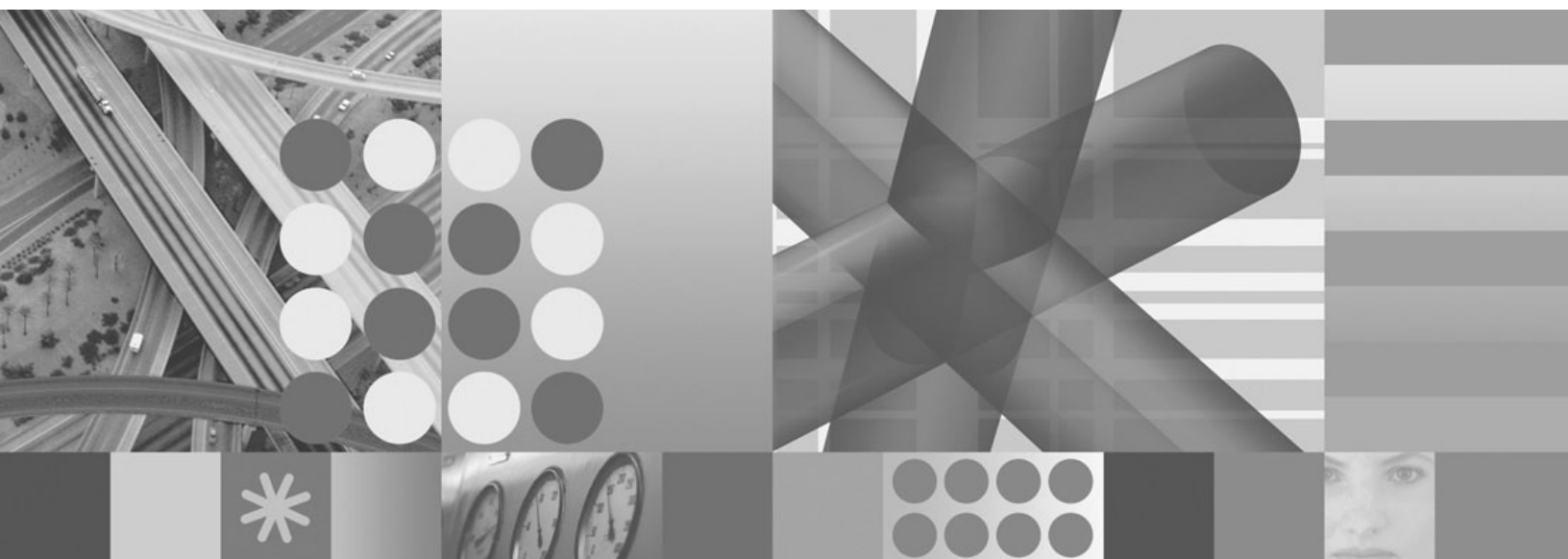




API and Command Programming Reference Guide



API and Command Programming Reference Guide

Note

Before using this information and the product it supports, read the information in Appendix B, "Notices," on page 71.

This edition applies to version 6, release 2, modification 0 of IBM Tivoli Monitoring (product number 5724-C04) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 2003, 2007. All rights reserved.**

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

Contents

Figures	v
Tables	vii
About this publication	ix
Intended audience	ix
Publications	ix
IBM Tivoli Monitoring library	ix
IBM Tivoli Open Process Automation Library (OPAL)	x
Accessing terminology online	x
Accessing publications online	x
Ordering publications	xi
Accessibility	xi
Tivoli technical training	xi
Support information	xi
Conventions used in this guide	xi
Typeface conventions	xii
Operating system-dependent variables and paths	xii
Chapter 1. API introduction	1
Application programming interfaces	1
Implementing the Tivoli Universal Agent APIs	1
Collecting Data: Interval-driven or demand-driven	2
The API client package	2
Invoking the APIs	3
Using program function calls	3
Using script file calls	3
Using manual commands	3
Specifying API Server Data Provider variables	3
Name and location of variables file	3
Specifying the host of the API Server Data Provider	4
Specifying the listening port for the API Server Data Provider	4
Specifying metafiles to the API Server Data Provider	4
Using a data provider configuration file	4
Using a console interface command	4
Using a program API call	4
Determining if the API Server Data Provider is operational	5
Using the verbose trace option	5
Bypassing API parameter validation	5
Chapter 2. API descriptions	7
Using the Tivoli Universal Agent API functions	7
Minimum program requirements	7
Starting a connection with the data provider (optional)	7
Handling applications with multiple attribute groups	7
Handling multiple row data input	8
dp_AcceptRequest	10
dp_AllocateBuffer	12
dp_AllocateHandle	13
dp_BeginInput	14
dp_BeginSample	16
dp_CancelRequest	17
dp_ClearBuffer	18

