PATROL® API Reference Manual

Version 3.5

January 15, 2003

bmcsoftware

Copyright 2003 BMC Software, Inc. All rights reserved.

BMC Software, the BMC Software logos, and all other BMC Software product or service names are registered trademarks or trademarks of BMC Software, Inc. All other trademarks belong to their respective companies.

BMC Software considers information included in this documentation to be proprietary and confidential. Your use of this information is subject to the terms and conditions of the applicable End User License Agreement for the product and the proprietary and restricted rights notices included in this documentation.

Restricted Rights Legend

U.S. Government Restricted Rights to Computer Software. UNPUBLISHED -- RIGHTS RESERVED UNDER THE COPYRIGHT LAWS OF THE UNITED STATES. Use, duplication, or disclosure of any data and computer software by the U.S. Government is subject to restrictions, as applicable, set forth in FAR Section 52.227-14, DFARS 252.227-7013, DFARS 252.227-7014, DFARS 252.227-7015, and DFARS 252.227-7025, as amended from time to time. Contractor/Manufacturer is BMC Software, Inc., 2101 CityWest Blvd., Houston, TX 77042-2827, USA. Any contract notices should be sent to this address.

Contacting BMC Software

You can access the BMC Software Web site at **http://www.bmc.com**. From this Web site, you can obtain information about the company, its products, corporate offices, special events, and career opportunities.

United States and Canada

Address BMC Software, Inc.
2101 CityWest Blvd.
Houston TX 77042-2827

Telephone 713 918 8800 or
800 841 2031

Fax 713 918 8000

Outside United States and Canada

Telephone (01) 713 918 8800

Fax (01) 713 918 8000

Customer Support

You can obtain technical support by using the Support page on the BMC Software Web site or by contacting Customer Support by telephone or e-mail. To expedite your inquiry, please see “Before Contacting BMC Software.”

Support Web Site

You can obtain technical support from BMC Software 24 hours a day, 7 days a week at **http://www.bmc.com/support.html**. From this Web site, you can

- read overviews about support services and programs that BMC Software offers
- find the most current information about BMC Software products
- search a database for problems similar to yours and possible solutions
- order or download product documentation
- report a problem or ask a question
- subscribe to receive e-mail notices when new product versions are released
- find worldwide BMC Software support center locations and contact information, including e-mail addresses, fax numbers, and telephone numbers

Support by Telephone or E-mail

In the United States and Canada, if you need technical support and do not have access to the Web, call 800 537 1813. Outside the United States and Canada, please contact your local support center for assistance. To find telephone and e-mail contact information for the BMC Software support center that services your location, refer to the Contact Customer Support section of the Support page on the BMC Software Web site at **www.bmc.com/support.html**.

Before Contacting BMC Software

Before you contact BMC Software, have the following information available so that Customer Support can begin working on your problem immediately:

- product information
  - product name
  - product version (release number)
  - license number and password (trial or permanent)
- operating system and environment information
  - machine type
  - operating system type, version, and service pack or other maintenance level such as PUT or PTF
  - system hardware configuration
  - serial numbers
  - related software (database, application, and communication) including type, version, and service pack or maintenance level

- sequence of events leading to the problem
- commands and options that you used
- messages received (and the time and date that you received them)
  - product error messages
  - messages from the operating system, such as `file system full`
  - messages from related software

BMC Software, Inc., Confidential and Proprietary Information

Contents

About This Manual xv

Chapter 1 Overview of the PATROL API

Introduction 1-2
Fitting the PATROL API into PATROL 1-2
Interacting with PATROL Agents 1-3
Integrating Applications and Managing the Enterprise 1-3
Accessing the PEM Engine 1-4
Connecting User Programs to the PATROL Agent 1-5
Retrieving Information from a PATROL Agent 1-6
Using Blocking or Non-blocking (Event-Driven) Functions . . . 1-6
Understanding Return Values for Blocking Functions 1-7
Finding Information about Functions for PATROL Agents 1-7
Using the PATROL API to Connect to the PEM Engine 1-10
Benefits of Connecting with the PATROL API 1-10
Receiving Events from the PEM Engine 1-11
Sending Events to the PEM Engine 1-12
Finding Information about Functions for the PEM Engine 1-12
Example of a PATROL API Blocking Program 1-14
Example of a PATROL API Non-Blocking Program 1-18

Chapter 2 Example Programs

Overview 2-2
Where to Find the Examples 2-2
Sending and Receiving Events from One PATROL Agent 2-3
Sending and Receiving Events from Multiple PATROL Agents 2-11
Working with Event Reports in Blocking Mode 2-22
Requesting Information in Blocking Mode 2-31

BMC Software, Inc., Confidential and Proprietary Information

Dumping Events Using Blocking Mode 2-43

Chapter 3 **Data Structures**

Overview of Type Definitions 3-3
PemnApplAttrArray 3-5
PemnApplAttrEnum 3-6
PemnApplObjHandle 3-7
PemnBDisconnectCallback 3-8
PemnClientData 3-9
PemnCommCallback 3-10
PemnCommHandle 3-11
PemnArchiveEventHandler 3-12
PemnEventHandle 3-13
PemnEventStatusEnum 3-14
PemnEventTypeEnum 3-15
PemnGetApplObjHandler 3-16
PemnGetEventClassHandler 3-17
PemnGetEventClassListHandler 3-18
PemnGetInstObjHandler 3-19
PemnGetListHandler 3-20
PemnGetParamObjHandler 3-21
PemnGetVarObjHandler 3-22
PemnInstAttrArray 3-23
PemnInstAttrEnum 3-24
PemnInstObjHandle 3-25
PemnParamAttrArray 3-26
PemnParamAttrEnum 3-27
PemnParamObjHandle 3-28
PemnQueryEventHandler 3-29
PemnReceiveEventHandler 3-30
PemnReportEventHandler 3-32
PemnReportHandle 3-33
PemnUserMessageHandler 3-34
PemnVarObjHandle 3-35

Chapter 4 **Event Access Functions**

PemnGetArgs() 4-2
PemnGetClassName() 4-3
PemnGetDesc() 4-4
PemnGetDiary() 4-5

BMC Software, Inc., Confidential and Proprietary Information

PemnGetEid() 4-6
PemnGetEventCatalogName() 4-7
PemnGetExpectancyStr() 4-8
PemnGetHandler() 4-9
PemnGetNode() 4-10
PemnGetNseverity() 4-11
PemnGetOrigin() 4-12
PemnGetOwner() 4-13
PemnGetSourceID() 4-14
PemnGetStatus() 4-15
PemnGetTime() 4-16
PemnGetType() 4-17

Chapter 5 Connection Management Functions
PemnBClose() 5-2
PemnBOpen() 5-3
PemnBPing() 5-5
PemnClose() 5-7
PemnGetCommAttributes() 5-8
PemnGetLocalPort() 5-10
PemnLoop() 5-11
PemnOpen() 5-12
PemnSetCommAttributes() 5-15
PemnSetLocalPort() 5-17
PemnSetNoAgentHeartbeat() 5-18

Chapter 6 Security Management Functions
PemnEncrypt() 6-2
PemnSendIdentity() 6-3

Chapter 7 Send and Receive Functions
PemnReceive() 7-2
PemnSend(), PemnBSend() 7-7

Chapter 8 Event Management Functions
PemnAddDiary() 8-2
PemnBEventScheduleAdd() 8-4
PemnBEventScheduleDelete() 8-7
PemnBEventScheduleList() 8-9
PemnBEventScheduleModify() 8-11

BMC Software, Inc., Confidential and Proprietary Information

PemnBEventScheduleResume() 8-14
PemnBEventScheduleSuspend() 8-16
PemnDefineEventClass() 8-18
PemnEventArchive(), PemnBEventArchive() 8-21
Specifying the Pemn(B)EventArchive() Output Format 8-27
PemnEventManage() 8-30
PemnEventManageIfMatch(), PemnBEventManageIfMatch() 8-32
PemnEventQuery(), PemnBEventQuery() 8-39
PemnEventRangeManage() 8-46
PemnEventRangeQuery() 8-49
PemnEventReport(), PemnBEventReport() 8-52
PemnFlushEvent() 8-58
PemnGetEventClass(), PemnBGetEventClass() 8-59
Specifying the PemnGetEventClass() Function Output Format .8-61
PemnGetEventClassList(), PemnBGetEventClassList() 8-63
PemnSetEventClassCmd() 8-66
PemnSetPersistentFilter(), PemnBSetPersistentFilter() 8-69

Chapter 9 Application Management Functions

PemnBPslExec() 9-2
PemnBSetVarObj() 9-4
PemnGetApplList(), PemnBGetApplList() 9-6
PemnGetApplObj(), PemnBGetApplObj() 9-9
PemnGetApplObjAttributes() 9-12
PemnGetApplObjInfo() 9-13
PemnGetComputerParamList(), PemnBGetComputerParamList() . .9-15
PemnGetComputerParamObj(), PemnBGetComputerParamObj() . .9-18
PemnGetInstList(), PemnBGetInstList() 9-21
PemnGetInstObj(), PemnBGetInstObj() 9-24
PemnGetInstObjAttributes() 9-27
PemnGetInstObjInfo() 9-28
PemnGetParamList(), PemnBGetParamList() 9-30
PemnGetParamObj(), PemnBGetParamObj() 9-33
PemnGetParamObjAttributes() 9-36
PemnGetParamObjInfo() 9-37
PemnGetVarList(), PemnBGetVarList() 9-39
PemnGetVarObj(), PemnBGetVarObj() 9-42

Chapter 10 Data Management Functions

PemnPutFile(), PemnBPutFile() 10-2

BMC Software, Inc., Confidential and Proprietary Information

PemnPutStream(), PemnBPutStream() 10-5

Chapter 11 Miscellaneous Functions

PemnCheckInit() 11-2
PemnGetLastMessage() 11-3
PemnRegisterUserMessageHandler() 11-4

Chapter 12 Specific Blocking Mode Functions

PemnBExitFunction() 12-2
PemnBGetTimeout() 12-3
PemnBSetTimeout() 12-4

Chapter 13 PATROL COM/DCOM Interface

Overview 13-3
COM/DCOM Files 13-4
Configuring the PATROL COM/DCOM Interface 13-4
    Configuring COM Servers 13-4
    Configuring DCOM Clients 13-6
Accessing the PATROL Agent COM Server 13-6
    Through the Dispatch Interface 13-7
    Through a Custom Interface Program 13-8
IPatrolAgent Interface Functions 13-9
    annotate() 13-11
    annotate_get() 13-13
    blackout() 13-15
    change_state() 13-17
    create() 13-19
    destroy() 13-21
    exists() 13-23
    get() 13-25
    getenv() 13-27
    get_psl_errno() 13-29
    get_ranges() 13-30
    get_vars() 13-32
    history() 13-34
    is_var() 13-36
    PslExecute() 13-38
    print() 13-40
    set() 13-41
    set_alarm_ranges() 13-43

BMC Software, Inc., Confidential and Proprietary Information

unset() 13-45

Appendix A **Using Your Own Event Loop**

How Your Program Waits for Activities A-1
The Library for the Default Event Loop A-2
Using the PATROL API from an X Application A-2
Using the PATROL API with a Custom Event Loop A-2

Appendix B **Hints**

Hint #1—Allocating Communication Handles B-2
Example of Correct Allocation B-2
Example of Incorrect Allocation B-3
Hint #2—Exiting the PATROL API Event Loop B-3
Hint#3—All Handles to Objects Are Not Stored or Freed B-4

Appendix C **Extending the Power of PATROL**

Introduction C-2
Capabilities for Knowledge Module Developers C-4
Interacting with a KM—Two-Way Communication C-4
Application Monitoring—Beyond the Physical Barrier C-8
Custom Browser of PATROL Objects C-9
Event Knowledge Broker C-10
Capabilities for Application Integrators C-12
Event Translation C-12
Event Correlation C-13
Event Routing C-15
Event Synchronization C-17
What Does the PATROL API Provide? C-19
PATROL API Availability C-20
Conclusion C-21

Appendix D **Example of annotate() Function**

annotate() Example D-2

Index

BMC Software, Inc., Confidential and Proprietary Information

Figures

Figure 1-1 Functions in the PATROL API. 1-5
Figure C-1 Event Management Architecture in PATROL C-3
Figure C-2 Program-Extended Data Collector. C-5
Figure C-3 Two-Way Communication in Interacting with a KM C-6
Figure C-4 Remote PSL Execution C-7
Figure C-5 Application Monitoring Beyond the Physical Barrier C-8
Figure C-6 Custom View of the PATROL Object Browser C-9
Figure C-7 Event Knowledge Broker. C-10
Figure C-8 Event Translator. C-13
Figure C-9 Event Correlation. C-14
Figure C-10 Event Forwarding. C-16
Figure C-11 Example of Event Synchronization C-18

BMC Software, Inc., Confidential and Proprietary Information

BMC Software, Inc., Confidential and Proprietary Information

Tables

Table 1-1 Return Values for Blocking Functions 1-7
Table 13-1 PATROL COM/DCOM Files 13-4
Table 13-2 PATROL Agent COM Variables 13-5
Table C-1 PATROL API Examples for KM Developers. C-4
Table C-2 PATROL API Examples for Application Integrators C-12

BMC Software, Inc., Confidential and Proprietary Information

BMC Software, Inc., Confidential and Proprietary Information

About This Manual

The *PATROL® API Reference Manual* describes the following interfaces:

- The PATROL Application Program Interface (API) is a series of functions defined in a C header file that allow a user-written, non-PATROL program to connect to a PATROL Event Manager or read a PATROL event log circular file.

- The PATROL Component Object Model/Distributed Component Object Model (COM/DCOM) Interface, a series of functions that allows collection of data from PATROL® Agents through COM-compatible applications.

Note

This book assumes that you are familiar with your host operating system, the C or C++ programming language and/or Visual Basic; the PATROL Event Manager event catalogs, event types, and persistent filter; Kerberos network authentication protocol; and COM/DCOM protocol.

How This Book Is Organized

This book is organized as follows. In addition, a glossary of terms and an index appear at the end of the book.

BMC Software, Inc., Confidential and Proprietary Information

Chapter/Appendix	Description
Chapter 1, “Overview of the PATROL API.”	Describes how the PATROL API fits into PATROL and what you can do with the PATROL API. This chapter also provides an overview of functions and two example programs.
Chapter 2, “Example Programs.”	Provides example programs written with the PATROL API and tells you where to find the electronic copies of the examples. You can use the examples as templates for your own programs.
Chapter 3, “Data Structures.”	Describes the data structures used within the PATROL API function definitions.
Chapter 4, “Event Access Functions.”	Describes the event access function calls that return the details of a PATROL event.
Chapter 5, “Connection Management Functions.”	Describes the connection management functions that open, close, and manage the connection between your program and the PATROL Agent.
Chapter 6, “Security Management Functions.”	Describes the security management functions that provide for account information encryption and identification.
Chapter 7, “Send and Receive Functions.”	Describes the send and receive functions that send PATROL events to, and receive PATROL events from, the PATROL Agent.
Chapter 8, “Event Management Functions.”	Describes the event management function calls.The event management calls contact the PEM engine logic.
Chapter 9, “Application Management Functions.”	Describes the application management function calls that return information about PATROL computer, application class, instance, parameter, and variable objects.
Chapter 10, “Data Management Functions.”	Describes the data management function calls that are used to send a stream of bytes to a PATROL Agent.
Chapter 11, “Miscellaneous Functions.”	Describes the Pemn miscellaneous function calls.
Chapter 12, “Specific Blocking Mode Functions.”	Describes the blocking mode functions for specific purposes such as exiting a function call and PATROL API Reference Manualretrieving the current blocking function default waiting time.
Chapter 13, “PATROL COM/DCOM Interface.”	Describes the PATROL COM/DCOM interface and describes its capabilities for accessing data from PATROL Agents through COM applications.

BMC Software, Inc., Confidential and Proprietary Information

Chapter/Appendix	Description
AppendixA, “Using Your Own Event Loop.”	Contains instructions on using your own event loop.
AppendixB, “Hints.”	Contains hints on allocating communication handles and exiting the PATROL API event loop.
AppendixC, “Extending the Power of PATROL.”	Provides an overview of the PATROL API, describes its capabilities for extending the power of PATROL by linking to user-created programs, and provides illustrations and brief examples of how the PATROL API can be used to provide comprehensive event management.
AppendixD, “Example of annotate() Function.”	Provides an example of how to use the `annotate()` function in a custom interface.

Related Documentation

BMC Software products offer several types of documentation:

- online and printed books
- online Help
- release notes

Online and Printed Books

The books that accompany BMC Software products are available in online format and printed format. You can view online books with Acrobat Reader from Adobe Systems. The reader is provided at no cost, as explained in “To Access Online Books.” You can also obtain additional printed books from BMC Software, as explained in “To Request Additional Printed Books.”

To Access Online Books

You can access online books from the documentation compact disc (CD) that accompanies your product or from the World Wide Web.

BMC Software, Inc., Confidential and Proprietary Information

Accessing Books with Acrobat Reader

Online books are formatted as Portable Document Format (PDF) files. You can view them, print them, or copy them to your computer by using Acrobat Reader 3.0 or later.

Note

In some cases, installation of Acrobat Reader and downloading the online books is an optional part of the product-installation process. For information about downloading the free reader from the Web, go to the Adobe Systems site at **http://www.adobe.com**. For information about installing the reader from the documentation CD, see the **readme.txt** file on the CD.

Accessing Books on the Web

To access any online book that BMC Software offers, visit the support page of the BMC Software Web site at **http://www.bmc.com/support**.

Note

To use the support page, you must be a registered user or have a temporary user name and a password.

If you are a registered user, proceed as follows:

1. Click **Log in to Support**.
2. Enter your user name and password.
3. To view books for a particular product, click the product name in the Product Directory.
4. Select the book that you want to view.

If you are not a registered user, choose one of the following options:

- To register, click **Request UserID**, complete the form, and click **Send Request.**

- To request a temporary user name and a password, contact your BMC Software sales representative.

To Request Additional Printed Books

BMC Software provides a core set of printed books with your product order. To request additional books, go to **http://www.bmc.com/support.**

Online Help

You can access Help for a product through the product's Help menu. The online Help provides information about the product's graphical user interface (GUI) and provides instructions for completing tasks.

Release Notes

Printed release notes accompany each BMC Software product. Release notes provide up-to-date information such as

- updates to the installation instructions
- last-minute product information

The latest versions of the release notes are also available on the Web at **http://www.bmc.com/support**.

Conventions

The following conventions are used in this book:

BMC Software, Inc., Confidential and Proprietary Information

- This book includes special elements called *notes*, *warnings*, *examples*, and *tips*:

Note

Notes provide additional information about the current subject.

Warning

Warnings alert you to situations that can cause problems, such as loss of data, if you do not follow instructions carefully.

Example

An example clarifies a concept discussed in text.

Tip

A tip provides useful information that may improve product performance or make procedures easier to follow.

- `All syntax, operating system terms, and literal examples are presented in this typeface.`

- In instructions, **boldface** type highlights information that you enter. File names, directories, and Web addresses also appear in boldface type.

- The symbol => connects items in a menu sequence. For example, **Actions** => **Create Test** instructs you to choose the Create Test command from the Actions menu.

- The symbol » denotes one-step instructions.

- In syntax, path names, or system messages, *italic* text represents a variable, as shown in the following examples:

  `The table` ***`table_name`*** `is not available.`

  system/*instance/file_name*

- In syntax, the following additional conventions apply:

  — A vertical bar (|) separating items indicates that you must choose one item. In the following example, you would choose *a*, *b*, or *c*:

  `a | b | c`

  — An ellipsis (. . .) indicates that you can repeat the preceding item or items as many times as necessary.

  — Square brackets ([ ]) around an item indicate that the item is optional.

- The following table shows equivalent mouse buttons for Unix users and Windows NT users:

BMC Software, Inc., Confidential and Proprietary Information

Unix Button	Windows NT Button	Description
MB1	left mouse button	Click this button on an icon or menu command to select that icon or command. Click MB1 on a command button to initiate action. Double-click an icon to open its container.
MB2	not applicable	Click this button on an icon to display the InfoBox for the icon. To simulate MB2 on a two-button mouse, simultaneously press the two buttons (MB1 and MB3).
MB3	right mouse button	Click this button on an icon to display its pop-up menu.

Note

If you have a one-button mouse (such as an Apple Macintosh mouse), assign MB1 to that button. You should also define a user-selectable combination of option and arrow keys to simulate MB2 and MB3. For details, refer to the documentation for your emulation software.

1

Overview of the PATROL API

This chapter describes how the PATROL API fits into PATROL and what you can do with the PATROL API. It also provides an overview of functions and two example programs.

Introduction 1-2
- Fitting the PATROL API into PATROL 1-2
- Interacting with PATROL Agents 1-3
- Integrating Applications and Managing the Enterprise 1-3
- Accessing the PEM Engine 1-4
- Connecting User Programs to the PATROL Agent 1-5

Retrieving Information from a PATROL Agent 1-6
- Using Blocking or Non-blocking (Event-Driven) Functions. . . . 1-6
- Understanding Return Values for Blocking Functions 1-7
- Finding Information about Functions for PATROL Agents. 1-7

Using the PATROL API to Connect to the PEM Engine 1-10
- Benefits of Connecting with the PATROL API 1-10
- Receiving Events from the PEM Engine 1-11
- Sending Events to the PEM Engine 1-12
- Finding Information about Functions for the PEM Engine 1-12

Example of a PATROL API Blocking Program 1-14

Example of a PATROL API Non-Blocking Program 1-18

BMC Software, Inc., Confidential and Proprietary Information

Introduction

The PATROL API, as defined in the C header file `pemapi.h`, allows your C programs to

- retrieve information from the agent about PATROL application instances, parameters, and variable objects
- register with the PEM engine in the agent to access all event management services

Fitting the PATROL API into PATROL

PATROL uses autonomous and intelligent agents to monitor databases and applications over the network. Agents provide the application monitoring and management infrastructure. The application knowledge is loaded from a Knowledge Module® (KM) written in a scripting language called PATROL® Scripting Language (PSL).

Event management is handled by a part of the agent called the PATROL Event Manager engine (PEM engine). The PEM engine manages events originating from the agent, knowledge modules running on the agent, or any application that is linked to the agent through the PATROL Application Program Interface (PATROL API).

BMC Software, Inc., Confidential and Proprietary Information

Interacting with PATROL Agents

You can write programs that interact with PATROL Agents and extend the Knowledge Module capabilities using the PATROL API. For example, using the PATROL API, you can write programs that

- interact with a Knowledge Module in a bidirectional way
- appear as single monitored entities to the user despite being distributed physically on different hosts
- load event knowledge from a knowledge broker and broadcast it to all interested hosts
- act as specialized application browsers of PATROL

Warning

The PATROL API is not presently thread safe. Unpredictable results may result if your application includes multiple requests through the same connection. Requests should be channeled one at a time through a single thread.

Integrating Applications and Managing the Enterprise

You can write programs that integrate applications and manage the enterprise using the PATROL API. For example, using the PATROL API, you can write programs that

- translate PATROL events into third party vendor events
- correlate events from different hosts, Knowledge Modules, and sources into a new compound event which initiates further actions by the PATROL Agent
- route events from one agent to another agent or to a custom program; the destination agent or program can provide specialized services (for example, archiving, capacity planning, event consolidation)
- synchronize events so an event translated into a peer vendor object can be tracked

BMC Software, Inc., Confidential and Proprietary Information

Accessing the PEM Engine

PEM engine features are accessible from any of several sources. Each source has its unique advantages according to the task you want to perform. The table below shows the source to use for the task you want.

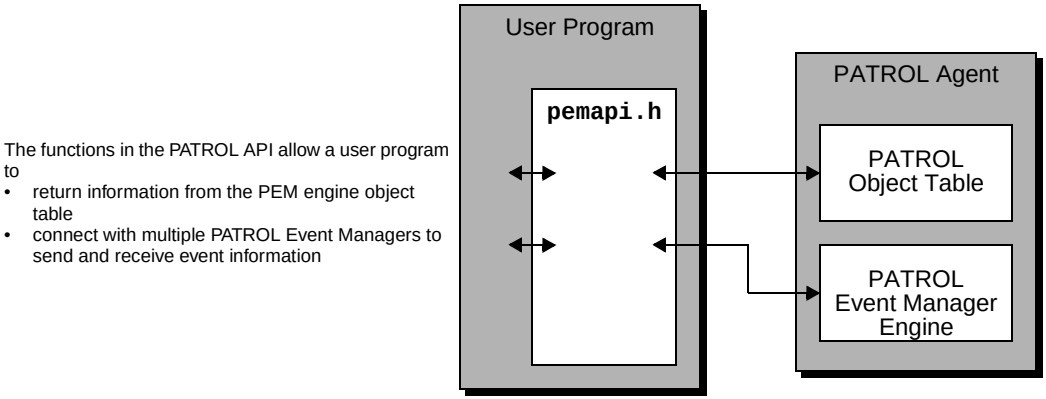
Task You Want to Perform	Source of Access to Use	Additional Information
Access PATROL from the command line.	The PATROL Command Line Interface (CLI)	—
Monitor a system using an interactive, user-friendly access.	A GUI such as the PATROL® Console or the PATROL Event Manager Console	The PATROL Developer Console allows you to define the conditions and actions that incoming events can trigger in a PATROL Agent.
Write PSL scripts that access events and objects.	The PATROL Script Language (PSL)	You can access events through function calls such as `event_query()` and `event_trigger()`. You can access objects through function calls such as `create()`, `get()`, and `destroy()`. These functions are designed for use by KM programmers writing PSL scripts. (Refer to the *PATROL Script Language Reference Manual* for details about PSL.)
Write C programs that include PATROL object monitoring and event management functionality.	This Application Program Interface (API)	—

BMC Software, Inc., Confidential and Proprietary Information

Connecting User Programs to the PATROL Agent

Figure 1-1 is a block diagram that represents the functions in the PATROL API that provide connections between a program and the PEM engine and PATROL event log. The topics that follow provide a more detailed introduction to the functions.

Figure 1-1 Functions in the PATROL API



BMC Software, Inc., Confidential and Proprietary Information

Retrieving Information from a PATROL Agent

The PATROL API provides the ability to retrieve information from the PATROL Agent object table for the active computer and application classes, instances, parameters, and variables. Through function calls, a user-written (non-PATROL) program has read access to the information collected and maintained by the PATROL Agent.

Using Blocking or Non-blocking (Event-Driven) Functions

The functions provided in the PATROL API support two different programming models, blocking and non-blocking (event driven).

If You Want Your Program to...	Use These Functions...	Additional Information
Continue executing statements.	PemnGet (Non-blocking)	When the return value arrives from the PATROL Agent, the API calls the handler specified in the PemnGet call to process the return value. The `PemnGet` functions were designed for event-driven programming, where the return value is large, asynchronous, or both, or when the execution speed of the user-written program is a design goal. Programs written using `PemnGet` function calls execute quickly and can address multiple agents at once (multiplex). However, they are harder to program because of the inability to predict how many program statements might execute before the return value arrives. See “Example of a PATROL API Non-Blocking Program” on page 1-18 for a simple example of a program written with `PemnGet` function calls.
Wait for the information from the PATROL Agent before executing any more statements.	PemnBGet (Blocking)	The `PemnBGet` return value is passed to the user-written program. The `PemnBGet` functions were designed for situations where the return value is quickly returned, or when the execution speed of the user-written program is not a design goal. Programs written using `PemnBGet` function calls are easier to program because of their synchronous nature. However, they execute more slowly than event-driven programs, and they cannot multiplex among agents. See “Example of a PATROL API Blocking Program” on page 1-14 for a simple example of a program written with `PemnBGet` function calls.

BMC Software, Inc., Confidential and Proprietary Information

Understanding Return Values for Blocking Functions

Blocking functions return a pointer to a character string that is set to one of the strings declared in the `pemapi.h`. These strings are essentially messages that tell you whether the function completed successfully. Table 1-1 presents a list of these messages.

Table 1-1 Return Values for Blocking Functions

Symbol / Return Value	Description
PEMN_ERROR	Returned if the function is unable to proceed due to invalid data input, such as a NULL pointer, or corrupt handle.
PEMN_FILE_ERROR	Returned if an error occurs during a data management operation.
PEMN_TIMEOUT	Returned when the blocking function waits for `iMsMaxTimeToWait` milliseconds without receiving the results from the PATROL Agent.
PEMN_NULL	Returned if the requested object does not exist.
PEMN_INTERRUPT	Returned if the blocking function is interrupted before completion.
PEMN_DISCONNECT	Returned if the connection with the PATROL Agent is broken while in the blocking function.
PEMN_OK	Returned if the operation successfully completed.

NULL as a Return Value for Blocking Functions

Calling a blocking function with an invalid parameter sometimes causes it to return NULL as the pointer. For example, the `PemnBEventQuery()` function returns NULL when an invalid value is passed as the `pcOriginFilter` parameter.

Finding Information about Functions for PATROL Agents

The following table tells you the function to use for the task you want to perform and provides a cross reference for finding detailed information.

BMC Software, Inc., Confidential and Proprietary Information

Information You Want to Retrieve	Function to Use	Blocking	Non-blocking	For More Information, See
open a connection with the PATROL Agent so you can retrieve information	PemnOpen()		•	“PemnOpen()” on page 5-12
	PemnBOpen()	•		“PemnBOpen()” on page 5-3
application class list	PemnGetApplList()		•	“PemnGetApplList(), PemnBGetApplList()” on page 9-6
	PemnBGetApplList()	•		
application class object	PemnGetApplObj()		•	“PemnGetApplObj(), PemnBGetApplObj()” on page 9-9
	PemnBGetApplObj()	•		
application class object summary	PemnGetApplObjInfo()		•	“PemnGetApplObjInfo()” on page 9-13
application instance list	PemnGetInstList()		•	“PemnGetInstList(), PemnBGetInstList()” on page 9-21
	PemnBGetInstList()	•		
application instance object	PemnGetInstObj()		•	“PemnGetInstObj(), PemnBGetInstObj()” on page 9-24
	PemnBGetInstObj()	•		
application instance object summary	PemnGetInstObjInfo()		•	“PemnGetInstObjInfo()” on page 9-28
instance parameter list	PemnGetParamList()		•	“PemnGetParamList(), PemnBGetParamList()” on page 9-30
	PemnBGetParamList()	•		
instance parameter object	PemnGetParamObj()		•	“PemnGetParamObj(), PemnBGetParamObj()” on page 9-33
	PemnBGetParamObj()	•		
instance parameter object summary	PemnGetParamObjInfo()		•	“PemnGetParamObjInfo ()” on page 9-37
parameter variable list	PemnGetVarList()		•	“PemnGetVarList(), PemnBGetVarList()” on page 9-39
	PemnBGetVarList()	•		
parameter variable object	PemnGetVarObj()		•	“PemnGetVarObj(), PemnBGetVarObj()” on page 9-42
	PemnBGetVarObj()	•		

BMC Software, Inc., Confidential and Proprietary Information

Information You Want to Retrieve	Function to Use	Blocking	Non-blocking	For More Information, See
computer instance parameter list	PemnGetComputerParam List()		•	“PemnGetComputer ParamList(), PemnBGetComputer ParamList()”on page 9-15
	PemnBGetComputerPara mList()	•		
computer parameter object	PemnGetComputerParam Obj()		•	“PemnGetComputer ParamObj(), PemnBGetComputer ParamObj()” on page 9-18
	PemnBGetComputerPara mObj()	•		
close the connection with the PATROL Agent	PemnClose()		•	“PemnClose()” on page 5-7
	PemnBClose()	•		“PemnBClose()” on page 5-2

BMC Software, Inc., Confidential and Proprietary Information

Using the PATROL API to Connect to the PEM Engine

The PATROL API provides an application integration tool that allows user-written (non-PATROL) programs to establish a connection with a PEM engine.

Each connection to the agent has its own handle, so the PATROL API functions allow a non-PATROL program to connect to any number of agents either on a single computer system or on multiple computer systems. The network transport layer used by the PATROL API functions is compatible with the Universal Datagram Protocol (UDP) and the Transmission Control Protocol (TCP).

Benefits of Connecting with the PATROL API

The benefits of using the PATROL API functions to connect a non-PATROL program to the PEM engine are as follows:

- You can integrate program-specific events from the non-PATROL program into the PATROL event space.
- You can view events generated by the non-PATROL program in the same way you view PATROL-generated events, that is, using any PATROL or PATROL Event Manager Console. You can close, acknowledge, or delete the events generated by the non-PATROL program.
- You can trigger application recovery actions in the PATROL Agent by associating the action with an event class and then sending an event of that class from the non-PATROL program. The PATROL Agent will execute the recovery action defined for the event.The recovery action may be any executable command or program.
- You can send events to the PATROL Event Manager from the non-PATROL program, and when those events escalate, the PATROL Agent will execute the escalation actions defined for their escalation.

BMC Software, Inc., Confidential and Proprietary Information

- A non-PATROL program can receive events from the PEM engine and can process the events (for example, dump or record them, or integrate them with a third-party product) in a way that is specific to the non-PATROL program.

- You can send events from the non-PATROL program to the Event Manager that the PATROL Agent will correlate with events from other PATROL and non-PATROL sources. The ability to correlate events from non-PATROL sources customizes and enhances the PATROL Agent's ability to detect, diagnose, and correct enterprise-wide warning and alarm conditions.

- You can use a non-PATROL program to correlate events from different agents and different Knowledge Modules.

Receiving Events from the PEM Engine

The PATROL API functions allow your program to receive events from the PEM engine. The PEM engine treats the PATROL API request for events as it would any other requestor, delivering all events that pass through its persistent filter. A second filter in your PATROL API functions allows you to further filter events, discarding those that don't match the filter, and calling an event handler routine for those that do.

BMC Software, Inc., Confidential and Proprietary Information

Sending Events to the PEM Engine

Using the PATROL API functions, your program can trigger any event defined in a PATROL event catalog, including the standard catalog and any user-defined catalogs. The PEM engine does not distinguish between events received from the PATROL API functions and events it receives through processing its own Knowledge Modules. Events that the PATROL Event Manager receives through the PATROL API functions that

- are viewable from any PATROL or PATROL Event Manager console
- can be closed, acknowledged, or deleted
- trigger notifications when notification actions are defined
- trigger event escalation when escalation actions are defined
- trigger the Acknowledge action when an event is acknowledged

Finding Information about Functions for the PEM Engine

The following table tells you the function to use for the task you want to perform and provides a cross reference for finding detailed information.

Task You Want to Perform	Function to Use	For More Information, See
Return connection attributes.	PemnGetCommAttributes()	“PemnGetCommAttributes()” on page 5-8
Set connection attributes.	PemnSetCommAttributes()	“PemnSetCommAttributes()” on page 5-15
Encrypt a password.	PemnEncrypt()	“PemnEncrypt()” on page 6-2
Open a connection with PATROL Event Manager.	PemnOpen()	“PemnOpen()” on page 5-12
Send user name and encrypted password to PATROL Agent.	PemnSendIdentity()	“PemnSendIdentity()” on page 6-3

Task You Want to Perform	Function to Use	For More Information, See
Set persistent filter for session.	PemnSetPersistentFilter()	“PemnSetPersistentFilter(), PemnBSetPersistentFilter()” on page 8-69
Add an event class to a PATROL event catalog.	PemnDefineEventClass()	“PemnDefineEventClass()” on page 8-18
Return fields from an event class definition.	PemnGetEventClass()	“PemnGetEventClass(), PemnBGetEventClass()” on page 8-59
Return a list of event classes from a PATROL event catalog.	PemnGetEventClassList()	“PemnGetEventClassList(), PemnBGetEventClassList()” on page 8-63
Send event to the PATROL Event Manager.	PemnSend()	“PemnSend(), PemnBSend()” on page 7-7
Filter events from the PATROL Event Manager.	PemnReceive()	“PemnReceive()” on page 7-2
Flush events to the PATROL event log.	PemnFlushEvent()	“PemnFlushEvent()” on page 8-58
Change the status of an event.	PemnEventManage()	“PemnEventManage()” on page 8-30
Add text to the diary of an event.	PemnAddDiary()	“PemnAddDiary()” on page 8-2
Return events that match the filter.	PemnEventQuery()	“PemnEventQuery(), PemnBEventQuery()” on page 8-39
Change event status in an ID range.	PemnEventRangeManage()	“PemnEventRangeManage() ” on page 8-46
Return events in an ID range.	PemnEventRangeQuery()	“PemnEventRangeQuery()” on page 8-49
Close the connection with PATROL Event Manager.	PemnClose()	“PemnClose()” on page 5-7
Monitor the connection for activity.	PemnLoop()	“PemnLoop()” on page 5-11

BMC Software, Inc., Confidential and Proprietary Information

Example of a PATROL API Blocking Program

The following program is an example of a simple PATROL API program using the blocking model. This example program is quite basic and limited. It is meant to show you the basic structure of a PATROL API program so you can begin to write your own.

The steps of the program are explained below. The sample code follows the explanations. Refer to the sample code as you read through the explanation of each step. For more involved examples, see Chapter 2, “Example Programs”

Explanation of Step 1: Encrypt the Password

The Password is sent to the agents encrypted.

Explanation of Step 2: Open the Connection with the PATROL Agent

`PemnBOpen()` returns successfully if *both* of these conditions are met (a boolean AND):

- connection is established
- username/password is validated

(Contrast this with the event driven programming shown in “Example of a PATROL API Non-Blocking Program” on page 1-18). You can specify a retry counter and a disconnect callback which can be executed if the session disconnects before `PemnBOpen()` completes.

Note

Step 2 (Open Connection) in the blocking mode merges step 1 (Request to Open) and step 3 (Validation) in event driven programming.

BMC Software, Inc., Confidential and Proprietary Information

Explanation of Step 3: Do the Work

Request an operation to be performed by the agent and wait until the agent sends back the results or a timeout occurs. You can use the default timeout or specify your own (in milliseconds).

You can repeat step 3 as often as you want.

Note

Step 3 (Do the Work) in the blocking mode merges step 4 (Request an Operation) and step 5 (Process the Answer Received from the PATROL Agent) in event driven programming.

Note

While waiting for results in step 3 (Do the Work) to return from one agent in the blocking mode, you cannot initiate or process any results from another agent. Processing results or initiating request from/to many agents is possible in event-driven programming.

Explanation of Step 4: Close the Connection with the PATROL Agent when Finished

`PemnBClose()` closes the connection.

Hint Regarding Transport Protocols

With blocking mode it may be better to use TCP protocol as a transport. The default transport protocol is UDP but can be changed to TCP using `PemnSetCommAttribute()`.

Sample Code for a Blocking Mode Program

```
/*
 * File: simple.c
```

```
 *-------------------------------------------------------------
 * Description: The “hello World” equivalent of the blocking-mode
API.
 *(see simple_nb.c for the non-blocking mode example).
 *All this program does is to connect
 *to a PATROL agent then dump the application list.
 *
 * API routines used:
 * PemnBOpen, PemnBClose, PemnBGetApplList, PemnEncrypt
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

staticPemnCommHandle  s_hComm =NULL;

int main(int argc, char *argv[])
{
char *pcOpenResult;
charcEncryptedPwd[126+1];
char    *pcPlainTextPwd = “mypassword”;/* put here your password */
char*pcUser = “user”;/* put here your user name */
char*pcHost = “nanou”;/* put here your host name */
intiPort = 1234;/* put here your port nb. */

/* Step 1: encrypt password
 * password is sent encrypted to agents:
 * encrypt the plain text password and store it in
 * the buffer ‘cEncryptedPwd’ */
if ( ! PemnEncrypt(cEncryptedPwd, 126, pcPlainTextPwd)){
printf(“Could not encrypt password\n”);
exit(1);
}
```

BMC Software, Inc., Confidential and Proprietary Information

```
/* Step 2: Open the connection with agent */
pcOpenResult = PemnBOpen(& s_hComm,
pcHost, iPort,/* agent details */
pcUser,/* account to log in the agent:
     * user name and encrypted password*/
cEncryptedPwd,
5, /* Loggin retry Counter */
NULL,  /* NO  disconnect callback */
NULL/* NO disconnect client data */
);

if (strcmp(PEMA_OK, pcOpenResult)!=0) {
printf("Cannot connect to PATROL agent (%s,%d): %s\n",
pcHost, iPort, pcOpenResult
);
exit(1);
}

/* Step 3: do the work
 * Request the application list and wait until the agent sends
 * back the results OR timeout */
if (s_hComm){
printf("Application List of PATROL agent (%s,%d)\n%s\n",
pcHost,
iPort,
PemnBGetApplList(s_hComm, USE_DEFAULT_TIMEOUT)
);
}

/* Step 4: Close the connection */
if (s_hComm) {
PemnBClose(s_hComm);
s_hComm = NULL;
}

return (0);
}
```

BMC Software, Inc., Confidential and Proprietary Information

Example of a PATROL API Non-Blocking Program

The following program is an example of a simple PATROL API program using the non-blocking model. This example program is quite basic and limited. It is meant to show you the basic structure of a PATROL API program so you can begin to write your own.

The steps of the program are explained below. The sample code follows the explanations. Refer to the sample code as you read through the explanation of each step. For more involved examples, see Chapter 2, “Example Programs.”

