CIMPLE 2.0.18 Release Notes

Karl Schopmeyer
18 May 2011

Table of Contents

1 Introduction

This document introduces CIMPLE 2.0.18 and explains what has changed since the last public release (CIMPLE 2.0.16). If you are unfamiliar with the major changes introduced by CIMPLE 2.0.16, you might want to review them on the cimplewbem web site.

This was done because we specifically included new functionality in this release.

NOTE: This release note covers only the version 2.0.18 release.

The next chapter describes what is new in this release. Chapter 3 discusses bugs fixed by this release. Chapter 4 explains how to migrate providers developed with earlier versions.

2 What's New?

This chapter covers new capabilities introduced by CIMPLE 2.0.18. Note that the release notes were not originally attached to the 2.0.0 release but were available with the 2.0.6 update to this release.

Note also that release notes were not consistently relese for versions between 2.0.6 and 2.0.16. This note covers changes made specifically between 2.0.16 and 2.0.18.

2.1 New Functionality Overview

The following new functionality has been implemented:

1. Increased control of logging through both runtime and compile time parameters

2. Addition of tracing for the adapters with implementation of this in the Pegasus adapter.

3. Addition of class to control dynamic configuration of CIMPLE providers from the providers themselves.

In addition there have been:

1. Extensions to test providers and unit tests. 2. Correction of some issues in the provider adapters, primarily the Pegasus adapter.

2.2 Logging Modifications

As the requirement for logging provider information moves from development to production releases of providers, more control over logging was requested.

This version implements:

1. a compile configuration parameter to compile of the log macros defined in the log.h file (CIMPLE_FATAL, CIMPLE_ERR, CIMPLE_WARN, CIMPLE_INFO, and CIMPLE_DBG. if the configure flag —-disable_log_macros is set all of these macros are completely disable. Note that direct calls to the log and vlog functions in log.h are NOT disabled but we encourage users to implement logging through the macros.

2. Extension of logging to allow the user to set maximum size limits on the output log file and to create a backup file each time this limit is reached. In addition, the capability exists to limit the number of backup files.

The runtime parameters to control log file size and number of backups are

MAX_LOG_FILE_SIZE=<integer> where integer is the approximate number of bytes in the log file before it is rolled over into an archive file. If this <integer> is set to zero, the log file is considered to be unlimited (i.e. the same behavior as earlier versions of CIMPLE). If this parameter is not included in the .cimplerc config file it is assumed to be zero(0).

MAX_LOG_BACKUP_FILES=<integer> where <integer> is between 0 and 9. This defines the maximum number of backup files that will be maintained. Each time the log messages file is rolled over it is renamed messages.1 and all existing archived files are incremented by one (i.e. the existing messages.1 becomes messages.2). The oldest file is removed if its archive number is greater than the value of MAX_LOG_BACKUP_FILEs

There is a special behavior if MAX_LOG_BACKUP_FILES =0. In this case, the log file is simple removed and restarted.

3. Addition of a runtime control for log output with a new runtime configuration parameter.

ENABLE_LOGGING="true" | "false" This runtime configuration variable absolutely controls the output of log files. If set to false, all logging is turned off.

2.3 Dynamic control of configuration parameters

A new class has been added to CIMPLE to allow providers to dynamically control at least some of the runtime configuration parameters from the provider iteself. Today this includes the following functions ad defined in src/cimple/CimpleConfig.h:

Set the log level to the level defined by the input level param level String parameter that defines log level. Must be one of the predefined constants —return true if set correctly Else false

static bool setLogLevel(String& level);

Set the log level to the level defined by the enum variable input —param level - Level to set —return bool true if operation executed.

static bool setLogLevel(Log_Level level);

Get the current log level —return Log_Level

static Log_Level getLogLevel();

Get the String that defines the current log level (i.e. DBG, WARN, etc.) —return String that defines the current log level

static const char getLogLevelString();

Set the HOME environment variable that defines the location of the various files used by CIMPLE. There is no corresponding get method. —param env_var —return bool true if accepted

static bool setHomeEnv(const charenv_var);

Set the maximum size in bytes of the log file before it is pruned and a new file started. —param newSize uint32 variable that defines the maximum size in bytes. If zero (0) the size is unlimited. This is the default unless the log size varaible is set either by this method or the CIMPLE config file. —return bool Returns true if the input is accepted.

static bool setLogFileMaxSize(uint32 newSize);

Get the value of the variable that defines the maximum size of the log file. —return uint32 - value of the maximum File size variable.

static uint32 getLogFileMaxSize();

