk space for each mount.

findOSInfo()

Finds the operating system version, OS release level and version, and hardware implementation data for the system.

Purpose

This function runs the `uname` command and parses its output for the operating system version, OS release level and version, and hardware implementation data for the system.

Syntax

`findOSInfo`

Input parameters

from *this*

copy from somewhere

Return values

String *\$oo*

Output from `uname` without the basic system information.

String *\$kk*

Operating system version

String *\$hh*

Hardware implementation represented as I for i386 hardware or Z for s390 hardware.

String *\$rr*

Operating system release level

String *\$vv*

Operating system release level version

telnetNFS()

Checks whether can telnet to the IP address of a mounted file system on the default port 2049.

Purpose

This function takes an IP as input and calls the `telnet` command to test whether remote connection is successful on the default telnet port 2049. It attempts the remote connection 10 times. If either the telnet action fails, the function returns a "FALSE" value; otherwise, it returns a "PASS" value.

Syntax

telnetNFS ipaddr

Input parameters

String *\$ipaddr*

The IP address to check whether a telnet can be performed.

Return values

String *"FALSE|TRUE"*

Returns the result of the telnet check. It returns "TRUE" if the check is successful; otherwise it returns "FALSE".

NFScheck()

Checks the NFS status of mounts on a UNIX-based system.

Purpose

This function takes a path as input and calls the mount command to get the list of mounted file systems. It calls the uname command to determine the operating system. It next calls the ping command to ping each mounted system and if it can ping, it then calls the telnetNFS function to check whether a remote connection can be performed. If either the ping or telnet actions fail, the function returns a "FALSE" value; otherwise, it returns a "PASS" value.

Syntax

NFScheck path

Input parameters

String *\$path*

Takes a valid path to a directory as its input.

Return values

Boolean value *TRUE or FALSE*

Returns TRUE if the NFS check is successful, that is, if it successfully pings the associated IP address or can use telnet to connect to the associated IP address for each filesystem; otherwise, it returns FALSE.

Example

This example of its usage is from the mes4Path() function:

```
# check if it's a path
path=`echo "$1" | sed -n '/^\//p'`
if [ -z "$path" ];then
    return 2;
else
    nfs_check_status=`NFScheck $path`
    if [ "$nfs_check_status" = "TRUE" ]; then
        case `uname` in
            ...
```

Appendix J. Other functions for UNIX systems

Prerequisite Scanner has a set of common functions in various files.

Table 41 outlines the set of functions in multiple files.

Table 41. Common functions in multiple files

Function	Description
"formatSizeDisplay()" on page 144	Takes the input parameter and appends or trims the fractional part of the input parameter to two decimal points, for example, 123MB to 123.00MB or 12.123MBs to 12.12MBs.
"versionCompare()" on page 144	Parses the input parameters that represent the actual and expected values for a prerequisite property and compares each part of the version to determine whether the prerequisite property passes the prerequisite check.

Table 42 outlines the set of functions in the UNIX-Linux/TAD722_impl.sh file for running checks for Tivoli License Compliance Manager and Tivoli Asset Discovery for Distributed.

Table 42. Common functions in TAD722_impl.sh

Function	Description
"checkSunOS()" on page 146	Checks whether the version of the Solaris operating system is for SPARC or X86 platforms.
"checkHpux()" on page 146	Checks whether the version of the HP-UX operating system is for IA64 or PARISC platforms.
"checkLinux()" on page 146	Checks whether the version of the Linux operating system is for System p®, System z, or x86 platforms.
"getSystemId()" on page 148	Calls different OS functions to check the platforms for the relevant operating system.
"getValue()" on page 147	Gets the value for a key in a specified file if the key exists.
"setValue()" on page 147	Sets the value for a key in a specified file if the prerequisite property exists.
"copyValue()" on page 147	Gets and sets the value for the prerequisite property (key) based on the product and operating system.
"parseDirParameter()" on page 149	Parses the parameter from the parameter list for the scanner's -p flag and puts its value in the list.
"getClosestExistingParentDir()" on page 148	Gets the closest parent directory or itself.

Table 42. Common functions in TAD722_impl.sh (continued)

Function	Description
"printDirSize()" on page 149	Checks the NFS status of the mounted file system and then gets the disk space of the file system or its parent directory

formatSizeDisplay()

Takes the input parameter and appends or trims the fractional part of the input parameter to two decimal points, for example, 123MB to 123.00MB or 12.123MBs to 12.12MBs

Purpose

This function counts the number of characters in the input parameter, checks whether it's a number or string, and splits the input part the whole and fractional parts. Depending upon the fractional part, it appends or trims it to two decimal places. It returns the result.