Explanation of Step 1: Request to Open a Connection with the PATROL Agent

When the connection is established, the Open Callback will be executed asynchronously. However, if the connection could not be established, the Close Callback will be executed.

Explanation of Step 2: Encrypt the Password

The password is sent to the agents encrypted.

Explanation of Step 3: Validate Username and Password

Send the username/encrypted password to agent using `PemnSendIdentity()`. You should send your account to the agent as a first task in the Open Callback; otherwise, the agent will not be able to authenticate you and the connection will drop.

BMC Software, Inc., Confidential and Proprietary Information

Explanation of Step 4: Request an Operation To Be Done

Request one or more operations to be performed by the agent. You can request as many operations as you want without having to wait for the results. When the agent sends back the results, the handlers you defined for the operation will be executed.

Note

The API remembers only the last handler you defined for each operation.

Explanation of Step 5: Process the Answer Received from the PATROL Agent

The answer received from the agent is processed.

Explanation of Step 6: Disconnect from the PATROL Agent(s) Using PemnClose()

Use the `PemnClose()` function to disconnect from the agent.

BMC Software, Inc., Confidential and Proprietary Information

Sample Code for a Non-Blocking (Event Driven) Mode Program

```
/*
 * File: simple_nb.c
 *-------------------------------------------------------------
 * Description: The “hello World” equivalent of the non-blocking API.
 *(see simple.c for the blocking mode example).
 * All this program does is to connect
 *to a PATROL agent then dump the application list.
 *
 * API routines used:
 * PemnClose, PemnOpen, PemnEncrypt, PemnLoop, PemnGetApplList,
 *PemnSendIdentity, PemnGetLastMessage
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#ifdef _WIN32
#  define strcasecmp_stricmp
#endif

#include <pemapi.h>

staticPemnCommHandle s_hComm;

static void _GetApplListHandler(
PemnCommHandle hComm,
char*pcList
)
{
/* Step 5 - process the list received from the agent then disconnect
*/
```

```
printf("Application List:\n %s\n", (pcList)?pcList:"NIL");
PemnClose(hComm);
}



static void _OpenCallback(
void*pTheObject,/* reserved for future use */
PemnClientData pClientData,
PemnCommHandle hComm
)
{
charcEncryptedPwd[126+1];
char    *pcPlainTextPwd = "my_password";/* put here your password */
char*pcUser = "user";/* put here your user name */

/* Step 2: encrypt password
 * This is the first thing you do in the Open callback.
 *
 * Password is sent encrypted to agents:
 * encrypt the plain text password and store it in
 * the buffer 'cEncryptedPwd' */
if (! PemnEncrypt(cEncryptedPwd, 126, pcPlainTextPwd)){
printf("Could not encrypt password\n");
exit(1);
}

/* Step 3: send the username/encrypted password to agent */
PemnSendIdentity(hComm, pcUser, cEncryptedPwd);


/* Step 4: do the work
 * Request the application list. Whenthe agent will
 * sends back the result '_GetApplListHandler' will be executed. */

if (PemnGetApplList(hComm, _GetApplListHandler)){
printf("request application List from the agent\n");
} else {
printf("Could not request application List from the agent\n");
}

}

static void _CloseCallback(
```

BMC Software, Inc., Confidential and Proprietary Information

```
void* pTheObject,/* reserved for future use */
PemnClientData pClientData,
PemnCommHandle  hComm
)
{
printf("Connection closed\n");
exit(0);
}

int main(int argc, char *argv[])
{
char*pcHost = "nanou";/* put here your host name */
intiPort = 1234;/* put here your port nb. */

/* Step 1: Request to Open a connection with agent
 * when the connection is established '_OpenCallback' will be
executed
 *if the connection could not be established then '_CloseCallback'
     *  will be executed */

PemnOpen(
iPort, pcHost,/* Port nb/ Host Node name */
_CloseCallback,/* Comm Close Callback   */
(PemnClientData)  NULL,/* NO Comm Close ClientData */
_OpenCallback,/* Comm Open  Callback   */
(PemnClientData)  NULL,/* NO Comm Open  Client Data   */
& s_hComm
);
if (s_hComm == NULL){
printf("Error -> %s\n", PemnGetLastMessage());
exit(1);
}

PemnLoop();
return (0);
}
```

Example Programs

This chapter provides example programs written with the PATROL API and tells you where to find the electronic copies of the examples. You can use the examples as templates for your own programs.

Overview 2-2
Where to Find the Examples 2-2
Sending and Receiving Events from One PATROL Agent 2-3
Sending and Receiving Events from Multiple PATROL Agents 2-11
Working with Event Reports in Blocking Mode 2-22
Requesting Information in Blocking Mode 2-31
Dumping Events Using Blocking Mode. 2-43

BMC Software, Inc., Confidential and Proprietary Information

Overview

The examples in this chapter are designed to provide a framework for you to follow when writing programs. Each example performs a common task. You can tailor each of these examples into a program that suits your specific needs.

Where to Find the Examples

The examples are shipped in electronic format with PATROL. You can find the examples in the `$PATROL_HOME/pemapi` directory. The following table lists the tasks for which the example programs are written and provides the names of the example programs and the sections in this chapter where you can find them.

Warning

You must use a *full* ANSI C compiler to compile these examples. A simple operating system compiler is not enough.

Task You Want to Perform	Program to Use	Section to Go to
Dump events using blocking mode.	`dump_events.c`	“Dumping Events Using Blocking Mode” on page 2-43
Send and Receive events from one PATROL Agent.	`send_receive.c`	“Sending and Receiving Events from One PATROL Agent” on page 2-3
Send and Receive events from three (more than one) PATROL Agents. (This example is very similar to send_receive except it connects with multiple agents instead of one.)	`multi_host.c`	“Sending and Receiving Events from Multiple PATROL Agents” on page 2-11

BMC Software, Inc., Confidential and Proprietary Information

Task You Want to Perform	Program to Use	Section to Go to
Send, query and generate event reports in blocking mode.	`blocking_mode.c`	“Working with Event Reports in Blocking Mode” on page 2-22
Use the blocking mode of the PATROL API to • request the agent for application list, instance list, parameter and variable list • request the agent for object's attribute: application, instance list, parameter and variable list	`patrol_obj.c`	“Requesting Information in Blocking Mode” on page 2-31

Sending and Receiving Events from One PATROL Agent

The following example is a program that sends and receives events from one PATROL Agent. Use this example as a framework as you write your program. You can find this example in electronic format in `$PATROL_HOME/pemapi/send_receive.c`. *You must use an ANSI C compiler to compile this program.*

```
/*
 * File: send_receive.c
 *-------------------------------------------------------------
 * Description: - Send and Receive events from one PATROL agent
 *
 * keywords: event driven, mono-agent connections
 *
 * API routines used:
 *PemnReceive, PemnSend
 *PemnEncrypt, PemnSendIdentity
 *PemnRegisterUserMessageHandler, PemnGetLastMessage
 *PemnOpen, PemnClose
 *PemnLoop
 *PemrGetEid, PemrGetNode, PemrGetOrigin, PemrGetType
 *PemrGetStatus, PemrGetDesc
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
```

BMC Software, Inc., Confidential and Proprietary Information

```
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

#define PROGNAME   “send_receive”



static int  s_iPort = 0;
static  int s_iMaxEventToReceive = 0;
static  int s_iEventReceived     = 0;
staticchar s_cHostName[512];
staticchar s_cUserName[512];
staticchar s_cPassword[512];
staticchar s_cPortNb[56];
staticPemnCommHandle  s_hComm =NULL;

/* forward declaration */
static void _SendEvent( PemnCommHandle hComm);
static void _MyMessageHandler(char *pcMessageToDisplay, int
iMsgType);

/*
 * Function:   _ProcessInput
 *-----------------------------------------------------------------
 * Purpose :
 *Process -h -p -u -w mandatory input arguments
 *-----------------------------------------------------------------
 */
static void _ProcessInput(int argc, char *argv[])
{
int  i;

 for (i = 1; i < argc; i++) {
int cond = (argv[i][0] == ‘-’ && argv[i][1]);
```

```
  if (!cond) continue;

     switch (argv[i][1]) {
       case ‘h’:
     case ‘H’:{
                        if(argv[i+1] == NULL){
_MyMessageHandler(
“Host name missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cHostName,argv[i+1]);
break;
}
     case ‘P’:
     case ‘p’:{
                    if(argv[i+1] == NULL){
_MyMessageHandler(
“Port nb. missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPortNb,argv[i+1]);
s_iPort = atoi( s_cPortNb);
break;
}
case ‘u’:
     case ‘U’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Username missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cUserName,argv[i+1]);
break;
}
    case ‘w’:
     case ‘W’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Password missing”,
PEMN_MSG_ERROR);
exit(1);
```

BMC Software, Inc., Confidential and Proprietary Information

```
}
strcpy(s_cPassword,argv[i+1]);
break;
}
}
}
if ( (strcmp(s_cHostName,"")==0)){
        _MyMessageHandler( "Should provide agent host name",
PEMN_MSG_ERROR);
}
if ( s_iPort <= 0){
        _MyMessageHandler( "Should provide agent port number",
PEMN_MSG_ERROR);
}
}

#define EVENT_TRACE_STR "\
Event received (real-time) nb=%d id=%d node=%s origin=%s type=%d
status=%d\n\
-->Description=%s\n\
-----------------------------------------------------------------\n"
/*
 * Function:   _ReceiveEventHandler
 *-----------------------------------------------------------------
 * Purpose :
 *This handler is called when an event is received.
 *-----------------------------------------------------------------
 */
static void _ReceiveEventHandler(
PemnCommHandle hComm,
PemrEventHandle oHEvent)
{
printf(EVENT_TRACE_STR,
s_iEventReceived++,
PemrGetEid   (oHEvent),
(PemrGetNode (oHEvent)) ? PemrGetNode(oHEvent) : "NIL",
(PemrGetOrigin(oHEvent)) ? PemrGetOrigin(oHEvent) : "NIL",
PemrGetType  (oHEvent),
PemrGetStatus(oHEvent),
(PemrGetDesc (oHEvent)) ? PemrGetDesc(oHEvent) : "NIL"
);

if (s_iEventReceived == s_iMaxEventToReceive) {
printf("%d events received- close connection\n",
```

```
s_iEventReceived
);
PemnClose( hComm);
} else {
_SendEvent(hComm);
}
}




/*
 * Function:   _RegisterToReceiveEvents
 *-------------------------------------------------------------------
 * Purpose :
 *Register to receive any event. When an event is received
 *the handler _ReceiveEventHandler will be called
 *-------------------------------------------------------------------
 */
static void _RegisterToReceiveEvents(void)
{
PemnReceive(
“”, /* StartTime: any  */
“”, /* EndTime: any  */
“O,A,C,E,D”, /* Status: any  */
“I,S,E,W,A,R”, /* Type: any  */
“”,/* Severity:    any     */
“”, /* Node: any  */
“”, /* Origin: any  */
“”, /* Pattern: any  */
“”,/* Range:any     */
“”,/* EvClass any     */
_ReceiveEventHandler
);
}

/*
 * Function:   _SendEvent
 *-------------------------------------------------------------------
 * Purpose :
 *Trigger Event “41” in Standard Event Catalog.
 *Provide 1 argument to event description
 *-------------------------------------------------------------------
 */
```

BMC Software, Inc., Confidential and Proprietary Information

```
static void _SendEvent(PemnCommHandle hComm)
{
PemnSend( hComm, /* Communication Handle to agent*/
STD_EVENT_CATALOG,/* event catalog name */
“41”, /* Event Class name */
“MyOrigin”, /* Event Origin */
NULL, NULL, /* No state change */
EV_TYPE_INFORMATION,/* event type */
3, /* event Severity */
PEMN_ARG_STRING,”MyArg”,/* Argument to event description */
NULL);

}

/*
 * Function:   _CommOpenCallback
 *-------------------------------------------------------------------
 * Purpose :
 *Called when the API framework open the connection
 *-------------------------------------------------------------------
 */
static void _CommOpenCallback(
void*pTheObject,/* reserved for future use */
void*pClientData,
PemnCommHandle hComm
)
{
char cEncryptedPassword[126];

/* Set the static variable: maximum number of event to receive */
s_iMaxEventToReceive = (int) pClientData;

/* Send the username/encrypted password to agent */
if ( ! PemnEncrypt(cEncryptedPassword, 126, s_cPassword)){
printf(“Could not encrypt password\n”);
exit(1);
}
PemnSendIdentity(hComm, s_cUserName, cEncryptedPassword);

/* before we send our first event. Register the filter
 * which defines the events we want to receive
        */
_RegisterToReceiveEvents();
_SendEvent(hComm);
```

```
}

/*
 * Function:   _CommCloseCallback
 *-----------------------------------------------------------------
 * Purpose :
 *Called when the API framework disconnect or after
 *I call PemnClose()
 *-----------------------------------------------------------------
 */
static void _CommCloseCallback(
void*pTheObject,/* reserved for future use */
void *pClientData,/* not used */
PemnCommHandle hComm
)
{
printf("Close connection\n");
printf("Exiting %s\n", PROGNAME);
exit(0);
}


/*
 * Function:   _MyMessageHandler
 *-----------------------------------------------------------------
 * Purpose :
 *My error/info/warning display routine
 *-----------------------------------------------------------------
 */
static void _MyMessageHandler(char *pcMessageToDisplay, int iMsgType)
{
switch(iMsgType){
case PEMN_MSG_WARN:
printf("WARNING: %s\n", pcMessageToDisplay);
break;
case PEMN_MSG_ERROR:
printf("ERROR: %s\n", pcMessageToDisplay);
break;
default:
printf("INFORMATION: %s\n", pcMessageToDisplay);
}
}
```

BMC Software, Inc., Confidential and Proprietary Information

```
#define SYNTAX_STR \
“Syntax:\n%s -h<hostname> -p<port-nb> -u<username> -w<password>\n” \
“Username /password is needed to log to the PATROL agent\n\n”









/*
 * Function:  main
 *-------------------------------------------------------------------
 * Purpose :
 *Registers message handler
 *Try to open connection to agent
 *-------------------------------------------------------------------
 */
int main(int argc, char *argv[])
{
/* display syntax */
printf(SYNTAX_STR, PROGNAME);

/* Process -h -p -u -w mandatory options */
_ProcessInput(argc, argv);

/* Register my own message handler to be called
         * by the API framework when error, information or warning
 * is displayed
         */
PemnRegisterUserMessageHandler(_MyMessageHandler);

/* Open the connection and register the two callbacks:
         * _CommCloseCallback - called when disconnect is detected
         * _CommOpenCallback  - called when connection complete
 * To the open callback we pass the number of events to send.
 */
PemnOpen(
    s_iPort, s_cHostName,/* Port nb/Agent Node name    */
    _CommCloseCallback,/* Comm. Close Callback      */
    (PemnClientData)  NULL,   /* No Close ClientData          */
    _CommOpenCallback,/* Comm. Open  Callback      */
    (PemnClientData)  10,/* Open  Client Data: 10 events   */
```

```
    & s_hComm/* Where to store the comm. handle */
    );

/* NOTE:
 *   ‘s_hComm’ should be a static varaible in your program.
         *    The API framework stores internally the address
         *    of ‘s_hComm’ and set it to NULL when disconnecting from
the
         *    the agent.
         *
     */
if (s_hComm == NULL) {
char cMsg[256];

sprintf(cMsg, “PemnOpen: %s\n”,
PemnGetLastMessage()
);

_MyMessageHandler(cMsg, PEMN_MSG_ERROR);
exit(0);
}

/* Enter the main loop (e.g select) */
PemnLoop();
return (0);
}
```

Sending and Receiving Events from Multiple PATROL Agents

The following example is a program that sends and receives events from multiple PATROL Agents. Use this example as a framework as you write your program. You can find this example in electronic format in the file `$PATROL_HOME/pemapi/multi_host.c`. *You must use an ANSI C compiler to compile this program.*

BMC Software, Inc., Confidential and Proprietary Information

```
/*
 * File: multi_host.c
 *-------------------------------------------------------------
 * Description: - Send and Receive events from 3 PATROL agents
 *This example is very similar to send_receive
 *except it connect with 3 agents instead of one.
 * keywords: event driven, multi-agent connections
 *
 * API routines used:
 *PemnReceive, PemnSend
 *PemnEncrypt, PemnSendIdentity
 *PemnRegisterUserMessageHandler, PemnGetLastMessage
 *PemnOpen, PemnClose
 *PemnLoop
 *PemrGetEid, PemrGetNode, PemrGetOrigin, PemrGetType
 *PemrGetStatus, PemrGetDesc
 *
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

typedef struct _HostInfo {
char cHostName[512];
char cUserName[126];
char cPassword[126];
int iPortNb;
intiEventReceived;
intiMaxEventToReceive;
intiNormalExit;
intiMaxRetry;
intiRetry;
PemnCommHandle  hComm;
} HostInfo, *HostInfoPt;
```

```
#define MAX_HOSTS 3

static HostInfo hosts[MAX_HOSTS];
staticchar s_cUserName[512];
staticchar s_cPassword[512];


/* forward declaration */
static void _SendEvent( PemnCommHandle hComm);
static void _MyMessageHandler(char *pcMessageToDisplay, int
iMsgType);

/*
 * Function:  _FindHost
 *-------------------------------------------------------------------
 * Purpose :
 *Given a communication handle find the host description
 *-------------------------------------------------------------------
 */
static HostInfoPt _FindHost(PemnCommHandle hComm)
{
int i;
for (i=0; i <MAX_HOSTS; i++){
if (hosts[i].hComm == hComm) return & hosts[i];
}
return NULL;
}

/*
 * Function:  _InitPatrolAgentDetails
 *-------------------------------------------------------------------
 * Purpose :
 *Encrypt the common password and store all agent details
 *      in the sattic array hosts[].
 *In real life:
 *The agent details will be read from a configuration file.
 *Also every agent may have differnet user name/password
 *-------------------------------------------------------------------
 */
static void _InitPatrolAgentDetails(void)
{
int i;
char cEncryptedPassword[126];
```

BMC Software, Inc., Confidential and Proprietary Information

```
/* Encrypt password */
if ( ! PemnEncrypt(cEncryptedPassword, 126, s_cPassword)){
printf("Could not encrypt password\n");
exit(1);
}

/* common details */
for (i=0; i < MAX_HOSTS; i++){
strcpy(hosts[i].cUserName, s_cUserName);
strcpy(hosts[i].cPassword, cEncryptedPassword);
hosts[i].iEventReceived    = 0;
hosts[i].iMaxEventToReceive = 10;
hosts[i].iNormalExit        = 0;
hosts[i].iMaxRetry          = 3;
hosts[i].iRetry             = 0;
}

/* specific details to each hosts */
strcpy(hosts[0].cHostName,"nanou");
hosts[0].iPortNb = 1234;

strcpy(hosts[1].cHostName,"belize");
hosts[1].iPortNb = 5678;

strcpy(hosts[2].cHostName,"nanou");
hosts[2].iPortNb =5678;
}


/*
 * Function:   _SendEvent
 *-------------------------------------------------------------------
 * Purpose :
 *Trigger Event "41" in Standard Event Catalog.
 *Provide 1 argument to event description
 *-------------------------------------------------------------------
 */
static void _SendEvent(PemnCommHandle hComm)
{
PemnSend( hComm, /* Communication Handle to agent*/
STD_EVENT_CATALOG,/* event catalog name */
"41", /* Event Class name */
"MyOrigin", /* Event Origin */
```

```
NULL, NULL, /* No state change */
EV_TYPE_INFORMATION,/* event type */
3, /* event Severity */
PEMN_ARG_STRING,”MyArg”,/* Argument to event description */
NULL);

}


#define EVENT_TRACE_STR \
“Event received (real-time) from agent %s %d\n”\
“nb=%d id=%d node=%s origin=%s type=%d status=%d\n”\
“-->Description=%s\n”\
“-----------------------------------------------------------------\n”
/*
 * Function:   _ReceiveEventHandler
 *-----------------------------------------------------------------
 * Purpose :
 *This handler is called when an event is received.
 *-----------------------------------------------------------------
 */
static void _ReceiveEventHandler(
PemnCommHandle hComm,
PemrEventHandle oHEvent)
{
 HostInfoPt pHost= _FindHost(hComm);

if (pHost == NULL){
_MyMessageHandler(
“_ReceiveEventHandler: Event received from unknown host”,
PEMN_MSG_ERROR);
exit(1);
}

printf(EVENT_TRACE_STR,
pHost->cHostName,
pHost->iPortNb,
(pHost->iEventReceived)++,
PemrGetEid   (oHEvent),
(PemrGetNode (oHEvent)) ? PemrGetNode(oHEvent) : “NIL”,
(PemrGetOrigin(oHEvent)) ? PemrGetOrigin(oHEvent) : “NIL”,
PemrGetType  (oHEvent),
PemrGetStatus(oHEvent),
(PemrGetDesc (oHEvent)) ? PemrGetDesc(oHEvent) : “NIL”
```

BMC Software, Inc., Confidential and Proprietary Information

```
);

if (pHost->iEventReceived > pHost->iMaxEventToReceive) {
printf("%d events received- close connection\n",
pHost->iEventReceived
);

/* We want to exit normaly*/
pHost->iNormalExit = 1;

PemnClose( hComm);
} else {
_SendEvent(hComm);
}
}


/*
 * Function:   _RegisterToReceiveEvents
 *-------------------------------------------------------------------
 * Purpose :
 *Register to receive any event. When an event is received
 *the handler _ReceiveEventHandler will be called
 *-------------------------------------------------------------------
 */
static void _RegisterToReceiveEvents(void)
{
PemnReceive(
"", /* StartTime: any  */
"", /* EndTime: any  */
"O,A,C,E,D", /* Status: any  */
"I,S,E,W,A,R", /* Type: any  */
"",/* Severity:    any     */
"", /* Node: any  */
"", /* Origin: any  */
"", /* Pattern: any  */
"",/* Range:any     */
"",/* EvClass any     */
_ReceiveEventHandler
);
}
```

```
/*
 * Function:   _CommOpenCallback
 *-------------------------------------------------------------------
 * Purpose :
 *Called when the API framework open the connection
 *-------------------------------------------------------------------
 */
static void _CommOpenCallback(
void*pTheObject,/* reserved for future use */
void*pClientData,
PemnCommHandle hComm
)
{
HostInfoPt  pHostInfo  = (HostInfoPt) pClientData;

PemnSendIdentity(hComm, pHostInfo->cUserName, pHostInfo->cPassword);
_SendEvent(hComm);
}
```

```
/*
 * Function:   _CommCloseCallback
 *-------------------------------------------------------------------
 * Purpose :
 *Called when the API framework disconnects or after
 *I call PemnClose()
 *-------------------------------------------------------------------
 */
static void _CommCloseCallback(
void*pTheObject,/* reserved for future use */
void *pClientData,
PemnCommHandle hComm
)
```

BMC Software, Inc., Confidential and Proprietary Information

```
{
HostInfoPt  pHostInfo  = (HostInfoPt) pClientData;
char *pcMyHostName = pHostInfo->cHostName;
intiMyPortNb     = pHostInfo->iPortNb;

if (pHostInfo->iNormalExit){
printf("Normal exit from %s %d\n",pcMyHostName,iMyPortNb);
return;
}

if (pHostInfo->iRetry >= pHostInfo->iMaxRetry){
printf("Max retry reached for %s %d\n",pcMyHostName, iMyPortNb);
return;

}

(pHostInfo->iRetry)++;

/* hMyComm should be equal hComm */
printf("Detect a disconnect on host %s port %d\n",
pcMyHostName, iMyPortNb);

printf(" Attempt # %d to re-establish the connection\n",
(pHostInfo->iRetry));
PemnOpen(
iMyPortNb,
pcMyHostName,/* Port nb/ Host Node name */
_CommCloseCallback,/* Comm Close Callback   */
(void*)  pHostInfo,   /* Comm Close ClientData       */
_CommOpenCallback,/* Comm Open  Callback   */
(void*)  pHostInfo,/* Comm Open  Client Data   */
& pHostInfo->hComm
);

}
```

```
/*
 * Function:   _ProcessInput
 *-------------------------------------------------------------------
 * Purpose :
 * Process -u -w:
 * common username/password to all agents we want to connect
 *-------------------------------------------------------------------
 */
static void _ProcessInput(int argc, char *argv[])
{
int  i;

 for (i = 1; i < argc; i++) {
int cond = (argv[i][0] == '-' && argv[i][1]);

  if (!cond) continue;

     switch (argv[i][1]) {
     case 'u':
     case 'U':{
                      if(argv[i+1] == NULL){
_MyMessageHandler(
"Username missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cUserName,argv[i+1]);
break;
}
    case 'w':
     case 'W':{
                     if(argv[i+1] == NULL){
_MyMessageHandler(
"Username missing",
PEMN_MSG_ERROR);
exit(1);
```

BMC Software, Inc., Confidential and Proprietary Information

```
}
strcpy(s_cPassword,argv[i+1]);
break;
}
}
}
}


/*
 * Function:   _MyMessageHandler
 *-------------------------------------------------------------------
 * Purpose :
 *My error/info/warning display routine
 *-------------------------------------------------------------------
 */
static void _MyMessageHandler(char *pcMessageToDisplay, int iMsgType)
{
switch(iMsgType){
case PEMN_MSG_WARN:
printf("WARNING: %s\n", pcMessageToDisplay);
break;
case PEMN_MSG_ERROR:
printf("ERROR: %s\n", pcMessageToDisplay);
break;
default:
printf("INFORMATION: %s\n", pcMessageToDisplay);
}
}

#define SYNTAX_STR \
"Syntax:\n%s -u<username> -w<password>\n" \
"Username /password is needed to log to the PATROL agent\n\n"

#define PROGNAME "multi_host"

/*
 * Function:  main
 *-------------------------------------------------------------------
 * Purpose :
 *Registers message handler
 *Build agent details and store them in hosts[]
 *Try to open connection to agents
 *-------------------------------------------------------------------
```

```
 */
int main(int argc, char *argv[])
{
int  i;

/* display syntax */
printf(SYNTAX_STR, PROGNAME);

/* Process -u -w:
 * common username/password to all agents we want to connect
 */
_ProcessInput(argc, argv);

/* Register my own message handler to be called
         * by the API framework when error, information or warning
 * is displayed
         */
PemnRegisterUserMessageHandler(_MyMessageHandler);

/* Build the hosts array from agent details */
_InitPatrolAgentDetails();

for (i=0; i < MAX_HOSTS; i++){
    PemnOpen(
hosts[i].iPortNb,
hosts[i].cHostName,   /* Port nb/ Host Node name */
_CommCloseCallback,   /* Comm Close Callback   */
(PemnClientData)&hosts[i], /* Comm Close ClientData     */
_CommOpenCallback,   /* Comm Open  Callback   */
(PemnClientData)&hosts[i], /* Comm Open  Client Data   */
& hosts[i].hComm
);
    if (hosts[i].hComm == NULL) {
char cMsg[256];

sprintf(cMsg, “Agent (%s,%d): %s\n”,
hosts[i].cHostName,
hosts[i].iPortNb,
PemnGetLastMessage()
);

_MyMessageHandler(cMsg, PEMN_MSG_ERROR);
    }
}
```

BMC Software, Inc., Confidential and Proprietary Information

```
/* Register to receive events for all agents*/
_RegisterToReceiveEvents();

/* Enter the main loop (e.g select) */
PemnLoop();
return (0);
}
```

Working with Event Reports in Blocking Mode

The following example is a program that illustrates sending, querying, and generating event reports in blocking mode. Use this example as a framework as you write your program. You can find this example in electronic format in `$PATROL_HOME/pemapi/blocking_mode.c`. *You must use an ANSI C compiler to compile this program*.

```
/*
 * File: blocking_mode.c
 *-------------------------------------------------------------
 * Description: - Illustrate the blocking mode API:
 * ->send, query and generate event reports.
 *
 * keywords: blocking mode, mono-agent connections
 *
 * API routines used:
 * PemnBOpen, PemnBClose
 * PemnBEventReport, PemnBSend, PemnBEventQuery,
 * PemnRegisterUserMessageHandler
 * PemnEncrypt
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
```

```
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

#define PROGNAME “blocking_mode”

static int  s_iPort=0;
static  int s_iIteration = 10;
staticchar s_cHostName[512];
staticchar s_cUserName[512];
staticchar s_cPassword[512];
staticchar s_cPortNb[56];
staticPemnCommHandle  s_hComm =NULL;



/*
 * Function:   _MyMessageHandler
 *-------------------------------------------------------------------
 * Purpose :
 *My error/info/warning display routine
 *-------------------------------------------------------------------
 */
static void _MyMessageHandler(char *pcMessageToDisplay, int iMsgType)
{
switch(iMsgType){
case PEMN_MSG_WARN:
printf(“WARNING: %s\n”, pcMessageToDisplay);
break;
case PEMN_MSG_ERROR:
printf(“ERROR: %s\n”, pcMessageToDisplay);
```

BMC Software, Inc., Confidential and Proprietary Information

```
break;
default:
printf("INFORMATION: %s\n", pcMessageToDisplay);
}
}


/*
 * Function:   _ProcessInput
 *-------------------------------------------------------------------
 * Purpose :
 *Process -h -p -u -w mandatory input arguments
 *-------------------------------------------------------------------
 */
static void _ProcessInput(int argc, char *argv[])
{
int  i;

 for (i = 1; i < argc; i++) {
int cond = (argv[i][0] == '-' && argv[i][1]);

  if (!cond) continue;

     switch (argv[i][1]) {
       case 'h':
 case 'H':{
                      if(argv[i+1] == NULL){
_MyMessageHandler(
"Host name missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cHostName,argv[i+1]);
break;
}
     case 'P':
     case 'p':{
                    if(argv[i+1] == NULL){
_MyMessageHandler(
"Port nb. missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPortNb,argv[i+1]);
```

```
s_iPort = atoi( s_cPortNb);
break;
}
     case ‘I’:
     case ‘i’:{
                    if(argv[i+1] != NULL){
s_iIteration = atoi( argv[i+1]);
}

if (s_iIteration <= 0) s_iIteration =10;
break;
}
case ‘u’:
 case ‘U’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Username missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cUserName,argv[i+1]);
break;
}
    case ‘w’:
 case ‘W’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Password missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPassword,argv[i+1]);
break;
}
}
}
if ( (strcmp(s_cHostName,””)==0)){
        _MyMessageHandler( “Should provide agent host name”,
PEMN_MSG_ERROR);
}
if ( s_iPort <= 0){
        _MyMessageHandler( “Should provide agent port number”,
PEMN_MSG_ERROR);
}
```

BMC Software, Inc., Confidential and Proprietary Information

```
}

#define SYNTAX_STR \
“Syntax:\n”\
“%s -h<hostname> -p<port-nb> -u<username> -w<password>
[-i<iteration\n”\
“Username/password is needed to log to the PATROL agent\n\n”


/*
 * Function:  _BlockingSend
 *-------------------------------------------------------------------
 * Purpose :
 *Send an event and block a maximum 1000 millisec
 *until the agent acknowldge receiving it.
 *
 *-------------------------------------------------------------------
 */
static void _BlockingSend(void)
{
char *pcSourceId;

printf(“>Send
started-----------------------------------------------\n”);
pcSourceId =PemnBSend( s_hComm,
1000,   /* Max Waiting time in millisec */
STD_EVENT_CATALOG,      /* event catalog name */
“41”,           /* Event Class name */
“MyOrigin”,         /* Event Origin*/
NULL, NULL,         /* No state change */
EV_TYPE_INFORMATION,    /* event type */
3,         /* event Severity */
PEMN_ARG_STRING,”MyArg”,/* Argument to event description*/
NULL);
printf(“>Send done %s\n”, pcSourceId);
}

static char* g_DEFAULT_EVENT_FORMAT =
“Id          : %{EV_ID}\n”
“Status      : %{EV_STATUS}\n”
“Type        : %{EV_TYPE}\n”
“Severity    : %{EV_NSEVERITY}\n”
“Time        : %{EV_TIME}\n”
“Node        : %{EV_NODE}\n”
```

```
“Origin      : %{EV_ORIGIN}\n”
“Catalog     : %{EV_CATALOG}\n”
“Class       : %{EV_CLASS_NAME}\n”
“Args        : %{EV_ARGS}\n”
“Description : %{EV_DESC}\n”;

/*
 * Function:  _BlockingQuery
 *-------------------------------------------------------------------
 * Purpose :
 *Query the event repository and use default waiting time
 *  (USE_DEFAULT_TIMEOUT) to block for results.
 *
 *-------------------------------------------------------------------
 */
static void _BlockingQuery(void)
{
char *pc;
printf(“>Query started
-------------------------------------------\n”);
pc = PemnBEventQuery(s_hComm,
12,/* Max count in query */
“\n\n”,/* events seperator   */
g_DEFAULT_EVENT_FORMAT,/* event print format */
“”, /* StartTime: any  */
“”, /* EndTime: any  */
“O,A,C,E,D”, /* Status: any  */
“W,A”, /* Type: only alarm and warning  */
“1”,/* Severity:    1 or higher     */
“”, /* Node: any  */
“”, /* Origin: any  */
“”, /* Pattern: any  */
“”,/* Range:any     */
“”,/* EvClass any     */
USE_DEFAULT_TIMEOUT
);
printf(“>Query done %s\n”, pc);
}
```

```
/*
 * Function:  _BlockingReport
 *-----------------------------------------------------------------
 * Purpose :
 *Request an event repository report and
 *wait a maximum of 3000 milliseconds for
 *results.
 *
 *-----------------------------------------------------------------
 */
static void _BlockingReport(void)
{
char *pc;
printf(“>Report started
-------------------------------------------\n”);
pc = PemnBEventReport(s_hComm,
“”,/* Reserved for future use */
“”, /* StartTime: any  */
“”, /* EndTime: any  */
“O,A,C,E,D”, /* Status: any  */
“W,A,I”, /* Type: only alarm, warning and info  */
“1”,/* Severity:  1 or higher    */
“”, /* Node: any  */
“”, /* Origin: any  */
“”, /* Pattern: any  */
“”,/* Range:any     */
“”,/* EvClass any     */
3000  /* 3 seconds */
);
printf(“>Report done %s\n”, pc);
}

/*
 * Function:  _MyDisconnectCallback
```

```
 *-----------------------------------------------------------------
 * Purpose :
 *Called if agent disconnect happens in
 * in the middle of PemnBOpen.
 *
 *-----------------------------------------------------------------
 */
static void _MyDisconnectCallback(
PemnCommHandle          hComm,
PemnClientData  pClientData
)
{
printf("_MyDisconnectCallback called: s_hComm is now invalid\n");
}



/*
 * Function:  _BlockingModeExec
 *-----------------------------------------------------------------
 * Purpose :
 *Open connection und block until success
 *Send event until it is received by agent.
 *Start a query and block until results arrive
 *Ask for a event report and block for results.
 *
 *-----------------------------------------------------------------
 */
static void _BlockingModeExec(char * pcEncryptedPassword)
{
char *pcOpenResult;

pcOpenResult = PemnBOpen(& s_hComm,
s_cHostName,s_iPort,/* agent details */
s_cUserName,/* account to log in the agent:
 * user name and encrypted password*/
pcEncryptedPassword,
5, /* Loggin retry Counter */
_MyDisconnectCallback,  /* Called if agent disconnect happens in
 * in the middle of PemnBOpen   */
NULL
);

if (strcmp(PEMA_OK, pcOpenResult)==0) {
```

BMC Software, Inc., Confidential and Proprietary Information

```
printf("PemnBOpen done: %s\n", pcOpenResult);

if (s_hComm) _BlockingSend();
if (s_hComm) _BlockingQuery();
if (s_hComm) _BlockingReport();

if (s_hComm) {
PemnBClose(s_hComm);
s_hComm = NULL;
printf("PemnBClose done\n\n");
}
} else {
printf("PemnBOpen failed %s\n", pcOpenResult);
}
}

/*
 * Function:  main
 *-------------------------------------------------------------------
 * Purpose :
 *Registers message handler.
 *Encrypt password.
 *execute <iteration> time the blocking mode opeartions
 *-------------------------------------------------------------------
 */
int main(int argc, char *argv[])
{
int i=0;
charcEncryptedPassword[126+1];

/* display syntax */
printf(SYNTAX_STR, PROGNAME);

/* Process -h -p -u -w mandatory options */
_ProcessInput(argc, argv);

/* Register my own message handler to be called
         * by the API framework when error, information or warning
 * is displayed
         */
 PemnRegisterUserMessageHandler(_MyMessageHandler);

/* passwords are sent encrypted to agents */
if ( ! PemnEncrypt(cEncryptedPassword, 126, s_cPassword)){
```

```
printf("Could not encrypt password\n");
exit(1);
}
while( (++i) <= s_iIteration){
_BlockingModeExec(cEncryptedPassword);
}
return (0);
}
```

Requesting Information in Blocking Mode

The following example is a program that illustrates using blocking mode to perform these tasks:

- request the agent for application list, instance list, parameter and variable list
- request the agent for an object's attribute: application, instance list, parameter and variable list

Use this example as a framework as you write your program. You can find this example in electronic format in `$PATROL_HOME/pemapi/patrol_obj.c`. *You must use an ANSI C compiler to compile this program.*

```
/*
 * File: patrol_obj.c
 *-------------------------------------------------------------
 * Description: Illustrate the blocking mode API:
 *->request the agent for application list, instance list,
 *parameter and variable list.
 *->request the agent for object's attribute:
 *application , instance list, parameter and variable list.
 *-> Execute remote PSL function.
 *NOTE:
 *The agent is assumed to run a UNIX KM. Modify
```

```
 *the program to access the list of objects in your KM.
 *
 * keywords: blocking mode, mono-agent connections
 *
 * API routines used:
 * PemnBOpen, PemnBClose
 *  PemnBGetApplList, PemnBGetParamList, PemnBGetInstList,
PemnBGetVarList
 *  PemnBGetApplObj, PemnBGetParamObj, PemnBGetInstObj,
PemnBGetVarObj
 * PemnRegisterUserMessageHandler
 * PemnEncrypt
 *  PemnBPslExec
 *-----------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-----------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

#define PROGNAME “patrol_obj”



static int  s_iPort=0;
static  int s_iIteration = 10;
staticchar s_cHostName[512];
staticchar s_cUserName[512];
staticchar s_cPassword[512];
staticchar s_cPortNb[56];
staticPemnCommHandle  s_hComm =NULL;

/*
 * Function:   _MyMessageHandler
 *-------------------------------------------------------------------
 * Purpose :
```

```
 *My error/info/warning display routine
 *-------------------------------------------------------------------
 */
static void _MyMessageHandler(char *pcMessageToDisplay, int iMsgType)
{
switch(iMsgType){
case PEMN_MSG_WARN:
printf("WARNING: %s\n", pcMessageToDisplay);
break;
case PEMN_MSG_ERROR:
printf("ERROR: %s\n", pcMessageToDisplay);
break;
default:
printf("INFORMATION: %s\n", pcMessageToDisplay);
}
}


/*
 * Function:   _ProcessInput
 *-------------------------------------------------------------------
 * Purpose :
 *Process -h -p -u -w mandatory input arguments
 *-------------------------------------------------------------------
 */
static void _ProcessInput(int argc, char *argv[])
{
int  i;

 for (i = 1; i < argc; i++) {
int cond = (argv[i][0] == '-' && argv[i][1]);

  if (!cond) continue;

     switch (argv[i][1]) {
       case 'h':
     case 'H':{
                        if(argv[i+1] == NULL){
_MyMessageHandler(
"Host name missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cHostName,argv[i+1]);
```

BMC Software, Inc., Confidential and Proprietary Information

```
break;
}
     case ‘P’:
     case ‘p’:{
                    if(argv[i+1] == NULL){
_MyMessageHandler(
“Port nb. missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPortNb,argv[i+1]);
s_iPort = atoi( s_cPortNb);
break;
}
     case ‘I’:
     case ‘i’:{
                    if(argv[i+1] != NULL){
s_iIteration = atoi( argv[i+1]);
}

if (s_iIteration <= 0) s_iIteration =10;
break;
}
case ‘u’:
 case ‘U’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Username missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cUserName,argv[i+1]);
break;
}
    case ‘w’:
 case ‘W’:{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
“Password missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPassword,argv[i+1]);
break;
```

```
}
}
}
if ( (strcmp(s_cHostName,"")==0)){
        _MyMessageHandler( "Should provide agent host name",
PEMN_MSG_ERROR);
}
if ( s_iPort <= 0){
        _MyMessageHandler( "Should provide agent port number",
PEMN_MSG_ERROR);
}
}

#define SYNTAX_STR \
"Syntax:\n"\
"%s -h<hostname> -p<port-nb> -u<username> -w<password>
[-i<iteration\n"\
"Username/password is needed to log to the PATROL agent\n\n"



static int _OK(PemnCommHandle hComm, char *pcMsg)
{
if (! hComm) {
printf("Connection is broken return from: %s\n", pcMsg);
return 0;
}
return 1;
}




/*
 * Function:  _BlockingGetList
 *-------------------------------------------------------------------
 * Purpose :
 *Request the list of applications, instances, parameters and
variables
 *
 *-------------------------------------------------------------------
 */
static void _BlockingGetList(void)
{
```