Set the internal variable that controls the maximum number of log files that CIMPLE will maintain. If the logFileMaxSize is set to a non-zero value, each time the log file reaches that size, it is closed, the name changed and a new log file started. This variable controls the maximum number of backup log files maintained. When this limit is reached, old files will be deleted. If the current value set by this variable is zero (0) no old files will be maintained and the current file will be deleted when it reaches the maximum size. Zero is the default value for the variable if it is not set by this method or the CIMPLE config file. The backup log files are numbered messages.n where n is the number of backup file, 1 being the newest. —param number uint32 integer between 0 and 9 that defines maximum number of backup files to be maintained. —return bool Returns true if the method is accepted. NOTE: Maximum number of backup files is 9. Also there is a cost to keeping many files because of the renaming process.

static bool setLogMaxBackupFiles(uint32 number);

Get the current value for the variable that controls the maximum number of log files maintained —return uint32

static uint32 getLogMaxBackupFiles();

Set the logging enabled state to either true or false. When the logging state is true, logging is enabled. Otherwise it is completely disabled. The default before being set by either this method or the CIMPLE configuration files is true. —param netState Boolean defining the new state —return bool

static bool setLogEnabledState(bool netState);

Get current logEnabledState —return bool Returns the current state

static bool getLogEnabledState();

Forces a reread of the CIMPLE configuration file. Normally this is for testing only.

static void readConfig();

Remove Log files. Cleans and restarts the log files. This removes any log backup files also. Use this ONLY if you want to completely delete log files. Removes all files in the log directory with the logfile name. —return true if operation successfully executed. False indicates some error in trying to remove files.

static bool removeLogFiles();

Enable or Disable the logging calls from log macros. This corresponds to the ENABLELOGGING runtime configure parameter in .cimplerc —param newState - The state to which this variable is to be set. If false, logging is immediatly disabled (no more log entries are generated) —return bool defining the previous state before newState is set into the variable

static bool setEnableLoggingState(bool newState);

Get the current state of the enable logging variable. —return true if logging enabled.

static bool getEnableLoggingState();

NOTE: Please consider this class experimental for this version as it is the first release of this interface.

2.4 Adapter Tracing

Created a generalized tracing capability (that uses the log for output) for the adapters. In the past, there was very little tracing capability for the Pegasus adapter and today the cmpi adapter has tracing permanently integrated and activated.

With this capability tracing can be controlled at the compile level with the configuration parameter –enable-adapter-trace which will enable compile of tracing statements.

When activated, tracing outputs trace statements from the defined adapter to the log file.

Activation of this capability is a compile time options with the –enable-adapter-trace option to configure and configure.bat. This retains the original behavior of disable tracing/logging of adapters with no option set for the Pegasus (C++) provider. The CMPI provider adapter retains its current and is not controlled through this configuration parameter today.

For this release only the Pegasus adapter has been modified for this capability and control is somewhat limited since it uses the same runtime parameters as logging with all tracing controlled through the severity option (all adapter traces are at the level LL_DBG). This will be extended in future releases to further separate adapter traces from provider tracing.

It is expected that adapter tracing will be primarily used in development and can be compiled off in release mode.

In addition functions have been added for both the cmpi and pegasus adapters to be able to print or log all the information in cmpi or pegasus instances as they pass through the adapters. Note that these functions have not been fully implemented in the providers themselves but that the functions exist with limited usage in the Pegasus provider.

2.5 Utility extensions (genclass, genprov, genmod, etc.)

Addition of a common parameter for genclass, genprov, genmod, genproj to allow the class list to be provided via a file input. This was provided because on windows at least, some users were reaching the limit of the cmd buffer with single command line inputs to these utilities.

The new option is -F <filename> where filename is the name of a file that contains the list of classes one class per line.

The original functionality using -f as an option in genclass has been maintained also for backward compatibilty.

3 Bug Fixes

This release fixes the bugs described below (all critical bugs were addressed by earlier maintenance releases).

See the ChangeLog and version diff file for more information about bug fixes.

4 Migration Notes

Always regenerate classes, providers, and modules when using a new version of CIMPLE. This is a trivial matter of running `genrpoj` as follows:

```
$ genproj MODULE-NAME CLASS-1 CLASS-2 ... CLASS-N
```

This regenerates the classes and module and will patch your providers if necessary. This operation will not require any rework on your part. Just regnerate, clean, and remake.

5 Platform Support

CIMPLE 2.0.18 supports the following platforms.

- Linux-X86 32-bit, GNU C++
- Linux-X86 64-bit, GNU C++
- Linux-IA64 64-bit, GNU C++
- Linux-S390 32-bit, GNU C++
- Linux-S390 64-bit, GNU C++
- Linux-PPC 32-bit, GNU C++
- Linux-PPC 64-bit, GNU C++
- Darwin-X86-32-bit, GNU C++
- Solaris-SPARC-64-bit, GNU C++
- Solaris-X86 32-bit and 64-bit, CC Compiler
- VxWorks-XScale-32-bit, GNU C++
- Windows-X86-32-bit, MSVC
- Windows-X86-64-bit, MSVC
- Solaris-X86-32bit and 64 bit CC compiler
- Solaris-SPARC 64-bit CC compiler
- Windows-x86-32-bit WMI provider MSVC