dp_ClearPriorData	19
dp_CloseSession	20
dp_Define	21
dp_EndInput	22
dp_EndSample	24
dp_FormatBufferData	25
dp_FreeBuffer.	27
dp_FreeHandle	28
dp_InputData	29
dp_OpenSession	31
dp_Ping	32
dp_ReceiveRequest	33
dp_Redefine	37
dp_ResetHandle	38
dp_ReturnData	39
dp_SendRequestResult	41
dp_SetSourceName	43
dp_ShowMessage	45
 Chapter 3. Console commands.	 47
Using API console commands	47
Command syntax	47
Minimum required command-line input.	47
KUMPAREQ (dp_AcceptRequest)	48
KUMPBGNI (dp_BeginInput)	49
KUMPDEFN (dp_Define).	51
KUMPENDI (dp_EndInput)	52
KUMPINPT (dp_InputData)	53
KUMPPING (dp_Ping).	55
KUMPRDFN (dp_Redefine).	56
KUMPRREQ (dp_ReceiveRequest)	57
KUMPRTDA (dp_ReturnData)	58
KUMPSHOW (dp_ShowMessage)	60
 Appendix A. Sample programs	 61
About the sample C program	61
APIs called in sample program	61
The RoutStat sample C program	62
About the sample shell script	66
The Rout_Cmd shell script	66
About the sample VBScript	67
The NetworkDrives sample VBScript	67
 Appendix B. Notices	 71
Trademarks	72
 Index	 73

Figures

1.	Components of the Tivoli Universal Agent API	1
2.	Sample program for combining multiple application data in batches	30
3.	The metafile RTstat.mdl	61
4.	The metafile RTstat.mdl	66
5.	The metafile NWDrives.mdl	67

Tables

1.	API client package.	2
2.	Name and location of variables file by operating system	4
3.	Tivoli Universal Agent APIs	8
4.	Tivoli Universal Agent API console commands	47

About this publication

The IBM® Tivoli® Universal Agent application programming interfaces (APIs) enable you to develop programs that send data to the API Server Data Provider for monitoring by IBM Tivoli Monitoring. The *IBM Tivoli Universal Agent API and Command Programming Reference Guide* contains the procedures for implementing the APIs and provides descriptions, syntax, and return status codes for the API calls and command-line interface commands.

Intended audience

This guide is designed for programmers who develop system monitoring applications. This guide assumes that you have performed the following:

- Installed the Tivoli Enterprise Portal and are familiar with the concepts, tasks, and features of the Tivoli Enterprise Portal.
- Installed the Tivoli Universal Agent and are familiar with the contents of the *IBM Tivoli Universal Agent User's Guide*.

If you are unfamiliar with these concepts and tasks, review the *IBM Tivoli Monitoring Administrator's Guide*. Additionally, review the online Tivoli Enterprise Portal Tour to become familiar with the features and capabilities of the product.

Publications

This section lists publications in the Tivoli Universal Agent library. It also describes how to access Tivoli publications online and how to order Tivoli publications.