Parent scripts

The following scripts contain the function:

- ./Unix-Linux/common.sh
- LCM_TAD_common.sh

Syntax

```
formatSizeDisplay val
```

Input parameters

Integer *val*

The value to round to two decimal points.

Return values

Integer *val*

Returns the value rounded to two decimal points.

versionCompare()

Parses the input parameters that represent the actual and expected values for a prerequisite property and compares each part of the version to determine whether the first value (actual) is greater than the second value(expected).

Purpose

This function checks first that the function receives two versions as input parameters. It uses awk to parse and split each version into its parts where "." is the delimiter for splitting the value into parts. It then performs a loop to compare each part of the first version against the same part of the second version and see whether they are equal.

Parent functions

Table 43. Parent functions calling `versionCompare`

Parent Function, Script	Description
db2.home_compare.sh	Compares the real and expected values for the disk space for DB2 HOME prerequisite property.
oracle.Client_compare.sh	Compares the real and expected values for the Oracle client prerequisite property.
os.locale_compare.sh	Compares the real and expected values for the OS locale prerequisite property.
os.MozillaVersion_compare.sh	Compares the real and expected values for Mozilla Firefox prerequisite property.
os.package.perl_compare.sh	Compares the real and expected values for Perl package prerequisite property. Calls itself.
os.RAMSize_compare.sh	Compares the real and expected values for the RAM prerequisite property.
os.space_compare.sh	Compares the real and expected values for available disk space prerequisite property.
OS_Version_compare.sh	Compares the real and expected values for the OS version prerequisite property.

Syntax

`versionCompare real expected`

Input parameters

String *\$real*

Contains the actual value for a prerequisite property.

String *\$expected*

Contains the expected value for a prerequisite property.

Return values

Integer *0*

Returns the 0 return code if the real and the expected values are equal. The parent function returns "PASS".

Special case: Returns the 0 return code and exits if the function receives empty input parameters.

Integer *-1*

Returns the -1 return code if the real value is less than the expected value. The parent function returns "FAIL".

Returns the -1 return code and exits if the function receives the second input parameter is empty.

Integer *1*

Returns the 1 return code if the real value is greater than the expected value. The parent function returns "PASS".

Returns the 1 return code and exits if the function receives the first input parameter is empty.

checkHpux()

Checks whether the version of the HP-UX operating system is for IA64 or PARISC platforms.

Purpose

This function uses the -m flag of the uname command to determine whether the HP-UX operating system is for IA64 or PARISC platforms.

Syntax

checkHpux

Return values

String *HPUXIA64|HPUXPARISC*

Returns "HPUXIA64" if the -m flag is "ia64"; otherwise, it returns "HPUXPARISC".

checkLinux()

Checks whether the version of the Linux operating system is for System p, System z, or x86 platforms.

Purpose

This function uses the -m flag of the uname command to determine whether the Linux operating system is for System p, System z, or x86 platforms.

Syntax

checkLinux

Input parameters

Return values

String *LINUXPSERIES|LINUXZSERIES|LINUXX86*

Returns "LINUXPSERIES" if the -m flag is "ppc64" or "ppc". It returns "LINUXZSERIES" if the value is "s390x" or "s390"; otherwise, it returns "LINUXX86".

checkSunOS()

Checks whether the version of the Solaris operating system is for SPARC or X86 platforms.

Purpose

This function uses the -p flag of the uname command to determine whether the Solaris operating system is for SPARC or X86 platforms.

Syntax

checkSunOS

Input parameters

Return values

String *SOLARISSPARC|SOLARISX86*

Returns "SOLARISSPARC" if the -p flag is "sparc"; otherwise, it returns "SOLARISX86".

getValue()

Gets the value for a key in a specified file if the key exists.

Purpose

Syntax

getValue key file

Input parameters

String *\$key*

Contains the key to set.

String *\$file*

Contains the name of the file that contains the key.

Return values

setValue()

Sets the value for a key in a specified file if the prerequisite property exists.

Purpose

Syntax

setValue key value file

Input parameters

String *\$key*

Contains the prerequisite property to set.

String *\$value*

Contains the value for the prerequisite property.

String *\$file*

Contains the name of the file that contains the prerequisite property.

Return values

copyValue()

Gets and sets the value for the prerequisite property (key) based on the product and operating system.