BMC Software, Inc., Confidential and Proprietary Information

```
char *pc;

printf(“>Get Appl. List started
-----------------------------------\n”);
pc = PemnBGetApplList(s_hComm, USE_DEFAULT_TIMEOUT);
printf(“>Get Appl. List done %s\n”, pc);

if (! _OK(s_hComm,”_BlockingGetList 1”)) return;
printf(“>Get Inst. List started
-----------------------------------\n”);
pc = PemnBGetInstList(s_hComm,
“FILESYSTEM”,
USE_DEFAULT_TIMEOUT
);
printf(“>Get Inst. List done %s\n”, pc);

if (! _OK(s_hComm,”_BlockingGetList 2”)) return;
printf(“>Get Param. List started --------------------------
-------\n”);
pc = PemnBGetParamList(s_hComm,
“FILESYSTEM”, “usr”,
USE_DEFAULT_TIMEOUT
);
printf(“>Get Param. List done %s\n”, pc);

if (! _OK(s_hComm,”_BlockingGetList 3”)) return;
printf(“>Get Var. List started -------------------------
-------\n”);
pc = PemnBGetVarList(s_hComm,
“/”,””,
USE_DEFAULT_TIMEOUT
);
printf(“>Get Var. List done %s\n”, pc);

}


/*
 * Function:  _BlockingGetObj
 *-------------------------------------------------------------------
 * Purpose :
 *Request the object’s attributes fo an application, an
 *  instance, a parameter and a variable
 *
```

```
 *-----------------------------------------------------------------
 */
static void _BlockingGetObj(void)
{
PemnApplObjHandle hApplObj;
PemnInstObjHandle hInstObj;
PemnParamObjHandle hParamObj;
PemnVarObjHandle hVarObj;

int i;
    PemnApplAttrArray *pApplAttr;
    PemnInstAttrArray *pInstAttr;
    PemnParamAttrArray *pParamAttr;



printf(“>Get Appl. Obj started
----------------------------------------\n”);
hApplObj = PemnBGetApplObj(s_hComm,
“FILESYSTEM”,
USE_DEFAULT_TIMEOUT
);

    pApplAttr = PemnGetApplObjAttributes(hApplObj);
if (pApplAttr) {
for (i=0; i < (int) PEMN_APPL_MAX_ATTR; i++){
printf(“attribute[%d]=<%s>\n”, i,
 ((*pApplAttr)[i])?(*pApplAttr)[i]:”NIL”);
}
} else {
printf(“Could not read object’s attributes\n”);
}
printf(“>Get Appl. Obj done\n”);

if (! _OK(s_hComm,”_BlockingGetObj 1”)) return;

printf(“>Get Inst. Obj started
----------------------------------------\n”);
hInstObj = PemnBGetInstObj(s_hComm,
“FILESYSTEM”, “usr”,
USE_DEFAULT_TIMEOUT
);
  pInstAttr = PemnGetInstObjAttributes(hInstObj);
if (pInstAttr) {
```

BMC Software, Inc., Confidential and Proprietary Information

```
for (i=0; i < (int) PEMN_INST_MAX_ATTR; i++){
printf("attribute[%d]=<%s>\n", i,
 ((*pInstAttr)[i])?(*pInstAttr)[i]:"NIL");
}
} else {
printf("Could not read object's attributes\n");
}

printf(">Get Inst. Obj
done\n--------------------------------------------");

if (! _OK(s_hComm,"_BlockingGetObj 2")) return;

printf(">Get Param. Obj started
---------------------------------------\n");
hParamObj = PemnBGetParamObj(s_hComm,
"FILESYSTEM", "usr", "FSCapacity",
USE_DEFAULT_TIMEOUT
);

    pParamAttr = PemnGetParamObjAttributes(hParamObj);

if (pParamAttr) {
for (i=0; i < (int) PEMN_PARAM_MAX_ATTR; i++){
printf("attribute[%d]=<%s>\n", i,
 ((*pParamAttr)[i])?(*pParamAttr)[i]:"NIL");
}
} else {
printf("Could not read object's attributes\n");
}

printf(">Get Param. Obj
done--------------------------------------------\n");

if (! _OK(s_hComm,"_BlockingGetObj 3")) return;

printf(">Get Var. Obj started
---------------------------------------\n");
hVarObj = PemnBGetVarObj(s_hComm,
"/sysBootTime",
USE_DEFAULT_TIMEOUT
);
printf(">Get var. Obj done: %s\n", hVarObj);
```

```
if (! _OK(s_hComm,"_BlockingGetObj 1")) return;

printf(">Get Var. Obj started
---------------------------------------\n");
hVarObj = PemnBGetVarObj(s_hComm,
"/serverLogPath",
USE_DEFAULT_TIMEOUT
);
printf(">Get var. Obj done: %s\n", hVarObj);

}


static void _BlockingExecPsl(PemnCommHandle hComm)
{
char *pcResult;

printf(">Exec. PSL started
---------------------------------------------\n");
pcResult = PemnBPslExec(hComm, USE_DEFAULT_TIMEOUT, 1,
"get_vars(\"/\");"
);
printf("> Exec. PSL cmd %s\n",
(pcResult) ? pcResult : "NIL"
);

if (! _OK(s_hComm,"_BlockingExecPsl")) return;

printf(">Exec. OS cmd started
----------------------------------------\n");
pcResult = PemnBPslExec(hComm, USE_DEFAULT_TIMEOUT, 1,
"system(\"cat /tmp/blago\");"
);
printf("> Exec. OS cmd %s\n",
(pcResult) ? pcResult : "NIL"
);
}

/*
 * Function:  _MyDisconnectCallback
 *-------------------------------------------------------------------
 * Purpose :
 *Called if agent disconnect happens in
 * in the middle of PemnBOpen.
```

BMC Software, Inc., Confidential and Proprietary Information

```
 *
 *-----------------------------------------------------------------
 */
static void _MyDisconnectCallback(
PemnCommHandle          hComm,
PemnClientData  pClientData
)
{
printf("_MyDisconnectCallback called: s_hComm is now invalid\n");
}
```

```
/*
 * Function:  _BlockingModeExec
 *-----------------------------------------------------------------
 * Purpose :
 *Open connection und block until success
 *Send event until it is received by agent.
 *Start a query and block until results arrive
 *Ask for a event report and block for results.
 *
 *-----------------------------------------------------------------
 */
static void _BlockingModeExec(char * pcEncryptedPassword)
{
char *pcOpenResult;

pcOpenResult = PemnBOpen(& s_hComm,
s_cHostName,s_iPort,/* agent details */
s_cUserName,/* account to log in the agent:
 * user name and encrypted password*/
```

```
pcEncryptedPassword,
5, /* Loggin retry Counter */
_MyDisconnectCallback,  /* Called if agent disconnect happens in
 * in the middle of PemnBOpen   */
NULL
);

if (strcmp(PEMA_OK, pcOpenResult)==0) {
printf("PemnBOpen done: %s\n", pcOpenResult);

if (s_hComm) _BlockingGetList();
if (s_hComm) _BlockingGetObj();
if (s_hComm) _BlockingExecPsl(s_hComm);

if (s_hComm) {
PemnBClose(s_hComm);
s_hComm = NULL;
printf("PemnBClose done\n\n");
}
} else {
printf("PemnBOpen failed %s\n", pcOpenResult);
}
}
```

BMC Software, Inc., Confidential and Proprietary Information

```
/*
 * Function:  main
 *-------------------------------------------------------------------
 * Purpose :
 *Registers message handler.
 *Encrypt password.
 *execute <iteration> time the blocking mode opeartions
 *-------------------------------------------------------------------
 */
int main(int argc, char *argv[])
{
int i=0;
charcEncryptedPassword[126+1];

/* The first thing we do is to init the PATROL API explicitly
     * since we want to return large amount of data in
PemnBExecPsl().
     * In this release: the maximum result returned by PemnBExecPsl()
     * is 32000 bytes.
     * If PemnInit() is not called the default for the API is 10K.
 */
    PemnInit(32000);

/* display syntax */
printf(SYNTAX_STR, PROGNAME);

/* Process -h -p -u -w mandatory options */
_ProcessInput(argc, argv);

/* Register my own message handler to be called
         * by the API framework when error, information or warning
 * is displayed
         */
 PemnRegisterUserMessageHandler(_MyMessageHandler);

/* passwords are sent encrypted to agents */
if ( ! PemnEncrypt(cEncryptedPassword, 126, s_cPassword)){
printf("Could not encrypt password\n");
exit(1);
}
while( (++i) <= s_iIteration){
```

```
_BlockingModeExec(cEncryptedPassword);
}
return (0);
}
```

Dumping Events Using Blocking Mode

The following example is a program that dumps events using blocking mode. Use this example as a framework as you write your program. You can find this example in electronic format in `$PATROL_HOME/pemapi/dump_events.c`. *You must use an ANSI C compiler to compile this program*.

```
/*
 * File: dump_events.c
 *-------------------------------------------------------------
 * Description: - Use blocking mode operations to dump events
 * keywords: blocking mode, mono-agent connections
 *
 * API routines used:
 * PemnBOpen, PemnBClose
 * PemnBEventArchive
 * PemnRegisterUserMessageHandler
 * PemnEncrypt
 *-------------------------------------------------------------
 * Author        :
 * Creation date :
 *
 * Last check-in date: $Date:$
 *
 * BMC Software, Inc.
 *-------------------------------------------------------------
 */
#include <time.h>
#include <string.h>
#include <stdlib.h>

#include <pemapi.h>

#define PROGNAME “dump_events”
```

BMC Software, Inc., Confidential and Proprietary Information

```
static int  s_iPort=0;
staticchar s_cHostName[512];
staticchar s_cUserName[512];
staticchar s_cPassword[512];
staticchar s_cDestination[512];
staticchar s_cPortNb[56];
staticPemnCommHandle  s_hComm =NULL;

/*
 * Function:   _MyMessageHandler
 *-------------------------------------------------------------------
 * Purpose :
 *My error/info/warning display routine
 *-------------------------------------------------------------------
 */
static void _MyMessageHandler(char *pcMessageToDisplay, int iMsgType)
{
switch(iMsgType){
case PEMN_MSG_WARN:
printf("WARNING: %s\n", pcMessageToDisplay);
break;
case PEMN_MSG_ERROR:
printf("ERROR: %s\n", pcMessageToDisplay);
break;
default:
printf("INFORMATION: %s\n", pcMessageToDisplay);
}
}


/*
 * Function:   _ProcessInput
 *-------------------------------------------------------------------
 * Purpose :
 *Process -h -p -u -w mandatory input arguments
 *Process -d optional destination
 *-------------------------------------------------------------------
 */
static void _ProcessInput(int argc, char *argv[])
{
int  i;

strcpy(s_cDestination,"/tmp/PEM.txt");
```

```
 for (i = 1; i < argc; i++) {
int cond = (argv[i][0] == '-' && argv[i][1]);

  if (!cond) continue;

     switch (argv[i][1]) {
       case 'd':
     case 'D':{
                        if(argv[i+1] == NULL){
_MyMessageHandler(
"Destination file missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cDestination,argv[i+1]);
break;
}
       case 'h':
     case 'H':{
                        if(argv[i+1] == NULL){
_MyMessageHandler(
"Host name missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cHostName,argv[i+1]);
break;
}
     case 'P':
     case 'p':{
                     if(argv[i+1] == NULL){
_MyMessageHandler(
"Port nb. missing",
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPortNb,argv[i+1]);
s_iPort = atoi( s_cPortNb);
break;
}
case 'u':
 case 'U':{
                        if(argv[i+1] == NULL) {
_MyMessageHandler(
```

BMC Software, Inc., Confidential and Proprietary Information

```
“Username missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cUserName,argv[i+1]);
break;
}
    case ‘w’:
 case ‘W’:{
                      if(argv[i+1] == NULL) {
_MyMessageHandler(
“Password missing”,
PEMN_MSG_ERROR);
exit(1);
}
strcpy(s_cPassword,argv[i+1]);
break;
}
}
}
if ( (strcmp(s_cHostName,””)==0)){
        _MyMessageHandler( “Should provide agent host name”,
PEMN_MSG_ERROR);
}
if ( s_iPort <= 0){
        _MyMessageHandler( “Should provide agent port number”,
PEMN_MSG_ERROR);
}
}

#define SYNTAX_STR \
“Syntax:\n”\
“%s -h<hostname> -p<port-nb> -u<username> -w<password> [-d
<destination>]\n”\
“Username/password is needed to log to the PATROL agent\n\n”



static char* g_DEFAULT_EVENT_FORMAT =
“Id          : %{EV_ID}\n”
“Status      : %{EV_STATUS}\n”
“Type        : %{EV_TYPE}\n”
“Severity    : %{EV_NSEVERITY}\n”
“Time        : %{EV_TIME}\n”
```

```
“Node        : %{EV_NODE}\n”
“Origin      : %{EV_ORIGIN}\n”
“Catalog     : %{EV_CATALOG}\n”
“Class       : %{EV_CLASS_NAME}\n”
“Args        : %{EV_ARGS}\n”
“Description : %{EV_DESC}\n”;
```

```
/*
 * Function:  _BlockingDump
 *-------------------------------------------------------------------
 * Purpose :
 *Dump events  matching the filter and use 1 min for finishing
 *
 *-------------------------------------------------------------------
 */
static void _BlockingDump(void)
{
char *pc;
printf(“>Dump events on %s started
--------------------------------\n”,
s_cDestination
```

```
);
pc = PemnBEventArchive(s_hComm,
g_DEFAULT_EVENT_FORMAT,/* event print format */
“\n\n”,/* events seperator   */
s_cDestination,
PEMN_WRITE,

“”, /* StartTime: any  */
“”, /* EndTime: any  */
“O,A,C,E,D”, /* Status: any  */
“W,A”, /* Type: only alarm and warning  */
“1”,/* Severity:    1 or higher     */
“”, /* Node: any  */
“”, /* Origin: any  */
“”, /* Pattern: any  */
“”,/* Range:any     */
“”,/* EvClass any     */
1*60*1000
);
printf(“>Dump events done %s\n”, pc);
}

/*
 * Function:  _MyDisconnectCallback
 *-------------------------------------------------------------------
 * Purpose :
 *Called if agent disconnect happens in
 * in the middle of PemnBOpen.
 *
 *-------------------------------------------------------------------
 */
static void _MyDisconnectCallback(
PemnCommHandle          hComm,
PemnClientData  pClientData
)
{
printf(“_MyDisconnectCallback called\n”);
}
```

```
/*
 * Function:  _DoDump
 *-----------------------------------------------------------------
 * Purpose :
 *Open connection und block until success
 *Send event until it is received by agent.
 *Start a query and block until results arrive
 *Ask for a event report and block for results.
 *
 *-----------------------------------------------------------------
 */
static void _DoDump(char * pcEncryptedPassword)
{
char *pcOpenResult;

pcOpenResult = PemnBOpen(& s_hComm,
s_cHostName,s_iPort,/* agent details */
s_cUserName,/* account to log in the agent:
 * user name and encrypted password*/
pcEncryptedPassword,
5, /* Loggin retry Counter */
_MyDisconnectCallback,  /* Called if agent disconnect happens in
 * in the middle of PemnBOpen   */
NULL
);

if (strcmp(PEMA_OK, pcOpenResult)==0) {
printf("PemnBOpen done: %s\n", pcOpenResult);

if (s_hComm) _BlockingDump();

if (s_hComm) {
PemnBClose(s_hComm);
s_hComm = NULL;
printf("PemnBClose done\n\n");
```

BMC Software, Inc., Confidential and Proprietary Information

```
}
} else {
printf("PemnBOpen failed %s\n", pcOpenResult);
}
}
```

```
/*
 * Function:  main
 *-------------------------------------------------------------------
 * Purpose :
 *Registers message handler.
 *Encrypt password.
 *execute <iteration> time the blocking mode opeartions
 *-------------------------------------------------------------------
 */
int main(int argc, char *argv[])
{
int i=0;
charcEncryptedPassword[126+1];
```

```
/* display syntax */
printf(SYNTAX_STR, PROGNAME);

/* Process -h -p -u -w mandatory options */
_ProcessInput(argc, argv);

/* Register my own message handler to be called
         * by the API framework when error, information or warning
 * is displayed
         */
 PemnRegisterUserMessageHandler(_MyMessageHandler);

/* passwords are sent encrypted to agents */
if ( ! PemnEncrypt(cEncryptedPassword, 126, s_cPassword)){
printf("Could not encrypt password\n");
exit(1);
}
_DoDump(cEncryptedPassword);

return (0);
}
```

BMC Software, Inc., Confidential and Proprietary Information

3

Data Structures

This chapter describes the data structures used within the PATROL API function definitions.

Overview of Type Definitions 3-3
PemnApplAttrArray 3-5
PemnApplAttrEnum 3-6
PemnApplObjHandle 3-7
PemnBDisconnectCallback 3-8
PemnClientData 3-9
PemnCommCallback 3-10
PemnCommHandle. 3-11
PemnArchiveEventHandler 3-12
PemnEventHandle 3-13
PemnEventStatusEnum 3-14
PemnEventTypeEnum 3-15
PemnGetApplObjHandler 3-16
PemnGetEventClassHandler 3-17
PemnGetEventClassListHandler 3-18
PemnGetInstObjHandler 3-19
PemnGetListHandler 3-20
PemnGetParamObjHandler 3-21
PemnGetVarObjHandler. 3-22
PemnInstAttrArray 3-23
PemnInstAttrEnum 3-24
PemnInstObjHandle 3-25
PemnParamAttrArray 3-26
PemnParamAttrEnum. 3-27
PemnParamObjHandle 3-28

BMC Software, Inc., Confidential and Proprietary Information

PemnQueryEventHandler 3-29
PemnReceiveEventHandler. 3-30
PemnReportEventHandler 3-32
PemnReportHandle 3-33
PemnUserMessageHandler 3-34
PemnVarObjHandle 3-35

Overview of Type Definitions

The following table summarizes the type definitions in the `pemapi.h` header.

Definition	Page
Routine Callback:	
PemnBDisconnectCallback PemnCommCallback	3-8 3-10
Enumeration:	
PemnEventStatusEnum PemnEventTypeEnum	3-14 3-15
Miscellaneous:	
PemnClientData	3-9
Routine Handler:	
The routine handlers define the routines that execute upon receiving a response from the agent. Contrast these with the object handles that define formal references to specific object types.	
PemnArchiveEventHandler PemnGetApplObjHandler PemnGetEventClassHandler PemnGetEventClassListHandler PemnGetInstObjHandler PemnGetListHandler PemnGetParamObjHandler PemnGetVarObjHandler PemnQueryEventHandler PemnReceiveEventHandler PemnReportEventHandler PemnUserMessageHandler	3-12 3-16 3-17 3-18 3-19 3-20 3-21 3-22 3-29 3-30 3-32 3-34
Object Handles:	
The object handles define formal references to specific object types. Contrast these with the routine handlers that define the routines that execute upon receiving a response from the agent.	

BMC Software, Inc., Confidential and Proprietary Information

Definition	Page
PemnApplObjHandle	3-7
PemnCommHandle	3-11
PemnEventHandle	3-13
PemnInstObjHandle	3-25
PemnParamObjHandle	3-28
PemnReportHandle	3-33
PemnVarObjHandle	3-35
Array Type and Indexes:	
PemnApplAttrArray	3-5
PemnApplAttrEnum	3-6
PemnInstAttrArray	3-23
PemnInstAttrEnum	3-24
PemnParamAttrArray	3-26
PemnParamAttrEnum	3-27

BMC Software, Inc., Confidential and Proprietary Information

PemnApplAttrArray

Define the read-only attributes of an application object.

Synopsis

```
typedef char* PemnApplAttrArray[PEMN_APPL_MAX_ATTR];
```

BMC Software, Inc., Confidential and Proprietary Information

PemnApplAttrEnum

Define the index to the read-only attributes of an application object.

Synopsis

```
typedef enum {
   PEMN_APPLNAME=0,      /* should be always ZERO */
   PEMN_APPLWORSTINST,
   PEMN_APPLSTATE,
   PEMN_APPLMASTER,
   PEMN_APPLMINOR,
   PEMN_APPL_MAX_ATTR    /* should be always last */
} PemnApplAttrEnum;
```

BMC Software, Inc., Confidential and Proprietary Information

PemnApplObjHandle

Define the PATROL application object handle.

Synopsis

```
typedef void *PemnApplObjHandle;
```

Description

The `PemnApplObjHandle` type is the abstract application type handle returned by the `PemnGetApplObj()` function. Use the `PemnGetApplObjAttributes()` function to access the application object attributes.

BMC Software, Inc., Confidential and Proprietary Information

PemnBDisconnectCallback

Define the disconnect callback type.

Synopsis

```
typedef void (*PemnBDisconnectCallback) (
   PemnCommHandle hComm,
   PemnClientData pClientData
);
```

Parameters

Parameter	Description
hComm	The communication handle.
pClientData	The client data.

Description

The `PemnBDisconnectCallback` defines the C routine that the API will call when a disconnect occurs during `PemnBOpen()` function execution.

For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnClientData

Define the generic client data type.

Synopsis

```
typedef void * PemnClientData;
```

BMC Software, Inc., Confidential and Proprietary Information

PemnCommCallback

Define the communication callback type.

Synopsis

```
typedef void (*PemnCommCallback)(
   void            *pTheObject, /* reserved */
   void            *pClientData,
   PemnCommHandle  hComm
);
```

Description

The API calls the programmer's defined callback when the connection to the PATROL Agent is opened or closed.

BMC Software, Inc., Confidential and Proprietary Information

PemnCommHandle

Define the communication handle type.

Synopsis

```
typedef void *PemnCommHandle;
```

Description

The `PemnCommHandle` communication handle is returned by the `PemnOpen()` function and is used in all subsequent calls to the PATROL Agent.

BMC Software, Inc., Confidential and Proprietary Information

PemnArchiveEventHandler

Define the event archive handler type.

Synopsis

```
typedef void (*PemnArchiveEventHandler) (
   PemnCommHandle hComm,
   char           *pcResult
);
```

Parameters

Parameter	Description
hComm	The communication handle.
pcResult	The event dump text.

Description

When the PATROL API receives the result of event archiving from the agent, it calls the `PemnArchiveEventHandler` registered by `PemnEventArchive`.

PemnEventHandle

Define the event handle type.

Synopsis

```
typedef void* PemnEventHandle;
```

Description

The `PemnEventHandle` is used by the program to access any event attributes.

BMC Software, Inc., Confidential and Proprietary Information

PemnEventStatusEnum

Define the status values type.

Synopsis

```
typedef enum {
   EV_TYPE_OPEN = 1, /*! should be one*/
   EV_TYPE_ACKNOWLEDGED,
   EV_TYPE_CLOSED,
   EV_TYPE_ESCALATED,
   EV_TYPE_DELETED
} PemnEventStatusEnum;
```

BMC Software, Inc., Confidential and Proprietary Information

PemnEventTypeEnum

Define the event type.

Synopsis

```
typedef enum {
EV_TYPE_INFORMATION = 1, /*! should be one*/
EV_TYPE_CHANGE_STATUS,
EV_TYPE_ERROR,
EV_TYPE_WARNING,
EV_TYPE_ALARM,
EV_TYPE_RESPONSE,
EV_TYPE_ACK,
EV_TYPE_MAX
} PemnEventTypeEnum;
```

BMC Software, Inc., Confidential and Proprietary Information

PemnGetApplObjHandler

Define the application object type handler.

Synopsis

```
typedef void (*PemnGetApplObjHandler)(
   PemnCommHandle     hComm,
   PemaApplObjHandle  hObj
);
```

Parameters

Parameter	Description
hComm	The communication handle.
hObj	The application object handle.

Description

`PemnGetApplObjHandler` defines the application object handler that the API calls whenever it receives an application object from the PEM engine.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetEventClassHandler

Define the event class handler type.

Synopsis

```
typedef void (*PemnGetEventClassHandler)(
   PemnCommHandle  hComm,
   char            *pcEventClassText
);
```

Parameters

Parameter	Description
hComm	The communication handle.
oEventHandle	A text string returned by the PATROL Agent that contains all the fields from the event catalog.

Description

`PemnGetEventClassHandler` defines the get event class handler type. The API calls the get event class list handler when it receives a response to a `PemnGetEventClass()` function call.

See Also

`PemnGetEventClass()` on page 8-59 uses the `PemnGetEventClassHandler` type to return the event handler.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetEventClassListHandler

Define the event class list handler type.

Synopsis

```
typedef void (*PemnGetEventClassListHandler)(
   PemnCommHandle  hComm,
   char*           pcEventClassList
);
```

Parameters

Parameter	Description
hComm	The communication handle.
pcEventClassList	A text string returned by the PATROL Agent that contains the list of event classes or • `ERROR` if the event catalog could not be found • `NONE` if no event classes exist

Description

`PemnGetEventClassListHandler` defines the get event class list handler type. The API calls the get event class list handler when it receives the response to a `PemnGetEventClassList()` function call.

See Also

`PemnGetEventClassList()` on page 8-63 uses the `PemnGetEventClassListHandler` type to return the list of event classes from the event catalog.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetInstObjHandler

Define the instance object handler.

Synopsis

```
typedef void (*PemnGetInstObjHandler)(
   PemnCommHandle     hComm,
   PemaInstObjHandle  hObj
);
```

Parameters

Parameter	Description
hComm	The communication handle.
hObj	The application instance object handle.

Description

`PemnGetInstObjHandler` defines the application instance handler that the API calls whenever it receives an application instance object from the PATROL Agent.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetListHandler

Define the get list object handler.

Synopsis

```
typedef void (*PemnGetListHandler)(
   PemnCommHandle  hComm,
   char            *pcList
);
```

Parameters

Parameter	Description
hComm	The communication handle.
pcList	A text string list that can have one of the following formats: **successful**: *full-object-name*\n *comp1*\n *comp2*\n . . . *compn*\n successful with empty list: *full-object-name*\n "NULL_OBJ"\n error (object not found): *full-object-name*\n "ERROR"\n

Description

`PemnGetListHandler` defines the list handler that the API calls whenever it receives a list object from the PEM engine.

PemnGetParamObjHandler

Define the parameter object handler.

Synopsis

```
typedef void (*PemnGetParamObjHandler)(
   PemnCommHandle      hComm,
   PemaParamObjHandle  hObj
);
```

Parameters

Parameter	Description
hComm	The communication handle.
hObj	The parameter object handle.

Description

`PemnGetParamObjHandler` defines the parameter object handler that the API calls whenever it receives a parameter object from the PATROL Agent.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetVarObjHandler

Define the variable object type handler.

Synopsis

```
typedef void (*PemnGetVarObjHandler)(
   PemnCommHandle    hComm,
   PemaVarObjHandle  hObj
);
```

Parameters

Parameter	Description
hComm	The communication handle.
hObj	The variable object handle.

Description

The API calls the PemnGetVarObjHandler type when it receives a variable object from the PATROL Agent.

PemnInstAttrArray

Define the read-only attributes of an application instance object.

Synopsis

```
typedef  char* PemnInstAttrArray[PEMN_INST_MAX_ATTR];
```

BMC Software, Inc., Confidential and Proprietary Information

PemnInstAttrEnum

Define the `Pemn` function application instance attribute type.

Synopsis

```
typedef enum {
PEMN_APPLINSTNAME=0,/* should be always ZERO */
PEMN_APPLINSTWORSTPARAM,
PEMN_APPLINSTSTATUS,
PEMN_APPLINSTRULESTATE,
PEMN_INSTCREATEICON,
PEMN_APPLINSTPARENT,
PEMN_INST_MAX_ATTR/* should be always last */
} PemnInstAttrEnum;
```

Description

The `PemnInstAttrEnum` type defines the index to the read-only attributes of an application instance object.

PemnInstObjHandle

Define the application instance object handle type.

Synopsis

```
typedef void *PemnInstObjHandle;
```

Description

The `PemnInstObjHandle` type is the abstract application instance handle that is returned by the `PemnGetInstObj()` function. Use the `PemnGetInstObjAttributes()` function to access the attributes of the application instance object.

BMC Software, Inc., Confidential and Proprietary Information

PemnParamAttrArray

Define the read-only attributes of a parameter object.

Synopsis

```
typedef  char* PemnParamAttrArray[PEMN_PARAM_MAX_ATTR];
```

BMC Software, Inc., Confidential and Proprietary Information

PemnParamAttrEnum

Define the parameter attribute type.

Synopsis

```
typedef enum {
   PEMN_PARAMNAME=0,     /* should be always ZERO */
   PEMN_PARAMCURRENTTIME,
   PEMN_PARAMPOLLINGINT,
   PEMN_PARAMRETRIES,
   PEMN_PARAMCURRENTVALUE,
   PEMN_PARAMSTATE,
   PEMN_PARAMOUTPUTMODE,
   PEMN_PARAMAUTOSCALE,
   PEMN_PARAM_Y_AXIS_MIN,
   PEMN_PARAM_Y_AXIS_MAX,
   PEMN_PARAM_MAX_ATTR   /* should be always last */
} PemnParamAttrEnum;
```

Description

The `PemnParamAttrEnum` type defines the index to the read-only attributes of a parameter object.

BMC Software, Inc., Confidential and Proprietary Information

PemnParamObjHandle

Define the `Pemn` function parameter object handle type.

Synopsis

```
typedef void *PemnParamObjHandle;
```

Description

The `PemnParamObjHandle` type is the abstract parameter handle returned by the `PemnGetParamObj()` or `PemnGetComputerParamObj()` functions. Use the `PemnGetParamObjAttributes()` functions to access the attributes of the parameter object.

PemnQueryEventHandler

Define the query event handler type.

Synopsis

```
typedef PemnReceiveEventHandler PemnQueryEventHandler;
```

Description

`PemnQueryEventHandler` defines the query event handler type. The `PemnQueryEventHandler` type is a synonym of the `PemnReceiveEventHandler` type. The API calls the query event handler when it receives a response to a query that matches its user-defined filter criteria. The API differentiates between events received normally and those received as the result of a query.

See Also

`PemnEventQuery()` on page 8-39 uses the `PemnQueryEventHandler` type to return events that match a specified filter.

`PemnEventRangeQuery()` on page 8-49 uses the `PemnQueryEventHandler` type to return events from a specified event ID range.

`PemnReceiveEventHandler` type on page 3-30 is the data type to which `PemnQueryEventHandler` is a synonym.

BMC Software, Inc., Confidential and Proprietary Information

PemnReceiveEventHandler

Define the `Pemn` function receive event handler type.

Synopsis

```
typedef void (*PemnReceiveEventHandler)(
   PemnCommHandle  hComm,
   PemnEventHandle oEventHandle
);
```

Note

The following synonyms are provided for reference to prior versions only. Please do not use them; use `PemnReceiveEventHandler` instead.

Synonyms for backward compatibility:

```
typedef PemnReceiveEventHandler PemnEventHandler;
typedef PemnReceiveEventHandler
   PemnEventReceivedHandler;
```

Parameters

Parameter	Description
hComm	The communication handle.
oEventHandle	The event handle.

Description

`PemnReceiveEventHandler` defines the receive event handler type. The API calls the receive event handler when it receives an event that

- matches its user-defined filter
- was *not* in response to a query

BMC Software, Inc., Confidential and Proprietary Information

PemnReportEventHandler

Define the event report handler type.

Synopsis

```
typedef void (*PemnReportEventHandler) (
   PemnCommHandle hComm,
   char           *pcReport
);
```

Parameters

Parameter	Description
hComm	The communication handle.
pcReport	The event report text.

BMC Software, Inc., Confidential and Proprietary Information

PemnReportHandle

Define the report handle type.

Synopsis

```
typedef char* PemnReportHandle;
```

Description

`PemnReportHandle` is the abstract PATROL variable type handle returned by the `PemnGetVarObj()` function.

Note

Programmers should use `PemnReportHandle` as an abstract type instead of a character type (`char *`) because the definition may change in future releases of the PATROL API.

BMC Software, Inc., Confidential and Proprietary Information

PemnUserMessageHandler

Define the message handler type.

Synopsis

```
typedef void (*PemnUserMessageHandler) (
   char  *pcMessageToDisplay,
   int   iMsgType
);
```

Parameters

Parameter	Description
pcMessageToDisplay	The message text.
iMsgType	The integer message type. The types defined in the pemapi.h header are 0 PEMN_MSG_WARN 1 PEMN_MSG_INFO 2 PEMN_MSG_ERROR

Description

`PemnUserMessageHandler` defines the user message handler type for the `Pemn` functions. The API calls the message handler when it receives a message from the PATROL Agent. The programmer can define his or her own message handler (see “PemnRegisterUserMessageHandler()” on page 11-4 for more information).

BMC Software, Inc., Confidential and Proprietary Information

PemnVarObjHandle

Define the variable object handle.

Synopsis

```
typedef char *PemnVarObjHandle;
```

Description

The `PemnVarObjHandle` is the abstract PATROL variable type handle returned by the `PemnGetVarObj()` function.

Note

Programmers should use `PemnVarObjHandle` as an abstract type instead of a character type (`char *`) because the definition may change in future releases of the PATROL API.

BMC Software, Inc., Confidential and Proprietary Information

BMC Software, Inc., Confidential and Proprietary Information

Event Access Functions

This chapter describes the event access function calls that return the details of a PATROL event.

PemnGetArgs() 4-2
PemnGetClassName() 4-3
PemnGetDesc() 4-4
PemnGetDiary() 4-5
PemnGetEid() 4-6
PemnGetEventCatalogName() 4-7
PemnGetExpectancyStr() 4-8
PemnGetHandler() 4-9
PemnGetNode() 4-10
PemnGetNseverity() 4-11
PemnGetOrigin() 4-12
PemnGetOwner() 4-13
PemnGetSourceID() 4-14
PemnGetStatus() 4-15
PemnGetTime() 4-16
PemnGetType() 4-17

BMC Software, Inc., Confidential and Proprietary Information

PemnGetArgs()

Return the list of event parameters.

Synopsis

```
#include <pemapi.h>

char* PemnGetArgs(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose arguments should be returned.

Description

The `PemnGetArgs()` function returns a read-only text string that is a list of the arguments for the event identified by `pEhandle`. The fields in the list of event arguments are separated by the tab string literal (`\t`).

PemnGetClassName()

Return the event class name for the event.

Synopsis

```
#include <pemapi.h>

char* PemnGetClassName(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose event class should be returned.

Description

The `PemnGetClassName()` function returns the read-only name of the event class for the event identified by `pEhandle`.

PemnGetDesc()

Return the event description.

Synopsis

```
#include <pemapi.h>

char *PemnGetDesc(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose description should be returned.

Description

The `PemnGetDesc()` function returns the read-only event description string for the event identified by `pEhandle`.

PemnGetDiary()

Return the content of the PATROL event diary.

Synopsis

```
#include <pemapi.h>

char* PemnGetDiary(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose event diary should be returned.

Description

The `PemnGetDiary()` function returns a read-only event diary string for the event identified by `pEhandle`. The event diary consists of text that is added by a user to annotate, clarify, or further describe the event.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetEid()

Return the PATROL event identifier.

Synopsis

```
#include <pemapi.h>

int    PemnGetEid(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose event identifier should be returned.

Description

The `PemnGetEid()` function returns the integer PATROL event identifier for the event identified by `pEhandle`. The PATROL event identifier is assigned sequentially by the PATROL Event Manager as events are created.

PemnGetEventCatalogName()

Return the PATROL event catalog name for the event.

Synopsis

```
#include <pemapi.h>

char* PemnGetEventCatalogName(
   PemnEventHandle pEhandle
);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose event catalog name should be returned.

Description

The `PemnGetEventCatalogName()` function returns the read-only name of the PATROL event catalog to which the event identified by `pEhandle` belongs.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetExpectancyStr()

Return the life expectancy of the event.

Synopsis

```
#include <pemapi.h>

char* PemnGetExpectancyStr(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose expectancy string should be returned.

Description

The `PemnGetExpectancyStr()` function returns the read-only life expectancy string for the event identified by `pEhandle`. The possible expectancy strings are:

- STORED
- DEL_IF_CLOSED
- DEL_IF_INFO
- DO_NOT_STORE

PemnGetHandler()

Return the user identifier of the event handler.

Synopsis

```
#include <pemapi.h>

char* PemnGetHandler(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose last event handler user ID should be returned.

Description

The `PemnGetHandler()` function returns the user identifier of the last person to modify the event diary of, or perform an ACKNOWLEDGE, CLOSE, or DELETE action on, the event identified by `pEhandle`.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetNode()

Return the host name that reported the PATROL event.

Synopsis

```
#include <pemapi.h>

char *PemnGetNode(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose originating host name should be returned.

Description

The `PemnGetNode()` function returns a pointer to the host name character string for the event identified by `pEhandle`. The host name identifies the computer system that created the PATROL event.

PemnGetNseverity()

Return the numeric severity of the event.

Synopsis

```
#include <pemapi.h>

int PemnGetNseverity(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose numeric severity should be returned.

Description

The `PemnGetNseverity()` function returns the numeric severity of the event identified by `pEhandle`. The numeric severity is an integer from `1` to `5`, `5` being the highest severity.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetOrigin()

Return the PATROL application name that reported the event.

Synopsis

```
#include <pemapi.h>

char *PemnGetOrigin(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose originating application, instance, or parameter name should be returned.

Description

The `PemnGetOrigin()` function returns a pointer to the originating application character string for the event identified by `pEhandle`. The originating application is the PATROL computer or application class that reported the event to the PATROL Event Manager.

PemnGetOwner()

Return the user identifier that owns the event.

Synopsis

```
#include <pemapi.h>

char* PemnGetOwner(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose owner user ID should be returned.

Description

The `PemnGetOwner()` function returns a text string that is the user identifier for the owner of the event identified by `pEhandle`. The event owner is usually a user account.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetSourceID()

Return the PATROL event source identifier.

Synopsis

```
#include <pemapi.h>

char* PemnGetSourceId(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose source ID should be returned.

Description

The `PemnGetSourceID()` function returns a string that is the source identifier for the PATROL event.

PemnGetStatus()

Return the PATROL Event Manager event status.

Synopsis

```
#include <pemapi.h>

int    PemnGetStatus(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose status should be returned.

Description

The `PemnGetStatus()` function returns the integer PATROL status assigned to the event identified by `pEhandle`. The status values are:

Value	Status	Meaning
1	`EV_TYPE_OPEN`	Open
2	`EV_TYPE_ACKNOWLEDGED`	Acknowledged
3	`EV_TYPE_CLOSED`	Closed
4	`EV_TYPE_ESCALATED`	Escalated
5	`EV_TYPE_DELETED`	Deleted

See Also

For information on defining the status values, see “PemnEventStatusEnum” on page 3-14.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetTime()

Return the event timestamp.