IBM Tivoli Monitoring library

The following documents provide information about the IBM Tivoli Monitoring product:

- *Introducing IBM Tivoli Monitoring*, GI11-4071
Introduces the components of IBM Tivoli Monitoring and also compares concepts and function of IBM Tivoli Monitoring to Tivoli Distributed Monitoring.
- *IBM Tivoli Monitoring: Upgrading from Tivoli Distributed Monitoring*, GC32-9462
Provides information on how to upgrade from Tivoli Distributed Monitoring.
- *IBM Tivoli Monitoring Installation and Setup Guide*, GC32-9407
Provides information about installing and setting up IBM Tivoli Monitoring and upgrading from OMEGAMON® V350 or V360.
- *IBM Tivoli Monitoring User's Guide*, SC32-9409
Complements the Tivoli Enterprise™ Portal online help. The guide provides hands-on lessons and detailed instructions for all Tivoli Enterprise Portal features.
- *IBM Tivoli Monitoring Administrator's Guide*, SC32-9408
Describes the support tasks and functions required for the Tivoli Enterprise Portal Server and clients, including Tivoli Enterprise Portal user administration.
- *IBM Tivoli Monitoring Services on z/OS Program Directory*
Provides installation information for installing a Tivoli Enterprise Monitoring Server on a z/OS® system.
- *Configuring IBM Tivoli Enterprise Monitoring Server on z/OS*, SC32-9463
Describes how to configure and customize the Tivoli Enterprise Monitoring Server running on a z/OS system.
- *IBM Tivoli Monitoring Problem Determination Guide*, GC32-9458
Provides information and messages to assist users with troubleshooting problems with the software.

- *Exploring IBM Tivoli Monitoring*, SC32-1803

Provides a series of exercises that help users explore IBM Tivoli Monitoring. After completing the activities in this workbook, you will have installed and configured your environment, explored both the graphical and command-line interfaces of the product, used several of the new features in this release of IBM Tivoli Monitoring (such as creating a monitor based on a custom script), and worked with several monitoring agents.

- *IBM Tivoli Universal Agent User's Guide*, SC32-9459

Introduces you to the IBM Tivoli Universal Agent, an agent of IBM Tivoli Monitoring. The IBM Tivoli Universal Agent enables you to use the monitoring and automation capabilities of IBM Tivoli Monitoring to monitor any type of data you collect.

- *IBM Tivoli Universal Agent API and Command Programming Reference Guide*, SC32-9461

Explains the procedures for implementing the IBM Tivoli Universal Agent APIs and provides descriptions, syntax, and return status codes for the API calls and command-line interface commands.

- *IBM Tivoli Monitoring: IBM Tivoli Monitoring 5.x Endpoint Agent User's Guide*

The IBM Tivoli Monitoring 5.x Endpoint Agent extends the capabilities of IBM Tivoli Monitoring V5.1.2. It enables data collected by deployed IBM Tivoli Monitoring V5.1.2 endpoints to be displayed in the Tivoli Enterprise Portal and stored in Tivoli Data Warehouse.

IBM Tivoli Open Process Automation Library (OPAL)

Additional information about Tivoli Universal Agent and other Tivoli products can be found at the IBM Tivoli Open Process Automation Library (OPAL) Web site, which contains many Tivoli Universal Agent monitoring solutions, white papers, and other information useful to Tivoli customers. The Web address is:
<http://catalog.lotus.com/wps/portal/tm>.

Accessing terminology online

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

<http://publib.boulder.ibm.com/tividd/glossary/tivologlossarymst.htm>

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

<http://www.ibm.com/software/globalization/terminology>

Accessing publications online

The documentation CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. Refer to the readme file on the CD for instructions on how to access the documentation.

The product CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. To access the publications using a Web browser, open the `infocenter.html` file. The file is in the appropriate publications directory on the product CD.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli Information Center Web site at <http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp>.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that allows Adobe® Reader to print letter-sized pages on your local paper.

Ordering publications

You can order many Tivoli publications online at the following Web site:

<http://www.elink.ibm.link.ibm.com/publications/servlet/pbi.wss>

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to <http://www.elink.ibm.link.ibm.com/public/applications/publications/cgibin/pbi.cgi>.
2. Select your country from the list and click **Go**.
3. Click **About this site** in the main panel to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate most features of the graphical user interface.