Purpose

This function calls the getValue() function to get the value for the specified prerequisite property for the product and operating system. It then calls the setValue() function to set the value for the prerequisite property in the prerequisite scanner file.

Syntax

copyValue key file

Input parameters

String *\$key*

Contains the key to get and set.

String *\$file*

Contains the name of the file that contains the key.

Return values

getSystemId()

Calls different OS functions to check the platforms for the relevant operating system.

Purpose

This function calls various OS functions to determine the platforms for the relevant operating system.

Syntax

getSystemId

Input parameters

Return values

String *AIX|Linux*

Returns "AIX" or "Linux" if the product is Tivoli License Compliance Manager and the OS is either AIX or Linux or "AIX" if the product is Tivoli Asset Discovery for Distributed and the OS is AIX.

getClosestExistingParentDir()

Gets the closest parent directory or itself.

Purpose

Syntax

getClosestExistingParentDir dirpath

Input parameters

String *\$dirpath*

Contains the path to obtain its parent directory or itself.

Return values

String *dirpath*

Returns the parent directory or itself

parseDirParameter()

Parses the parameter from the parameter list for the scanner's -p flag and puts its value in the list.

Purpose

Syntax

Input parameters

String

Return values

printDirSize()

Checks the NFS status of the mounted file system and then gets the disk space of the file system or its parent directory.

Purpose

This function first calls the NFScheck function to determine the NFS status of the directory. If the status is true, it calls the getClosestExistingParentDir function to return the directory or its parent directory, and then use the df command to get the amount of free disk space. It finally calls the formatSizeDisplay function to round the value to decimal points.

Syntax

`printDirSize dirpath`

Input parameters

String *\$dirpath*

Contains the path to the directory for which to get the free disk space.

Return values

Integer *dsize*

Returns the amount of free disk space to within two decimal points.

String *"NFS_NOT_AVAILABLE"*

Returns that the mounted filesystem is not available.

Appendix K. Logging utility functions for UNIX systems

Prerequisite Scanner has a set of common logging functions in the `/lib/common_function.sh` file for writing debugging and trace data to log files.

Table 44 describes the logging utilities.

Table 44. Logging utility functions on UNIX systems

Function	Description	Input parameters
<code>wrlTrace log_str1 log_str2</code>	Writes <i>log_str1</i> and <i>log_str2</i> strings to the trace file, with the timestamp	<i>log_str1</i> and <i>log_str2</i> , trace strings that represent the action and collector being executed and which to log in the trace file. For example: <pre> ~wrlTrace Starting os.lib~ ~wrlTrace Executing os.lib~ ~wrlDebug Starting os.lib~ ~wrlDebug Expected libXp ` ss=~./os.lib libXp libXp~ ~wrlTrace Finished os.lib~ echo "os.lib.libXp=\$ss" ~wrlDebug Finished os.lib~ ~wrlDebug OutPutValueIs \$ss~ ~wrlTrace Done os.lib~ </pre>
<code>wrlTraceFuncStart fn_name</code>	Passes the <i>fn_name</i> function to <code>wrlTrace()</code>	<i>fn_name</i> , trace string that represents the function that is just called. For example: <pre>~wrlTraceFuncStart "\$1"~</pre>
<code>wrlTraceFuncExit fn_name</code>	Passes the <i>fn_name</i> function to <code>wrlTrace()</code>	<i>fn_name</i> , trace string that represents the function that just completed. For example: <pre>~wrlTraceFuncExit "\$1"~</pre>
<code>wrlDebug log_str1 log_str2</code>	Passes <i>log_str1</i> and <i>log_str2</i> strings to <code>wrlDebugGeneric()</code>	<i>log_str1</i> and <i>log_str2</i> , debug strings that represent the action and collector being executed and which to log in the debug file. For example: <pre> ~wrlTrace Starting os.lib~ ~wrlTrace Executing os.lib~ ~wrlDebug Starting os.lib~ ~wrlDebug Expected libXp ` ss=~./os.lib libXp libXp~ ~wrlTrace Finished os.lib~ echo "os.lib.libXp=\$ss" ~wrlDebug Finished os.lib~ ~wrlDebug OutPutValueIs \$ss~ ~wrlTrace Done os.lib~ </pre>
<code>wrlDebugFuncStart fn_name</code>	Passes the <i>fn_name</i> function to <code>wrlDebug()</code>	<i>fn_name</i> , debug string that represents the function that is just called. For example: <pre>~wrlDebugFuncStart "\$1"~</pre>
<code>wrlDebugFuncExit fn_name</code>	Passes the <i>fn_name</i> function to <code>wrlDebug()</code>	<i>fn_name</i> , debug string that represents the function that just completed. For example: <pre>~wrlDebugFuncExit "\$1"~</pre>
<code>wrlDebugFuncReturn result_value</code>	Writes the returned <i>result_value</i> result for the function to the log file	<i>result_value</i> , debug string that represents the returned value from the function. For example: <pre>~wrlDebugFuncReturn "\$versionCompare"~</pre>