Synopsis

```
#include <pemapi.h>

time_t PemnGetTime(PemnEventHandle  pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose timestamp should be returned.

Description

The `PemnGetTime()` function returns the integer timestamp for the event identified by `pEhandle`. The timestamp is expressed as the number of seconds that have elapsed since 00:00:00 GMT, January 1, 1970.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetType()

Return the PATROL event type.

Synopsis

```
#include <pemapi.h>

int PemnGetType(PemnEventHandle pEhandle);
```

Parameter

Parameter	Description
pEhandle	The handle for the event whose type should be returned.

Description

The `PemnGetType()` function returns the integer PATROL type assigned to the event identified by `pEhandle`. The type values are:

Value	Event Type	Meaning
1	`EV_TYPE_INFORMATION`	Information
2	`EV_TYPE_CHANGE_STATUS`	State change
3	`EV_TYPE_ERROR`	Error
4	`EV_TYPE_WARNING`	Warning
5	`EV_TYPE_ALARM`	Alarm
6	`EV_TYPE_RESPONSE`	Response
7	`EV_TYPE_ACK`	Acknowledgment

BMC Software, Inc., Confidential and Proprietary Information

See Also

For information on defining the event types, see “PemnEventTypeEnum” on page 3-15.

5

Connection Management Functions

This chapter describes the connection management functions that open, close, and manage the connection between your program and the PATROL Agent.

PemnBClose() 5-2
PemnBOpen() 5-3
PemnBPing() 5-5
PemnClose() 5-7
PemnGetCommAttributes() 5-8
PemnGetLocalPort() 5-10
PemnLoop() 5-11
PemnOpen() 5-12
PemnSetCommAttributes() 5-15
PemnSetLocalPort() 5-17
PemnSetNoAgentHeartbeat() 5-18

BMC Software, Inc., Confidential and Proprietary Information

PemnBClose()

Close a connection to a PATROL Agent and block until it is done.

Synopsis

```
#include <pemapi.h>

char* PemnBClose(PemnCommHandle hComm);
```

Parameter

Parameter	Description
hComm	The communication handle for the connection which the `PemnBClose()` function should close. The `PemnBOpen()` function sets the handle when the communication connection successfully opens.

Description

The `PemnBClose()` function closes the communication session to a PATROL Agent represented by `hComm`. The `PemnBClose()` function blocks and waits for the session to close.

For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnBOpen()

Open a connection to a PATROL Agent and wait for it to be established.

Synopsis

```
#include <pemapi.h>

char* PemnBOpen(
PemnCommHandle*phComm,
char*pcHostName,
intiPort,
char*pcUserName,
char*pcEncryptedPassword,
intiMaxRetry,
PemnBDisconnectCallbackpDisconnectCallback,
PemnClientDatapClientData
);
```

Parameters

Parameter	Description
phComm	A pointer to a pointer to the handle for this communication connection. The PATROL API stores the communication handle at the address pointed to by `*phComm` when the connection to the PATROL Agent successfully opens. IMPORTANT: A variable holding the communication handle should be either statically defined in the user program or allocated on the heap. It should *never* be allocated on the stack.
pcHostName	Host name on which the PATROL Agent you wish to connect to is running.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
iPort	Port number to which the PATROL Agent you wish to connect to is listening. If the value of this parameter is zero, this function opens a connection using the default port number of 3181.
pcUserName	User name used to start the PATROL Agent session. The `pcUserName` and `pcEncryptedPassword` values together must be a valid account for the PATROL Agent.
pcEncryptedPassword	Password generated by the `PemnEncrypt()` function that is used to start the PATROL Agent session. The `pcEncryptedPassword` and `pcUserName` values together must be a valid account for the PATROL Agent.
iMaxRetry	The maximum number of times that the `PemnBOpen()` function will retry to connect to the PATROL Agent.
pDisconnectCallback	Pointer to a programmer-defined callback routine that the PATROL API will call if the PATROL Agent disconnects during the blocking open session.
pClientData	Data to be passed to `pDisconnectCallback`.

Description

The `PemnBOpen()` function opens a session under `pcUserName` and `pcEncryptedPassword` with the PATROL Agent on `pcHostName` using port `iPort`. The `PemnBOpen()` function is a blocking call that waits until the session connection is open and authenticated by the PATROL Agent. The `PemnBOpen()` function returns the read-only PATROL API message string indicating the status of the function call.

For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnBPing()

Ping a PATROL Agent.

Synopsis

```
#include <pemapi.h>

char* PemnBPing(
char*pcHostName,
intiPort,
intiMsMaxTimeToWait
);
```

Parameters

Parameter	Description
pcHostName	Name of the host to which the PATROL Agent ping is to be sent.
iPort	Port number on `pcHostName` to which the PATROL Agent ping is to be sent.
iMsMaxTimeToWait	Number of milliseconds to wait for a response to the PATROL Agent ping, excluding time for DNS lookup.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnBPing()` function determines whether a PATROL Agent is running on `pcHostName` port number `iPort`. The `PemnBPing()` function will wait up to `iMsMaxTimeToWait` milliseconds for a response from the PATROL Agent. If the `PemnBPing()` function receives a response from the PATROL Agent within the wait time interval, it returns the status of the agent as follows:

- `OK\n`
- `WARN\n`
- `ALARM\n`
- `OFFLINE\n`
- `VOID\n`

Otherwise, it returns the string `PEMN_INTERRUPT`.

For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Example

The following is an example of the `PemnBPing()` function:

```
(void) PemnCheckInit();
for (i = 0; i < 12; i++) {
printf("ping to %s on port %d is: %s\n",
"myhost",
100 + i,
PemnBPing("myhost",100+i,300)
);
}
```

BMC Software, Inc., Confidential and Proprietary Information

PemnClose()

Close the connection with the PATROL Agent.

Synopsis

```
#include <pemapi.h>

void PemnClose(PemnCommHandle hComm);
```

Parameter

Parameter	Description
hComm	The communication handle for the connection that the `PemnClose()` function should close. The `PemnOpen()` function sets the handle value when the communication connection successfully opens.

Description

The `PemnClose()` function requests to close the connection to the PATROL Agent that is specified by the communication handle `hComm`. After the connection is closed, the close callback defined in `PemnOpen()` is called.

Warning

An attempt to use a `hComm` communication handle that has been closed using the `PemnClose()` function will produce unpredictable results.

See Also

`PemnOpen()` on page 5-12 to open a connection to a PATROL Agent.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetCommAttributes()

Return communication connection attributes.

Synopsis

```
#include <pemapi.h>

char* PemnGetCommAttributes(
int*piHeartbeat,
int*piTimeout,
int*piRetries
);
```

Parameters

Parameter	Description
piHeartbeat	A pointer to the storage location for the output of the integer millisecond (ms) value of the communication connection heartbeat. Valid range: 5000 ms (5 seconds) – 18000000 ms (5 hours). The pointer can be NULL if no output value is required. Default: 600000 ms (10 minutes)
piTimeout	A pointer to the storage location for the output of the integer millisecond value of the communication connection timeout. Valid range: 300 ms (0.3 seconds) – 180000 ms (3 minutes). The pointer can be NULL if no output value is required. Default: 2000 ms (2 seconds)
piRetries	A pointer to the storage location for the output of the integer value of the communication connection retries. Valid range: 3 – 10 The pointer can be NULL if no output value is required. Default: 5

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnGetCommAttributes()` function returns pointers to the current heartbeat, timeout, retries, and transport values for communication connections to PATROL Agents.

The returned string is a pointer to the string that specifies the type of network transport protocol in use for the PATROL Agent communication session. The network transport protocol can be one of the following:

- `PEMN_TCP_TRANSPORT`
- `PEMN_UDP_TRANSPORT`

The default value is `PEMN_UDP_TRANSPORT`.

See Also

`PemnSetCommAttributes()` on page 5-15 to modify the communication connection attributes

BMC Software, Inc., Confidential and Proprietary Information

PemnGetLocalPort()

Return the current local port number for communicating with a PATROL Agent across a firewall.

Synopsis

```
#include <pemapi.h>

int PemnGetLocalPort(void);
```

Description

The `PemnGetLocalPort()` function returns the local port number used to communicate with a PATROL Agent across a firewall.

PemnLoop()

Place the program in a loop waiting on activity from the communication connection or from timer-related activity.

Synopsis

```
#include <pemapi.h>

void PemnLoop(void);
```

Description

The `PemnLoop()` function places the program into a wait loop on activity coming from the communication connection. Refer to AppendixB, “Hints,” for information on using your own event loops.

BMC Software, Inc., Confidential and Proprietary Information

PemnOpen()

Open a connection to a PATROL Event Manager or PATROL Agent.

Synopsis

```
#include <pemapi.h>

void PemnOpen(
intiPort,
char*pcRemoteHost,
PemnCommCallbackpCloseCallback,
PemnClientDatapCloseClientData,
PemnCommCallbackpOpenCallback,
PemnClientDatapOpenClientData,
PemnCommHandle*phComm);
```

Parameters

Parameter	Description
iPort	The integer port number on which the PATROL Agent is listening.
pcRemoteHost	The character string that is the name of the computer system on which the PATROL Event Manager is active.
pCloseCallback	A callback routine that the `PemnOpen()` function calls if the connection attempt is unsuccessful.
pCloseClientData	The data for `pCloseCallback`.
pOpenCallback	A callback routine that the `PemnOpen()` function calls when the connection successfully opens.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
`pOpenClientData`	The data for `pOpenCallback`.
`phComm`	A pointer to a pointer to the handle for this communication connection. The PATROL API stores the communication handle at the address pointed to by `*phComm` when the connection to the PATROL Agent successfully opens. `PemnOpen()` returns `NULL` if the connection fails. When the agent session connection fails, results will vary depending on whether the connection was attempted using UDP or TCP: • With UDP, the agent session connection returns a 0 and all callback functions are registered. • With TCP, the agent session connection returns an error and no callback functions are registered. IMPORTANT: A variable holding the communication handle should be either statically defined in the user program or allocated on the heap. It should *never* be allocated on the stack.

Description

The `PemnOpen()` function attempts to open a communication connection to a PATROL Event Manager or PATROL Agent on the host referenced by `*pcRemoteHost` using the port number `iPort`.

If the connection is successful, the `PemnOpen()` function places the output value of the communication handle address into the storage location that is pointed to by `*phComm` and calls the callback routine `pOpenCallback` with `pOpenClientData`.

The callback routine `pOpenCallback` should immediately call the `PemnSendIdentity()` function to authenticate the connection.

If the connection is unsuccessful, the `PemnOpen()` function calls the callback routine `pCloseCallback` with `pCloseClientData`.

BMC Software, Inc., Confidential and Proprietary Information

See Also

`PemnClose()` on page 5-7 to close the connection with the PATROL Event Manager or PATROL Agent.

BMC Software, Inc., Confidential and Proprietary Information

PemnSetCommAttributes()

Set communication connection attributes.

Synopsis

```
#include <pemapi.h>

void PemnSetCommAttributes(
intiHeartbeat,
intiTimeout,
intiRetries
char*pcTransport
);
```

Parameters

Parameter	Description
iHeartbeat	An integer millisecond (ms) value of the communication connection heartbeat. Valid range: 5000 ms (5 seconds) – 18000000 ms (5 hours). The pointer can be NULL if no value is to be returned. Default: 600000 ms (10 minutes) NOTE: The value will be rounded to the nearest second unless it falls between 0 and 100 ms, exclusively; in that case, it will be rounded up to one second. The special value of 0 can be used to disable this parameter.
iTimeout	An integer millisecond value of the communication connection timeout. Valid range: 300 ms (0.3 seconds) – 180000 ms (3 minutes) Default: 2000 ms (2 seconds) The special value of zero can be used to disable this parameter. If 0 is specified for this parameter, the existing setting will not be changed.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
iRetries	An integer value of the communication connection retries. Valid range: 3 – 10 Default: 5 If 0 is specified for this parameter, the existing setting will not be changed.
pcTransport	A string specifying the type of network transport protocol used for the PATROL Agent communication session. Valid range: • PEMN_TCP_TRANSPORT • PEMN_UDP_TRANSPORT Default: PEMN_UDP_TRANSPORT

Description

The `PemnSetCommAttributes()` function sets the heartbeat, timeout, retries, an transport attributes for the communication connections to PATROL Agents.

See Also

`PemnGetCommAttributes()` on page 5-8 to return the current communication connection attributes.

PemnSetLocalPort()

Set the local port number used to communicate with a PATROL Agent across a firewall.

Synopsis

```
#include <pemapi.h>

void PemnSetLocalPort(int ilocalPort);
```

Parameter

Parameter	Description
iLocalPort	Integer port number that is to be used as a local port when communicating with a PATROL Agent across a firewall.

Description

The `PemnSetLocalPort()` function sets the local port number iLocalPort used to communicate with a PATROL Agent across a firewall.

BMC Software, Inc., Confidential and Proprietary Information

PemnSetNoAgentHeartbeat()

Disable the heartbeat feature.

Synopsis

```
#include <pemapi.h>

void PemnSetNoAgentHeartbeat(void);
```

Description

The `PemnSetNoAgentHeartbeat()` function instructs a PATROL Agent to stop using the heartbeat feature to verify the connection with the host that ran `PemnSetNoAgentHeartbeat()`.

Though it disables the heartbeat from the agent to the host, this function allows the heartbeat traffic to continue from the host to the agent. You would want to disable the agent to host heartbeat if it is causing the connected to timeout too often. Losing this side of the heartbeat is not typically significant if using `PemnSetNoAgentHeartbeat()` with a TCP connection, which would usually alert a PATROL Agent when a connection breaks. You should avoid using this function with UDP.

To successfully disable the heartbeat, you need to call this function before opening the connection to the agent. To re-enable the heartbeat, restart the application that called `PemnSetNoAgentHeartbeat()`.

BMC Software, Inc., Confidential and Proprietary Information

Security Management Functions

This chapter describes the security management that provide for account information encryption and identification.

PemnEncrypt() 6-2
PemnSendIdentity() 6-3

BMC Software, Inc., Confidential and Proprietary Information

PemnEncrypt()

Encrypt a password.

Synopsis

```
#include <pemapi.h>

int PemnEncrypt(
   char  *buff,
   int   iLen,
   char  *pcPasswd);
```

Parameters

Parameter	Description
buff	The buffer that will contain the encrypted password.
iLen	The length of the password. This value should be `iLen` ≥ 50.
pcPasswd	The password that is to be encrypted.

Description

The `PemnEncrypt()` function encrypts the password `pcPasswd`. The result can be stored in a configuration file.

PemnSendIdentity()

Send user name and encrypted password to the PATROL Agent.

Synopsis

```
#include <pemapi.h>

int PemnSendIdentity(
   PemnCommHandle  hComm,
   char            *pcUserName,
   char            *pcEncryptedPasswd);
```

Parameters

Parameter	Description
hComm	The handle of the communication session to which the `PemnSendIdentity()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcUserName	User name for access to the PATROL Agent.
pcEncryptedPasswd	Encrypted password for access to the PATROL Event Manager or PATROL Agent. You can encrypt a password using the `PemnEncrypt()` function. (See page 6-2 for more information about the `PemnEncrypt()` function.)

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnSendIdentity()` function sends the user name `pcUserName` and the encrypted password `pcEncryptedPasswd` to the PATROL Agent using the communication handle `hComm`.

The `PemnSendIdentity()` function should be executed immediately after the `PemnOpen()` function to validate access to the PATROL Agent. The PATROL Agent will not accept event requests otherwise.

The `PemnSendIdentity()` function returns TRUE if the identity information is sent to the PATROL Agent. The function returns false if the identity information is not sent to the PATROL Agent.

See Also

`PemnOpen()` on page 5-12 to open a connection to a PATROL Agent.

Send and Receive Functions

This chapter describes the send and receive functions that send PATROL events to, and receive PATROL events from, the PATROL Agent.

PemnReceive() 7-2
PemnSend(), PemnBSend() 7-7

BMC Software, Inc., Confidential and Proprietary Information

PemnReceive()

Register to receive events from the PATROL Event Manager that match the filter criteria.

Synopsis

```
#include <pemapi.h>

void PemnReceive(
char*pcStartTimeFilter,
char*pcEndTimeFilter,
char*pcStatusMaskFilter,
char*pcTypeMaskFilter,
char*pcNseverityFilter,
char*pcNodeFilter,
char*pcOriginFilter,
char*pcPatternFilter,
char*pcEidrange,
char*pcEvClass,
PemnEventReceivedHandlerpEventHandler
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
pcStartTimeFilter	A string that is the timestamp that defines the lower (least recent) bound of the event filter. Format is one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcEndTimeFilter	A string that is the timestamp that defines the upper (most recent) bound of the event filter. Format is one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcStatusMaskFilter	A string of one or more status tags that are separated by commas. Valid range: O OPEN A ACKNOWLEDGED E ESCALATED C CLOSED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED. • `"O,A,C,E,D"` matches all statuses. • `"O,C"` matches only statuses OPEN and CLOSED.
pcTypeMaskFilter	A string of one or more event type tags that are separated by commas. Valid range: I INFORMATION S STATE_CHANGE E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION. • `"I,S,E,W,A,R"` matches all event types. • `"W,A"` matches only event types WARNING and ALARM.
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity.
pcNodeFilter	A string that is the event host name filter. The `""` string will match every host name.
pcOriginFilter	A string that is the event origin filter. The origin can be an application or a host name. The `""` string will match every PATROL application class.
pcPatternFilter	A string that is the event description filter. The `""` string will match any event description.
pcEidrange	A string that defines the range of PATROL event IDs that are eligible for this `PemnReceive()` function. Specify the `pcEidrange` character string as follows: `x` event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events

Parameter	Description
pcEvClass	A string that is the event class filter. Valid range: an event class name, or `""` indicating all event classes.
pEventHandler	The name of the event handler that the PATROL API calls when it receives an event matching the filter criteria.

Description

The `PemnReceive()` function registers a filter to receive events from the PATROL Agent. The API calls the event handler `pEventHandler` for those that match the filter criteria. (Non-matching events are discarded.)

In effect, events that reach `pEventHandler` are double filtered in that the agent only sends events over the communication connection that pass its own persistent filter.

Note

You can use the `PemnSetPersistentFilter()` function to set the persistent filter that the PEM engine uses for a session that was opened by the `PemnOpen()` or `PemnBOpen()` function. The persistent filter is lost when you execute `PemnClose()` to end the session.

BMC Software, Inc., Confidential and Proprietary Information

See Also

`PemnSend()` on page 7-7 to send a PATROL event to the PATROL Event Manager.

`PemnSetPersistentFilter()` on page 8-69 to set the PATROL Event Manager persistent filter for this session.

BMC Software, Inc., Confidential and Proprietary Information

PemnSend(), PemnBSend()

Send a PATROL event to the PATROL Event Manager.

Synopsis

```
#include <pemapi.h>

int PemnSend(
PemnCommHandlehComm,
char*pcEventCatalogName,
char*pcEventClassName,
char*pcEventOrigin,
char*pcOldState,
char*pcNewState,
PemEventTypeEnumeEventType,
intiEventNseverity,
Event parameters
);

char* PemnBSend(
PemnCommHandlehComm,
intiMsMaxTimeToWait
char*pcEventCatalogName,
char*pcEventClassName,
char*pcEventOrigin,
char*pcOldState,
char*pcNewState,
PemEventTypeEnumeEventType,
intiEventNseverity,
Event parameters
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The communication handle of the connection to the Event Manager to which the event is sent. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBSend()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.
pcEventCatalogName	A character string that identifies the event catalog to which this event belongs. The PATROL Event Manager maintains a default event catalog called the standard event catalog `"STD"`. Additionally, each application class can have its own event catalog and event types.
pcEventClassName	A character string that identifies the event class to which the event is sent.
pcEventOrigin	A character string that identifies the application instance or class that produced this event.
pcOldState	A character string that identifies the state of the application instance or class prior to the event. Valid range: • NULL • "VOID" • "OFFLINE" • "OK" • "WARN" • "ALARM"

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcNewState	A character string that identifies the state of the application instance or class after the event. Valid range: • NULL • "VOID" • "OFFLINE" • "OK" • "WARN" • "ALARM"
eEventType	The type of event that is being sent. Valid range: • EV_TYPE_INFORMATION • EV_TYPE _CHANGE_STATUS • EV_TYPE_ERROR • EV_TYPE_WARNING • EV_TYPE_ALARM • EV_TYPE_RESPONSE
iEventNseverity	An integer that identifies the event severity. Valid range: 1 (lowest) – 5 (highest)
Event parameters	Parameters passed to the event to form the event description. The last parameter should be NULL. Valid range: 0 – 20 parameters

Description

The `PemnSend()` and `PemnBSend()` functions send a PATROL event with the specified information to the PATROL Agent whose communication handle is `hComm`. The `PemnSend()` and `PemnBSend()` functions allow you to include a dynamic number of event parameters.

The `PemnSend()` function does not wait for an acknowledgment from the PATROL Agent that it received the event.

BMC Software, Inc., Confidential and Proprietary Information

The `PemnBSend()` function causes the API to loop while waiting for an acknowledgment from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBSend()` function waits up to `iMsMaxTimeToWait` milliseconds. Then one of the following occurs:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBSend()` function call.
- If the `PemnBSend()` function wait time expires, the `PemnBSend()` function returns an error message and the API resumes execution with the statement that follows the `PemnBSend()` function call.
- The `PemnSend()` function returns the event source ID.
- The `PemnBSend()` function returns the event source ID as a read only string.

For a list of return values for blocking functions, see Table 1-1 on page 1-7.

See Also

`PemnReceive()` on page 7-2 to register to receive events from the PATROL Event Manager log that match the filter criteria.

8

Event Management Functions

This chapter describes the event management function calls. These function calls contact the PEM engine logic:

PemnAddDiary() 8-2
PemnBEventScheduleAdd() 8-4
PemnBEventScheduleDelete() 8-7
PemnBEventScheduleList() 8-9
PemnBEventScheduleModify() 8-11
PemnBEventScheduleResume() 8-14
PemnBEventScheduleSuspend() 8-16
PemnDefineEventClass() 8-18
PemnEventArchive(), PemnBEventArchive() 8-21
PemnEventManage() 8-30
PemnEventManageIfMatch(), PemnBEventManageIfMatch() 8-32
PemnEventQuery(), PemnBEventQuery() 8-39
PemnEventRangeManage() 8-46
PemnEventRangeQuery() 8-49
PemnEventReport(), PemnBEventReport() 8-52
PemnFlushEvent() 8-58
PemnGetEventClass(), PemnBGetEventClass() 8-59
PemnGetEventClassList(), PemnBGetEventClassList() 8-63
PemnSetEventClassCmd() 8-66
PemnSetPersistentFilter(), PemnBSetPersistentFilter() 8-69

BMC Software, Inc., Confidential and Proprietary Information

PemnAddDiary()

Add text to the diary of a PATROL event.

Synopsis

```
#include <pemapi.h>

int PemnAddDiary(
   PemnCommHandle   hComm,
   PemnEventHandle  pEh,
   char             *pcTextToAdd);
```

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnAddDiary()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pEh	The handle for the event to which the diary text is added.
pcTextToAdd	The character string that contains the text that is to be added to the event diary.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnAddDiary()` function sends a request to the PEM engine to add the content of character string `pcTextToAdd` to the diary for the event with handle `pEh`. The PEM engine is identified by the communication handle `hComm`.

The `PemnAddDiary()` function returns a 1 if the request was sent successfully to the PEM engine and a 0 if it was not.

See Also

`PemnEventManage()` on page 8-30 to change the status of an event.

BMC Software, Inc., Confidential and Proprietary Information

PemnBEventScheduleAdd()

Add the specified event to the PATROL Agent schedule queue for triggering at a specific time.

Synopsis

```
#include <pemapi.h>

char *PemnBEventScheduleAdd(
   PemnCommHandle     hComm,
   char               *pcTime,
   char               *pcEventCatalogName,
   char               *pcEventClassName,
   char               *pcEventOrigin,
   PemnEventTypeEnum  eEventType,
   int                iEventNseverity,
   int                iMsMaxTimeToWait
   Event Parameters
);
```

Parameter

Parameter	Description
hComm	Handle of the communication connection to which the `PemnBEventScheduleAdd()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcTime	String specifying the GMT time that the PATROL Agent will schedule the event. The format is `"MMddhhmm[yy]"`
pcEventCatalogName	Character string name of the catalog to which the event belongs. Enter `STD_EVENT_CATALOG` if using the PATROL standard event catalog. **Note:** The `pcEventCatalogName` should be previously defined to the PATROL Event Manager.

Parameter	Description
pcEventClassName	Character string name of the class to which the event belongs. **Note:** The `pcEventClassName` should be previously defined to the PATROL Event Manager.
pcEventOrigin	Character string containing the application class or instance name that is identified as having created the event.
eEventType	Event type as displayed in the PATROL Event Manager. **Valid Values** • EV_TYPE_INFORMATION • EV_TYPE _CHANGE_STATUS • EV_TYPE_ERROR • EV_TYPE_WARNING • EV_TYPE_ALARM
iEventNseverity	Integer value indicating the severity of the event. **Valid Values** 1–5 (5 is most severe).
iMsMaxTimeToWait	Integer number of milliseconds that this function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.
Event Parameters	Parameters belonging to the event that you intend to schedule. The last parameter should be NULL. **Valid Values** 0 – 20 parameters

Description

The `PemnBEventScheduleAdd()` function inserts an event into the PATROL Agent schedule queue for triggering at a specified time. When the event is triggered, its notification command is executed. Scheduling an event implies that you are scheduling the OS or PSL Notification command corresponding to that event for execution.

BMC Software, Inc., Confidential and Proprietary Information

If successful, this function returns a scheduling ID. You can use this ID with the following functions to affect a change in the scheduling of an event:

- `PemnBEventScheduleDelete()`
- `PemnBEventScheduleSuspend()`
- `PemnBEventScheduleResume()`
- `PemnBEventScheduleModify()`

The `PemnBEventScheduleAdd()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventScheduleAdd()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleAdd()` function call.

- If the `PemnBEventScheduleAdd()` function wait time expires, the `PemnBEventScheduleAdd()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleAdd()` function call.

The `PemnBEventScheduleAdd()` function returns a scheduling ID if the event was successfully scheduled and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnBEventScheduleDelete()

Remove the specified event from the PATROL Agent schedule queue.

Synopsis

```
#include <pemapi.h>

char *PemnBEventScheduleDelete(
   PemnCommHandle  hComm,
   char            *pcSchId,
   int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to whichthe`PemnBEventScheduleDelete()`function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcSchId	A number that identifies the event which is to be deleted. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventScheduleDelete()` function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnBEventScheduleDelete()` function removes the event identified by `pcSchId` from the PATROL Agent schedule queue. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function. The `PemnBEventScheduleDelete()` function returns the string `PEMN_OK` if successful or an error messages string if not.

The `PemnBEventScheduleDelete()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, this function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleDelete()` function call.
- If the wait time expires, the `PemnBEventScheduleDelete()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleDelete()` function call.

If the `PemnBEventScheduleDelete()` function does not complete successfully, it returns an API message string. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnBEventScheduleList()

List all events scheduled for execution.

Synopsis

```
#include <pemapi.h>

char *PemnBEventScheduleList(
   PemnCommHandle  hComm,
   int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnBEventScheduleList()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventScheduleList()` function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnBEventScheduleList()` function returns a list of events contained in the schedule queue of the PATROL Agent identified by the `hComm` parameter.

BMC Software, Inc., Confidential and Proprietary Information

The `PemnBEventScheduleList()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventScheduleList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleList()` function call.

- If the `PemnBEventScheduleList()` function wait time expires, the `PemnBEventScheduleList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleList()` function call.

If the `PemnBEventScheduleList()` function does not complete successfully, it returns an API message string. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnBEventScheduleModify()

Modify a scheduled event.

Synopsis

```
#include <pemapi.h>

char *PemnBEventScheduleModify(
   PemnCommHandle     hComm,
   char               *pcSchId,
   char               *pcTime,
   char               *pcEventCatalogName,
   char               *pcEventClassName,
   char               *pcEventOrigin,
   PemnEventTypeEnum  eEventType,
   int                iEventNseverity,
   int                iMsMaxTimeToWait
   Event Parameters
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to whichthe`PemnBEventScheduleModify()`function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcSchId	A number that identifies the event which is to be modified. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function.
pcTime	A character string specifying the GMT time that the PATROL Agent will schedule the event. The format is `"MMddhhmm[yy]"`.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcEventCatalogName	A character string name of the catalog to which the event belongs. Enter `STD_EVENT_CATALOG` if using the PATROL standard event catalog. **Note:** The `pcEventCatalogName` should be previously defined to the PATROL Event Manager.
pcEventClassName	A character string name of the class to which the event belongs. **Note:** The `pcEventClassName` should be previously defined to the PATROL Event Manager.
pcEventOrigin	Character string containing the application class or instance name that is identified as having created the event.
eEventType	The event type as displayed in the PATROL Event Manager. **Valid Values** • EV_TYPE_INFORMATION • EV_TYPE _CHANGE_STATUS • EV_TYPE_ERROR • EV_TYPE_WARNING • EV_TYPE_ALARM
iEventNseverity	Integer value indicating the severity of the event. **Valid Values** 1–5 (5 is most severe).
iMsMaxTimeToWait	An integer number of milliseconds that this function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.
Event Parameters	Parameters belonging to the event that you intend to schedule. The last parameter should be NULL. **Valid Values** 0 – 20 parameters

Description

The `PemnBEventScheduleModify()` function modifies the properties of an event in the PATROL Agent schedule queue. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function. The `PemnBEventScheduleModify()` function returns the string `PEMN_OK` if successful or an error messages string if not.

The `PemnBEventScheduleModify()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, this function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleModify()` function call.

- If the wait time expires, the `PemnBEventScheduleModify()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleModify()` function call.

If the `PemnBEventScheduleModify()` function does not complete successfully, it returns an API message string. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnBEventScheduleResume()

Resume an event that was previously suspended.

Synopsis

```
#include <pemapi.h>

char *PemnBEventScheduleResume(
   PemnCommHandle  hComm,
   char            *pcSchId,
   char            *pcTime,
   int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to whichthe`PemnBEventScheduleResume()`function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcSchId	A number that identifies the event which is to be resumed. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function.
pcTime	A character string specifying the GMT time that the PATROL Agent will schedule the event. The format is `"MMddhhmm[yy]"`
iMsMaxTimeToWait	An integer number of milliseconds that the `PemnBEventScheduleResume()` function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnBEventScheduleResume()` function reinstates an event that has been suspended in the PATROL Agent schedule queue. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function. The `PemnBEventScheduleResume()` function returns the string `PEMN_OK` if successful or an error messages string if not.

The `PemnBEventScheduleResume()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, this function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleResume()` function call.

- If the wait time expires, the `PemnBEventScheduleResume()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleResume()` function call.

If the `PemnBEventScheduleResume()` function does not complete successfully, it returns an API message string. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnBEventScheduleSuspend()

Suspend an event that is in the PATROL Agent schedule queue.

Synopsis

```
#include <pemapi.h>

char* PemnBEventScheduleSuspend(
   PemnCommHandle  hComm,
   char            *pcSchId,
   int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnBEventScheduleSuspend()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcSchId	A number that identifies the event which is to be suspended. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventScheduleSuspend()`functionwillwait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnBEventScheduleSuspend()` function suspends an event identified by `pcSchId`. A suspended event remains in the schedule queue of the PATROL Agent, but the agent does not trigger the event at its scheduled time. The `pcSchId` parameter is returned by the `PemnBEventScheduleAdd()` function. The `PemnBEventScheduleSuspend()` function returns the string `PEMN_OK` if successful or an error messages string if not.

The `PemnBEventScheduleSuspend()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, this function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventScheduleSuspend()` function call.

- If the wait time expires, the `PemnBEventScheduleSuspend()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventScheduleSuspend()` function call.

If the `PemnBEventScheduleSuspend()` function does not complete successfully, it returns an API message string. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnDefineEventClass()

Define an event class for a PATROL event catalog.

Synopsis

```
#include <pemapi.h>

int PemnDefineEventClass(
PemnCommHandlehComm,
char*pcEventCatalog,
char*pcEventClass,
char*pcOwner,
char*pcDesc,
char*pcExpertAdvice,
char*pcReserved1,
char*pcReserved2,
char*pcReserved3,
char*pcOperation
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	Communication handle for the connection to which the `PemnDefineEventClass()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pcEventCatalog	String that contains the name of the event catalog in which the event class should be created.
pcEventClass	String that contains the name of the event class that should be created.
pcOwner	String that contains the owner of the event class.
pcDesc	String that contains the text that appears in the **Event Description** field.
pcExpertAdvice	String that contains the text that appears in the **Expert Advice** field.
pcReserved1	String reserved for future use.
pcReserved2	String reserved for future use.
pcReserved3	String reserved for future use.
pcOperation	String indicating the type of event class operation that the `PemnDefineEventClass()` function should perform. Valid Range: PEMN_EC_OPERATIONS_ADD — add the event class if it does not exist PEMN_EC_OPERATIONS_REPLACE — replace the event class PEMN_EC_OPERATIONS_DELETE — delete the event class

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnDefineEventClass()` function manages the creation and deletion of event classes:

If `pcOperation` =	Then ...
`ADD`	The `PemnDefineEventClass()` function adds event class `pcEventClass` to catalog `pcEventCatalog` only if it does not already exist. (It does nothing otherwise).
`REPLACE`	The `PemnDefineEventClass()` function replaces event class `pcEventClass` in catalog `pcEventCatalog` if it exists, or adds it if it does not exist.
`DELETE`	The `PemnDefineEventClass()` function deletes event class `pcEventClass` from `pcEventCatalog`.

The `PemnDefineEventClass()` function returns 1 if it was successfully sent to the PATROL Agent and 0 if it was not.

Note

The resulting event class will be lost when the Agent exits the definition.

PemnEventArchive(), PemnBEventArchive()

Archive PATROL events that match the specified filter criteria to a file.

Synopsis

```
#include <pemapi.h>

int PemnEventArchive(
   PemnCommHandlehComm,
   char*pcEvFormat,
   char*pcEvSep,
   char*pcFileName,
   char*pcFileOperation,
   char*pcStartTimeFilter,
   char*pcEndTimeFilter,
   char*pcStatusFilter,
   char*pcTypeFilter,
   char*pcNseverityFilter,
   char*pcNodeFilter,
   char*pcOriginFilter,
   char*pcPatternFilter,
   char*pcEidrange,
   char*pcEvClass,
   PemnArchiveEventHandlerpArchiveEventHandler
);

char* PemnBEventArchive(
   PemnCommHandlehComm,
   char*pcEvFormat,
   char*pcEvSep,
   char*pcFileName,
   char*pcFileOperation,
   char*pcStartTimeFilter,
   char*pcEndTimeFilter,
   char*pcStatusFilter,
   char*pcTypeFilter,
   char*pcNseverityFilter,
   char*pcNodeFilter,
   char*pcOriginFilter,
   char*pcPatternFilter,
   char*pcEidrange,
   char*pcEvClass,
   int iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Definition
hComm	The communication handle for the connection to which the `PemnEventArchive()` and `PemnBEventArchive()` functions are issued. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pcEvFormat	Format string used to present each event entry. For more information, see “Specifying the Pemn(B)EventArchive() Output Format” on page 8-27. Default: `""` is equivalent to the string `"%s %s %s %s %s %s %s %s\n"` where the eight strings returned are (in order): • event ID assigned by the PEM • event status • event type • event timestamp • host name that produced the event • application class or instance that produced the event • text string from the event description field • text string from the event diary field
pcEvSep	Character string used to separate events in the archive file. If `""` is specified, the `PemnBEventArchive()` or `PemnEventArchive()` function uses the newline character (`\n`) to separate events.
pcFileName	String that is the name of the file where the archived events should be written. If the file name begins with a path separator `/` or `\` (depending on the operating system), the `event_archive()` function assumes that the path name is a full path name. Otherwise, the `event_archive()` function assumes that the file name is a file in the `$PATROL_HOME/log` directory.

Parameter	Definition
pcFileOperation	String that specifies the file access used for archiving events. Valid range: • `PEMN_WRITE` overwrites existing file contents (if any) • `PEMN_APPEND` appends to existing file contents (if any)
***FILTER*:**	
pcStartTimeFilter	Time end-point that specifies the oldest event timestamp that is valid for the event archival. Valid range is `""` indicating January 1, 1970 at 00:00:00, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Definition
pcEndTimeFilter	Time end-point that specifies the most recent event timestamp that is valid for the event archival. Valid range is `""` indicating all event timestamps in the repository, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcStatusFilter	Event status that is valid for the event archival. Valid range: O OPEN A ACKNOWLEDGED C CLOSED E ESCALATED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED. • `"O,A,C,E,D"` or `""` matches all statuses. • `"O,C"` matches only statuses OPEN and CLOSED.

Parameter	Definition
pcTypeFilter	Event type that is valid for the event archival. Valid range: I INFORMATION S CHANGE_STATUS E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION. • `"I,S,E,W,A,R"` or `""` matches all event types. • `"W,A"` matches only event types WARNING and ALARM.
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity..
pcNodeFilter	Computer system name that is valid for the event archival. Valid range is `""`, indicating all computer systems listed in the PEM repository or a host name character string.
pcOriginFilter	Application instance or class name that is valid for the event archival. Valid range is `""` for all application classes or a character string, indicating a specific application instance or class.
pcPatternFilter	Character string within the event description field that is valid for the event archival. Valid range is `""`, indicating any characters or a character string.
pcEidrange	String that defines the range of PATROL event IDs that are valid for the event archival. Valid range: x event ID x x/y event IDs between and including x and y -/y event IDs less than and including y x/- event IDs greater than and including x -/- all events where x and y are integers such that $0 \le x \le y \le 2{,}147{,}483{,}647$

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Definition
pcEvClass	Event class that is valid for the event archival. Valid range is `""`, indicating all event classes or a character string specifying a specific event class.
pArchive EventHandler	The name of the handler that the `PemnEventArchive()` function calls when it receives a response from the PATROL Agent.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventArchive()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnEventArchive()` and `PemnBEventArchive()` functions write PATROL events that match the filter criteria from the PATROL event repository to the file identified by `pcFileName` using `pcEvFormat` and `pcEvSep`.

The `PemnEventArchive()` function does not wait for an acknowledgment from the PATROL Agent that the archival is complete. When a response arrives, the API calls `pArchiveEventHandler`.

The `PemnBEventArchive()` function causes the API to loop while waiting for an acknowledgment from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventArchive()` function waits up to `iMsMaxTimeToWait` milliseconds. Then one of the following occurs:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventArchive()` function call.
- If the `PemnBEventArchive()` function wait time expires, the `PemnBEventArchive()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventArchive()` function call.

The `PemnEventArchive()` function returns 1 if the event archival request was successfully sent to the PATROL Agent and 0 if it was not. The `PemnBEventArchive()` function returns the string `PEMN_OK` if the request was successfully processed by the PATROL Agent, and returns an API message string otherwise. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Specifying the Pemn(B)EventArchive() Output Format

The `PemnEventArchive()` and `PemnBEventArchive()` function `pcEvFormat` parameter is similar to the specification string used for the standard C library `printf()` function. The `pcEvFormat` parameter can contain alphanumeric characters for use as titles and field names, and string literals for spacing, tabbing, and carriage control.

PATROL macro variables within the `pcEvFormat` parameter identify the fields that the `PemnEventArchive()` or `PemnBEventArchive()` function returns. The following table describes the marco variables that are available to the `PemnEventArchive()` and `PemnBEeventArchive()` functions.

PEM Macro	Definition
`%{EV_ID}`	A sequential integer identifier assigned by the PATROL Event Manager upon receipt of the event.
`%{EV_STATUS}`	Event status. Valid range: • OPEN • ACKNOWLEDGED • CLOSED • ESCALATED • DELETED
`%{EV_TYPE}`	Event type. Valid range: • INFORMATION • STATE_CHANGE • ERROR • WARNING • ALARM • RESPONSE

BMC Software, Inc., Confidential and Proprietary Information

PEM Macro	Definition
`%{EV_TIME}`	Time stamp indicating the system clock time at the moment the event was produced. The timestamp is a 24-character string with the format: *day month date hh:mm:ss yyyy* For example, `Sun Sep 16 01:03:52 1973`
`%{EV_NODE}`	Host name that produced the event.
`%{EV_ORIGIN}`	Application instance or class that produced the event.
`%{EV_DESC}`	The text string description that was produced for the event.
`%{EV_DIARY}`	The text string that was entered into the diary for the event.
`%{EV_HANDLER}`	The user ID of the person who performed the last acknowledge, close, or delete action on the event.
`%{EV_OWNER}`	User ID that owns the event. Default: PATROL account user ID
`%{EV_NAME}`	Name of the event within the PATROL event class.
`%{EV_CLASS_NAME}`	Name of the PATROL event class within the PATROL event catalog to which the event belongs.
`%{EV_EXPECTANCY}`	Life expectancy and disposition of the event. Valid range: • STORED • DEL_IF_CLOSED • DEL_IF_INFO • DO_NOT_STORE
`%{EV_NSEVERITY}`	Numeric severity of the event. Event severity is predefined for all event classes in the STANDARD catalog.
`%{EV_ARGS}`	Character string that presents the event parameters separated by tab characters (`\t`).
`%{EV_ARG1}`	PATROL Event Manager first dynamic parameter.
`%{EV_ARG2}`	PATROL Event Manager second dynamic parameter.
`%{EV_CATALOG}`	Name of the PATROL event catalog to which the event belongs.
`%{EV_EXPERT_ADVICE}`	Text string from the **Expert Advice** edit window for this event catalog and class.
`%{EV_SNMP_SUPPORT}`	Text string from the **SNMP Support** field for this event catalog and class. Valid range: • NO_TRAP • SEND_TRAP

BMC Software, Inc., Confidential and Proprietary Information

PEM Macro	Definition
`%{EV_CTG_DESC}`	Text string from the **Description** edit window for this event catalog and class.
`%{EV_ESCL_TEXT}`	Text string from the **Escalation Command** edit window for this event catalog and class.
`%{EV_NOTIFY_TEXT}`	Text string from the **Notification Command** edit window for this event catalog and class.
`%{EV_ACK_TEXT}`	Text string from the **Acknowledge Command** edit window for this event catalog and class.

BMC Software, Inc., Confidential and Proprietary Information

PemnEventManage()

Change the status of a PATROL event.

Synopsis

```
#include <pemapi.h>

int PemnEventManage(
PemnCommHandlehComm,
PemnEventHandlepEh,
char*pcNewStatus);
```

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnEventManage()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pEh	The handle for the event whose status is to be changed.
pcNewStatus	The character string that contains the new status that is to be assigned to the event. Valid range: • PEMN_ACKNOWLEDGED • PEMN_CLOSED • PEMN_DELETED

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnEventManage()` function sends a request to the PEM engine to change the status of the event with handle `pEh` to the new state defined in `pcNewStatus`. The PEM engine is identified by the communication handle `hComm`.

The `PemnEventManage()` function returns a 1 if the request was sent successfully to the PEM engine and 0 if it was not.

Managing events using the `PemnEventManage()` function follows the same rules used for managing events from the PATROL Console or the PATROL Event Manager (PEM). For example, you cannot acknowledge an event that is closed.