For additional information, see the Accessibility Appendix in the user's guide for this product.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

<http://www.ibm.com/software/tivoli/education/>

Support information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:

Online

Go to the IBM Software Support site at <http://www.ibm.com/software/support/probsub.html> and follow the instructions.

IBM Support Assistant

The IBM Support Assistant (ISA) is a free local software serviceability workbench that helps you resolve questions and problems with IBM software products. The ISA provides quick access to support-related information and serviceability tools for problem determination. To install the ISA software, go to <http://www.ibm.com/software/support/isa>.

Problem determination guidance

For more information about resolving problems, see the problem determination information for this product.

Conventions used in this guide

This guide uses several conventions for special terms and actions, and operating system-dependent commands and paths.

Typeface conventions

This guide uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (for example, "Use the word *that* to introduce a restrictive clause.")
- New terms in text (except in a definition list)
- Variables and values you must provide

Monospace

- Code and other examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

The publications in this library use the UNIX[®] convention for specifying environment variables and for directory notation.

When using the Windows[®] command line, replace `$variable` with `%variable%` for environment variables and replace each forward slash (/) with a backslash (\) in directory paths. The names of environment variables are not always the same in Windows and UNIX environments. For example, `%TEMP%` in Windows environments is equivalent to `$TMPDIR` in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. API introduction

This chapter provides an introduction to the following Tivoli Universal Agent application programming interface concepts:

- Implementing APIs
- Invoking APIs
- Specifying transport mode between client and server
- Defining metafiles to an API Server Data Provider

Application programming interfaces

The Tivoli Universal Agent Application Programming Interfaces (APIs) provide the basic tools for developing programs that collect, format, and send user-defined data to the Tivoli Universal Agent. Once the data has been received by the Tivoli Universal Agent, you can use the data with all standard IBM Tivoli Monitoring features, such as situation evaluation, real-time reporting, automation, and history.

This chapter provides a brief description of how the APIs are implemented. It also discusses how to invoke the API functions, how to specify environment variables for the API Server Data Provider, and how to specify which API metafiles that you want it to load.

Implementing the Tivoli Universal Agent APIs

C/C++ calling programs using the Tivoli Universal Agent API functions rely upon services provided by the Tivoli Universal Agent runtime dynamic library. These services are commonly referred to as the API client package and the calling programs are referred to as API client programs.

The implementation of the Tivoli Universal Agent API client package varies among operating systems. Generally speaking, however, the client side of the API package is constructed as a dynamic link library (DLL) or UNIX shared library that exports API functions.

You can use the API client package to build simple or complex programs that collect and send data to the API Server Data Provider. The API Server Data Provider functions as a server daemon running inside the Tivoli Universal Agent process that listens for client connections and responds to client requests. API client programs can reside on the same host as the API Server Data Provider or they can run at a remote location. Figure 1 on page 1 illustrates the relationships among the components.

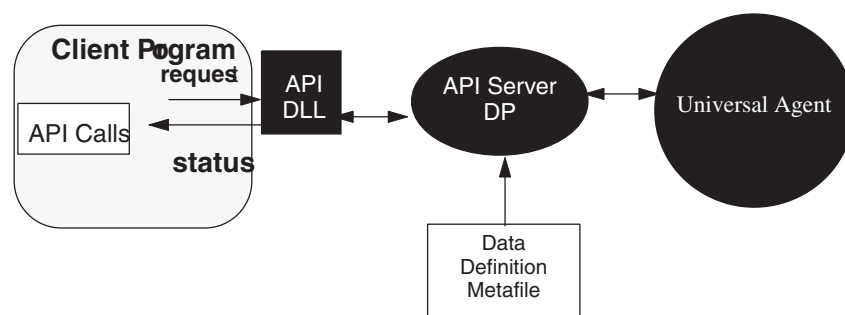


Figure 1. Components of the Tivoli Universal Agent API

The Tivoli Universal Agent also supports a command-line interface that allows you to invoke a subset of the API calls through console commands.