Table 44. Logging utility functions on UNIX systems (continued)

Function	Description	Input parameters
<code>wr1DebugFuncParam param1 param2</code>	Passes the <i>param1</i> and <i>param2</i> parameters to <code>wr1DebugFunc()</code>	<i>param1</i> and <i>param2</i> , debug strings that represent parsed section titles, parsed qualifiers or parsed input arguments to called functions. For example: `wr1DebugFuncParam "OSArch" "\$3"~`
<code>wr1DebugGeneric formatspec log_str1 log_str2</code>	Writes <i>log_str1</i> and <i>log_str2</i> strings to the debug file, formatted by the <i>formatspec</i> string argument	<ul style="list-style-type: none"> <i>log_str1</i> and <i>log_str2</i>, strings that represent specific data to log on a line in the debug file <i>formatspec</i>, the string argument to place after the timestamp but before the left alignment of the log strings and newline character For example: `wr1DebugGeneric "" "\$1" "\$2"~`
<code>wr1DebugFunc str</code>	Passes a tab character and the <i>str</i> input parameter to <code>wr1DebugGeneric()</code>	<i>str</i> , string that represents data to log, that is, the status of a check or an action being performed. For example: `wr1DebugFunc "Reading config file and parsing using parse array..." ~`
<code>wr1LogFuncStart str</code>	Passes the <i>str</i> input parameter to <code>wr1TraceFuncStart()</code> and <code>wr1DebugFuncStart()</code>	<i>str</i> , string that represents data to log, that is, the name of the function being called. For example: `wr1LogFuncStart "main()"~`
<code>wr1LogFuncExit str</code>	Passes the <i>str</i> input parameter to <code>wr1TraceFuncExit()</code> and <code>wr1DebugFuncExit()</code>	<i>str</i> , string that represents data to log, that is, the name of the function being exited. For example: `wr1LogFuncExit "main()"~`

Notices

This information was developed for products and services offered in the U.S.A. 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 give 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 might 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.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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. 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 IBM's application programming interfaces.

If you are viewing this information in softcopy form, the photographs and color illustrations might not be displayed.

Trademarks

IBM, the IBM logo, and ibm.com[®] are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine and Cell/B.E. are trademarks of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

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

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

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

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Support information and feedback

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides different ways for you to obtain the support you need, such as online or IBM Support Assistant. You can also provide feedback or submit product requests for enhancements.

Online

The following sites contain troubleshooting information:

- Go to IBM Prerequisite Scanner page at IBM Support Portal.
- Go to the Prerequisite Scanner topics at Service Management Connect. Feel free to contribute to these topics.

Use the following sites to provide feedback, submit requests, or discuss Prerequisite Scanner:

- Go to the Prerequisite Scanner topics at Prerequisite Scanner at Service Management Connect. Feel free to contribute to these topics.
- Use the Integrated Service Management Message Board at Service Management Connect.
- Submit or review product request enhancements for Prerequisite Scanner at Tivoli RFE Community.

IBM Support Assistant

The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to <http://www.ibm.com/software/support/isa>

Index

A

- access permissions qualifiers
 - description 8, 95
- adding
 - prerequisite properties, custom 45
 - prerequisite properties, predefined 45
 - product codes 43
 - sections 45
- application subtypes
 - description 5, 95

C

- categories
 - Autonomic Deployment Engine 91
 - common 86
 - connectivity 92
 - DB2 92
 - installed software 106
 - Internet Explorer 93
 - MS SQL Server 92
 - network 93
 - operating system 95
 - Oracle 94
 - prerequisite properties 1, 4
 - UNIX network 107
 - user 106
 - Windows network 107
- codename.cfg
 - adding product codes 43
 - description 12
 - updating 43
- collectors
 - description 20
 - UNIX
 - creating 21, 52
 - description 21
 - format 21
 - inputs 109
 - location 21
 - naming conventions 21
 - packageTest.sh, updating 21, 52, 53
 - predefined 109
 - rules 21
 - shell 21
 - standard output 21
 - Windows
 - common 20, 48
 - creating 20, 48, 50
 - description 20
 - format 20
 - location 20
 - naming conventions 20
 - product-specific 20, 50
 - rules 20, 48
 - standard output 20
 - VBScript 20