See Also

`PemnAddDiary()` on page 8-2 to add text to the diary of a PATROL event.

BMC Software, Inc., Confidential and Proprietary Information

PemnEventManageIfMatch(), PemnBEventManageIfMatch()

Request the PATROL Agent to acknowledge, close, or delete events that match the specified filter criteria.

Synopsis

```
#include <pemapi.h>

int PemnEventManageIfMatch(
   PemnCommHandlehComm,
   int iMaxCount,
   char*pcNewStatus,
   char*pcStartTimeFilter,
   char*pcEndTimeFilter,
   char*pcStatusFilter,
   char*pcTypeFilter,
   char*pcNseverityFilter,
   char*pcNodeFilter,
   char*pcOriginFilter,
   char*pcPatternFilter,
   char*pcEidrange,
   char*pcEvClass
);

char* PemnBEventManageIfMatch(
   PemnCommHandlehComm,
   int iMaxCount,
   char*pcNewStatus,
   char*pcStartTimeFilter,
   char*pcEndTimeFilter,
   char*pcStatusFilter,
   char*pcTypeFilter,
   char*pcNseverityFilter,
   char*pcNodeFilter,
   char*pcOriginFilter,
   char*pcPatternFilter,
   char*pcEidrange,
   char*pcEvClass,
   int iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Definition
hComm	The communication handle for the connection to which the `PemnEventManageIfMatch()` and`PemnBEventManageIfMatch()`functionsare issued. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
iMaxCount	An integer that limits the number of events that will be managed. Specifying `iMaxCount` = 0 causes `iMaxCount` to default to 10.
pcNewStatus	The character string that contains the new status that is to be assigned to events that match the filter criteria. Valid range: PEMN_ACKNOWLEDGED PEMN_CLOSED PEMN_DELETED **Note:** entering an invalid value for this parameter causes the `PemnBEventManageIfMatch()` function to return NULL.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Definition
***FILTER*:**	
pcStartTimeFilter	Time end-point that specifies the oldest event timestamp that is valid for the event archival. Valid range is `""` indicating January 1, 1970 at 00:00:00, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

Parameter	Definition
pcEndTimeFilter	Time end-point that specifies the most recent event timestamp that is valid for the event archival. Valid range is `""` indicating all event timestamps in the repository, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcStatusFilter	Event status that is valid for the event archival. Valid range: O OPEN A ACKNOWLEDGED C CLOSED E ESCALATED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED. • `"O,A,C,E,D"` or `""` matches all statuses. • `"O,C"` matches only statuses OPEN and CLOSED.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Definition
pcTypeFilter	Event type that is valid for the event archival. Valid range: I INFORMATION S CHANGE_STATUS E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION. • `"I,S,E,W,A,R"` or `""` matches all event types. • `"W,A"` matches only event types WARNING and ALARM.
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity.
pcNodeFilter	Computer system name that is valid for the event archival. Valid range is `""`, indicating all computer systems listed in the PEM repository or a host name character string.
pcOriginFilter	Application instance or class name that is valid for the event archival. Valid range is `""` for all application classes or a character string, indicating a specific application instance or class.
pcPatternFilter	Character string within the event description field that is valid for the event archival. Valid range is `""`, indicating any characters or a character string.
pcEidrange	String that defines the range of PATROL event IDs that are valid for the event archival. Valid range: x event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events where x and y are integers such that $0 \le x \le y \le 2{,}147{,}483{,}647$

Parameter	Definition
pcEvClass	Event class that is valid for the event archival. Valid range is `""`, indicating all event classes or a character string specifying a specific event class.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventArchive()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnEventManageIfMatch()` and `PemnBEventManageIfMatch()` functions send a request to the PATROL Agent identified by the session handle `hComm` to change the status of the next `iMaxCount` events that match the specified filter criteria to `pcNewStatus`.

Note

Choose the value of `iMaxCount` with care. Large `iMaxCount` values cause more work for the PATROL Agent and can impact its performance.

The `PemnEventManageIfMatch()` function does not wait for a response from the PATROL Agent.

The `PemnBEventManageIfMatch()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventManageIfMatch()` function will wait up to `iMsMaxTimeToWait` milliseconds. Then one of the following occurs:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventManageIfMatch()` function call.

BMC Software, Inc., Confidential and Proprietary Information

- If the `PemnBEventManageIfMatch()` function wait time expires, the `PemnBEventManageIfMatch()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventManageIfMatch()` function call.

The `PemnEventManageIfMatch()` function returns 1 if the event management request was successfully sent to the PATROL Agent and 0 if it was not.

The `PemnBEventManageIfMatch()` function returns the string `PEMN_OK` if the request was successfully executed by the PATROL Agent and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnEventQuery(), PemnBEventQuery()

Return PATROL events that match the specified filter.

Synopsis

```
#include <pemapi.h>

int PemnEventQuery(
PemnCommHandlehComm,
intiQueryLength,
char*pcStartTimeFilter,
char*pcEndTimeFilter,
char*pcStatusFilter,
char*pcTypeFilter,
char*pcNseverityFilter
char*pcNodeFilter,
char*pcOriginFilter,
char*pcPatternFilter
char*pcEidrange,
char*pcEvClass,
PemnQuryEventHandlerpQueryEventHandler
);

char PemnBEventQuery(
PemnCommHandlehComm,
intiQueryLength,
char*pcEvSep,
char*pcEvFormat,
char*pcStartTimeFilter,
char*pcEndTimeFilter,
char*pcStatusFilter,
char*pcTypeFilter,
char*pcNseverityFilter
char*pcNodeFilter,
char*pcOriginFilter,
char*pcPatternFilter
char*pcEidrange,
char*pcEvClass,
intiMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnEventQuery()` or `PemnBEventQuery()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
iQueryLength	An integer that limits the number of events that will be returned. Specifying `iQueryLength=0` causes `iQueryLength` to default to 10.
pcEvSep	A string that is used to separate each event in the list of events returned by the event query. Valid range: • `""` indicating that a newline character `\n` will separate the entries • any valid characters including string literals
pcEvFormat	Format string used to present each event entry. For more information, see “Specifying the Pemn(B)EventArchive() Output Format” on page 8-27. Default: `""` is equivalent to the string `"%s %s %s %s %s %s %s %s\n"` where the eight strings returned are (in order): • event ID assigned by the PEM • event status • event type • event timestamp • host name that produced the event • application class or instance that produced the event • text string from the event description field • text string from the event diary field

Parameter	Description
***FILTER*:**	
pcStartTimeFilter	Time end-point that specifies the oldest event timestamp that is valid for the event archival. Valid range is `""` indicating January 1, 1970 at 00:00:00, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcEndTimeFilter	Time end-point that specifies the most recent event timestamp that is valid for the event archival. Valid range is `""` indicating all event timestamps in the repository, or one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss year* • PSL `date()`: *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are as follows: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month*— Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date*—01 to 31 *hh* — 00 to 23 *mm* and *ss*—00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcStatusFilter	Event status that is valid for the event archival. Valid range: O OPEN A ACKNOWLEDGED C CLOSED E ESCALATED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED. • `"O,A,C,E,D"` or `""` matches all statuses. • `"O,C"` matches only statuses OPEN and CLOSED.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcTypeFilter	A string of one or more event type tags that are separated by commas. Valid range: I INFORMATION S STATE_CHANGE E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION • `"I,S,E,W,A,R"` matches all event types • `"W,A"` matches only event types WARNING and ALARM
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity.
pcNodeFilter	A string that is the event host name filter. The `""` string will match every host name.
pcOriginFilter	A string that is the event origin filter. The origin can be an application or host name. The `""` string will match every PATROL application class.
pcPatternFilter	A string that is the event description filter. The `""` string will match any event description.
pcEidrange	A string that defines the range of PATROL event IDs that are eligible for this `pemnEventQuery()` function. Specify the `pcEidrange` character string as follows: x event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events
pcEvClass	A string that is the event class filter. Valid range: an event class name or `""` indicating all event classes.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pQueryEventHandler	The name of the event handler that the `PemnEventQuery()` function calls when its filter criteria matches an event from the PATROL Event Manager.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventQuery()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnEventQuery()` and `PemnBEventQuery()` functions send a request to the PATROL Agent to return up to `iQueryLength` events that match the specified filter parameters.

The `PemnEventQuery()` function does not wait for a response from the PATROL Agent. When a response arrives, the API calls `pQueryEventHandler`.

The `PemnBEventQuery()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventQuery()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBEventQuery()` function call.
- If the `PemnBEventQuery()` function wait time expires, the `PemnBEventQuery()` function returns an error message and the API resumes execution with the statement that follows the `PemnBEventQuery()` function call.

The `PemnEventQuery()` function returns a 1 if the query was sent successfully to the PEM engine and 0 if it was not. The `PemnBEventQuery()` function returns up to `iQueryLength` events formatted using `pcEvSep` and `pcEvFormat` if the query was successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Note

Choose the value of `iQueryLength` with care. Large `iQueryLength` values cause more work for the PEM engine and can impact its performance.

Under some circumstances, the `PemnBEventQuery()` function returns NULL when an invalid parameter is used to call this function. For example, it returns NULL when an invalid origin is passed as the `pcOriginFilter` parameter.

See Also

`PemnEventRangeQuery()` on page 8-49 to return PATROL events from a specified event ID range.

BMC Software, Inc., Confidential and Proprietary Information

PemnEventRangeManage()

Change the status of PATROL events within a specified event ID range.

Synopsis

```
#include <pemapi.h>

int PemnEventRangeManage(
   PemnCommHandle  hComm,
   char            *pcEidRange,
   char            *pcNewStatus,
   char            *pcUsrName,
   int             iMaxCount);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnEventRangeManage()` and `PemnBEventRangeManage()` functions are issued. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pcEidRange	A character string that defines the range of PATROL event IDs that are eligible for this `PemnEventRangeManage()` function. Specify the `pcEidRange` character string as follows: `x` event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events
pcNewStatus	The character string that contains the new status that is to be assigned to the event range. Valid range: PEMN_ACKNOWLEDGED PEMN_CLOSED PEMN_DELETED
pcUsrName	A character string that contains the user name of the person or process changing the event status. The content of character string `pcUsrName` is placed in the event diary of each event whose status is changed.
iMaxCount	An integer that limits the number of events that will be affected by the status change. Specifying `iMaxCount = 0` causes `iMaxCount` to default to 10.

Description

The `PemnEventRangeManage()` function sends a request to the PEM engine to assign `pcNewStatus` as the new event status for up to `iMaxCount` events whose event IDs are within the range specified by the content of the character string `pcEidRange`. The `PemnEventRangeManage()` function also requests that the content of

BMC Software, Inc., Confidential and Proprietary Information

the character string `pcUsrName` be added to the event diary of each modified event. All the events belong to the PEM engine whose communication handle is `hComm`.

The `PemnEventRangeManage()` function returns a 1 if the request was sent successfully to the PEM engine and a 0 if it was not.

Note

Choose the value of `iMaxCount` with care. Large `iMaxCount` values cause more work for the PEM engine and can impact its performance.

Managing events using the `PemnEventRangeManage()` function follow the same rules used for managing events from the PATROL Console or the PATROL Event Manager (PEM) Console. For example, you cannot acknowledge an event that is closed.

PemnEventRangeQuery()

Return PATROL events from a specified event ID range.

Synopsis

```
#include <pemapi.h>

int PemnEventRangeQuery(
   PemnCommHandle         hComm,
   char                   *pcEidRange,
   int                    iQueryLength
   PemnQueryEventHandler  pQueryEventHandler);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnEventRangeQuery()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pcEidRange	A character string that defines the range of PATROL event IDs that are eligible for this `PemnEventRangeQuery()` function. Specify the `pcEidRange` character string as follows: `x` event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events
iQueryLength	An integer that limits the number of events that will be affected by the status change. Specifying `iQueryLength = 0` causes `iQueryLength` to default to the `pemMaxQuery` value defined for the PATROL Event Manager in its `config.default` file. (The default `pemMaxQuery` value is 10).
pQueryEventHandler	The name of the event handler that the `PemnEventRangeQuery()` function calls when its filter criteria matches an event returned from the PATROL Event Manager.

Description

The `PemnEventRangeQuery()` function requests the PEM engine to return up to `iQueryLength` PATROL events whose event IDs are within the range specified by the content of the character string `pcEidRange`. The events are sent by the PATROL Agent and those that match the event ID range criteria are received by the event handler `pQueryEventHandler`.

BMC Software, Inc., Confidential and Proprietary Information

The `PemnEventRangeQuery()` function returns a 1 if the query was sent successfully to the PATROL Agent a 0 if it was not.

Note

Choose the value of `iQueryLength` with care. Large `iQueryLength` values cause more work for the PATROL Agent and can impact its performance.

See Also

`PemnEventQuery()` on page 8-39 to return PATROL events that match the specified filter.

BMC Software, Inc., Confidential and Proprietary Information

PemnEventReport(), PemnBEventReport()

Return a report summarizing all events that match the specified filter criteria.

Synopsis

```
int PemnEventReport(
   PemnCommHandle          hComm,
   char                    *pcFormat,  /* reserved */
   char                    *pcStartTimeFilter,
   char                    *pcEndTimeFilter,
   char                    *pcStatusFilter,
   char                    *pcTypeFilter,
   char                    *pcNseverityFilter,
   char                    *pcNodeFilter,
   char                    *pcOriginFilter,
   char                    *pcPatternFilter,
   char                    *pcEidrange,
   char                    *pcEvClass,
   PemnReportEventHandler  pReportEventHandler
);

char* PemnBEventReport(
   PemnCommHandle  hComm,
   char            *pcFormat,          /* reserved */
   char            *pcStartTimeFilter,
   char            *pcEndTimeFilter,
   char            *pcStatusFilter,
   char            *pcTypeFilter,
   char            *pcNseverityFilter,
   char            *pcNodeFilter,
   char            *pcOriginFilter,
   char            *pcPatternFilter,
   char            *pcEidrange,
   char            *pcEvClass,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the `PemnEventReport()` function is submitted. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
pcFormat	Field reserved for future use. (The event report format will be user-definable.)
pcStartTimeFilter	A string that is the timestamp that defines the lower (least recent) bound of the event filter. Format: one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL date(): *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month* — Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date* — 01 to 31 *hh* — 00 to 23 *mm* and *ss* — 00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcEndTimeFilter	A string that is the timestamp that defines the upper (most recent) bound of the event filter. Format: one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL date(): *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month* — Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date* — 01 to 31 *hh* — 00 to 23 *mm* and *ss* — 00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcStatusFilter	A string of one or more status tags that are separated by commas. Valid range: O OPEN A ACKNOWLEDGED E ESCALATED C CLOSED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED • `"O,A,C,E,D"` matches all statuses • `"O,C"` matches only statuses OPEN and CLOSED

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcTypeFilter	A string of one or more event type tags that are separated by commas. Valid range: I INFORMATION S STATE_CHANGE E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION • `"I,S,E,W,A,R"` matches all event types • `"W,A"` matches only event types WARNING and ALARM
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity.
pcNodeFilter	A string that is the event host name filter. The `""` string will match every host name.
pcOriginFilter	A string that is the event origin filter. The origin can be an application or host name. The `""` string will match every PATROL application class.
pcPatternFilter	A string that is the event description filter. The `""` string will match any event description.
pcEidrange	A string that defines the range of PATROL event IDs that are eligible for this `pemnEventQuery()` function. Specify the `pcEidrange` character string as follows: x event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events
pcEvClass	A string that is the event class filter. Valid range: an event class name or `""` indicating all event classes.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
[nb]pReportEventHandler	The name of the event handler that the `PemnEventReport()` function calls when its filter criteria matches an event from the PATROL Event Manager. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBEventReport()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnEventReport()` and `PemnBEventReport()` functions cause the PATROL Agent to generate a report summarizing events that match the specified filter criteria.

The `PemnEventReport()` function does not wait for a response from the PATROL Agent. When a response arrives, the API calls `pReportEventHandler`.

The `PemnBEventReport()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBEventReport()` function will wait up to `iMsMaxTimeToWait`milliseconds:

- if a response arrives, the API resumes execution at the statement that follows the `PemnBEventReport()` function call
- if the `PemnBEventReport()` function wait time expires, the `PemnBEventReport()` function returns a timeout error message and the API resumes execution with the statement that follows the `PemnBEventReport()` function call

BMC Software, Inc., Confidential and Proprietary Information

The `PemnEventReport()` function returns 1 if the event report request was successfully sent to the PATROL Agent and 0 if it was not. The `PemnBEventReport()` function returns the event report if successful, or an API message string if not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

The following is an example of the text string returned from the PATROL Agent using the `PemnEventReport()` and `PemnBEventReport()` functions.

```
----------------------------------------------------------------------
Number of events in the Repository: 821
Number of events in cache: 134
Number of events on disk: 687

Status
  OPEN: 821
  ACKNOWLEDGED: 0
  CLOSED: 0
  ESCALATED: 0
  DELETED: 0

Type
  ALARM: 3
  WARNING: 10
  ERROR: 0
  STATE_CHANGE: 806
  INFORMATION: 2
  RESPONSE: 0

Time
  First event: Tue Mar 19 03:34:43 1996
  Last  event: Tue Mar 19 08:25:51 1996

Event Id
  Smallest Id: 14595
  Greatest Id: 16069

Event size:
  Average event size: 225 bytes
  Maximum event size: 287 bytes
```

BMC Software, Inc., Confidential and Proprietary Information

PemnFlushEvent()

Flush PATROL events to the event log on disk.

Synopsis

```
#include <pemapi.h>

void PemnFlushEvent(PemnCommHandle hComm);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnFlushEvent()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.

Description

The `PemnFlushEvent()` function instructs the PEM engine on the connection `hComm` to write all the events accumulated in its internal buffers to the log file, usually `PEM.log`.

Note

You do not need to flush events in order to use the `PemnReceive()` function.

PemnGetEventClass(), PemnBGetEventClass()

Return selected fields from the event class.

Synopsis

```
#include <pemapi.h>

int PemnGetEventClass(
   PemnCommHandle            hComm,
   char                      *pcEventCatalog,
   char                      *pcEventClass,
   char                      *pcEvFormat,
   PemnGetEventClassHandler  pEventClassHandler
);

char* PemnBGetEventClass(
   PemnCommHandle            hComm,
   char                      *pcEventCatalog,
   char                      *pcEventClass,
   char                      *pcEvFormat,
   int                       iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetEventClass()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcEventCatalog	The string that is the name of the event catalog (usually an application name) to which the event class belongs.
pcEventClass	The string that is the event class name.
pcEvFormat	Format string used to present event class information. See “Specifying the PemnGetEventClass() Function Output Format” on page 8-61 for more information.
pEventClassHandler	The name of the event handler that the `PemnGetEventClass()` function calls when it receives a response from the PATROL Event Manager.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetEventClass()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnGetEventClass()` and `PemnBGetEventClass()` functions request that the PEM engine return a character string containing fields from the event class `pcEventClass` in catalog `pcEventCatalog`. The returning character string is formatted as specified in `pcEvFormat`.

The `PemnGetEventClass()` function does not wait for a response from the PATROL Agent. When a response arrives, the API calls `pEventClassHandler`.

The `PemnBGetEventClass()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBGetEventClass()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- if a response arrives, the API resumes execution at the statement that follows the `PemnBGetEventClass()` function call
- if the `PemnBGetEventClass()` function wait time expires, the `PemnBGetEventClass()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetEventClass()` function call

The `PemnGetEventClass()` function returns 1 if the event class field request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetEventClass()` function returns the event class information formatted using `pcEvFormat` if successful and an API message string if not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Under some circumstances, the `PemnBGetEventClass()` function returns NULL when an invalid parameter is used to call this function. For example, it returns NULL when you use NULL as the `hComm` parameter or the NULL string ("") as `pcEvFormat` or `pcEventClass`.

Specifying the PemnGetEventClass() Function Output Format

The `PemnGetEventClass()` and `PemnBGetEventClass()` functions `pcEvFormat` parameter is similar to the specification string for the standard C library `printf()` function. The `pcEvFormat` parameter can contain alphanumeric characters for use as titles and field names, and string literals for spacing, tabbing, and carriage control.

BMC Software, Inc., Confidential and Proprietary Information

PATROL macro variables within the `pcEvFormat` parameter identify the fields that the `PemnGetEventClass()` or `PemnBGetEventClass()` functions return.

Note

The PATROL macro variables for the `PemnGetEventClass()` and `PemnBGetEventClass()` functions return information from the event catalog. To view this information from the PATROL Developer Console you can open the **Event Editor** dialog box.

Macro Variable	Definition
`%{EV_EXPERT_ADVICE}`	Text string from the **Expert Advice** edit window for this event catalog and class.
`%{EV_SNMP_SUPPORT}`	Text from the **SNMP Support** field for this event catalog and class. Valid range: NO_TRAP SEND_TRAP
`%{EV_CTG_DESC}`	Text string from the **Description** edit window for this event catalog and class.
`%{EV_ESCL_TEXT}`	Text string from the **Escalation Command** edit window for this event catalog and class.
`%{EV_NOTIFY_TEXT}`	Text string from the **Notification Command** edit window for this event catalog and class.
`%{EV_ACK_TEXT}`	Text string from the **Acknowledge Command** edit window for this event catalog and class.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetEventClassList(), PemnBGetEventClassList()

Return the list of event classes from an event catalog.

Synopsis

```
#include <pemapi.h>

int PemnGetEventClassList(
   PemnCommHandle                hComm,
   char                          *pcEventCatalog,
   PemnGetEventClassListHandler  pEventClassListHandler
);

char* PemnBGetEventClassList(
   PemnCommHandle                hComm,
   char                          *pcEventCatalog,
   int                           iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the event class list function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcEventCatalog	The string that is the name of the event catalog (usually an application name) whose event classes should be returned.
pEventClassList-Handler	The name of the event handler that the `PemnGetEventClassList()` function calls when it receives a response from the PATROL Event Manager.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetEventClassList()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnGetEventClassList()` and `PemnBGetEventClassList()` functions request that the PEM engine return a list of event classes from the event catalog `pcEventCatalog`.

The `PemnGetEventClassList()` function *does not* wait for a response from the PATROL Agent. When a response arrives, the API calls `pEventClassListHandler`.

The `PemnBGetEventClassList()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBGetEventClassList()` function waits up to `iMsMaxTimeToWait` milliseconds:

- if a response arrives, the API resumes execution at the statement that follows the `PemnBGetEventClassList()` function call

- if the `PemnBGetEventClassList()` function wait time expires, the `PemnBGetEventClassList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetEventClassList()` function call

The `PemnGetEventClassList()` function returns 1 if the event class field request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetEventClassList()` function returns the requested list of event classes if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Under some circumstances, the `PemnBGetEventClassList()` function returns NULL when an invalid parameter is used to call this function. For example, it returns NULL when you use NULL as the `hComm` parameter.

BMC Software, Inc., Confidential and Proprietary Information

PemnSetEventClassCmd()

Define a command in an event class.

Synopsis

```
#include <pemapi.h>

int PemnSetEventClassCmd(
   PemnCommHandle  hComm,
   char            *pcEventCatalog,
   char            *pcEventClass,
   char            *pcCmdType,
   char            *pcCmdText,
   char            *pcOperation
   char            *pcSpecificData
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication session to which the `PemnSetPersistentFilter()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcEventCatalog	String that is the name of the event catalog (usually an application name) that contains `pcEventClass`.
pcEventClass	String that is the name of the event class to which a command is to be added.
pcCmdType	Command type that is being defined. Valid range: PEMN_OS_NOTIFICATIONPEMN_PSL_NOTIFICATION PEMN_OS_ESCALATIONPEMN_PSL_ESCALATION PEMN_OS_MGT PEMN_PSL_MGT
pcCmdText	The statements of the OS or PSL command.
pcOperation	String indicating the type of event class operation thatthe`PemnDefineEventClass()`functionshould perform. Valid Range: PEMN_EC_OPERATIONS_ADD — add the command if it does not exist PEMN_EC_OPERATIONS_REPLACE — replace the command PEMN_EC_OPERATIONS_DELETE — delete the command
pcSpecificData	String containing additional data specific to the command.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnSetEventClassCmd()` function manages the creation and deletion of event class commands:

- If `pcOperation = ADD`, the `PemnSetEventClassCmd()` function adds a `pcCmdType` command with `pcCmdText` to `pcEventClass` and `pcEventCatalog` only if it does not already exist (it does nothing otherwise).

- If `pcOperation = REPLACE`, the `PemnSetEventClassCmd()` function replaces event class `pcCmdType` with `pcCmdText` in `pcEventClass` and `pcEventCatalog` if it exists, or adds it if it does not exist.

- If `pcOperation = DELETE`, the `PemnSetEventClassCmd()` function deletes event class `pcCmdType` from `pcEventClass` and `pcEventCatalog`.

Note

Commands added using the `PemnSetEventClassCmd()` function remain while the PATROL Agent is running but are not saved as part of the PATROL Agent and will be lost when the PATROL Agent is restarted.

The `PemnSetEventClassCmd()` function returns 1 if it was successfully sent to the PATROL Agent and 0 if it was not.

PemnSetPersistentFilter(), PemnBSetPersistentFilter()

Set the PATROL Event Manager persistent filter for this session.

Synopsis

```
#include <pemapi.h>

int PemnSetPersistentFilter(
   PemnCommHandle  hComm
   char            *pcStartTimeFilter,
   char            *pcEndTimeFilter,
   char            *pcStatusMaskFilter,
   char            *pcTypeMaskFilter,
   char            *pcNseverityFilter,
   char            *pcNodeFilter,
   char            *pcOriginFilter,
   char            *pcPatternFilter,
   char            *pcEidrange,
   char            *pcEvClass
);

char* PemnBSetPersistentFilter(
   PemnCommHandle  hComm
   char            *pcStartTimeFilter,
   char            *pcEndTimeFilter,
   char            *pcStatusFilter,
   char            *pcTypeFilter,
   char            *pcNseverityFilter,
   char            *pcNodeFilter,
   char            *pcOriginFilter,
   char            *pcPatternFilter,
   char            *pcEidrange,
   char            *pcEvClass,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication session to which the `PemnSetPersistentFilter()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcStartTimeFilter	A string that is the timestamp that defines the lower (least recent) bound of the event filter. Format: one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL date(): *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month* — Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date* — 01 to 31 *hh* — 00 to 23 *mm* and *ss* — 00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.
pcEndTimeFilter	A string that is the timestamp that defines the upper (most recent) bound of the event filter. Format: one of the following strings: • PSL backward compatible: *MMddhhmmyyyy* • RFC-822: *day*, *date month year hh*:*mm*:*ss* • UNIX: *day month date hh*:*mm*:*ss yyyy* • PSL date(): *day month date hh*:*mm*:*ss yyyy* where the valid ranges of the parameters are: *day* — Sun Mon Tue Wed Thu Fri Sat *MM* — 01 to 12 *month* — Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec *dd* or *date* — 01 to 31 *hh* — 00 to 23 *mm* and *ss* — 00 to 59 *yyyy* — 1902 to 2037. In the PSL compatibility format the current year is used when *yyyy* is omitted.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcStatusFilter	A string of one or more status tags that are separated by commas. Valid range: O OPEN A ACKNOWLEDGED E ESCALATED C CLOSED D DELETED For example: • `"O,A,C,D"` matches all statuses except ESCALATED • `"O,A,C,E,D"` matches all statuses • `"O,C"` matches only statuses OPEN and CLOSED
pcTypeFilter	A string of one or more event type tags that are separated by commas. Valid range: I INFORMATION S STATE_CHANGE E ERROR W WARNING A ALARM R RESPONSE For example: • `"S,E,W,A,R"` matches all event types except INFORMATION • `"I,S,E,W,A,R"` matches all event types • `"W,A"` matches only event types WARNING and ALARM
pcNseverityFilter	A string that is the event severity. Valid range: a number 1 – 5 with 5 being the most severe, or `""` matching every event severity.
pcNodeFilter	A string that is the event host name filter. The `""` string will match every host name.
pcOriginFilter	A string that is the event origin filter. The origin can be an application or host name. The `""` string will match every PATROL application class.
pcPatternFilter	A string that is the event description filter. The `""` string will match any event description.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcEidrange	A string that defines the range of PATROL event IDs that should be passed by the filter. Specify the `pcEidrange` character string as follows: x event ID x `x/y` event IDs between and including x and y `-/y` event IDs less than and including y `x/-` event IDs greater than and including x `-/-` all events
pcEvClass	A string that is the event class filter. Valid range: an event class name or `""` indicating all event classes.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBSetPersistentFilter()` function will wait for a reply from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnSetPersistentFilter()` and `PemnBSetPersistentFilter()` functions set the persistent filter that the PEM engine will use to filter events that it sends to this session using the communication handle `hComm`.

The `PemnSetPersistentFilter()` function does not wait for a response from the PATROL Agent.

The `PemnBSetPersistentFilter()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBSetPersistentFilter()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- if a response arrives, the API resumes execution at the statement that follows the `PemnBSetPersistentFilter()` function call

- if the `PemnBSetPersistentFilter()` function wait time expires, the `PemnBSetPersistentFilter()` function returns an error message and the API resumes execution with the statement that follows the `PemnBSetPersistentFilter()` function call

The `PemnSetPersistentFilter()` function returns 1 if the new filter was successfully sent to the PEM engine and 0 if it was not. The `PemnBSetPersistentFilter()` function returns the string `PEMN_OK` if the new filter was successfully set in the PATROL Agent and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Under some circumstances, the `PemnBSetPersistentFilter()` function returns NULL when an invalid parameter is used to call this function. For example, it returns NULL when you use NULL as the `hComm` parameter.

Changing the persistent filter for one `hComm` handle will not change the persistent filter for other `hComm` handles. When the `PemnClose()` function closes the communication connection with the PEM engine, the persistent filter settings for the session are lost.

See Also

`PemnReceive()` on page 7-2 to register to receive events from the PATROL Event Manager log that match the filter criteria.

BMC Software, Inc., Confidential and Proprietary Information

Application Management Functions

This chapter describes the application management function calls that return information about PATROL computer, application class, instance, parameter, and variable objects.

PemnBPslExec() 9-2
PemnBSetVarObj() 9-4
PemnGetApplList(), PemnBGetApplList() 9-6
PemnGetApplObj(), PemnBGetApplObj() 9-9
PemnGetApplObjAttributes() 9-12
PemnGetApplObjInfo() 9-13
PemnGetComputerParamList(), PemnBGetComputerParamList() . . 9-15
PemnGetComputerParamObj(), PemnBGetComputerParamObj() . . 9-18
PemnGetInstList(), PemnBGetInstList() 9-21
PemnGetInstObj(), PemnBGetInstObj() 9-24
PemnGetInstObjAttributes() 9-27
PemnGetInstObjInfo() 9-28
PemnGetParamList(), PemnBGetParamList() 9-30
PemnGetParamObj(), PemnBGetParamObj() 9-33
PemnGetParamObjAttributes() 9-36
PemnGetParamObjInfo() 9-37
PemnGetVarList(), PemnBGetVarList() 9-39
PemnGetVarObj(), PemnBGetVarObj() 9-42

BMC Software, Inc., Confidential and Proprietary Information

PemnBPslExec()

Execute the specified PSL script.

Synopsis

```
#include <pemapi.h>

char* PemnBPslExec(
   PemnCommHandle  hComm,
   int             iMsMaxTimeToWait,
   int             iWaitForResults,
   char            *pcText
);
```

Parameters

Parameter	Description
hComm	The communication handle for the connection to which the PSL script is sent. The `PemnOpen()` or `PemnBOpen()` function returns the handle when the communication connection successfully opens.
iMsMaxTimeToWait	Number of milliseconds to wait for a response from the PATROL Agent. The minimum permitted value is 100. Default: The value specified by the `PemnBSetTimeout()` function call or 60000 milliseconds (1 minute).
iWaitForResults	Flag specifying whether the PemnBPslExec() function should wait for the result of the PSL script execution. Valid range: 0 Wait only for acknowledgment that the PATROL Agent received the PSL script 1 Wait for the result of the PSL script execution (blocking)
pcText	Text of the PSL script that is to be executed by the PATROL Agent.

Description

The `PemnBPslExec()` function sends the PSL script `pcText` to the PATROL Agent identified by the communication handle `hComm`. If `iWaitForResults` = 1, the `PemnBPslExec()` function will wait for, and return, the read-only result of the PSL script execution. Otherwise, the `PemnBPslExec()` function will return a read-only API message string indicating the status of the operation. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

The PemnBPslExec() function and the API do not perform any compilation or syntax checks on `pcText` before sending it to the PATROL Agent. That is the user's responsibility. (Hint: use the PATROL Console to edit the PSL script and verify the syntax.)

BMC Software, Inc., Confidential and Proprietary Information

PemnBSetVarObj()

Request the PATROL Agent to set a PATROL variable value.

Synopsis

```
#include <pemapi.h>

PemnVarObjHandle PemnBSetVarObj(
   PemnCommHandle  hComm,
   char            *pcVarName,
   char            *pcVarValue,
   int             iMsMaxTimeToWait
);
```

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnBSetVarObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcVarName	Name of the PATROL variable whose value is to be set.
pcVarValue	Value to which the PSL variable is to be set.
iMsMaxTimeToWait	The maximum number of milliseconds that the `PemnBSetVarObj()` function will wait for a response from the PATROL Agent.

Description

The `PemnBSetVarObj()` function requests that the PATROL Agent identified by the connection handle `hComm` set variable `pcVarName` to the value `pcVarValue`. The `PemnBSetVarObj()` function will wait `iMsMaxTimeToWait` milliseconds for a response from the PATROL Agent. The `PemnBSetVarObj()` returns the API message string indicating the status of the operation. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetApplList(), PemnBGetApplList()

Return a list of a computer's application classes from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetApplList(
   PemnCommHandle      hComm,
   PemnGetListHandler  pGetListHandler
);

char* PemnBGetApplList(
   PemnCommHandle      hComm,
   int                 iMsMaxTimeToWait
);
```

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetApplList()` or `PemnBGetApplList()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
[nb]pGetListHandler	The name of the handler that the `PemnGetApplList()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetApplList()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetApplList()` and `PemnBGetApplList()` functions request that the PEM engine return a list of the application classes from the computer system connected to handle `hComm`.

The `PemnGetApplList()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetListHandler`.

The `PemnBGetApplList()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetApplList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetApplList()` function call.

BMC Software, Inc., Confidential and Proprietary Information

- If the `PemnBGetApplList()` function wait time expires, the `PemnBGetApplList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetApplList()` function call.

The `PemnGetApplList()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetApplList()` function returns the requested application list if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

PemnGetApplObj(), PemnBGetApplObj()

Return an application object from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetApplObj(
   PemnCommHandle         hComm,
   char                   *pcApplTypeName,
   PemnGetApplObjHandler  pGetObjHandler
);

PemnApplObjHandle PemnBGetApplObj(
   PemnCommHandle  hComm,
   char            *pcApplTypeName,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetApplObj()` or `PemnBGetApplObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcApplTypeName	String that is the name of the application class to be returned.
[nb]pGetObjHandler	The name of the handler that the `PemnGetApplObj()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetApplObj()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetApplObj()` and `PemnBGetApplObj()` functions request that the PEM engine return the application class object identified by `pcApplTypeName`. Once the application object handle (`hObj`) is received, use the `PemnGetApplObjAttributes()` function to access the attributes of the application object.

The `PemnGetApplObj()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetObjHandler`.

The `PemnBGetApplObj()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetApplObj()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetApplObj()` function call.

- If the `PemnBGetApplObj()` function wait time expires, the `PemnBGetApplObj()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetApplObj()` function call.

The `PemnGetApplObj()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetApplObj()` function returns the requested application object handle if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetApplObjAttributes()

Return the attributes of an application.

Synopsis

```
PemnApplAttrArray *PemnGetApplObjAttributes(
   PemnApplObjHandle  hObj
);
```

Parameter

Parameter	Description
hObj	The handle for the application object for which the attributes should be returned.

Description

The `PemnGetApplObjAttributes()` function returns the attributes of the application class object identified by the object handle `hObj`. The returned attributes are in the form of a structure defined by the `PemnApplAttrEnum` type definition (see “PemnApplAttrEnum” on page 3-6 for more information). The application class attributes include the following:

- worst instance
- state
- master revision
- minor revision

Note

The returned information is read-only. *Do not* try to free it or write to it.

PemnGetApplObjInfo()

Return a PATROL application information string.

Synopsis

```
#include <pemapi.h>

char *PemnGetApplObjInfo(
   PemnApplObjHandle  hObj,
   char               *pcFormatString
);
```

Parameter

Parameter	Description
hObj	The handle for the application object for which an information string should be returned.
pcFormatString	String reserved for future use.

Description

The `PemnGetApplObjInfo()` function returns a character string that contains information about the PATROL application object identified by the handle `hObj`.

The `PemnGetApplObjInfo()` function return value is formatted as follows:

BMC Software, Inc., Confidential and Proprietary Information

appname`\t`*worstinstance*`\t`*state*`\t`*masterrev*`\t`*minorrev*

Note

The format of the returned string may change in future PATROL API releases.

Field	Description
appname	Full object name of the application.
worstinstance	Full object name of the instance of this application that has the worst object state.
state	The state of the application object. Valid range: OFFLINE OK WARN ALARM
masterrev	Major version number of the Knowledge Module for this application.
minorrev	Minor version number of the Knowledge Module for this application.

PemnGetComputerParamList(), PemnBGetComputerParamList()

Return a list of the computer parameter names from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetComputerParamList(
   PemnCommHandle      hComm,
   PemnGetListHandler  pGetListHandler,
);

char* PemnBGetComputerParamList(
   PemnCommHandle  hComm,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetComputerParamList()` or `PemnBGetComputerParamList()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
[nb]pGetListHandler	The name of the handler that the `PemnGetComputerParamList()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetComputerParamList()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetComputerParamList()` and `PemnBGetComputerParamList()` functions request that the PEM engine return a list of the computer parameters from the computer system connected to handle `hComm`.

The `PemnGetComputerParamList()` function does not wait for a response from the PEM engine. When a response arrives, the API calls `pGetListHandler`.

The `PemnBGetComputerParamList()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetComputerParamList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetComputerParamList()` function call.

- If the `PemnBGetComputerParamList()` function wait time expires, the `PemnBGetComputerParamList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetComputerParamList()` function call.

The `PemnGetComputerParamList()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetComputerParamList()` function returns the requested computer parameter list if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetComputerParamObj(), PemnBGetComputerParamObj()

Return a computer parameter object from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetComputerParamObj(
   PemnCommHandle          hComm,
   char                    *pcParamName,
   PemnGetParamObjHandler  pGetObjHandler,
);

PemnParamObjHandle PemnBGetComputerParamObj(
   PemnCommHandle  hComm,
   char            *pcParamName,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetComputerParamObj()` or `PemnBGetComputerParamObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcParamName	String that is the name of the computer parameter to be returned.
[nb]pGetObjHandler	The name of the handler that the `PemnGetComputerParamObj()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetComputerParamObj()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetComputerParamObj()` and `PemnBGetComputerParamObj()` functions request that the PEM engine return the computer parameter object identified by `pcParamName`. Once the computer parameter object handle (`hObj`) is received, use the `PemnGetParamObjAttributes()` function to access the attributes of the computer parameter object.

The `PemnGetComputerParamObj()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetObjHandler`.

BMC Software, Inc., Confidential and Proprietary Information

The `PemnBGetComputerParamObj()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetComputerParamObj()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetComputerParamObj()` function call.

- If the `PemnBGetComputerParamObj()` function wait time expires, the `PemnBGetComputerParamObj()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetComputerParamObj()` function call.

The `PemnGetComputerParamObj()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetComputerParamObj()` function returns the requested computer parameter object handle if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetInstList(), PemnBGetInstList()

Return a list of application instance names from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetInstList(
   PemnCommHandle      hComm,
   char                *pcApplTypeName,
   PemnGetListHandler  pGetListHandler,
);

char* PemnBGetInstList(
   PemnCommHandle  hComm,
   char            *pcApplTypeName,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetInstList()` or `PemnBGetInstList()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcApplTypeName	String that is the name of the application class from which the instance list is to be returned.
[nb]pGetListHandler	The name of the handler that the `PemnGetInstList()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetInstList()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetInstList()` and `PemnBGetInstList()` functions request that the PEM engine return a list of the application instances for the application class `pcApplTypeName` from the computer system connected to handle `hComm`.

The `PemnGetInstList()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetListHandler`.

The `PemnBGetInstList()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetInstList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetInstList()` function call.

- If the `PemnBGetInstList()` function wait time expires, the `PemnBGetInstList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetInstList()` function call.

The `PemnGetInstList()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetInstList()` function returns the requested instance list if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetInstObj(), PemnBGetInstObj()

Return an application instance object from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetInstObj(
   PemnCommHandle         hComm,
   char                   *pcApplTypeName,
   char                   *pcApplInstName,
   PemnGetInstObjHandler  pGetObjHandler
);

PemnInstObjHandle PemnBGetInstObj(
   PemnCommHandle  hComm,
   char            *pcApplTypeName,
   char            *pcApplInstName,
   int             iMsMaxTimeToWait
);
```

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetInstObj()` or `PemnBGetInstObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcApplTypeName	String that is the name of the application class from which the instance is to be returned.
pcApplInstName	String that is the name of the application instance to be returned.
[nb]pGetObjHandler	The name of the handler that the `PemnGetInstObj()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetInstObj()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetInstObj()` and `PemnBGetInstObj()` functions request that the PEM engine return the application instance object for application class `pcApplTypeName` and instance `pcApplInstName`. Once the application instance object handle (`hObj`) is received, use the `PemnGetInstObjAttributes()` function to access the attributes of the application instance object.