Collecting Data: Interval-driven or demand-driven

Universal Agent has traditionally relied on interval-driven data collection (also referred to as sample-driven). Data providers collect data at fixed intervals for each application table. In this mode, all report, history, and situation processing uses the most recently collected data, which is cached in memory.

The API Server Data Provider supports *interval-driven* and *demand-driven* data collection. Collecting data on demand offers newer data and potentially reduces unnecessary processing overhead.

Note: The Tivoli Enterprise Portal client workspace API application refreshes take longer to complete because the report request is sent to the API client program where the data rows are collected and sent back to the API Server Data Provider. This might take several seconds to complete. With interval-driven collection, the Tivoli Enterprise Portal client workspace refresh completes faster because the data rows are retrieved from the Tivoli Universal Agent memory, where the data has already been collected and cached, without initiating a report request to the API client program.

The API client package

The API client package requires only a common C runtime environment and TCP/IP with a socket interface. The client package consists of the following:

- A library containing the binary executables of the API functions.
- A C header file.
- A set of command-line interface programs.

To use the API Server Data Provider, you need to install the API client package on the same machine where the client program will run, as well as on the computer where you compile the client program if they are not the same.

To install the client package:

1. Navigate to one of the following directories:
 - For Windows operating systems, \IBM\ITM\TMAITM6
 - For UNIX operating systems, /\$install_dir/\$ARCH/um/binwhere:

\$ARCH

Specifies the platform architecture. For example: aix516, li6243, or sol283.

2. Copy the files listed in Table 1 to the target machine directory where you want to compile and run your API client program.

Table 1. API client package

Part name	Description
KUMPAPI.h	API C header file
KUMPAPI.lib (Windows operating systems only)	API function export library file
KUMPAPI.dll or libkumpapi	API function runtime dynamic link library
KUMPAREQ	dp_AcceptRequest console command
KUMPBGNI	dp_BeginInput console command
KUMPDEFN	dp_Define console command
KUMPENDI	dp_EndInput console command
KUMPINPT	dp_InputData console command
KUMPPING	dp_Ping console command
KUMPRDFN	dp_Redefine console command

Table 1. API client package (continued)

Part name	Description
KUMPRREQ	dp_ReceiveRequest console command
KUMPRTDA	dp_ReturnData console command
KUMPSHOW	dp_ShowMessage console command

Note: For UNIX operating systems, the libkumpapi shared library is located in the `/$install_dir/tmaitm6/$ARCH/lib` directory.

Invoking the APIs

You can invoke the Tivoli Universal Agent API functions in any of the following ways:

- Program function calls
- Script file calls
- Manual commands

Using program function calls

You can develop or modify C/C++ programs that invoke the APIs directly as subroutines. The API functions are thread-safe, but they are synchronous in nature. Therefore, a program's calling thread is blocked until the API call is returned.

Using script file calls

You can develop or modify a script or batch file that calls the API command-line interface programs. The script file provides a degree of automation for commonly used processing procedures. However, the API command-line interface implements only a subset of the API suite because of the discontinuous nature of the command interface.

Using manual commands

You can enter API commands directly from a system console. This method is particularly useful because it allows you to send data or create events whenever the need arises. For example, a command might be issued in response to customer support receiving an urgent call from a critical customer site. The customer service representative enters a KUMPINPT API command with message text. This would cause a previously defined IBM Tivoli Monitoring situation to become true and the on-duty action team to be notified.

Specifying API Server Data Provider variables

The following sections provide details for specifying variables for the API Server Data Provider.

Name and location of variables file

Environment variables for the API Server Data Provider all begin with the string KUMP_API. Unless otherwise specified, the default values of the variables are in force. To change the default values, you must enter the appropriate variable and the desired value in a variables file. The name and location of the variables file differs by operating system.