- command-line interface
 - output format 23, 61
 - running Prerequisite Scanner 61, 67
- common
 - collectors, UNIX 52
 - collectors, Windows 20, 48
 - evaluators, UNIX 22
 - evaluators, Windows 22
- common category
 - description 4
 - predefined prerequisite properties 86
- configuration files
 - checks, UNIX 42
 - checks, Windows 41
 - creating 44
 - description 13
 - example 13, 44
 - file extension, .cfg 13, 44
 - format 13, 44
 - location 13, 44
 - naming conventions 13, 44
 - operating systems, supported 13, 44
 - predefined 81
 - prerequisite properties 13, 44
 - product versions 13, 44
 - rules 13, 44
 - sections 13, 14, 44
 - standard output 13, 44
- connectivity category
 - description 4, 92
- CPU name 86
- CPU section
 - description 14
- CPUArch section
 - description 14
- creating
 - collectors, UNIX 21, 52
 - collectors, Windows 20
 - common 48
 - product-specific 50
 - configuration files 44
 - evaluators, UNIX 22, 59
 - evaluators, Windows 22, 55

D

- DB2 category
 - description 4
 - predefined prerequisite properties 92
- de category
 - predefined prerequisite properties 91
- debug parameter
 - description 61
 - logging utility functions 151
 - logging utility sub routines 129
 - precheck.log 23, 61, 69, 129
 - prs.debug 23, 61, 71, 151
- debugging
 - debug 23
 - log files 23, 69, 71
 - Prerequisite Scanner 23, 69

- detail parameter
 - description 61
 - output formats 23, 61
- directory subtypes
 - description 5, 95
- Disk 86

E

- enhancements 36
- environment variables category
 - description 4
- environment variables section
 - description 14
- evaluators
 - UNIX
 - creating 22, 59
 - description 22
 - format 22
 - location 22
 - naming conventions 22
 - rules 22, 59
 - shell 22, 59
 - standard output 22
 - Windows
 - common 22
 - creating 22, 55
 - description 22
 - format 22
 - location 22
 - naming conventions 22
 - rules 22, 55
 - standard output 22
 - VBScript 22, 55
- extending
 - checks, UNIX 42
 - checks, Windows 41
 - tasks, UNIX 42
 - tasks, Windows 41

F

- file system qualifiers
 - description 8, 95
- format
 - collectors, UNIX 21
 - collectors, Windows 20
 - configuration files 13, 44
 - evaluators, UNIX 22
 - evaluators, Windows 22
 - prerequisite properties 1
 - sections 14

I

- IBM Support Assistant 157
- installation directories 39, 40, 68
- installed software category
 - description 4

- installed software category *(continued)*
 - predefined prerequisite properties 106
- installing 39, 40
- Internet Explorer category
 - description 4
 - predefined prerequisite properties 93
- ISA 157

L

- library subtypes
 - description 5, 95
- location
 - collectors, UNIX 21
 - collectors, Windows 20, 48
 - evaluators, UNIX 22, 59
 - evaluators, Windows 22, 55
- log file
 - output format 23
 - precheck.log 23, 69
 - prs.debug 23, 71
 - prs.trc 23, 71
- logging utility functions
 - prs.debug 151
 - prs.trc 151
- logging utility sub routines
 - precheck.log 129

M

- Memory 86
- MS SQL Server category
 - predefined prerequisite properties 92

N

- naming conventions
 - collectors, UNIX 21
 - collectors, Windows 20
 - configuration files 13, 44
 - evaluators, UNIX 22
 - evaluators, Windows 22
 - prerequisite properties 1
 - sections 14
- network category
 - description 4
 - predefined prerequisite properties 93

O

- operating system category
 - description 4
 - predefined prerequisite properties 95
- Oracle category
 - description 4
 - predefined prerequisite properties 94
- os category
 - See* operating system category
- OS Version 86
- OSArch section
 - description 14
- OSType section
 - description 13, 14

- output formats
 - command-line interface 23
 - location 23
 - log file 23
 - return codes 74
 - text file 23
- outputDir parameter
 - description 61