The `PemnGetInstObj()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetObjHandler`.

BMC Software, Inc., Confidential and Proprietary Information

The `PemnBGetInstObj()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetInstObj()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetInstObj()` function call.

- If the `PemnBGetInstObj()` function wait time expires, the `PemnBGetInstObj()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetInstObj()` function call.

The `PemnGetInstObj()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetInstObj()` function returns the requested instance object handle if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetInstObjAttributes()

Return the attributes of an application instance.

Synopsis

```
PemnInstAttrArray *PemnGetInstObjAttributes(
   PemnInstObjHandle  hObj
);
```

Parameter

Parameter	Description
hObj	The handle for the application instance object for which the attributes should be returned.

Description

The `PemnGetInstObjAttributes()` function returns the attributes of the application instance object identified by the object handle `hObj`. The returned attributes are in the form of a structure defined by the `PemnInstAttrEnum` type definition (see “PemnInstAttrEnum” on page 3-24 for more information). The application instance attributes include the following:

- worst parameter
- status
- rule state
- create icon
- parent name

Note

The returned information is read-only. *Do not* try to free it or write to it.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetInstObjInfo()

Return a PATROL application instance information string.

Synopsis

```
#include <pemapi.h>

char *PemnGetInstObjInfo(
   PemnInstObjHandle  hObj
   char               *pcFormatString
);
```

Parameter

Parameter	Description
hObj	The handle for the application object for which an information string should be returned.
pcFormatString	String reserved for future use.

Description

The `PemnGetInstObjInfo()` function returns a character string that contains information about the PATROL application instance object identified by the handle `hObj`.

The `PemnGetInstObjInfo()` function return value is formatted as follows:

instancename\t*worstparam*\t*status*\t*rulestate*\t*createicon*\t*parent*

Note

The format of the returned string may change in future PATROL API releases.

Field	Description
appname	Full object name of the application.
worstparam	Full object name of the parameter of this application that has the worst object state.
status	The state of the application instance object. Valid range: VOID OFFLINE OK WARN ALARM
rulestate	The state of the rule.
createicon	Flag that indicates whether an icon should be created and displayed for this instance. Valid range: 0 do not create an icon 1 create an icon
parent	Name of the object that is the parent of this instance. Valid range: instance name or NULL string.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetParamList(), PemnBGetParamList()

Return a list of the application instance parameter names from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetParamList(
   PemnCommHandle      hComm,
   char                *pcApplTypeName,
   char                *pcApplInstName,
   PemnGetListHandler  pGetListHandler,
);

char* PemnBGetParamList(
   PemnCommHandle  hComm,
   char            *pcApplTypeName,
   char            *pcApplInstName,
   int             iMsMaxTimeToWait
);
```

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetParamList()` or `PemnBGetParamList()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcApplTypeName	String that is the name of the application class from which the parameter list is to be returned.
pcApplInstName	String that is the application instance name from which the parameter list is to be returned.
[nb]pGetListHandler	The name of the handler that the `PemnGetParamList()` function calls when it receives a response from the PEM engine. This parameter is used in nonblocking mode only.
[b]iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetParamList()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement. This parameter is used in blocking mode only.

Description

The `PemnGetParamList()` and `PemnBGetParamList()` functions request that the PEM engine return a list of parameters for application class `pcApplTypeName` and instance `pcApplInstName` from the computer system connected to handle `hComm`.

The `PemnGetParamList()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetListHandler`.

The `PemnBGetParamList()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetParamList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

BMC Software, Inc., Confidential and Proprietary Information

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetParamList()` function call.

- If the `PemnBGetParamList()` function wait time expires, the `PemnBGetParamList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetParamList()` function call.

The `PemnGetParamList()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetParamList()` function returns the requested parameter list if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetParamObj(), PemnBGetParamObj()

Return a application instance parameter object from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetParamObj(
   PemnCommHandle          hComm,
   char                    *pcApplTypeName,
   char                    *pcApplInstName,
   char                    *pcParamName,
   PemnGetParamObjHandler  pGetObjHandler
);

PemnParamObjHandle PemnBGetParamObj(
   PemnCommHandle  hComm,
   char            *pcApplTypeName,
   char            *pcApplInstName,
   char            *pcParamName,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetParamObj()` or `PemnBGetParamObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcApplTypeName	String that is the name of the application class from which the parameter is to be returned.
pcApplInstName	String that is the application instance name from which the parameter is to be returned.
pcParamName	String that is the name of the application instance parameter to be returned.
pGetObjHandler	The name of the handler that the `PemnGetParamObj()` function calls when it receives a response from the PEM engine.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetParamObj()` function will wait for a reply from the PEM engine before posting an error condition and resuming program execution with the next statement.

Description

The `PemnGetParamObj()` and `PemnBGetParamObj()` functions request that the PEM engine return the application instance parameter object identified by application class `pcApplTypeName`, instance `pcApplInstName`, and parameter `pcParamName`. Once the parameter object handle (`hObj`) is received, use the `PemnGetParamObjAttributes()` function to access the attributes of the parameter object.

The `PemnGetParamObj()` function *does not* wait for a response from the PEM engine. When a response arrives, the API calls `pGetObjHandler`.

The `PemnBGetParamObj()` function causes the API to loop while waiting for a response from the PEM engine. If `iMsMaxTimeToWait` is specified, the `PemnBGetParamObj()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetParamObj()` function call.

- If the `PemnBGetParamObj()` function wait time expires, the `PemnBGetParamObj()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetParamObj()` function call.

The `PemnGetParamObj()` function returns 1 if the application class list request was successfully sent to the PEM engine and 0 if it was not. The `PemnBGetParamObj()` function returns the requested parameter object handle if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetParamObjAttributes()

Return the attributes of a parameter.

Synopsis

```
PemnParamAttrArray *PemnGetParamObjAttributes(
   PemnParamObjHandle  hObj
);
```

Parameter

Parameter	Description
hObj	The handle for the parameter object for which the attributes should be returned.

Description

The `PemnGetParamObjAttributes()` function returns the attributes of the parameter object identified by the object handle `hObj`. The returned attributes are in the form of a structure defined by the `PemnParamAttrEnum` type definition (see “PemnParamAttrEnum” on page 3-27 for more information). The parameter attributes include the following (see table in “PemnGetParamObjInfo()” on page 9-37):

- timestamp
- polling interval
- retries
- current value
- state
- output mode
- auto scale setting
- y-axis minimum
- y-axis maximum

PemnGetParamObjInfo()

Return a PATROL parameter information string.

Synopsis

```
#include <pemapi.h>

char *PemnGetParamObjInfo(
   PemnParamObjHandle  hObj
   char                *pcFormatString
);
```

Parameter

Parameter	Description
hObj	The handle for the parameter object for which an information string should be returned.
pcFormatString	String reserved for future use.

Description

The `PemnGetParamObjInfo()` function returns a character string that contains information about the PATROL parameter object identified by the handle `hObj`.

The `PemnGetInstObjInfo()` function return value is formatted as follows:

paramname`\t`*time*`\t`*pollintvl*`\t`*retries*`\t`*value*`\t`*state*`\t`*outmode*`\t`
autoscale`\t`*xaxis*`\t`*yaxis*

Field	Description
paraname	Full object name of the parameter.
time	24-character time stamp of the most current parameter value.
pollintvl	Number of seconds between successive polls of this parameter.
retries	Number of times the PATROL Agent will attempt to poll the parameter (unsuccessfully) before reporting an error.
value	Most recent value reported by the parameter.
state	Most recent state of the parameter. Valid range: OK, WARN, or ALARM.
outmode	Parameter output type. The return value is a number between 0 and 7 as follows: 0 NONE 1 TEXT 2 GAUGE 3 GRAPH 4 CONSOLE 5 STATEBOOLEAN 6 STOPLIGHT 7 CUSTOM
autoscale	State of the flag that allows the PATROL Console to automatically scale the parameter graph or gauge based on the parameter value.
xaxis	X axis value for the parameter.
yaxis	Y axis for the parameter.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetVarList(), PemnBGetVarList()

Return a list of a computer's variables from the PEM engine.

Synopsis

```
#include <pemapi.h>

int PemnGetVarList(
   PemnCommHandle       hComm,
   char                 *pcNode,
   char                 *pcKeyword,
   PemnGetListHandler   pGetListHandler,
);

char* PemnBGetVarList(
   PemnCommHandle  hComm,
   char            *pcNode,
   char            *pcKeyword,
   int             iMsMaxTimeToWait
);
```

BMC Software, Inc., Confidential and Proprietary Information

Parameters

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetVarList()` or `PemnBGetVarList()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcNode	String that is the object name whose variable list should be returned.
pcKeyword	Keyword that specifies the information that is to be returned for the object. Valid range: • `nodes` — return node list only • `subnodes` — return a node and subnode list • `leaves` — return a list of only leaves (this option is equivalent to the PATROL Version 3.0 `get_vars(`*object*`)` function) • `all` — return a list of all children (this option is equivalent to the PATROL Version 3.1 `get_vars(`*object*`,1)` function) **Note:** Currently this function cannot validate the value of the pcKeyword parameter. Entering an invalid pcKeyword may make this function fail without returning any sort of error message.
pGetListHandler	The name of the handler that the `PemnGetVarList()` function calls when it receives a response from the PEM.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetVarList()` function will wait for a reply from the PEM before posting an error condition and resuming program execution with the next statement.

BMC Software, Inc., Confidential and Proprietary Information

Description

The `PemnGetVarList()` and `PemnBGetVarList()` functions request that the PEM return a list of variables for the computer system identified by `pcNode`.

The `PemnGetVarList()` function *does not* wait for a response from the PEM. When a response arrives, the API calls `pGetListHandler`.

The `PemnBGetVarList()` function causes the API to loop while waiting for a response from the PEM. If `iMsMaxTimeToWait` is specified, the `PemnBGetVarList()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetVarList()` function call.
- If the `PemnBGetVarList()` function wait time expires, the `PemnBGetVarList()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetVarList()` function call.

The `PemnGetVarList()` function returns 1 if the application class list request was successfully sent to the PEM and 0 if it was not. The `PemnBGetVarList()` function returns the requested variable list if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnGetVarObj(), PemnBGetVarObj()

Return a variable object from the PEM.

Synopsis

```
#include <pemapi.h>

int PemnGetVarObj(
   PemnCommHandle        hComm,
   char                  *pcVarName,
   PemnGetVarObjHandler  pGetObjHandler,
);

PemnVarObjHandle PemnBGetVarObj(
  PemnCommHandle  hComm,
  char            *pcVarName,
  int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnGetVarObj()` or `PemnBGetVarObj()` function is issued. The `PemnOpen()` function returns the communication handle when the connection successfully opens.
pcVarName	String that is the name of the variable to be returned.
pGetObjHandler	The name of the handler that the `PemnGetVarObj()` function calls when it receives a response from the PEM.
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBGetVarList()` function will wait for a reply from the PEM before posting an error condition and resuming program execution with the next statement.

Description

The `PemnGetVarObj()` and `PemnBGetVarObj()` functions request that the PEM return the variable object identified by `pcVarName`.

The `PemnGetVarObj()` function *does not* wait for a response from the PEM . When a response arrives, the API calls `pGetObjHandler`.

The `PemnBGetVarObj()` function causes the API to loop while waiting for a response from the PEM. If `iMsMaxTimeToWait` is specified, the `PemnBGetVarObj()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBGetVarObj()` function call.

BMC Software, Inc., Confidential and Proprietary Information

- If the `PemnBGetVarObj()` function wait time expires, the `PemnBGetVarObj()` function returns an error message and the API resumes execution with the statement that follows the `PemnBGetVarObj()` function call.

The `PemnGetVarObj()` function returns 1 if the application class list request was successfully sent to the PEM and 0 if it was not. The `PemnBGetVarObj()` function returns the requested variable object handle if successful and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Data Management Functions

This chapter describes the data management function calls that are used to send a stream of bytes to a PATROL Agent.

PemnPutFile(), PemnBPutFile() 10-2
PemnPutStream(), PemnBPutStream() 10-5

BMC Software, Inc., Confidential and Proprietary Information

PemnPutFile(), PemnBPutFile()

Send a file to a remote PATROL Agent.

Synopsis

```
#include <pemapi.h>

int PemnPutFile(
   PemnCommHandle  hComm,
   char            *pcSrcFullFileName,
   char            *pcDstFileName,
   char            *pcCatalogName,
   char            *pcClassName,
   char            *pcId
);

char* PemnBPutFile(
   PemnCommHandle  hComm,
   char            *pcSrcFullFileName,
   char            *pcDstFileName,
   char            *pcCatalogName,
   char            *pcClassName,
   char            *pcId,
   int             iMsMaxTimeToWait
);
```

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnPutFile()` or `PemnBPutFile()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
pcSrcFullFileName	The name and path of the file that you want to copy to the remote PATROL Agent.

Parameter	Description
pcDstFileName	The name that you want this file to have on the remote PATROL Agent. **Note:** You cannot specify a directory path with this parameter.
pcCatalogName	Optional event catalog name for the event generated by this put file operation. Valid range: PATROL event catalog name or `""`.
pcClassName	Optional event class name for the event generated by this put file operation. Valid range: PATROL event class name or `""`.
pcId	A character string used to identify the `pcDstFileName` parameter. Valid range: any character string or `""`.
iMsMaxTimeToWait	An Integer number of milliseconds that the `PemnBPutFile()` function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

You can use the `PemnPutFile()` or `PemnBPutFile()` function to copy `pcSrcFullFileName` from the host on which the function executes to `pcDstFileName` on the PATROL Agent host identified by `hComm`. If the *PATROL_REMOTE* environment variable is defined for the remote host, the file is written to ***$PATROL_REMOTE*** (or ***%PATROL_REMOTE%*** on Windows). If the *PATROL_REMOTE* is not defined, the file is written to ***$PATROL_HOME*/remote/** on the remote PATROL Agent.

`PemnPutFile()` and `PemnBPutFile()` send an event to the remote PATROL Agent using the `pcCatalogName` and `pcClassName` parameters. Using the following rules, you can choose which event to send or allow the function to send the standard event, RemProcess. If `pcCatalogName` and `pcClassName` are NULL (`""`), no event is

BMC Software, Inc., Confidential and Proprietary Information

generated. If `pcCatalogName` is NULL, an event is generated with the standard event catalog and `pcClassName`. If the `pcClassName` is NULL, the event uses `pcCatalaogName` and the RemProcess class. If neither parameter is NULL, the event is determined by `pcCatalogName` and `pcClassName`.

Regardless of whether you define your own event or use RemProcess, the event must carry three dynamic arguments: **FileName**, **Operation** (reserved for future use), and **FileId**.

The `PemnPutFile()` function does not wait for a response from the PATROL Agent. It returns 1 if the file was successfully sent to the PATROL Agent and 0 if it was not.

The `PemnBPutFile()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBPutFile()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBPutFile()` function call.

- If the `PemnBPutFile()` function wait time expires, the `PemnBPutFile()` function returns an error message and the API resumes execution with the statement that follows the `PemnBPutFile()` function call.

The `PemnBPutFile()` function returns the string `PEMN_OK` if the file was successfully sent to the PATROL Agent and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnPutStream(), PemnBPutStream()

Send a character stream to a PATROL Agent.

Synopsis

```
#include <pemapi.h>

int PemnPutStream(
   PemnCommHandle  hComm,
   int             iStreamSize,
   char            *pcStreamData,
   char            *pcCatalogName,
   char            *pcClassName,
   int             iTag,       /* reserved for future use*/
   char            *pcFileName,
   char            *pcOperation,
   char            *pcStreamId,
   char            *pcAccess
);

char* PemnBPutStream(
   PemnCommHandle  hComm,
   int             iStreamSize,
   char            *pcStreamData,
   char            *pcCatalogName,
   char            *pcClassName,
   int             iTag,       /* reserved for future use*/
   char            *pcFileName,
   char            *pcOperation,
   char            *pcStreamId,
   char            *pcAccess
   int             iMsMaxTimeToWait
   );
```

BMC Software, Inc., Confidential and Proprietary Information

Parameter

Parameter	Description
hComm	The handle of the communication connection to which the `PemnPutStream()` or `PemnBPutStream()` function is issued. The `PemnOpen()` or `PemnBOpen()` function returns the communication handle when the connection successfully opens.
iStreamSize	Integer number of characters that will be sent to the PATROL Agent in the character stream. Valid range: 0 – 8096
pcStreamData	Character string containing the character stream.
pcCatalogName	Optional event catalog name for the event generated by the character stream operation. Valid range: PATROL event catalog name or `""`. If `pcCatalogName = pcClassName = ""` then no event is generated. If `pcCatalogName = ""` or `pcClassName = ""`, an event with the standard catalog and `RemProcess` class is generated.
pcClassName	Optional event class name for the event generated by the character stream operation. Valid range: PATROL event class name or `""`. If `pcCatalogName = pcClassName = ""` then no event is generated. If `pcCatalogName = ""` or `pcClassName = ""`, an event with the standard catalog and `RemProcess` class is generated.
iTag	Integer reserved for future use.
pcFileName	Character file name in which the PATROL Agent will buffer the character stream it receives.
pcOperation	String indicating the type of stream operation that the `PemnPutStream()` or `PemnBPutStream()` function should perform. Valid range: • PEMN_WRITE — PATROL Agent will write character stream to `pcFileName` • PEMN_APPEND — PATROL Agent will append character stream to `pcFileName` **Note:** Entering invalid data for this parameter, most often, causes the character stream to be appended to `pcFileName`.
pcStreamId	Character identifier for the character stream.

BMC Software, Inc., Confidential and Proprietary Information

Parameter	Description
pcAccess	Character string that assigns the file permissions of `pcFileName`. For this parameter, you would enter the decimal equivalent of any octal value that can be used with the chmod command. You could, for example, enter "511" which is equal to the octal value 777 (rwxrwxrwx). Specifying the NULL string for this parameter causes the file to be created with the permission 644 (rw-r--r--). Specifying an invalid string causes the file to be created without any permission setting (---------).
iMsMaxTimeToWait	Integer number of milliseconds that the `PemnBPutStream()` function will wait for a response from the PATROL Agent before posting an error condition and resuming program execution with the next statement.

Description

The `PemnPutStream()` and `PemnBPutStream()` functions send a character stream `pcStreamData` of size `iStreamSize` to the PATROL Agent identified by `hComm` with the instruction `pcOperation` that the PATROL Agent either write or append the character stream to the file `pcFileName`. The `PemnPutStream()` and `PemnBPutStream()` function call can generate an optional event (either PATROL- or user-defined) using event catalog `pcCatalogName` and `pcClassName`.

The `PemnPutStream()` function does not wait for a response from the PATROL Agent.

The `PemnBPutStream()` function causes the API to loop while waiting for a response from the PATROL Agent. If `iMsMaxTimeToWait` is specified, the `PemnBPutStream()` function will wait up to `iMsMaxTimeToWait` milliseconds:

- If a response arrives, the API resumes execution at the statement that follows the `PemnBPutStream()` function call

BMC Software, Inc., Confidential and Proprietary Information

- If the `PemnBPutStream()` function wait time expires, the `PemnBPutStream()` function returns an error message and the API resumes execution with the statement that follows the `PemnBPutStream()` function call.

The `PemnPutStream()` function returns 1 if the character stream was successfully sent to the PATROL Agent and 0 if it was not. The `PemnBPutStream()` function returns the string `PEMN_OK` if the character stream was successfully sent to the PATROL Agent and an API message string if it was not. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Miscellaneous Functions

This chapter describes the `Pemn` miscellaneous function calls.

PemnCheckInit() 11-2
PemnGetLastMessage() 11-3
PemnRegisterUserMessageHandler() 11-4

BMC Software, Inc., Confidential and Proprietary Information

PemnCheckInit()

Verify initialization, or initialize the PEM library.

Synopsis

```
include <pemapi.h>

int PemnCheckInit(void);
```

Description

The `PemnCheckInit()` function verifies that the PEM library is initialized and initializes it if it is not. The `PemnCheckInit()` function returns 1 if it found the PEM library was initialized, and 0 if it found that it was not initialized.

PemnGetLastMessage()

Return the last message issued by the API.

Synopsis

```
char * PemnGetLastMessage(void)
```

Description

The `PemnGetLastMessage()` function returns the last error, warning, or information message string issued by the API.

BMC Software, Inc., Confidential and Proprietary Information

PemnRegisterUserMessageHandler()

Register a message processing function with the API.

Synopsis

```
void PemnRegisterUserMessageHandler(
   PemnUserMessageHandler  pMessageHandler
);
```

Parameter

Parameter	Description
pMessageHandler	Character string name of the message handling routine.

Description

The `PemnRegisterUserMessageHandler()` function registers a user-created routine with the API that the API will call to process information, warning, and error messages. The message handler uses the type definition `PemnUserMessageHandler` (see "PemnUserMessageHandler" on page 3-34 for more information.)

Specific Blocking Mode Functions

This chapter describes the blocking mode functions for specific purposes such as exiting a function call and retrieving the current blocking function default waiting time.

PemnBExitFunction() 12-2
PemnBGetTimeout() 12-3
PemnBSetTimeout() 12-4

BMC Software, Inc., Confidential and Proprietary Information

PemnBExitFunction()

Exit a function that is blocked awaiting a response.

Synopsis

```
#include <pemapi.h>

void PemnBExitFunction(void);
```

Description

The `PemnBExitFunction()` function can be called from the program signal handler (for example, `SIGINT`) to abort a blocking mode operation. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnBGetTimeout()

Return the default waiting time in milliseconds for blocking operations.

Synopsis

```
#include <pemapi.h>

int PemnBGetTimeout(void);
```

Description

The `PemnBGetTimeout()` function returns the integer value in milliseconds that is currently in use as the default waiting time for blocking mode operations. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

BMC Software, Inc., Confidential and Proprietary Information

PemnBSetTimeout()

Set the default waiting time in milliseconds for blocking operations.

Synopsis

```
#include <pemapi.h>

void PemnBSetTimeout(int iNewDefaultTimeout);
```

Parameter

Parameter	Description
iNewDefaultTimeout	Number of milliseconds that blocking mode functions will wait for a response from the PATROL Agent. The minimum permitted value is 100. Default: If no `iNewDefaultTimeout` has been specified, the blocking functions use 60000 milliseconds, or 1 minute.

Description

The `PemnBSetTimeout()` function sets the default timeout value used by the blocking functions to wait for a response from the PATROL Agent. For a list of return values for blocking functions, see Table 1-1 on page 1-7.

Example

The following example shows how the default wait time is used in a function that executes a PSL statement in the PATROL Agent:

```
PemnBPslExec(hComm,USE_DEFAULT_TIMEOUT,0,"GET_VARS(\"/\");")
```

PATROL COM/DCOM Interface

This chapter provides information about accessing the PATROL Agent through the Component Object Model/Distributed Component Object Model (COM/DCOM) interface.

The following topics are discussed:

Overview 13-3
COM/DCOM Files 13-4
Configuring the PATROL COM/DCOM Interface 13-4
    Configuring COM Servers 13-4
    Configuring DCOM Clients 13-6
Accessing the PATROL Agent COM Server 13-6
    Through the Dispatch Interface 13-7
    Through a Custom Interface Program 13-8
IPatrolAgent Interface Functions 13-9
    annotate() 13-11
    annotate_get() 13-13
    blackout() 13-15
    change_state() 13-17
    create() 13-19
    destroy() 13-21
    exists() 13-23
    get() 13-25
    getenv() 13-27
    get_psl_errno() 13-29
    get_ranges() 13-30
    get_vars() 13-32
    history() 13-34

BMC Software, Inc., Confidential and Proprietary Information

is_var() 13-36
PslExecute() 13-38
print() 13-40
set() 13-41
set_alarm_ranges() 13-43
unset() 13-45

Overview

This chapter describes the Component Object Model/Distributed Component Object Model (COM/DCOM) programming interface. The COM/DCOM interface, which is available for Windows NT only, provides developers of COM-compatible applications access to data collected by one or more PATROL Agents.

The PATROL Agent for Windows NT is a local COM server. The agent provides COM\DCOM functionality through the IPatrolAgent, an interface that allows you to access and use PATROL data through COM-compatible applications.

For additional information about programming using the COM/DCOM protocol, refer to the Microsoft World Wide Web site at **http://www.microsoft.com**.

An application can access the IPatrolAgent Interface by

- agent's dispatch interface using a VisualBasic (VB) script or another script that supports OLE automation
- custom interface using a C++/C program.

This chapter describes

- files provided to support COM functionality
- accessing the IPatrolAgent interface using either a dispatch interface or a custom interface
- functions that comprise the COM API

Note

Because most of the functions in the COM/DCOM API are based on PSL functions, you should use this chapter along with the *PATROL Script Language Reference Manual*.

BMC Software, Inc., Confidential and Proprietary Information

COM/DCOM Files

PATROL provides the COM/DCOM files listed in Table 13-1.

Table 13-1 PATROL COM/DCOM Files

File name	Description
ipa.h	This file defines the IPatrolAgent interface. It is installed in the $PATROL_HOME/com directory.
ipa_i.c	This file defines the UUID for the agent COM server. It is installed in the $PATROL_HOME/com directory.
ipa.tlb	This file is the type library. If you use this file, you must place it in the $PATROL_HOME/bin directory on the server machine. In addition, on client machines, you must set the location of the file in the Windows NT registry using the PATROL Client Configuration utility (PCCFG.EXE).

Configuring the PATROL COM/DCOM Interface

Before you can use your application (DCOM client) to access data on one or more PATROL Agents (COM servers), you must configure the machines hosting PATROL Agents (COM servers) and the machines hosting your application (DCOM clients).

Configuring COM Servers

You configure the machines hosting PATROL Agents (COM servers) either by setting the PATROL Agent COM configuration variables and/or by running the Microsoft DCOM configuration utility (DCOMCFG.EXE).

PATROL Agent COM Configuration Variables

The two PATROL Agent configuration variables described in Table 13-2, “PATROL Agent COM Variables,” on page 13-5 apply to the COM server:

Table 13-2 PATROL Agent COM Variables

Variable	Description
comUserAllowed	Controls who can access the PATROL Agent COM server. Listed users, members of groups, and the user who started the agent may access the agent COM server. The following values are valid: • Empty String (default) • Group Names • User Names Each entry must be separated by the pipe character (|) and no space because a space is a valid character for group and user names. Example: `Patrol|Domain Users|patrol-adm` This variable, supported only on Windows NT 4.0, takes precedence over any security set by the Microsoft DCOM Configuration Utility. If you set this variable to the empty string, security is set by the DCOM Configuration Utility. NOTE: If you set this variable, it is not necessary to run the DCOM configuration utility.
comSecurityLevel	Controls the level of information provided by the PATROL Agent to the DCOM client. The following values are valid: • N - No DCOM interface allowed (default) • R - DCOM client can receive data from the client • W - DCOM client can receive and change data on the client • F - DCOM client can receive and change data and use PslExecute() to run OS commands. NOTE: You must this variable if you want to use the COM interface.

For detailed information about these variables and procedures for setting them, refer to the *PATROL Agent Reference Manual*.

BMC Software, Inc., Confidential and Proprietary Information

Microsoft DCOM Configuration Utility

You run the Microsoft DCOM Utility on each PATROL Agent with which your DCOM application needs to communicate but only if you do not set the comUserAllowed variable. The comUserAllowed variable takes precedence over configuration set by the Microsoft DCOM Utility.

For information on obtaining the Microsoft DCOM Utility and instructions for using it to configure PATROL Agents, refer to the *PATROL Agent Reference Manual*.

Configuring DCOM Clients

You configure the machines hosting your application (DCOM clients) to communicate with the PATROL Agent by running the PATROL Client Configuration Utility (PCCFG.EXE). You do not need to do so, however, if the PATROL Agent is installed on the same machine that is running your DCOM application.

For instructions on configuring your machine using the PATROL Client Configuration Utility, refer to the *PATROL Agent Reference Manual*

Accessing the PATROL Agent COM Server

You access the PATROL Agent COM server through the IPatrolAgent interface. Definition of the IPatrolAgent interface (in the ipa.h file) includes 18 functions that are based on PSL functions:

- `annotate()`
- `annotate_get()`
- `blackout()`
- `change_state()`
- `create()`
- `destroy()`
- `exists()`
- `get()`
- `getenv()`

- `get_ranges()`
- `get_vars()`
- `history()`
- `is_var()`
- `print()`
- `PslExceute()`
- `set()`
- `set_alarm_ranges()`
- `unset()`

In addition to the functions based on PSL functions, the IPatrolAgent interface also includes the following function that allows you to get the error number of the PSL process:

- `get_psl_errno()`

How you invoke the IPatrolAgent interface functions and the format of their return values depends on whether you access the PATROL Agent COM server with a dispatch interface, such as a VB script, or with custom interface, such as a C++/C program. For detailed information about the IPatrolAgent interface functions see “IPatrolAgent Interface Functions” on page 13-9.

Through the Dispatch Interface

A client can access the PATROL Agent COM server through the dispatch interface using any script that supports OLE automation, such as a VB script. The dispatch interface is the same interface through which a PSL script accesses the PATROL Agent using PSL functions. Therefore, your script must invoke the IPatrolAgent interface functions in the same manner as a PSL script invokes PSL functions. When you develop your script, observe the following guidelines:

- For the IPatrolAgent interface functions that are based on PSL functions, use strings for the parameters. (These functions will also take strings as their return values.)

- Do not include a parameter with the [out, retval] attribute; this is the return value this is the return value used by a custom interface.

BMC Software, Inc., Confidential and Proprietary Information

- You do not need to include a parameter with an [optional] attribute.

The result is in the same format as returned from PSL function calls. See the *PATROL Script Language Reference Manual* for detailed descriptions of the functionality, parameters, return values, and return value formats of the PSL functions on which the IPatrolAgent interface functions are based.

Through a Custom Interface Program

A client can also access the PATROL Agent COM server through a custom interface, such as a C++/C program, that invokes the PSL-based interface functions. A custom interface must invoke the IPatrolAgent interface functions as they are defined. Therefore, when you develop your program, observe the following guidelines:

- To conform with PSL function prototype, each the PSL-based IPatrolAgent interface functions must take Unicode BSTR strings as its parameters.

- If a parameter has the [optional] attribute, you cannot omit it from the string, but you can pass NULL to the parameter.

- If a parameter is required, you cannot pass NULL to it; if you do, the function will return E_INVALIDARG.

IPatrolAgent interface functions will always return HRESULT, which indicates whether the call to the PSL function was successful. HRESULT does not reflect the result of the PSL function. HRESULT will return one of the following values:

- S_OK—the IPatrolAgent interface function call to the PSL function was successful. (You must check the return value in the parameter pbstrResult for the result of the PSL function's execution.)

- E_POINTER—the IPatrolAgent interface function encountered the NULL pointer and cannot return a value from the PSL function to pbstrResult.

- E_INVALIDARG—one or more of the required parameters is NULL or invalid.

- E_ACCESSDENIED—the COM security level set in comSecurityLevel parameter of the agent configuration database does not permit execution of the requested the action.

The last parameter of the IPatrolAgent interface is a Unicode string defined as BSTR * pbstrResult. This parameter, which has the [out, retval] attribute, usually holds the return value from the PSL function.

See the *PATROL Script Language Reference Manual* for detailed descriptions of the functionality, parameters, return values, and return value formats of the PSL functions on which the IPatrolAgent interface functions are based.

IPatrolAgent Interface Functions

This section provides the following information about each of the IPatrolAgent interface functions:

- a brief description of the function

- an example of how to invoke the function using a custom interface (a C++/C program)

- a table listing parameters of the functions using the dispatch interface and custom interface names, the attributes defined for each parameter, and brief description of the parameter

- a table listing the codes returned by HRESULT in a custom interface

For details about invoking a function using a dispatch interface (VB script), detailed information about the parameters for each PSL-based function, and details about the return values of PSL functions, consult the *PATROL Script Language Reference Manual*.

BMC Software, Inc., Confidential and Proprietary Information

Examples of Invoking IPatrolAgent Interface Functions

To illustrate the differences between invoking an IPatrolAgent interface function from a dispatch interface and a custom interface, this section shows how to invoke the `history()` function from both types of interface.

Dispatch Interface

You invoke the `history()` function through the dispatch interface as you would in a PSL script:

`Result=object.history(`“*parameter*”,“*format*”,“*number*”`);`

Because they are optional, you can omit *format* and *number.*

Custom Interface

You invoke the `history()` function through a custom interface as follows:

```
HRESULT history(BSTR bstrItem,
                BSTR bstrFormat,
                BSTR bstrCount,
                BSTR* pbstrResult);
```

In a custom interface, you cannot omit the optional parameters, but you can pass NULL to them.

BMC Software, Inc., Confidential and Proprietary Information

annotate()

This function provides contextual information for a PATROL parameter.

Synopsis

```
HRESULT annotate(BSTR bstrName,
                 BSTR bstrFormat,
                 VARIANT* pvarData,
                 BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `annotate()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	parameter	[in]	The name of the parameter for which annotation information is specified.
bstrFormat	format	[in, optional]	A mnemonic indicating the type of display for the specified annotation information.
pvarData	datan	[in, optional]	A safe array of VARIANT. The type of each VARIANT element is the Unicode BSTR string. See AppendixD, "Example of annotate() Function," for an example of how to construct this argument for a custom interface.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain NULL if `annotate()` is successful and an error message if it is not.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter pvarData is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

annotate_get()

This function returns the annotation for a parameter value.

Synopsis

```
HRESULT annotate_get(BSTR bstrName,
                     BSTR bstrTimestamp,
                     BSTR* pbstrValue);
```

Parameters

The following table describes the parameters that `annotate_get()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	parameter	[in]	The name of the parameter for which annotation information is specified.
bstrTimestamp	timestamp	[in, optional]	The BSTR string that identifies the timestamp of the parameter value for which annotation information should be returned.
pbstrValue	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain the annotation for the parameter value at timestamp.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

blackout()

This function hides PATROL object state changes for a specific period of time.

Synopsis

```
HRESULT blackout(BSTR bstrInst,
                 BSTR bstrPeriod,
                 BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `blackout()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrInst	object	[in, optional]	The name of the PATROL object whose state changes are to be hidden.
bstrPeriod	period	[in, optional]	The length of time the PATROL object is to remain blacked out.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The way you specify the bstrInst and bstrPeriod determines the result: • The parameter pbstrResult will contain, separated by new line characters, a list of all PATROL objects that are currently blacked out if bstrInst and bstrPeriod are NULL. It will contain the blackout status for the bstrInst if bstrInst is not NULL, and bstrPerid is NULL. • The parameter pbstrResult will contain OK if either bstrInst and bstrPeriod is not NULL.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

BMC Software, Inc., Confidential and Proprietary Information

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

change_state()

This function changes the rule state of a PATROL application instance.

Synopsis

```
HRESULT change_state(BSTR bstrName,
                     BSTR bstrState,
                     BSTR bstrDescr,
                     BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `change_state()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
BstrName	object	[in]	The name of the application instance whose state is to be changed.
bstrState	state	[in]	The operational state to which bstrName will be set.
bstrDescr	description	[in, optional]	An optional text string that can be used to explain the state change action.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain either the new state or NULL if the function returns an error.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter bstrName or bstrState is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

create()

This function creates a PATROL object.

Synopsis

```
HRESULT create(BSTR bstrName,
               BSTR bstrLabel,
               BSTR bstrState
               BSTR bstrDescr,
               BSTR bstrParent
               BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `create()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Return Values**
bstrName	object	[in]	The alphanumeric identifier for the object to be created.
bstrLabel		[in, optional]	An optional text string that can be used to identify the object. *Label* appears below the object's icon.
bstrState		[in, optional]	The optional state of the object that is assigned at its creation.
bstrDescr		[in, optional]	An optional text string that can be used to introduce the newly created object.
bstrParent		[in, optional]	An optional text string that specifies the parent to which the created object will belong.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain the bstrName if it creates the instance or NULL if an error occurs.

*Attributes are included in the definition of the IPatrolAgent interface.

BMC Software, Inc., Confidential and Proprietary Information

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

destroy()

This function destroys a PATROL object.

Synopsis

```
HRESULT destroy(BSTR bstrName,
                BSTR bstrDescr,
                BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `destroy()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Return Values**
bstrName	object	[in]	The alphanumeric identifier for the object to be destroyed.
bstrDescr	description	[in, optional]	An optional text string that can be used to explain why the object was destroyed.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain TRUE if `destroy()` is successful and FALSE if the function returns an error.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

exists()

This function verifies the existence of a PSL object.

Synopsis

```
HRESULT exists(BSTR bstrName,
               BSTR bstrInherit,
               BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `exists()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	object	[in]	The alphanumeric identifier for the object whose existence is being verified.
bstrInherit	inherit	[in, optional]	The string to control whether it will search inheritance in the agent object hierarchy to verify the existence of the object. For TRUE do not search the inheritance hierarchy; for FALSE search the inheritance hierarchy.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter contains TRUE if the object exists, and FALSE if it does not.
*Attributes are included in the definition of the IPatrolAgent interface.			

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

get()

This function returns the current value of a variable.

Synopsis

```
HRESULT get(BSTR bstrVariable,
            BSTR *pbstrResult);
```

Parameters

The following table describes the parameters that `get()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrVariable	variable	[in]	The name of the variable whose current value will be returned.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain the value for bstrVariable.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrValue is NULL pointer.

Return Value	Description
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

getenv()

This function returns the string value of a PSL environment variable.

Synopsis

```
HRESULT getenv(BSTR bstrName,
               BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `getenv()` contains.

Parameter			
Custom Interface	**PSL Function**	**Attributes***	**Description**
bstrName	variable	[in]	The name of object whose value is to be returned.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain the string value of the variable in the environment of the PSL script.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.

Return Value	Description
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

get_psl_errno()

This function returns the error number of the PSL process. This is a help function; there is no PSL function for it.

Synopsis

```
HRESULT get_psl_errno(long* plRet);
```

Parameters

The following table describes the parameters that `get_psl_errno()` contains.

Parameter	Attribute*	Description
plRet	[out, retval]	The error number for the PSL process.
*Attributes are defined in the file **ipa.c**		

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter plRet is a NULL pointer.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

get_ranges()

This function returns the Border, Alarm1, and Alarm2 range settings for a PATROL parameter.

Synopsis

```
HRESULT get_ranges(BSTR bstrName,
                   BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `get_ranges()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	param	[in, optional]	The name of the PATROL parameter whose Border, Alarm1, and Alarm2 range settings are to be returned.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain the current setting for Border, Alarm1, or Alarm2 range settings for a PATROL parameter.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

get_vars()

This function returns the list of variables for a PSL object.

Synopsis

```
HRESULT get_vars(BSTR bstrObj,
                 BSTR bstrKeyword ,
                 BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `get_vars()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrObj	Object	[in, optional]	The optional name of the object whose variables are to be listed. The default is the current object.
bstrKeyword	Keyword	[in, optional]	An optional keyword that specifies the information that is to be returned for bstrObj.
pbstrResult	NA	[out,retval]	The result of the call to the PSL function. The parameter will contain the list of the variables of bstrObj or the list of variables for the current object if bstrObj is omitted. The parameter will contain NULL if bstrObj does not exist.
*Attributes are included in the definition of the IPatrolAgent interface.			

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

history()

This function returns history information from the PATROL history database.

Synopsis

```
HRESULT history(BSTR bstrItem,
                BSTR bstrFormat,
                BSTR bstrCount,
                BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `history()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrItem	parameter	[in, optional]	The name of the object whose history is to be returned.
bstrFormat	format	[in, optional]	An optional string to specify the format of each history() function entry.
bstrCount	number	[in, optional]	An optional string to limit the number of entries the history function will return. The default is 50.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain a list of data points followed by a number of entries separated by new line characters.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult always contains a NULL pointer.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

is_var()

This function verifies that a PSL object variable exists.

Synopsis

```
HRESULT is_var(BSTR bstrName,
               BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `is_var()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	object	[in]	The name of the object that is to be verified as a variable.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain TRUE if the object exists and is a variable, and FALSE if the object does not exist or is not a variable.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.

Return Value	Description
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

PslExecute()

This function starts a PSL process. PslExecute() will return its result before the script finishes running.

Synopsis

```
HRESULT PslExceute(BSTR bstrName,
                   BSTR bstrScript,
                   BSTR *pbstrErrlist,
                   BSTR *pbstrWarnlist,
                   BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `PslExecute()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrName	name	[in]	The BSTR string that provides a descriptive identifier for the new PSL process.
bstrScript	PslScript	[in]	The string that contains the PSL commands to be executed.
pbstrErrlist	errorlist	[out, optional]	An optional variable to which the PSL interpreter or compiler will return error messages created while compiling the PSL script.
pbstrWarnlist	warninglist	[out, optional]	An optional variable to which the PSL interpreter/compiler will return warning messages generated while compiling the PSL script.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain 0 if the script has been successfully created and is executing, 1 for compilation errors, and 2 for a process creation error.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	One of the required parameters bstrName and bstrScript is not provided.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

BMC Software, Inc., Confidential and Proprietary Information

print()

This function prints text to a computer output window on the PATROL console.