Table 2 contains the name and location of the variables file on each of the supported operating systems. Consult Appendix D of the *Tivoli Universal Agent User's Guide* for further information on Tivoli Universal Agent variables.

Table 2. Name and location of variables file by operating system

Operating system	Location	Name/Member
UNIX	\$install_dir/config/	um.ini
Windows	IBM\ITM\TMAITM6\	KUMENV

Specifying the host of the API Server Data Provider

The default mode of the API client assumes that the API Server Data Provider resides on the same system. If the API Server Data Provider is running on a remote system, you must set the environment variable `KUMP_API_DPAPI_HOST` to the host name of the API Server Data Provider. You can set it in the program or login script of the client host, or as part of the workstation's configuration.

Specifying the listening port for the API Server Data Provider

The default listening port for the API Server Data Provider is 7600. If this port is already in use, or you would prefer that a different port be used, you can set the environment variable `KUMP_API_DPAPI_PORT` to the preferred port. If you set this variable for the API Server Data Provider, you must set the same variable value on the API client side so that the client can successfully connect to the listening port.

If an alternate instance of Universal Agent has been started, port 7600 will not be in effect. You can determine the API listening port being used for the alternate instance from the API Server Data Provider UAGENT DPLOG report. You would then specify that value in the `KUMP_API_DPAPI_PORT` environment variable on the API client side.

Specifying metafiles to the API Server Data Provider

The data provider must load the appropriate data definition metafiles in order to support application data. The following are the three methods of defining the application metafile to the data provider:

- Configuration file
- Console command
- Program API call

Using a data provider configuration file

At startup, the data provider looks for a configuration file named `KUMPCNFG`. `KUMPCNFG` is a free-form text file that contains a list of metafile names. (On case-sensitive systems, the file name `KUMPCNFG` must be all uppercase.)

If it discovers the `KUMPCNFG` file, the data provider automatically loads and validates the data definitions in the metafiles listed in `KUMPCNFG`. A `KUMPCNFG` file can be shared by multiple data providers.

Consult the *Tivoli Universal Agent User's Guide* for information on the configuration file.

Using a console interface command

An application data definition can be added or updated dynamically at data provider runtime. You can use the Tivoli Universal Agent console interface command `IMPORT` to load a new metafile or the command `REFRESH` to revise an existing metafile.

Using a program API call

A calling program can control the activation of a metafile in a manner similar to the console commands. The program calls `dp_Define`, requesting that the API Server Data Provider load a specific metafile. It calls `dp_Redefine` to refresh a currently active metafile. The calling program can always include `dp_Define` calls in its logic so that it has total control over its operating environment.

Determining if the API Server Data Provider is operational

The quickest way to determine the status of the API Server Data Provider is by calling `dp_Ping` or the equivalent command-line interface program, `KUMPPING`. If either returns `FALSE`, the server is not active, and there is no need to proceed further with any API procedure.

Using the verbose trace option

The verbose runtime option acts like a trace facility that enables you to perform problem determination when using the Tivoli Universal Agent API. This service is turned on by the special environment variable `KUMP_API_VERBOSE=Yes/No`. If verbose is enabled, a detailed API flow trace is generated.

The output can be redirected to a file. For example,

```
SET KUMP_API_VERBOSE=Y ^>DPAPI.LOG
```

enables the verbose mode and writes the trace details to the file `DPAPI.LOG`.

Note that this variable must be set in the API client program's environment and not in the Universal Agent environment file.

If you want to supplement API tracing with your own program tracing, you can use the `dp_WriteVerboseMessage` API call, which has the following function signature:

```
int dp_WriteVerboseMessage(char *);
```

You pass this function a NULL-terminated string that contains additional tracing information to help with program diagnosis. The string you pass to `dp_WriteVerboseMessage` is incorporated into the API verbose trace along with all of the Tivoli Universal Agent-provided trace messages.

The `dp_WriteVerboseMessage` function always returns a 0 status code. If you call this function when API verbose tracing is not activated, the function call is ignored.