P

- p flag
 - description 61
- package subtypes
 - description 5, 95
- packageTest.sh
 - collectors, UNIX 21
 - updating 53
- path names 68
- path parameter
 - description 61
- precheck.log
 - debug parameter 23, 61, 69, 129
 - debugging log file 23, 69, 129
 - logging utility sub routines 129
- prereq_checker
 - flags 61, 67
 - parameters 61, 67
 - running 67
 - syntax 61, 67
- Prerequisite Checker Wiki 157
- prerequisite properties
 - adding, custom 45
 - adding, predefined 45
 - categories 1, 4, 45, 47, 86, 91, 92, 93, 94, 95, 106, 107
 - collectors 20, 21
 - configuration files 13, 44
 - description 1
 - evaluators 22
 - format 1, 45, 47
 - naming conventions 1, 45, 47
 - qualifiers 1, 8
 - reference 85
 - subtypes 1, 45, 47
 - types 1
 - updating, custom 47
 - updating, predefined 47
 - updating, qualifier values 47
- Prerequisite Scanner
 - architecture 1, 34
 - batch 1
 - binary 61
 - collectors 20
 - configuration files 81
 - debugging 23
 - description 1
 - enhancements 36
 - extending 41, 42
 - installation directories 39, 40, 68
 - installing 39, 40
 - new features 36
 - output formats 23
 - prerequisite properties 1
 - prerequisites 39
 - product codes 12, 43, 77
 - results 23

- Prerequisite Scanner *(continued)*
 - return codes 74
 - root directory 68
 - running 61, 67
 - scanning process 34
 - script syntax 61
 - shell 1
 - uninstalling 40
 - VBScript 1
 - version 36
- prerequisites 39
- product codes
 - codename.cfg 12, 43, 77
 - configuration files 81
 - description 12
 - parameter 12, 61
 - predefined 77
 - Prerequisite Scanner script 12, 61
- product versions
 - configuration files 13, 44
 - parameter 12, 61
 - Prerequisite Scanner script 12, 61
 - product codes 12
- product-specific
 - collectors, Windows 20, 48, 50
- prs.debug
 - debug parameter 23, 61, 71, 151
 - debugging log file 23, 71, 151
 - logging utility functions 151
- prs.trc
 - logging utility functions 151
 - trace log file 23, 71, 151
 - trace parameter 23, 61, 71, 151

Q

- qualifiers
 - format 8
 - naming conventions 8
 - predefined 8, 95
 - prerequisite properties 1, 8
 - rules 8

R

- results
 - command-line interface 23
 - log file 23
 - text file 23
- return codes 74
- rules
 - collectors, UNIX 21
 - collectors, Windows 20, 48
 - configuration files 13, 44
 - evaluators, UNIX 22, 59
 - evaluators, Windows 22, 55
 - product codes 43
 - product codes, 12
- running
 - Prerequisite Scanner 61, 67

S

- scanning process 34
- script subtypes
 - description 5, 95

- scripts
 - batch 1
 - shell 1
 - VBScript 1
- sections
 - adding 45
 - configuration files 13, 14, 44
 - description 14
 - format 14
 - naming conventions 14
 - section categories 14
- service subtypes
 - description 5, 95
- Software Support 157
- special characters
 - prerequisite properties 1
 - Prerequisite Scanner script 61
- standard output
 - collectors, UNIX 21
 - collectors, Windows 20
 - configuration files 13, 44
 - evaluators, UNIX 22
 - evaluators, Windows 22
- subtypes
 - prerequisite properties 1, 5
- support assistant 157

T

- text file
 - output format 23
 - output formats 23
 - results 23
 - results.txt 23
- trace parameter
 - description 61
 - logging utility functions 151
 - prs.trc 23, 61, 71, 151
- type qualifiers
 - description 8, 95
- types
 - collectors 20
 - evaluators 22
 - prerequisite properties 1

U

- unit qualifiers
 - description 8, 95
- UNIX network category
 - predefined prerequisite properties 107
- updating
 - packageTest.sh 53
 - prerequisite properties, custom 47
 - prerequisite properties, predefined 47
 - qualifier values 47
 - qualifiers 8
- user category
 - description 4
 - predefined prerequisite properties 106

V

- VBScript
 - collectors, Windows 20
 - evaluators, Windows 22

W

- Windows network category
 - predefined prerequisite properties 107
- Windows Script Host 20, 22

X

- xmlResult
 - XML result parameter 61



Printed in USA