Synopsis

```
HRESULT print(BSTR bstrMsg);
```

Parameters

The following table describes the parameters that `print()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrMsg	text1...textn	[in]	The text to be printed.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

set()

This function assigns a value to a variable.

Synopsis

```
HRESULT set(BSTR bstrObj,
            BSTR pbstrValue);
```

Parameters

The following table describes the parameters that `set()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
bstrObj	variable	[in]	The name of a variable in the PATROL Agent object hierarchy to which value is assigned.
pbstrResult	value	[in]	The result of the call to the PSL function. The parameter will contain the value that is assigned to pbstrValue.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain NULL if `set()` is successful or it will contain an error message if not.

*Attributes are included in the definition of the IPatrolAgent interface.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_INVALIDARG	One of the parameters is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

set_alarm_ranges()

This function sets the ranges for the alarm conditions for a particular application or PATROL parameter.

Synopsis

```
HRESULT set_alarm_ranges(BSTR bstrAlarmRangeValue,
                         BSTR
bstrGlobalParamClassName,
                         BSTR bstrApplName,
                         BSTR bstrParamOid,
                         BSTR bstrParamFullPath,
                         BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `set_alarm_ranges()` contains.

Parameter			
Custom Interface	**Dispatch Interface**	**Attributes***	**Description**
BstrAlarmRangeValue	new_ranges	[in]	A list of alarm range settings separated by new line characters.
bstrGlobalParamClassName	param	[in]	The name of the PATROL parameter to be monitored.
bstrApplName	appl	[in, optional]	An optional string for the name of the application.
bstrParamOid	param_oid	[in, optional]	An optional string for the PATROL parameter object ID.
bstrParamFullPath	path	[in, optional]	An optional string for the name of the path to the PATROL parameter.
pbstrResult	NA	[out, retval]	The result of the call to the PSL function. The parameter will contain NULL if the function is successful or will contain an error message if the function is unsuccessful.

*Attributes are included in the definition of the IPatrolAgent interface.

BMC Software, Inc., Confidential and Proprietary Information

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.
E_INVALIDARG	One of the required parameters, bstrAlarmRangeValue or bstrGlobalParamClassName, is not provided.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

unset()

This function removes a PATROL object that was previously assigned by the `set()` function.

Synopsis

```
HRESULT unset(BSTR bstrName,
              BSTR* pbstrResult);
```

Parameters

The following table describes the parameters that `unset()` contains.

Parameter	Description
bstrName	[in] the name of PATROL object that is to be removed by unset() function.
pbstrResult	[out, retval] The result of the call to the PSL function. It will always contain NULL.

Refer to the *PATROL Script Language Reference Manual* for details about the values contained in pbstrResult.

Return Values

HRESULT for this function returns the following codes:

Return Value	Description
S_OK	The call to the PSL function was successful. Check the parameter pbstrResult for the return value from the PSL function.
E_POINTER	The parameter pbstrResult is a NULL pointer.

BMC Software, Inc., Confidential and Proprietary Information

Return Value	Description
E_INVALIDARG	The parameter bstrName is invalid.
E_ACCESSDENIED	• The user is not permitted to access the COM server. Check the comUserAllowed setting in the agent configuration file or use the DCOMCFG.EXE utility to determine whether the user has access. • The security level does not permit the requested action. Check the comSecurityLevel setting in the agent configuration file.

Using Your Own Event Loop

This appendix contains instructions on using your own event loop. For most applications, the event loop provided by the PATROL API is simple and sufficient. However, if you are trying to use the PATROL API from a program with an existing event loop (X, WIN32 or custom event loop) you may wish to use your own.

The following topics are discussed:

How Your Program Waits for Activities. A-1
The Library for the Default Event Loop. A-2
Using the PATROL API from an X Application A-2
Using the PATROL API with a Custom Event Loop A-2

How Your Program Waits for Activities

At the lower level, the program using the API loops waits for activities happening on communication sockets or timers firing off. At the higher level the program is waiting for agent response to some solicited requests (for example, PemGetApplList()) or to registered PATROL events. Please note the high level term PATROL event and don't confuse it with the low level term in event loop.

BMC Software, Inc., Confidential and Proprietary Information

The Library for the Default Event Loop

The PATROL API default event loop is provided in a separate library called `libhandlers.a` or `handlers.lib`. The header file associated with this library is called `handlers.h`.

Using the PATROL API from an X Application

To use the PATROL API from an X-based program:

1. Make sure `libxhandlers.a` and `libpemapi.a` are installed in the same directory.

2. Link with `libxhandlers.a`, which is shipped with your agent, instead of linking with `libhandlers.a`.

3. Create your application context and assign it (or copy it) to the global variable `g_AppContext` (see the header file `handlers.h`).

Using the PATROL API with a Custom Event Loop

To use the PATROL API from an application with a custom event loop:

1. Don't link your program with `libhandlers.a` shipped with the agent.

2. Read the `handlers.h` header file and implement the routines defined in the header file using your own custom event loop routine.

   Timers in `handlers.h` fire only once (the Xt model) while in some systems (for example, Win32) timers keep firing continuously.

BMC Software, Inc., Confidential and Proprietary Information

Hints

This appendix contains the following hints on allocating communication handles and exiting the PATROL API event loop:

Hint #1—Allocating Communication Handles B-2
Example of Correct Allocation B-2
Example of Incorrect Allocation B-3
Hint #2—Exiting the PATROL API Event Loop B-3
Hint#3—All Handles to Objects Are Not Stored or Freed. B-4

BMC Software, Inc., Confidential and Proprietary Information

Hint #1—Allocating Communication Handles

The communication handles passed by address to by PemnOpen() should never be allocated on the stack. They should be static or allocated on the heap.

Example of Correct Allocation

In this example of correct allocation, `s_hComm` is static and will remain in memory while the program is running.

```
static  PemnCommHandle      s_hComm =NULL;

PemnOpen(1234, "MyHostName",    /* Port nb/ Host Node name  */
   _CommCloseProc,     /* Comm Close Callback      */
   (PemnClientData) NULL, /* No Comm Close ClientData   */ sCommOpenProc,
   /* Comm Open  Callback      */
   (PemnClientData)  NULL, /* No Comm Open  Client Data    */
   & s_hComm
   );
```

BMC Software, Inc., Confidential and Proprietary Information

Example of Incorrect Allocation

In this example of incorrect allocation, `hComm` is allocated on the stack and will be deallocated when we exit the scope of `f()`.

When the connection is broken, the PATROL API tries to put NULL in `hComm` but `hComm` is no longer in the program memory. A crash is likely.

```
void f()
{
   PemnCommHandle hComm =NULL;

   PemnOpen(1234, "MyHostName", /* Port nb/ Host Node name  */
      _CommCloseProc, /* Comm Close Callback      */
      (PemnClientData) NULL, /* No Comm Close ClientData   */ sCommOpenProc,
      /* Comm Open  Callback      */
      (PemnClientData)  NULL, /* No Comm Open  Client Data    */
      & hComm
      );
}
```

Hint #2—Exiting the PATROL API Event Loop

To exit the PATROL API event loop, use the `g_MainLoopInterrupt` function defined in `handlers.h`. You can control your event processing using the function `OneEvent()`.

To implement the function g_MainLoopInterrupt within the MainLoop() function in `handlers.h`, follow this example.

```
void
MainLoop (void)
{
while (!g_MainLoopInterrupt) {
OneEvent(ALL_EVENTS);
}
}
```

BMC Software, Inc., Confidential and Proprietary Information

Hint#3—All Handles to Objects Are Not Stored or Freed

All handles are defined in the context of a function handler. Outside the scope of the handler, the handle is not defined.

BMC Software, Inc., Confidential and Proprietary Information

C

Extending the Power of PATROL

This appendix provides a white paper of the PATROL Application Programming Interface (PATROL API), describes its capabilities for extending the power of PATROL by linking to user-created programs, and provides illustrations and brief examples of how the PATROL API can be used to provide comprehensive event management.

This appendix assumes a familiarity with PATROL concepts and terms. The term *program* is used to indicate the in-house software you write using the PATROL API.

This appendix discusses the following topics:

Introduction C-2
Capabilities for Knowledge Module Developers C-4
Interacting with a KM—Two-Way Communication C-4
Application Monitoring—Beyond the Physical Barrier C-8
Custom Browser of PATROL Objects C-9
Event Knowledge Broker C-10
Capabilities for Application Integrators C-12
Event Translation C-12
Event Correlation C-13
Event Routing C-15
Event Synchronization C-17
What Does the PATROL API Provide? C-19
PATROL API Availability C-20
Conclusion C-21

BMC Software, Inc., Confidential and Proprietary Information

Introduction

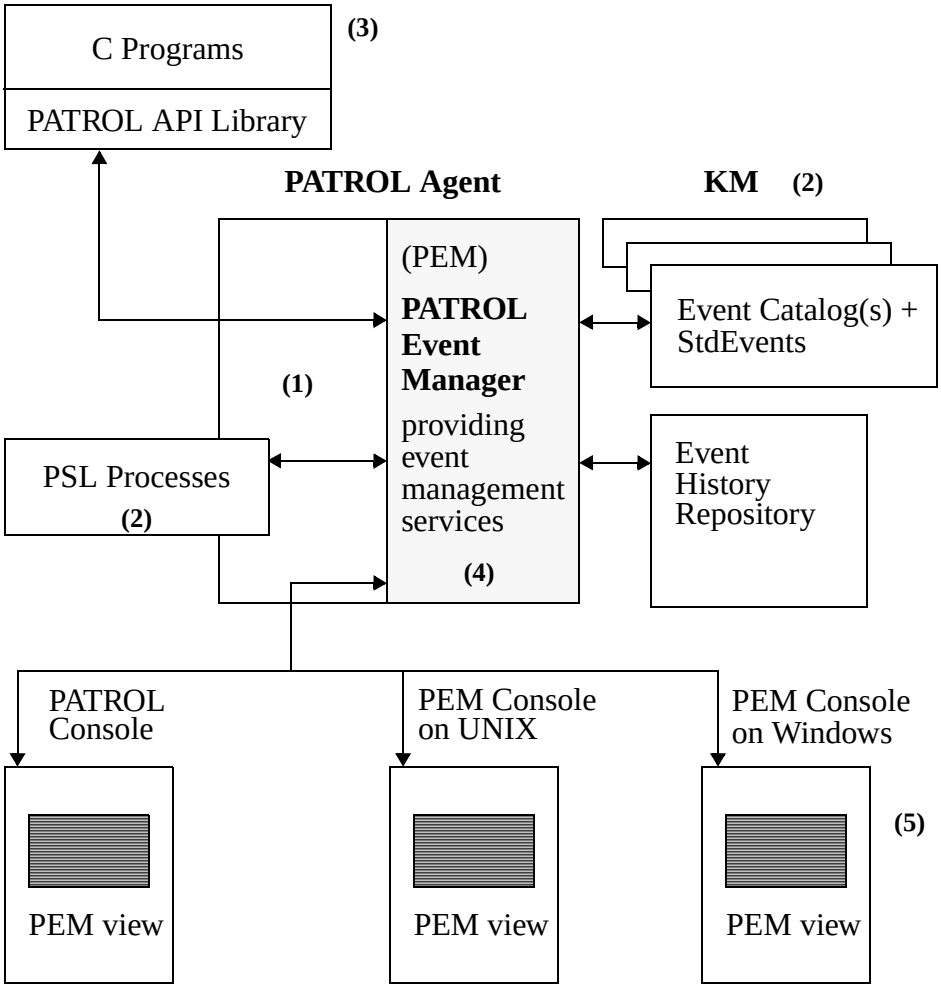
PATROL is a powerful application management tool. It uses autonomous and intelligent agents to monitor databases and applications over the network. While agents provide the application monitoring and management infrastructure, the application knowledge itself is loaded from a Knowledge Module (KM) written in a scripting language called the PATROL Script Language (PSL).

Event management is handled by a sub-module of the agent called the PATROL Event Manager (PEM), The PEM manages events originating from the Agent, a KM running on the Agent, or any application that is linked to the agent through the PATROL API.

Figure C-1 on page C-3 illustrates how the PEM manages events.

BMC Software, Inc., Confidential and Proprietary Information

Figure C-1 Event Management Architecture in PATROL



The source of events can be one of the following:

- the Agent (1)
- the Knowledge Module (2)
- any application using the PATROL API (3)

Events are stored and managed in the PEM (4) but can be viewed and accessed from different consoles (5).

BMC Software, Inc., Confidential and Proprietary Information

Capabilities for Knowledge Module Developers

The first part of this appendix is of interest to KM developers. It shows how you can interact with PATROL Agents and extend the KM capabilities using the PATROL API.

Table C-1 lists several examples that illustrate this interaction:

Table C-1 PATROL API Examples for KM Developers

Example	Description	See Page...
Interacting with a KM—Two-Way Communication	Shows how a custom written application interacts in a bidirectional way with a KM.	C-4
Application Monitoring—Beyond the Physical Barrier	Shows how an application distributed physically on different hosts appears as a single monitored entity to the user.	C-8
Custom Browser of PATROL Objects	Shows how you can write your own specialized application browser of PATROL.	C-9
Event Knowledge Broker	Shows how event knowledge (sub-set of a KM) can be loaded from a knowledge broker and broadcasted to all interested hosts.	C-10

Interacting with a KM—Two-Way Communication

PSL is a very powerful scripting language. In most situations, PSL can express and capture the whole of the application knowledge. However, in some situations, using a C/C++ program can add value to PSL. In a way, the PATROL API allows a KM to be extended using a C/C++ program.

For example, Figure C-2 illustrates how the program can act as an extended collector of data. First, A program allows the KM to obtain richer data, data from other APIs, or data with finer granularity.

For simple data, the program can use the PSL `set()` function to set KM variables. (For more information on the PSL `set()` function, refer to the PATROL *Script Language Reference Manual*.)

BMC Software, Inc., Confidential and Proprietary Information

For complex data, data is collected and sent to the PATROL Agent where it will be further processed by the PSL notification command belonging to the KM. The Notification command can create new parameters, increment the value of some variables, etc.

Second, the program can act as event originator and request the KM to execute PSL/OS commands on behalf of the event.

Figure C-2 Program-Extended Data Collector

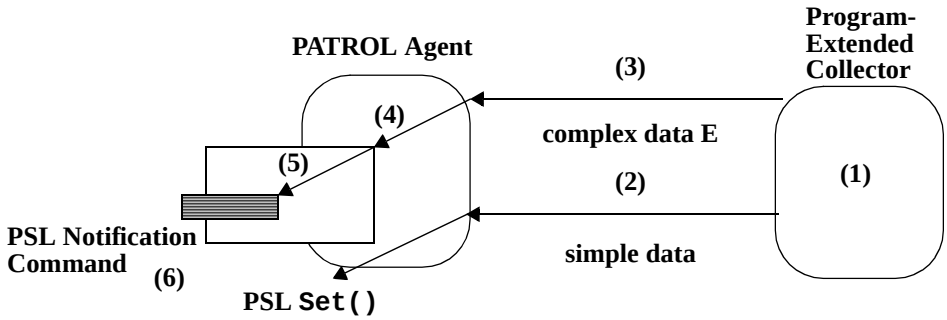


Figure C-2 on page C-5 illustrates the program flow that occurs when the program serves as an extended collector of data:

1. The extended collector program collects granular or pertinent data.

2. Simple data are passed to the KM using the PSL `Set()` function.

3. Complex data E is sent in an INFO event to the PATROL Agent.

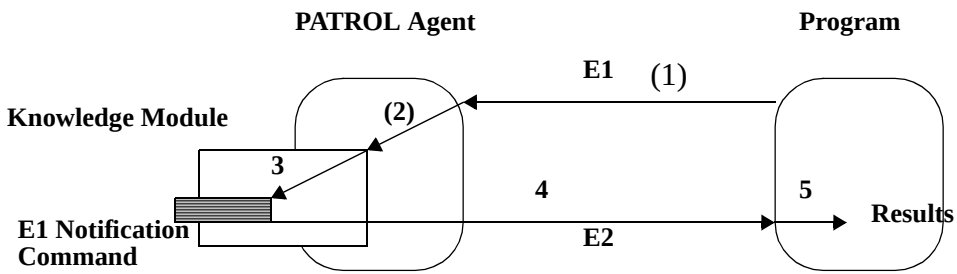
4. The PATROL Agent fetches in the KM the Notification command of complex data E.

5. The agent executes the PSL Notification command.

6. The PSL Notification command increments counters or sets variables in the KM, etc.

Finally, the program can be heavily synchronized with the knowledge as described in Figure C-3.

Figure C-3 Two-Way Communication in Interacting with a KM



1. Application sends an event E1.

2. Agent fetches in the KM the Notification command of E.

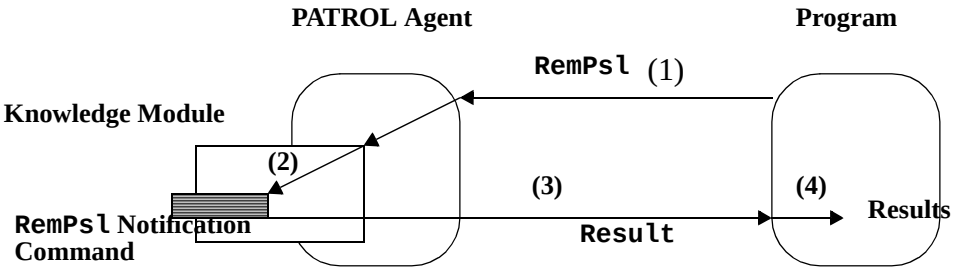
3. The agent executes the PSL Notification command.

4. The PSL Notification command calls variables and objects in the KM, execute other PSL functions then stores any results in event E2 and sends it back to the program.

5. The program listens to event E2 and captures back the results.

As a final example of program and KM interaction, a remote PSL or OS execution can be initiated by the program. The command executes on the agent and results returned to the program. For convenience, the PATROL API provides a wrapper function called `PemnBPslExec()` which allows the program to dispatch a PSL statement to the remote agent for execution and wait for results.

Figure C-4 Remote PSL Execution



1. A special event called '`RemPsl`' is sent to the agent. This event contains as an argument the PSL statement to be executed.

2. The agent receives the `RemPsl` event and executes its notification command. The notification command does noting other than reading the PSL statement from `RemPsl` and executing it.

3. The results are captured and returned to the program using another event called `Result`.

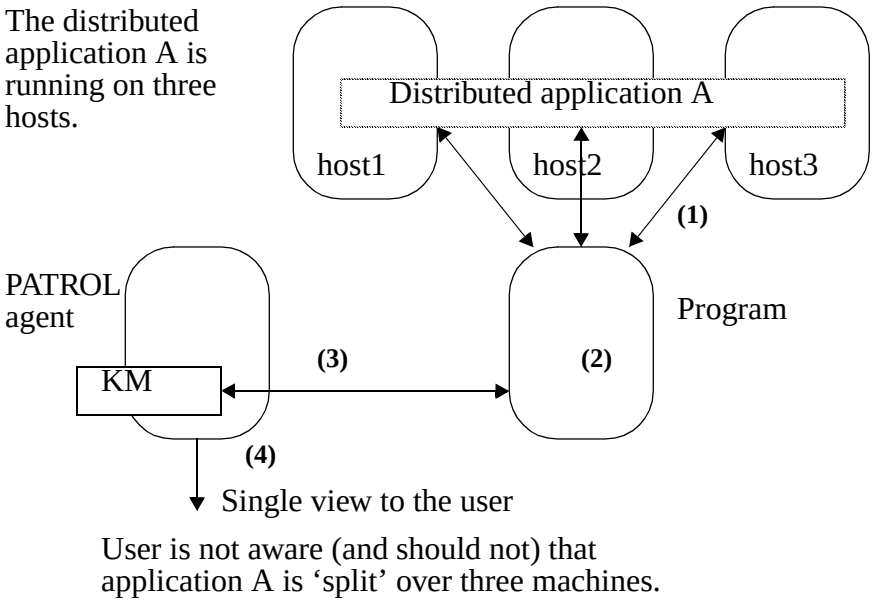
4. The program receives the Result event and extracts the PSL command results.

BMC Software, Inc., Confidential and Proprietary Information

Application Monitoring—Beyond the Physical Barrier

An application can be spread over different machines. As an example, a user may access information in three replicated databases over three hosts. From a user perspective, the application is down (alarm) when the user cannot access the information. One database could be down or temporarily unavailable but this is not an alarming condition to the user as it is always possible to access the same information from the other databases. One KM running on one agent (instead of three) can detect this kind of situation. The notion of ONE application running on ONE host is being replaced by an application that *is a container accessing the services of other applications on same or different machines.*

Figure C-5 Application Monitoring Beyond the Physical Barrier

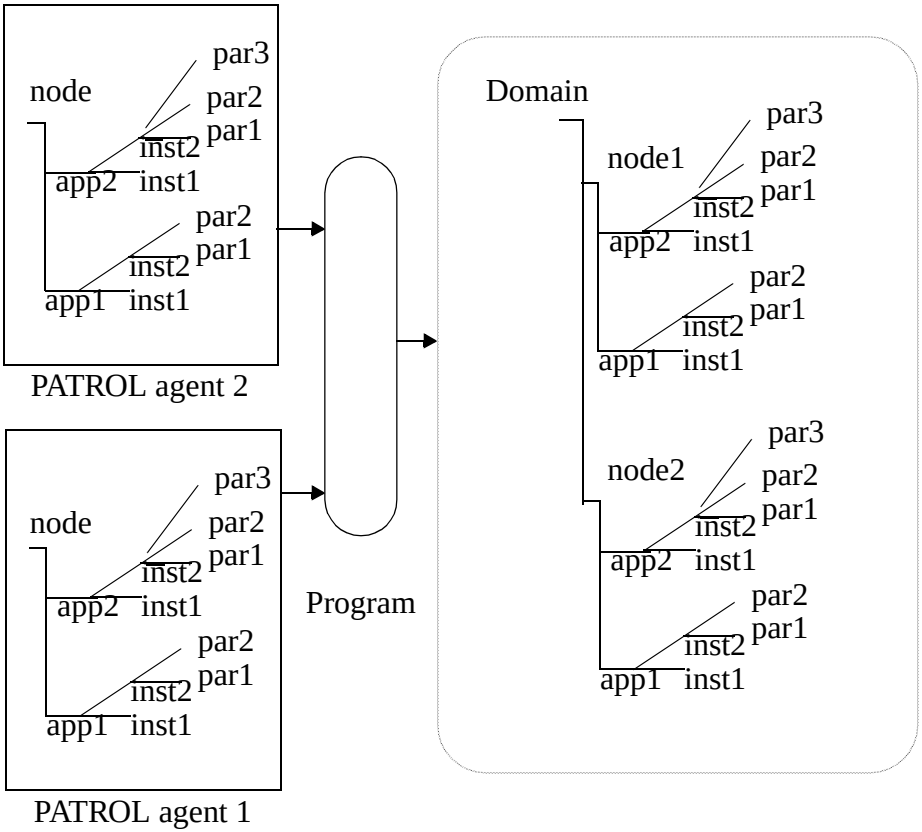


1. The program interact with a all components of application A.
2. The program massages and unifies information.
3. The program interacts with the KM.
4. User is managing a single object not three different objects.

Custom Browser of PATROL Objects

A program can recreate the PATROL object hierarchy and view it in a custom way. The list of applications, instances, parameters or variables can be obtained directly by the API in addition to object attributes: status and value. Figure C-6 shows a custom view of PATROL where agents are structured into domains.

Figure C-6 Custom View of the PATROL Object Browser



The program can reconstruct the whole PATROL hierarchy and display it in a custom hierarchy (e.g domain, node, application, instance, parameter) or in a third party software format.

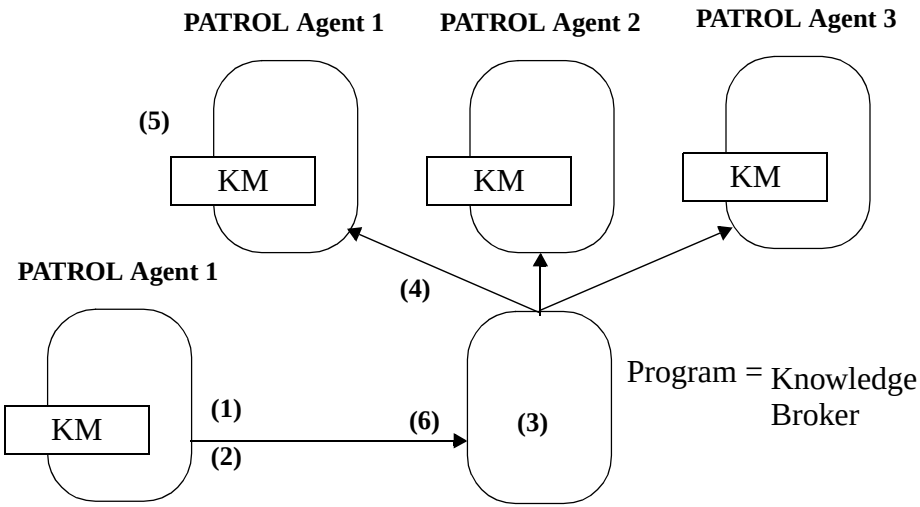
BMC Software, Inc., Confidential and Proprietary Information

Event Knowledge Broker

KMs contain the static knowledge of an application. However, in some situations it makes sense to create the knowledge dynamically by the program. This event knowledge is volatile in the sense it will disappear once the agent exits. This dynamic knowledge (compared to the traditional static knowledge) can be used in event forwarding, routing, translation or synchronization.

As an example, a program acting as an event router can interrogate an agent looking for the event catalog used for routing events. When the knowledge is found, it is sent to some other relevant agents (client agents). The program then disconnects with the original agent and interacts with the client agents to route events.

Figure C-7 Event Knowledge Broker



1. Agent 1 is interrogated by the program for a particular set of event knowledge.

2. Event knowledge is read from the KM in Host 1.

3. The program creates any additional event knowledge needed for functioning. The program fetches the list of all agents requiring this knowledge.

4. The program send them the new event knowledge.

5. The volatile knowledge is used to control events.

6. The connection with agent 1 is disconnected.

BMC Software, Inc., Confidential and Proprietary Information

Capabilities for Application Integrators

The second part of this appendix is of interest to application integrators. It shows how the PATROL API can be used for application integration and enterprise management.

Table C-2 lists several examples that illustrate this interaction:

Table C-2 PATROL API Examples for Application Integrators

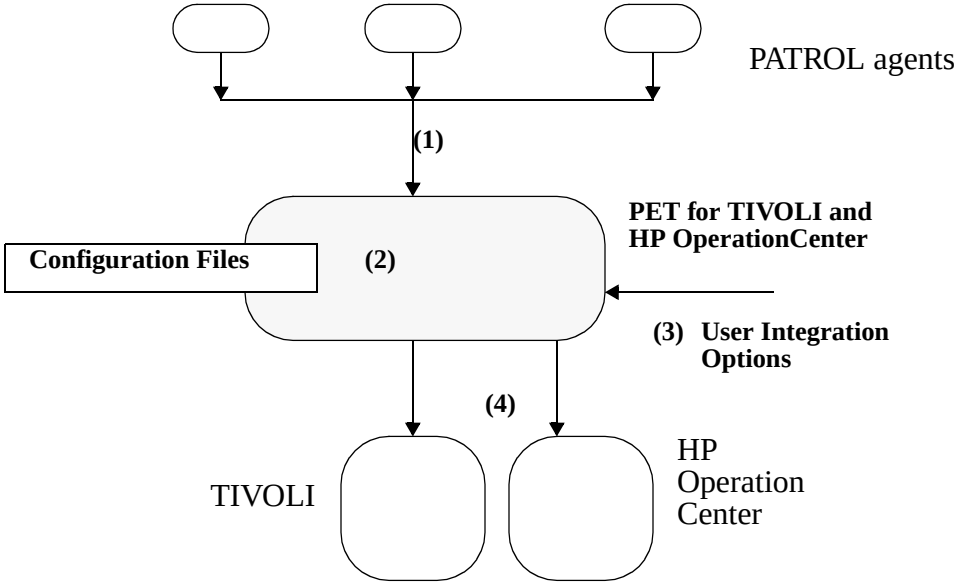
Example	Description	See Page...
Event Translation	Illustrates how PATROLVIEW translates PATROL events into third party vendor events.	C-12
Event Correlation	Shows how events from different hosts, KMs and sources are correlated into a new compound event which initiates further PATROL agent actions.	C-13
Event Routing	Shows how events can be routed from one agent to another agent or to a custom program. The destination agent or program provides specialized services: e.g archiving, capacity planning, event consolidation, etc.	C-15
Event Synchronization	Shows how an event translated into a peer vendor object can be tracked. Whenever this event changes status the peer vendor object can be modified to reflect the change synchronously.	C-17

Event Translation

PATROL events can be re-mapped and formatted for management framework (see Figure C-8). A PATROL Event Translator (PET) listens to events coming from the agents. If an event matches certain criteria, it will be formatted and re-mapped to be inserted as an event of the management framework.

BMC Software, Inc., Confidential and Proprietary Information

Figure C-8 Event Translator



As exemplified in Figure C-6, a PET can be used to translate PATROL events into HP Operation Center or TIVOLI events. First, events are collected from PATROL Agents (1). Then, based on configuration (2) and user integration options (3), events are translated into events that are specific to TIVOLI or HP OpC (4).

Event Correlation

The PATROL API provides the necessary infrastructure for event correlation. Simple event correlation can be done in-house using an event correlation (see Figure C-9 on page C-14) program:

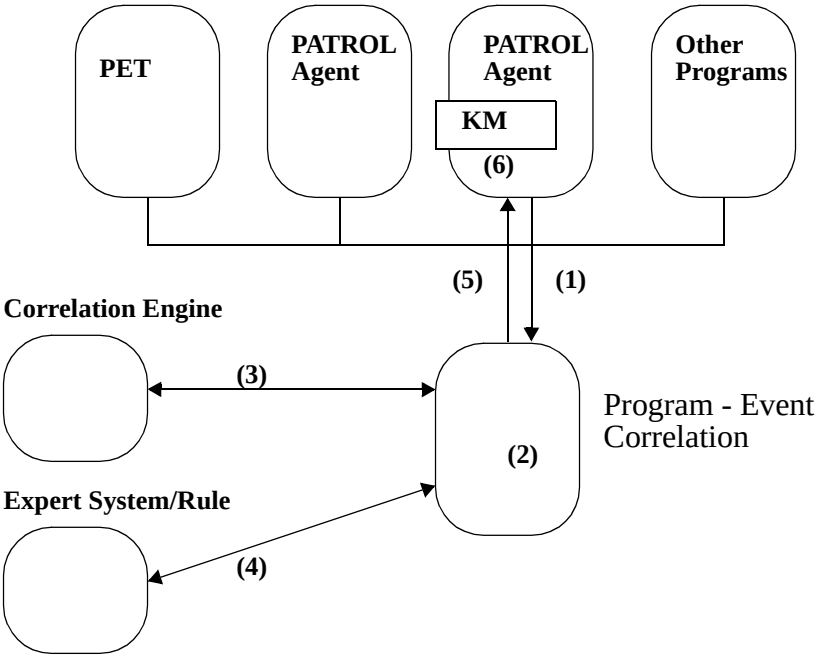
- Correlate events originated from different hosts.
- Correlate events originated in different KMs.
- Correlate events originated from other programs (for example, the PET and event routers).
- Correlate events where the correlation logic is not complex.

BMC Software, Inc., Confidential and Proprietary Information

Complex correlation may necessitate the integration with existing products:

- For complicated correlation logic, the program may use the service of a correlation engine.
- For rule based correlation, the program may use the services of an expert system.

Figure C-9 Event Correlation



(1) Events are originated from PATROL agents, PET or other programs.

(2) The program filters, massages and events and correlates event from different hosts, KM or other programs. For complex correlation it may use the services of (3) a correlation engineer (4) an expert system.

(5) The Event Correlation program triggers new events resulting from the correlation and send these events to PATROL agents.

(6) These compound events may trigger new actions by PSL notification commands.

For example: Assuming that the KM defines three event classes EvClass1, EvClass2 and EvClass3. The event correlator program can evaluate this boolean condition:

```
if ( ((EvClass1) && (EvClass2.origin == ORACLE)) ||
(EvClass3.cardinality==5))
```

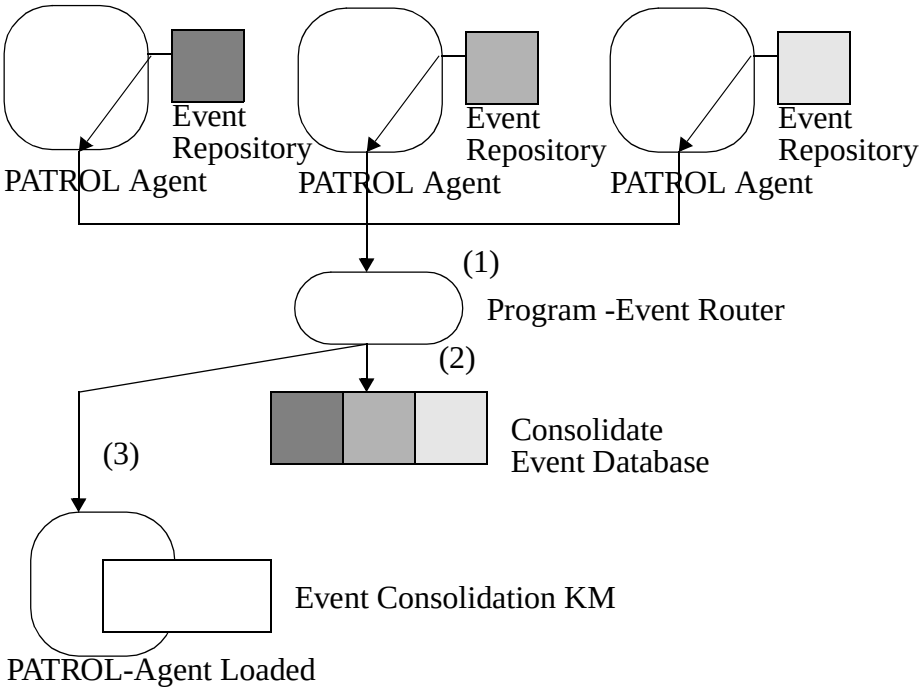
In other words the program can detect the condition when one event (EvClass3) has occurred 5 times *or* that event (EvClass1) has occurred *and* also event (EvClass2) has occurred in the ORACLE KM.

Event Routing

The PEM engine in the agent processes and manages all events that are local to the agent. In some situations, it may be judicious to centralize the processing and management of events. Events are sent to a local consolidation program where they will be stored in a relation database for history, capacity planning or archiving. Assuming that the user wants archiving of all alarm and warning events in a single relational database, the program registers to receive alarms and warnings from PATROL agents then it calls an embedded SQL statement to insert the events in the RDBMS. The program may do further event aggregation and send the new event to an agent with an Event Consolidation KM. The KM in this case will execute the pertinent SQL statements.

BMC Software, Inc., Confidential and Proprietary Information

Figure C-10 Event Forwarding



1. Events managed by different agents are sent to the program.

2. The event router filters, consolidate and store events in a company-wide database for archiving, capacity planning or future analysis.

3. The consolidated events are sent to an agent with a special KM which executes Notification commands generating report and possibly triggering new events.

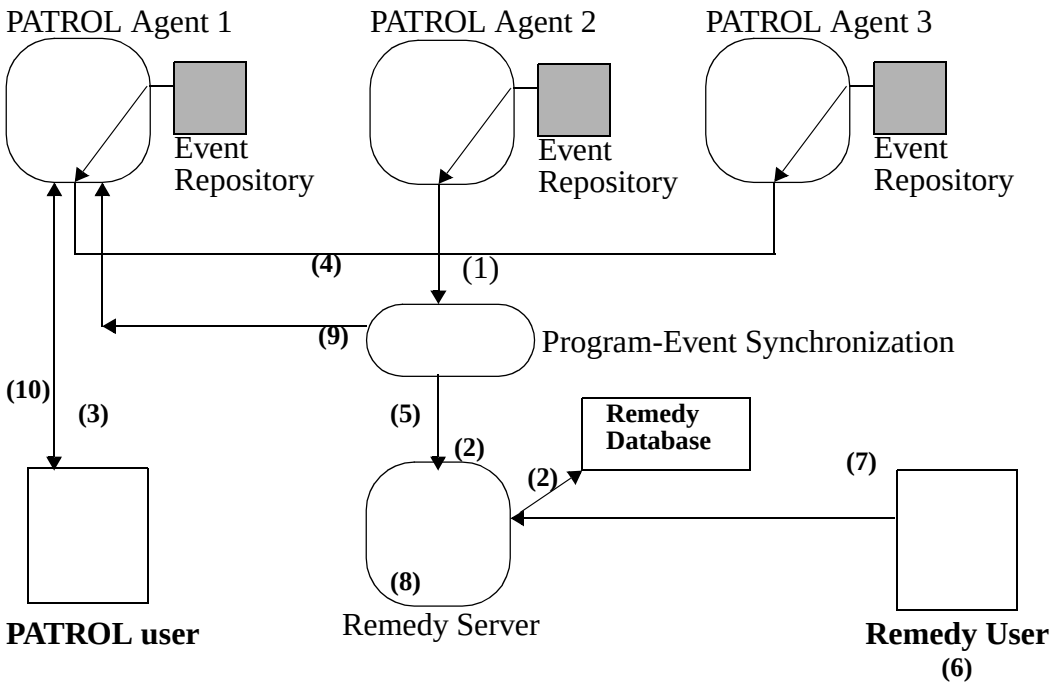
Event Synchronization

Translating a PATROL event into a peer object belonging to other applications allows easy integration. However, sometimes there is a need to track the state of the event or the peer object to enrich the integration. Tracking the state of the peer object and keeping it synchronized with the state of the original event is called *event synchronization*.

For Example: After translating a PATROL event into a Remedy trouble ticket, the user wants to avoid the situation where an operator using PATROL Console closes the event but the peer trouble ticket remains unchanged. The user wants the trouble ticket to be closed as well. This is how event synchronization resolves the problem: The program translating the event registers the event with the PATROL agent and marks it as ‘Remedy Integrated’. Later when the event changes status, the program detects the fact that a trouble ticket has already been created and has changed status; it then notifies Remedy of the new status. The program can also poll the status of the Remedy trouble ticket and updates the status of the peer PATROL event when the trouble ticket changes status.

BMC Software, Inc., Confidential and Proprietary Information

Figure C-11 Example of Event Synchronization



1. A sub- set of PATROL event is captured by the Program.
2. For every ‘interesting’ event create a peer Remedy trouble ticket.
3. A PATROL user closes the event on agent 1.
4. The program is notified of the change.
5. Remedy is notified of the new status.
6. The Remedy user is notified.
7. Going the other direction, the Remedy user changes [1]the status of the trouble ticket.

1. Steps 1 to 6 are provided by the current version of PEM Console for Remedy. Steps 7 to 10 may be provided in a future version.

BMC Software, Inc., Confidential and Proprietary Information

8. The Program (reading log files in a periodical fashion) detects the new status of the trouble ticket.

9. The program closes/acknowledges the PATROL event using the API.

10. The PATROL user is notified of the change.

What Does the PATROL API Provide?

The following services are provided by the PATROL API:

- Connection Management

  — Connection and disconnection with PATROL Agents.
  — Password and user name authentication and encryption.

- Event Management

  — Send events to agent.Receive events from agent.
  — Close/Acknowledge/Delete or add diary to events.
  — Generate reports.
  — Query past events from agent.

- PATROL Objects Access

  — Get PATROL object list: list of applications, instances, parameters or variables.

  — Get PATROL object attributes: applications, instances, parameters or variables. As an example, the value of a parameter, the status of any objects, the name of the worst parameter in an instance can be returned by the PATROL API.

- Event Knowledge

  — Get the list of event classes in the KM.
  — Get the description of an event class: the text of PSL/OS notification commands, the expert advice text.

BMC Software, Inc., Confidential and Proprietary Information

- Event Synchronization

  Register an event with the agent to track and update its status in the integrated product.

- Event Driven and Blocking Mode

  For every service provided by the PATROL API, there is a blocking mode version and an event driven version. In the blocking mode version the program callers waits for the results until they are received or a time-out occurs. In the event driven version the program registers callback which are executed when the results arrive.

  In general, blocking mode programming is easier to write while event driven programming is better in performance and multi-host connection support.

PATROL API Availability

The PATROL API has been used already in PATROLVIEW to integrate with TIVOLI and HP Operation Center. PATROL Event Translator (PET) registers with one or more agents and then translates PATROL events to TIVOLI or HP Operation center events.

The PATROL Command Line Interface (CLI), which is character based, also uses the PATROL API to access and manage PATROL objects and events in the Agent.

Connection and event management services (see Section) are already provided in PATROL v3.0. PATROL object access and event knowledge services are implemented for v3.1. Event synchronization services are used only internally in the PEM for Remedy products. The PATROL API is available on all PATROL Agent platforms: UNIX, NT, and so forth.

BMC Software, Inc., Confidential and Proprietary Information

Conclusion

PATROL Application Management strength resides in its KM coverage and depth. The API adds to this strength by extending the power and inter-operability of KMs. The API is offered to KM developers and customers alike.

Today applications interact more and more with each other. The API provides the mechanisms to open and integrate PATROL agent technology with potentially any other product.

Customers always prefer a finished solution, but in a changing world with emerging new technologies and products, an open API allows the customer to develop in-house solutions. The PATROL API gives them the tools.

BMC Software, Inc., Confidential and Proprietary Information

D

Example of annotate() Function

This appendix provides an the following example:

annotate() Example. D-2

BMC Software, Inc., Confidential and Proprietary Information

annotate() Example

The following example shows of how to use the `annotate()` function in a custom interface.