Bypassing API parameter validation

Each API call requires a number of input parameters. On every call, the API routine must validate the presence and content of each parameter. The validation protects the integrity of the API runtime routine and guards against unacceptable input values and runtime exceptions that can affect the API client program.

The API validation consumes limited processing overhead. However, even this overhead can be avoided when the client program has been thoroughly tested and there is little danger to bypassing parameter validation checks.

To bypass validation checks, set the environment variable `KUMP_API_BYPASS_VAL` to `YES`.

Chapter 2. API descriptions

This chapter provides syntax, semantics, descriptions, and return codes for the Tivoli Universal Agent APIs. It assumes familiarity with the discussion of the API Server Data Provider provided in the *IBM Tivoli Universal Agent User's Guide*.

Using the Tivoli Universal Agent API functions

This section discusses the programming requirements for contacting a data provider and formatting and sending data. The API calls are summarized in Table 3 on page 8.

Minimum program requirements

At a minimum, a C/C++ program needs to implement the following procedure to communicate with the API Server Data Provider:

1. Call `dp_AllocateHandle` to allocate a data provider handle.
2. Call `dp_AllocateBuffer` to allocate a data provider data buffer.
3. Call `dp_BeginInput` to identify the application and attribute group of interest.
4. Repeatedly call `dp_FormatBufferData` to move application data into the data provider buffer.
5. Call `dp_InputData` to send the application data record to the API Server.
6. Call `dp_ClearBuffer` to reset the buffer content.
7. Repeat Steps 4, 5, and 6 as long as data is available.
8. Call `dp_EndInput` to complete the application input procedure.
9. Call `dp_FreeBuffer` to release buffer storage.
10. Call `dp_FreeHandle` to free the buffer and handle storage.

Starting a connection with the data provider (optional)

The function `dp_OpenSession` establishes a connection with the data provider. Establishing a connection is not a requirement since, by default, the API Server Data Provider uses a connectionless model. However, by establishing a connection, you avoid the overhead required for connectionless communication management for the life of the application data exchange.

To start a connection:

1. Call `dp_AllocateHandle` to allocate a data provider handle.
2. Call `dp_OpenSession` to establish data provider connectivity.

Check the returned status code to determine if the connection was made successfully. Use the data provider handle for the remaining data exchange.

To close the connection:

1. Call `dp_CloseSession` to end data provider connectivity.
2. Call `dp_FreeHandle` to release the allocated handle.

Handling applications with multiple attribute groups

A data provider handle identifies a single attribute group. Therefore, you need multiple data provider handles for an application that contains multiple attribute groups. Since data provider buffers are owned by handles, data provider buffers cannot be shared among attribute groups.

To input data for an application with multiple attribute groups:

1. Call `dp_AllocateHandle` to allocate a data provider handle.

2. Call `dp_AllocateBuffer` to allocate a data provider data buffer.
3. Call `dp_BeginInput` to identify the application and attribute group of interest.
4. Repeat Steps 1, 2, and 3 for each attribute group.
5. Repeatedly call `dp_FormatBufferData` to move application data into the data provider buffer.
6. Call `dp_InputData` to send an application data record to API Server Data Provider.
7. Call `dp_ClearBuffer` to reset the buffer content.
8. Repeat Steps 5, 6, and 7 as long as data is available.
9. Repeat Steps 5, 6, 7, and 8 for each attribute group.
10. Call `dp_EndInput` to complete the application input procedure.
11. Call `dp_FreeBuffer` to release buffer storage.
12. Call `dp_FreeHandle` to free buffer and handle storage.
13. Repeat Steps 10, 11, and 12 for each attribute group.

Handling multiple row data input

A typical data source can consist of multiple data records. The calls `dp_BeginSample` and `dp_EndSample` act as brackets surrounding a group of data records.