```
// the following 3 functions are helper functions to show you
// how to use annotate() function in IPatrolAgent interface
HRESULT CreateVariant4BSTR(VARIANT *pvar, BSTR bstr)
{
    VariantInit (pvar);
    V_VT(pvar) = VT_BSTR;
    V_BSTR(pvar) = bstr;

    return S_OK;
}

HRESULT CreateSafeArray(SAFEARRAY **ppsa, int numElements, BSTR* pbstr)
{
    SAFEARRAYBOUND rgsabound[1];
    VARIANT var;
    long ix[1];

    if (NULL == ppsa || 0 == numElements)
        return E_FAIL;


    rgsabound[0].lLbound = 0;
        rgsabound[0].cElements = numElements;
        *ppsa = SafeArrayCreate(VT_VARIANT, 1, rgsabound);

    if (NULL == *ppsa)
        return E_OUTOFMEMORY;

    for (int i=0; i<numElements; i++) {
        CreateVariant4BSTR(&var, pbstr[i]);
        ix[0] = i;
               SafeArrayPutElement(*ppsa, ix, &var);
    }

    return S_OK;
}


HRESULT AnnotateArg(VARIANT* pvar)
{
    int count=3;
```

```
    BSTR pbstr[3];
    SAFEARRAY *psa;

    pbstr[0]= SysAllocString (L"progarg1\n100");
    pbstr[1]= SysAllocString (L"progarg2\n200");
    pbstr[2]= SysAllocString (L"progarg3\n300");

    CreateSafeArray(&psa, count, pbstr);
    VariantInit (pvar);
    V_VT(pvar) = VT_ARRAY | VT_VARIANT;
    V_ARRAY(pvar) = psa;


    return S_OK;
}

void main()
{
    BSTR bstrRet=0, bstrName, bstrFormat;
    VARIANT v;

                        .....
        /*get an interface pointer of IPatrolAgent, setup bstrName, bstrFormat*/
                        ......
                        .......

         /* prepare for the pvarData*/
    AnnotateArg(&v);
    hr = pIPatrolAgent->annotate(bstrName,bstrFormat,&v,&bstrRet);

                        .....
}
```

BMC Software, Inc., Confidential and Proprietary Information

BMC Software, Inc., Confidential and Proprietary Information

Index

A

accessing PATROL Agent COM server through dispatch interface 13-7
annotate() 13-11
annotate_get() 13-13
application management functions
- PemnBGetApplList() return application class list (blocking) 9-6
- PemnBGetApplObj() return application class object (blocking) 9-9
- PemnBGetComputerParamList() return computer parameter list (blocking) 9-15
- PemnBGetComputerParamObj() return computer parameter (blocking) 9-18
- PemnBGetInstList() return application instance list (blocking) 9-21
- PemnBGetInstObj() return application instance object (blocking) 9-24
- PemnBGetParamList() return instance parameter list (blocking) 9-30
- PemnBGetParamObject() return parameter object 9-33
- PemnBGetVarList() return variable list (blocking) 9-39
- PemnBGetVarObj() return variable object (blocking) 9-42
- PemnBPslExec() execute PSL script (blocking) 9-2
- PemnBSetVarObj() set PATROL variable (blocking) 9-4
- PemnGetApplInstObjAttributes() return instance object attributes 9-27
- PemnGetApplList() return application class list 9-6
- PemnGetApplObj() return application class object 9-9
- PemnGetApplObjAttributes() return application class object attributes 9-12
- PemnGetApplObjInfo() return application object info 9-13
- PemnGetComputerParamList() return computer parameter list 9-15
- PemnGetComputerParamObj() return computer parameter object 9-18
- PemnGetInstList() return application instance list 9-21
- PemnGetInstObj() return application instance object 9-24
- PemnGetInstObjInfo() return instance object information 9-28
- PemnGetParamList() return instance parameter list 9-30
- PemnGetParamObj() return parameter object 9-33

BMC Software, Inc., Confidential and Proprietary Information

PemnGetParamObjAttributes() return parameter object attributes 9-36
PemnGetParamObjInfo() return parameter object info 9-37
PemnGetVarList() return variable list 9-39
PemnGetVarObj() return variable object 9-42
array type and index definitions
typedef PemnApplAttrArray 3-5
typedef PemnApplAttrEnum 3-6
typedef PemnInstAttrArray 3-23
typedef PemnInstAttrEnum 3-24
typedef PemnParamAttrArray 3-26, 3-27

B

blackout() 13-15
blocking functions 1-6
pointer 1-7
return values 1-7

C

change_state() 13-17
COM/DCOM files 13-4
COM/DCOM protocol 13-3
comSecurityLevel 13-5
comUserAllowed 13-5
configuring COM servers 13-4
Configuring DCOM clients 13-6
connection functions
PemnBPing() ping PATROL Agent 5-5
PemnClose() close connection 5-7
PemnGetCommAttributes() get connection attributes 5-8
PemnGetLocalPort() return local port number 5-10
PemnLoop() monitor for connection activity 5-11
PemnOpen() open connection 5-12
PemnSetCommAttributes() set connection attributes 5-15
PemnSetLocalPort() set local port number 5-17
PemnSetNoAgentHeartbeat() disable the heartbeat feature 5-18
create() 13-19
custom interface
example of invoking function in 13-10
to access PATROL Agent COM server 13-8

D

data management functions
PemnBPutFile() send file to remote agent (blocking) 10-2
PemnBPutStream() send character stream (blocking) 10-5
PemnPutFile() send file to remote agent 10-2
PemnPutStream() send character stream 10-5
data structures
overview 3-3
type definitions (table) 3-3
DCOM clients, configuring 13-6
DCOMCFG.EXE
Microsoft DCOM configuration utility 13-4
destroy() 13-21
dispatch interface
example of invoking function in 13-10
guidelines for using 13-7
to access PATROL Agent COM server 13-7

BMC Software, Inc., Confidential and Proprietary Information

E

enumeration definitions
- typedef PemnEventStatusEnum 3-14
- typedef PemnEventTypeEnum 3-15

event access functions
- PemnGetArgs() return event arguments 4-2
- PemnGetClassName() return event class 4-3
- PemnGetDesc() return event description 4-4
- PemnGetDiary() get event diary 4-5
- PemnGetEid return PATROL event identifier 4-6
- PemnGetEventCatalogName() return event catalog name 4-7
- PemnGetExpectancyStr() return event life expectancy 4-8
- PemnGetHandler() return event last user ID 4-9
- PemnGetNode() return event host name 4-10
- PemnGetNSeverity() return numeric severity 4-11
- PemnGetOrigin() return event application class 4-12
- PemnGetOwner() return owner user ID 4-13
- PemnGetSourceID() return event source ID 4-14
- PemnGetStatus() return event status 4-15
- PemnGetTime() return event timestamp 4-16
- PemnGetType() return event type 4-17

event management functions
- PemnAddDiary() add text to event diary 8-2
- PemnBEventArchive() write events to a file (blocking) 8-21
- PemnBEventManageIfMatch() change matching event status 8-32
- PemnBEventQuery() return filtered events (blocking) 8-39
- PemnBEventReport() summarize filtered events (blocking) 8-52
- PemnBEventScheduleAdd() add event to agent schedule queue 8-4
- PemnBEventScheduleDelete() remove event from agent schedule queue 8-7
- PemnBEventScheduleList() list all events scheduled for execution 8-9
- PemnBEventScheduleModify() modify a scheduled event 8-11
- PemnBEventScheduleResume() resume an event that was suspended 8-14
- PemnBEventScheduleSuspend() suspend an event in the agent schedule queue 8-16
- PemnBGetEventClass() return event class info (blocking) 8-59
- PemnBGetEventClassList() list event classes (blocking) 8-63
- PemnBSetPersistentFilter() set session event filter (blocking) 8-69
- PemnDefineEventClass() define event class 8-18
- PemnEventManage() change event status 8-30
- PemnEventManageIfMatch() change matching event status 8-32
- PemnEventQuery() return filtered events 8-39
- PemnEventRangeManage() change event range status 8-46
- PemnEventRangeQuery() return range of events 8-49

BMC Software, Inc., Confidential and Proprietary Information

PemnEventReport() summarize filtered events 8-52
PemnEvnetArchive() write events to a file 8-21
PemnFlushEvent() flush events to log 8-58
PemnGetEventClass() return event class info 8-59
PemnGetEventClassList() list event classes 8-63
PemnSetEventClassCmd() define event class command 8-66
PemnSetPersistentFilter() set session event filter 8-69
event-driven (non-blocking) functions 1-6
exists() 13-23

F

functions
PemnAddDiary() add text to event diary 8-2
PemnBEventArchive() write events to a file (blocking) 8-21
PemnBEventManageIfMatch() change matching event status (blocking) 8-32
PemnBEventQuery() return filtered events (blocking) 8-39
PemnBEventReport() summarize filtered events (blocking) 8-52
PemnBEventScheduleAdd() add event to agent schedule queue 8-4
PemnBEventScheduleDelete() remove event from agent schedule queue 8-7
PemnBEventScheduleList() list all events scheduled for execution 8-9
PemnBEventScheduleModify() modify a scheduled event 8-11
PemnBEventScheduleResume() resume an event that was suspended 8-14
PemnBEventScheduleSuspend() suspend an event in the agent schedule queue 8-16
PemnBExitFunction() exit blocked function 12-2
PemnBGetApplList() return application class list (blocking) 9-6
PemnBGetApplObj() return application class object (blocking) 9-9
PemnBGetComputerParamList() return computer parameter list (blocking) 9-15
PemnBGetComputerParamObj() return computer parameter (blocking) 9-18
PemnBGetEventClass() return event class info (blocking) 8-59
PemnBGetEventClassList() list event classes (blocking) 8-63
PemnBGetInstList() return application instance list (blocking) 9-21
PemnBGetInstObj() return application instance object (blocking) 9-24
PemnBGetParamList() return instance parameter list (blocking) 9-30
PemnBGetParamObj() return parameter object (blocking) 9-33
PemnBGetTimeout() return max wait time default 12-3
PemnBGetVarList() return variable list (blocking) 9-39
PemnBGetVarObj() return variable object (blocking) 9-42
PemnBPing() ping PATROL Agent 5-5
PemnBPslExec() execute PSL script (blocking) 9-2
PemnBPutFile() send file to remote agent (blocking) 10-2
PemnBPutStream() send character stream (blocking) 10-5

BMC Software, Inc., Confidential and Proprietary Information

PemnBSend() send PATROL event (blocking) 7-7
PemnBSetPersistentFilter() set session event filter (blocking) 8-69
PemnBSetTimeout() set max wait time default 12-4
PemnBSetVarObj() set PATROL variable (blocking) 9-4
PemnCheckInit() check PEM initialization 11-2
PemnClose() close connection 5-7
PemnDefineEventClass() define event class 8-18
PemnEncrypt() encrypt password 6-2
PemnEventArchive() write events to a file 8-21
PemnEventManage() change event status 8-30
PemnEventManageIfMatch() change matching event status 8-32
PemnEventQuery() return filtered events 8-39
PemnEventRangeManage() change event range status 8-46
PemnEventRangeQuery() return range of events 8-49
PemnEventReport() summarize filtered events 8-52
PemnFlushEvent() flush events to log 8-58
PemnGetApplInstObjAttributes() return instance object attributes 9-27
PemnGetApplList() return application class list 9-6
PemnGetApplObj() return application class object 9-9
PemnGetApplObjAttributes() return application class object attributes 9-12
PemnGetApplObjInfo() return application class object info 9-13
PemnGetArgs() return event arguments 4-2
PemnGetClassName() return event class 4-3
PemnGetCommAttributes() get connection attributes 5-8
PemnGetComputerParamList() return computer parameter list 9-15
PemnGetComputerParamObj() return computer parameter object 9-18
PemnGetDesc() return event description 4-4
PemnGetDiary() get event diary 4-5
PemnGetDourceID() return event source ID 4-14
PemnGetEid() return PATROL event identifier 4-6
PemnGetEventCatalogName() return event catalog name 4-7
PemnGetEventClass() return event class info 8-59
PemnGetEventClassList() list event classes 8-63
PemnGetExpectancyStr() return event life expectancy 4-8
PemnGetHandler() return event last user ID 4-9
PemnGetInstList() return application instance list 9-21
PemnGetInstObj() return application instance object 9-24
PemnGetInstObjInfo() return instance information 9-28
PemnGetLastMessage() return last API message 11-3
PemnGetLocalPort() return local port number 5-10
PemnGetNode() return event host name 4-10

BMC Software, Inc., Confidential and Proprietary Information

PemnGetNSeverity() return numeric severity 4-11
PemnGetOrigin() return event application class 4-12
PemnGetOwner() return owner user ID 4-13
PemnGetParamList() return instance parameter list 9-30
PemnGetParamObj() return parameter object 9-33
PemnGetParamObjAttributes() return parameter object attributes 9-36
PemnGetParamObjInfo() return parameter object info 9-37
PemnGetStatus() return event status 4-15
PemnGetTime() return event timestamp 4-16
PemnGetType() return event type 4-17
PemnGetVarList() return variable list 9-39
PemnGetVarObj() return variable object 9-42
PemnLoop() monitor for connection activity 5-11
PemnOpen() open connection 5-12
PemnPutFile() send file to remote agent 10-2
PemnPutStream() send character stream 10-5
PemnReceive() filter incoming events 7-2
PemnRegisterUserMessageHandler() register user message process 11-4
PemnSend() send PATROL event 7-7
PemnSendIdentity() send account info 6-3
PemnSetCommAttributes() set connection attributes 5-15
PemnSetEventClassCmd() define event class command 8-66
PemnSetLocalPort() set local port number 5-17
PemnSetNoAgentHeartbeat() disable the heartbeat feature 5-18
PemnSetPersistentFilter() set session event filter 8-69

G

get() 13-25
get_psl_errno() 13-29
get_ranges() 13-30
get_vars() 13-32
getenv() 13-27

H

HRESULT 13-8

I

interface for COM/DCOM 13-3
Introduction to the API 1-2
ipa.h 13-4
ipa.tlb
    description of file 13-4
ipa_i.c 13-4
IPatrolAgent interface 13-3
IPatrolAgent interface function
    annotate() 13-11
    annotate_get() 13-13
    blackout() 13-15
    change_state() 13-17
    create() 13-19
    destroy() 13-21
    exists() 13-23

BMC Software, Inc., Confidential and Proprietary Information

get() 13-25
get_psl_errno() 13-29
get_ranges() 13-30
get_vars() 13-32
getenv() 13-27
is_var() 13-36
print() 13-40
PslExecute() 13-38
set() 13-41
set_alarm_ranges() 13-43
unset() 13-45
is_var() 13-36

M

Microsoft DCOM configuration utility 13-4, 13-6
miscellaneous definitions
typedef PemnClientData 3-9
miscellaneous functions
PemnCheckInit() check PEM initialization 11-2
PemnGetLastMessage() return last API message 11-3
PemnRegisterUserMessageHandler() register user message process 11-4

N

non-blocking (event-driven) functions 1-6

O

object handle definitions
PemnEventHandle 3-13
typedef PemnApplObjHandle 3-7
typedef PemnCommHandle 3-11
typedef PemnInstObjHandle 3-25
typedef PemnParamObjHandle 3-28
typedef PemnReportHandle 3-33
typedef PemnVarObjHandle 3-35

P

PATROL Agent COM server
accessing through custom interface 13-8
PATROL Agent COM variables
comSecurityLevel 13-5
comUserAllowed 13-5
descriptions of 13-5
PATROL COM/DCOM files 13-4
PATROL Event Manager
receiving events from 1-11
sending events to 1-12
Pemn functions
benefits of 1-10
execution sequence 1-7, 1-12
introduction 1-10
models
blocking 1-6
event-driven 1-6
PemnAddDiary() add text to event diary 8-2
PemnBEventArchive() write events to a file (blocking) 8-21
PemnBEventManageIfMatch() change matching event status (blocking) 8-32
PemnBEventQuery() return filtered events (blocking) 8-39
PemnBEventReport() summarize filtered events (blocking) 8-52
PemnBEventScheduleAdd() add event to agent schedule queue 8-4
PemnBEventScheduleDelete() remove event from agent schedule queue 8-7
PemnBEventScheduleList() list all events scheduled for execution 8-9
PemnBEventScheduleModify() modify a scheduled event 8-11

BMC Software, Inc., Confidential and Proprietary Information

PemnBEventScheduleResume() resume an event that was suspended 8-14
PemnBEventScheduleSuspend() suspend an event in the agent schedule queue 8-16
PemnBExitFunction() exit blocked function 12-2
PemnBGetApplList() return application class list (blocking) 9-6
PemnBGetApplObj() return application class object (blocking) 9-9
PemnBGetComputerParamList() return computer parameter list (blocking) 9-15
PemnBGetComputerParamObj() return computer parameter (blocking) 9-18
PemnBGetEventClass() return event class info (blocking) 8-59
PemnBGetEventClassList() list event classes (blocking) 8-63
PemnBGetInstList() return application instance list (blocking) 9-21
PemnBGetInstObj() return application instance object (blocking) 9-24
PemnBGetInstObjInfo() return instance information 9-28
PemnBGetParamList() return instance parameter list (blocking) 9-30
PemnBGetParamObj() return parameter object (blocking) 9-33
PemnBGetTimeout() return max wait time default 12-3
PemnBGetVarList() return variable list (blocking) 9-39
PemnBGetVarObj() return variable object (blocking) 9-42
PemnBPing() ping PATROL Agent 5-5
PemnBPslExec() execute PSL script (blocking) 9-2
PemnBPutFile() send file to remote agent (blocking) 10-2
PemnBPutStream() send character stream (blocking) 10-5
PemnBSend() send PATROL event (blocking) 7-7
PemnBSetPersistentFilter() set session event filter (blocking) 8-69
PemnBSetTimeout() set max wait time default 12-4
PemnBSetVarObj() set PATROL variable (blocking) 9-4
PemnCheckInit() check PEM initialization 11-2
PemnClose() close connection 5-7
PemnDefineEventClass() define event class 8-18
PemnEncrypt() encrypt password 6-2
PemnEventArchive() write events to a file 8-21
PemnEventManage() change event status 8-30
PemnEventManageIfMatch() change matching event status 8-32
PemnEventQuery() return filtered events 8-39
PemnEventRangeManage() change event range status 8-46
PemnEventRangeQuery() return range of events 8-49
PemnEventReport() summarize filtered events 8-52
PemnFlushEvent() flush events to log 8-58
PemnGetApplList() return application class list 9-6
PemnGetApplObj() return application class object 9-9
PemnGetApplObjAttributes() return application class object attributes 9-12
PemnGetApplObjInfo() return application object info 9-13
PemnGetArgs() return event arguments 4-2

BMC Software, Inc., Confidential and Proprietary Information

PemnGetClassName() return event class 4-3
PemnGetCommAttributes() get connection attributes 5-8
PemnGetComputerParamList() return computer parameter list 9-15
PemnGetComputerParamObj() return computer parameter object 9-18
PemnGetDesc() return event description 4-4
PemnGetDiary() get event diary 4-5
PemnGetEid() return PATROL event identifier 4-6
PemnGetEventCatalogName() return event catalog name 4-7
PemnGetEventClass() return event class info 8-59
PemnGetEventClassList() list event classes 8-63
PemnGetExpectancyStr() return event life expectancy 4-8
PemnGetHandler() return event last user ID 4-9
PemnGetInstList() return application instance list 9-21
PemnGetInstObj() return application instance object 9-24
PemnGetInstObjAttributes() return instance object attributes 9-27
PemnGetLastMessage() return last API message 11-3
PemnGetLocalPort() return local port number 5-10
PemnGetNode() return event host name 4-10
PemnGetNSeverity() return numeric severity 4-11
PemnGetOrigin() return event application class 4-12
PemnGetOwner() return owner user ID 4-13
PemnGetParamList() return instance parameter list 9-30
PemnGetParamObj() return parameter object 9-33
PemnGetParamObjAttributes() return parameter object attributes 9-36
PemnGetParamObjInfo() return parameter object info 9-37
PemnGetSourceID() return event source ID 4-14
PemnGetStatus() return event status 4-15
PemnGetTime() return event timestamp 4-16
PemnGetType() return event type 4-17
PemnGetVarList() return variable list 9-39
PemnGetVarObj() return variable object 9-42
PemnLoop() monitor for connection activity 5-11
PemnOpen() open connection 5-12
PemnPutFile() send file to remote agent 10-2
PemnPutStream() send character stream 10-5
PemnReceive() filter incoming events 7-2
PemnRegisterUserMessageHandler() register user message process 11-4
PemnSend() send PATROL event 7-7
PemnSendIdentity() send account info 6-3
PemnSetCommAttributes() set connection attributes 5-15
PemnSetEventClassCmd() define event class command 8-66
PemnSetLocalPort() set local port number 5-17
PemnSetNoAgentHeartbeat() disable the heartbeat feature 5-18
PemnSetPersistentFilter() set session event filter 8-69
print() 13-40
PslExecute() 13-38

BMC Software, Inc., Confidential and Proprietary Information

R

return values for blocking functions 1-7
routine callback definitions
typedef PemnBDisconnectCallback 3-8
typedef PemnCommCallback 3-10
routine handler definitions
typedef PemnDumpEventHandler 3-12
typedef PemnGetApplObjHandler 3-16
typedef PemnGetEventClassHandler 3-17
typedef PemnGetEventClassListHandler 3-18
typedef PemnGetInstObjHandler 3-19
typedef PemnGetListHandler 3-20
typedef PemnGetParamObjHandler 3-21
typedef PemnGetVarObjHandler 3-22
typedef PemnQueryEventHandler 3-29
typedef PemnReceiveEventHandler 3-30
typedef PemnReportEventHandler 3-32
typedef PemnUserMessage Handler 3-34

S

security functions
PemnEncrypt() encrypt password 6-2
PemnSendIdentity() send account info 6-3
send/receive functions
PemnBSend() send PATROL event (blocking) 7-7
PemnReceive() filter incoming events 7-2
PemnSend() send PATROL event 7-7
set() 13-41
set_alarm_ranges() 13-43
special blocking functions
PemnBExitFunction() exit blocked function 12-2
PemnBGetTimeout() return max wait time default 12-3
PemnBSetTimeout() set max wait time default 12-4

T

type definitions
PemnCommCallback 3-10
PemnGetEventClassHandler 3-17
typedef PemaGetInstObjHandler 3-19
typedef PemnApplAttrArray 3-5
typedef PemnApplAttrEnum 3-6
typedef PemnApplObjHandle 3-7
typedef PemnApplObjHandler 3-16
typedef PemnBDisconnectCallback 3-8
typedef PemnClientData 3-9
typedef PemnCommHandle 3-11
typedef PemnDumpEventHandle 3-12
typedef PemnEventHandle 3-13
typedef PemnEventStatusEnum 3-14
typedef PemnEventTypeEnum 3-15
typedef PemnGetEventClassListHandler 3-18
typedef PemnGetListHandler 3-20
typedef PemnGetParamObjHandler 3-21
typedef PemnGetVarObjHandler 3-22
typedef PemnInstAttrArray 3-23
typedef PemnInstAttrEnum 3-24
typedef PemnInstObjHandle 3-25
typedef PemnParamAttrArray 3-26
typedef PemnParamAttrEnum 3-27
typedef PemnParamObjHandle 3-28
typedef PemnQueryEventHandler 3-29
typedef PemnReceiveEventHandler 3-30
typedef PemnReportEventHandler 3-32
typedef PemnReportHandle 3-33
typedef PemnUserMessageHandler 3-34

BMC Software, Inc., Confidential and Proprietary Information

typedef PemnVarObjHandle 3-35
type definitions (table) 3-3
typedef PemaGetInstObjHandler 3-19
typedef PemnApplAttrArray 3-5
typedef PemnApplAttrEnum 3-6
typedef PemnApplObjHandle 3-7
typedef PemnBDisconnectCallback 3-8
typedef PemnClientData 3-9
typedef PemnCommCallback 3-10
typedef PemnCommHandle 3-11
typedef PemnDumpEventHandle 3-12
typedef PemnEventHandle 3-13
typedef PemnEventStatusEnum 3-14
typedef PemnEventTypeEnum 3-15
typedef PemnGetApplObjHandler 3-16
typedef PemnGetEventClassHandler 3-17
typedef PemnGetEventClassListHandler 3-18
typedef PemnGetListHandler 3-20
typedef PemnGetParamObjHandler 3-21
typedef PemnGetVarObjHandler 3-22
typedef PemnInstAttrArray 3-23
typedef PemnInstAttrEnum 3-24
typedef PemnInstObjHandle 3-25
typedef PemnParamAttrArray 3-26
typedef PemnParamAttrEnum 3-27
typedef PemnParamObjHandle 3-28
typedef PemnQueryEventHandler 3-29
typedef PemnReceiveEventHandler 3-30
typedef PemnReportEventHandler 3-32
typedef PemnReportHandle 3-33
typedef PemnUserMessageHandler 3-34
typedef PemnVarObjHandle 3-35

U

Unicode string 13-9
unset() 13-45

BMC Software, Inc., Confidential and Proprietary Information

END USER LICENSE AGREEMENT
NOTICE

BY OPENING THE PACKAGE, INSTALLING, PRESSING "AGREE" OR "YES" OR USING THE PRODUCT, THE ENTITY OR INDIVIDUAL ENTERING INTO THIS AGREEMENT AGREES TO BE BOUND BY THE FOLLOWING TERMS. IF YOU DO NOT AGREE WITH ANY OF THESE TERMS, DO NOT INSTALL OR USE THE PRODUCT, PROMPTLY RETURN THE PRODUCT TO BMC OR YOUR BMC RESELLER, AND IF YOU ACQUIRED THE LICENSE WITHIN 30 DAYS OF THE DATE OF YOUR ORDER CONTACT BMC OR YOUR BMC RESELLER FOR A REFUND OF LICENSE FEES PAID. IF YOU REJECT THIS AGREEMENT, YOU WILL NOT ACQUIRE ANY LICENSE TO USE THE PRODUCT.

This Agreement ("**Agreement**") is between the entity or individual entering into this Agreement ("You") and BMC Software Distribution, Inc., a Delaware corporation located at 2101 CityWest Blvd., Houston, Texas, 77042, USA or its affiliated local licensing entity ("BMC"). "You" includes you and your Affiliates. "Affiliate" is defined as an entity which controls, is controlled by or shares common control with a party. IF MORE THAN ONE LICENSE AGREEMENT COULD APPLY TO THE PRODUCT, THE FOLLOWING ORDER OF LICENSE AGREEMENT PRECEDENCE APPLIES: (1) WEB BASED LICENSE AGREEMENT WITH BMC, (2) WRITTEN LICENSE AGREEMENT WITH BMC, (3) SHRINK-WRAP LICENSE AGREEMENT WITH BMC PROVIDED WITH THE PRODUCT, AND (4)THIS ELECTRONIC LICENSE AGREEMENT WITH BMC. In addition to the restrictions imposed under this Agreement, any other usage restrictions contained in the Product installation instructions or release notes shall apply to Your use of the Product.

PRODUCT AND CAPACITY. "Software" means the object code version of the computer programs provided, via delivery or electronic transmission, to You. Software includes computer files, enhancements, maintenance modifications, upgrades, updates, bug fixes, and error corrections.

"Documentation" means all written or graphical material provided by BMC in any medium, including any technical specifications, relating to the functionality or operation of the Software.

"Product" means the Software and Documentation.

"License Capacity" means the licensed capacity for the Software with the pricing and other license defining terms, including capacity restrictions, such as tier limit, total allowed users, gigabyte limit, quantity of Software, and/or other capacity limitations regarding the Software. For licenses based on the power of a computer, You agree to use BMC's current computer classification scheme, which is available at http://www.bmc.com or can be provided to You upon request.

ACCEPTANCE. The Product is deemed accepted by You, on the date that You received the Product from BMC.

LICENSE. Subject to the terms of this Agreement, as well as Your payment of applicable fees, BMC grants You a non-exclusive, non-transferable, perpetual (unless a term license is provided on an order) license for each copy of the Software, up to the License Capacity, to do the following:

(a) install the Software on Your owned or leased hardware located at a facility owned or controlled by You in the country where You acquired the license;

(b) operate the Software solely for processing Your own data in Your business operations; and

(c) make one copy of the Software for backup and archival purposes only (collectively a **"License"**).

If the Software is designed by BMC to permit you to modify such Software, then you agree to only use such modifications or new software programs for Your internal purposes or otherwise consistent with the License. BMC grants You a license to use the Documentation solely for Your internal use in Your operations.

LICENSE UPGRADES. You may expand the scope of the License Capacity only pursuant to a separate agreement with BMC for such expanded usage and Your payment of applicable fees. There is no additional warranty period or free support period for license upgrades.

RESTRICTIONS: You agree to **NOT**:

(a) disassemble, reverse engineer, decompile or otherwise attempt to derive any Software from executable code;

(b) distribute or provide the Software to any third party (including without limitation, use in a service bureau, outsourcing environment, or processing the data of third parties, or for rental, lease, or sublicense); or

(c) provide a third party with the results of any functional evaluation or benchmarking or performance tests, without BMC's prior written approval, unless prohibited by local law.

TRIAL LICENSE. If, as part of the ordering process, the Product is provided on a trial basis, then these terms apply: (i) this license consists solely of a non-exclusive, non-transferable evaluation license to operate the Software for the period of time specified from BMC or, if not specified, a 30 day time period (**"Trial Period"**) only for evaluating whether You desire to acquire a capacity-based license to the Product for a fee; and (ii) Your use of the Product is on an AS IS basis without any warranty, and **BMC, ITS AFFILIATES AND RESELLERS, AND LICENSORS DISCLAIM ANY AND ALL WARRANTIES (INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT) AND HAVE NO LIABILITY WHATSOEVER RESULTING FROM THE USE OF THIS PRODUCT UNDER THIS TRIAL LICENSE ("Trial License")**. BMC may terminate for its convenience a Trial License upon notice to You. When the Trial Period ends, Your right to use this Product automatically expires. If You want to continue Your use of the Product beyond the Trial Period, contact BMC to acquire a capacity-based license to the Product for a fee.

TERMINATION. This Agreement shall immediately terminate if You breach any of its terms. Upon termination, for any reason, You must uninstall the Software, and either certify the destruction of the Product or return it to BMC.

OWNERSHIP OF THE PRODUCT. BMC or its Affiliates or licensors retain all right, title and interest to and in the BMC Product and all intellectual property, informational, industrial property and proprietary rights therein. BMC neither grants nor otherwise transfers any rights of ownership in the BMC Product to You. Products are protected by applicable copyright, trade secret, and industrial and intellectual property laws. BMC reserves any rights not expressly granted to You herein.

CONFIDENTIAL AND PROPRIETARY INFORMATION. The Products are and contain valuable confidential information of BMC (**"Confidential Information"**). Confidential Information means non-public technical and non-technical information relating to the Products and Support, including, without limitation, trade secret and proprietary information, and the structure and organization of the Software. You may not disclose the Confidential Information to third parties. You agree to use all reasonable efforts to prevent the unauthorized use, copying, publication or dissemination of the Product.

WARRANTY. Except for a Trial License, BMC warrants that the Software will perform in substantial accordance with the Documentation for a period of one year from the date of the order. This warranty shall not apply to any problems caused by software or hardware not supplied by BMC or to any misuse of the Software.

EXCLUSIVE REMEDY. BMC's entire liability, and Your exclusive remedy, for any defect in the Software during the warranty period or breach of the warranty above shall be limited to the following: BMC shall use reasonable efforts to remedy defects covered by the warranty or replace the defective Software within a reasonable period of time, or if BMC cannot remedy or replace such defective copy of the Software, then BMC shall refund the amount paid by You for the License for that Software. BMC's obligations in this section are conditioned upon Your providing BMC prompt access to the affected Software and full cooperation in resolving the claim.

DISCLAIMER. EXCEPT FOR THE EXPRESS WARRANTIES ABOVE, THE PRODUCT IS PROVIDED "AS IS." BMC, ITS AFFILIATES AND LICENSORS SPECIFICALLY DISCLAIM ALL OTHER WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. BMC DOES NOT WARRANT THAT THE OPERATION OF THE SOFTWARE WILL BE UNINTERRUPTED OR ERROR FREE, OR THAT ALL DEFECTS CAN BE CORRECTED.

DISCLAIMER OF DAMAGES. IN NO EVENT IS BMC, ITS AFFILIATES OR LICENSORS LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, PUNITIVE OR CONSEQUENTIAL DAMAGES RELATING TO OR ARISING OUT OF THIS AGREEMENT, SUPPORT, AND/OR THE PRODUCT (INCLUDING, WITHOUT LIMITATION, LOST PROFITS, LOST COMPUTER USAGE TIME, AND DAMAGE OR LOSS OF USE OF DATA), EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND IRRESPECTIVE OF ANY NEGLIGENCE OF BMC OR WHETHER SUCH DAMAGES RESULT FROM A CLAIM ARISING UNDER TORT OR CONTRACT LAW.

LIMITS ON LIABILITY. BMC'S AGGREGATE LIABILITY FOR DAMAGES IS LIMITED TO THE AMOUNT PAID BY YOU FOR THE LICENSE TO THE PRODUCT.

SUPPORT. If Your order includes support for the Software, then BMC agrees to provide support (24 hours a day/7 days a week) (**"Support"**). You will be automatically re-enrolled in Support on an annual basis unless BMC receives notice of termination from You as provided below. There is a free support period during the one year warranty period.

(a) **Support Terms.** BMC agrees to make commercially reasonable efforts to provide the following Support: (i) For malfunctions of supported versions of the Software, BMC provides bug fixes, patches or workarounds in order to cause that copy of the Software to operate in substantial conformity with its then-current operating specifications; and (ii) BMC provides new releases or versions, so long as such new releases or versions are furnished by BMC to all other enrolled Support customers without additional charge. BMC may refuse to provide Support for any versions or releases of the Software other than the most recent version or release of such Software made available by BMC. Either party may terminate Your enrollment in Support upon providing notice to the other at least 30 days prior to the next applicable Support anniversary date. If You re-enroll in Support, BMC may charge You a reinstatement fee of 1.5 times what You would have paid if You were enrolled in Support during that time period.

(b) **Fees.** The annual fee for Support is 20% of the Software's list price less the applicable discount or a flat capacity based annual fee. BMC may change its prices for the Software and/or Support upon at least 30 days notice prior to Your support anniversary date.

VERIFICATION. If requested by BMC, You agree to deliver to BMC periodic written reports, whether generated manually or electronically, detailing Your use of the Software in accordance with this Agreement, including, without limitation, the License Capacity. BMC may, at its expense, perform an audit, at your facilities, of Your use of the Software to confirm Your compliance with the Agreement. If an audit reveals that You have underpaid fees, You agree to pay such underpaid fees. If the underpaid fees exceed 5% of the fees paid, then You agree to also pay BMC's reasonable costs of conducting the audit.

EXPORT CONTROLS. You agree not to import, export, re-export, or transfer, directly or indirectly, any part of the Product or any underlying information or technology except in full compliance with all United States, foreign and other applicable laws and regulations.

GOVERNING LAW. This Agreement is governed by the substantive laws in force, without regard to conflict of laws principles: (a) in the State of New York, if you acquired the License in the United States, Puerto Rico, or any country in Central or South America; (b) in the Province of Ontario, if you acquired the License in Canada (subsections (a) and (b) collectively referred to as the **"Americas Region"**); (c) in Singapore, if you acquired the License in Japan, South Korea, Peoples Republic of China, Special Administrative Region of Hong Kong, Republic of China, Philippines, Indonesia, Malaysia, Singapore, India, Australia, New Zealand, or Thailand (collectively, **"Asia Pacific Region"**); or (d) in the Netherlands, if you acquired the License in any other country not described above. The United Nations Convention on Contracts for the International Sale of Goods is specifically disclaimed in its entirety.

ARBITRATION. ANY DISPUTE BETWEEN YOU AND BMC ARISING OUT OF THIS AGREEMENT OR THE BREACH OR ALLEGED BREACH, SHALL BE DETERMINED BY BINDING ARBITRATION CONDUCTED IN ENGLISH. IF THE DISPUTE IS INITIATED IN THE AMERICAS REGION, THE ARBITRATION SHALL BE HELD IN NEW YORK, U.S.A., UNDER THE CURRENT COMMERCIAL OR INTERNATIONAL, AS APPLICABLE, RULES OF THE AMERICAN ARBITRATION ASSOCIATION. IF THE DISPUTE IS INITIATED IN A COUNTRY IN THE ASIA PACIFIC REGION, THE ARBITRATION SHALL BE HELD IN SINGAPORE, SINGAPORE UNDER THE CURRENT UNCITRAL ARBITRATION RULES. IF THE DISPUTE IS INITIATED IN A COUNTRY OUTSIDE OF THE AMERICAS REGION OR ASIA PACIFIC REGION, THE ARBITRATION SHALL BE HELD IN AMSTERDAM, NETHERLANDS UNDER THE CURRENT UNCITRAL ARBITRATION RULES. THE COSTS OF THE ARBITRATION SHALL BE BORNE EQUALLY PENDING THE ARBITRATOR'S AWARD. THE AWARD RENDERED SHALL BE FINAL AND BINDING UPON THE PARTIES AND SHALL NOT BE SUBJECT TO APPEAL TO ANY COURT, AND MAY BE ENFORCED IN ANY COURT OF COMPETENT JURISDICTION. NOTHING IN THIS AGREEMENT SHALL BE DEEMED AS PREVENTING EITHER PARTY FROM SEEKING INJUNCTIVE RELIEF FROM ANY COURT HAVING JURISDICTION OVER THE PARTIES AND THE SUBJECT MATTER OF THE DISPUTE AS NECESSARY TO PROTECT EITHER PARTY'S CONFIDENTIAL INFORMATION, OWNERSHIP, OR ANY OTHER PROPRIETARY RIGHTS. ALL ARBITRATION PROCEEDINGS SHALL BE CONDUCTED IN CONFIDENCE, AND THE PARTY PREVAILING IN ARBITRATION SHALL BE ENTITLED TO RECOVER ITS REASONABLE ATTORNEYS' FEES AND NECESSARY COSTS INCURRED RELATED THERETO FROM THE OTHER PARTY.

U.S. GOVERNMENT RESTRICTED RIGHTS. The Software under this Agreement is "commercial computer software" as that term is described in 48 C.F.R. 252.227-7014(a)(1). If acquired by or on behalf of a civilian agency, the U.S. Government acquires this commercial computer software and/or commercial computer software documentation subject to the terms of this Agreement as specified in 48 C.F.R. 12.212 (Computer Software) and 12.211 (Technical Data) of the Federal Acquisition Regulations (**"FAR"**) and its successors. If acquired by or on behalf of any agency within the Department of Defense (**"DOD"**), the U.S. Government acquires this commercial computer software and/or commercial computer software documentation subject to the terms of this Agreement as specified in 48 C.F.R. 227.7202 of the DOD FAR Supplement and its successors.

MISCELLANEOUS TERMS. You agree to pay BMC all amounts owed no later than 30 days from the date of the applicable invoice, unless otherwise provided on the order for the License to the Products. You will pay, or reimburse BMC, for taxes of any kind, including sales, use, duty, tariffs, customs, withholding, property, value-added (VAT), and other similar federal, state or local taxes (other than taxes based on BMC's net income) imposed in connection with the Product and/or the Support. This Agreement constitutes the entire agreement between You and BMC and supersedes any prior or contemporaneous negotiations or agreements, whether oral, written or displayed electronically, concerning the Product and related subject matter. No modification or waiver of any provision hereof will be effective unless made in a writing signed by both BMC and You. You may not assign or transfer this Agreement or a License to a third party without BMC's prior written consent. Should any provision of this Agreement be invalid or unenforceable, the remainder of the provisions will remain in effect. The parties have agreed that this Agreement and the documents related thereto be drawn up in the English language. Les parties exigent que la présente convention ainsi que les documents qui s'y rattachent soient rédigés en anglais.

Notes

23145