To handle a multiple row sample after a data provider handle has been allocated:

1. Call `dp_BeginSample` to identify the beginning of a multiple row sample.
2. Repeatedly call `dp_FormatBufferData` to move application data into the buffer.
3. Call `dp_InputData` to send application data record to the data provider.
4. Call `dp_ClearBuffer` to reset the contents of the buffer.
5. Repeat Steps 2, 3, and 4 for the next data record.
6. Call `dp_EndSample` to mark the end of the multiple row sample.
7. Repeat Steps 1 through 6 for the next sample.

Table 3. Tivoli Universal Agent APIs

Name	Description
Data Definition	
<code>dp_Define</code>	Defines a metafile to the data provider
<code>dp_Redefine</code>	Refreshes a metafile
<code>dp_SetSourceName</code>	Associates an unique source name with an application attribute group for data input
Session	
<code>dp_OpenSession</code>	Initiates a connection between a client program and the API Server Data Provider
<code>dp_CloseSession</code>	Ends the connection between a client program and the API Server Data Provider
Data Formatting	
<code>dp_AllocateBuffer</code>	Allocates a data transport buffer
<code>dp_FreeBuffer</code>	Releases an allocated data transport buffer
<code>dp_ClearBuffer</code>	Clears an allocated buffer of application data and control information
<code>dp_FormatBufferData</code>	Formats a data field in an allocated buffer for transmission
<code>dp_BeginSample</code>	Indicates the beginning of a multiple row data sample
<code>dp_EndSample</code>	Indicates the completion of a multiple row data sample
Data Transfer	

Table 3. Tivoli Universal Agent APIs (continued)

Name	Description
dp_AllocateHandle	Allocates a handle for data exchange with a data provider
dp_FreeHandle	Frees a previously allocated handle
dp_ResetHandle	Resets the status of an allocated handle for reuse
dp_BeginInput	Identifies an attribute group for data input
dp_EndInput	Indicates the completion of data input for an attribute group
dp_InputData	Sends the data buffer to the API Server Data Provider
dp_ClearPriorData	Clears the Tivoli Universal Agent cache of any previously collected data for the attribute group
Status	
dp_Ping	Checks the operational status of the API Server Data Provider
dp_ShowMessage	Displays the text of an API return code status message
Request Activity	
dp_AcceptRequest	Indicates that an application program is willing and able to accept requests from a data provider as the result of IBM Tivoli Monitoring automation actions or activities
dp_CancelRequest	Signals the API Server Data Provider that there are no data rows to return in response to a report or situation request.
dp_ReceiveRequest	Receives request from the data provider
dp_ReturnData	Sends the data buffer to the API Server Data Provider in response to a report or situation request received by the dp_ReceiveRequest API.
dp_SendRequestResult	Sends result and status of an automation action request to the data provider.

dp_AcceptRequest

Description

dp_AcceptRequest allows a client program to indicate its ability to accept inbound requests from the data provider that arise as the result of IBM Tivoli Monitoring automation actions or activities. This call is valid only after the application program has completed the dp_BeginInput API call and an association between an API handle and an application attribute group has been established.

For programs that support multiple attribute groups simultaneously, a dp_AcceptRequest call is required for each attribute group that supports inbound requests. If all the supported attribute groups are capable of accepting inbound requests, one dp_AcceptRequest call with a NULL handle accomplishes the same result as a separate dp_AcceptRequest call for each attribute group.

The dp_AcceptRequest API must be used for a client program to receive a request from the API Server Data Provider. It must be completed successfully before any dp_ReceiveRequest, dp_ReturnData, dp_SendRequestResult, or dp_CancelRequest API calls.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_AcceptRequest(dp_handle_t InHandle, int EnableRequest, int ReplyWaitTime, int* Call_Status)
```

Input parameters

The following are required dp_AcceptRequest input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the dp_AllocateHandle API call. If the API handle is NULL, this API call affects all handles allocated by the client program.

int EnableRequest

A value of KUMP_API_TRUE indicates that the client program is ready to accept inbound requests. A value of KUMP_API_REPORT (a superset of KUMP_API_TRUE) indicates that the client program is ready to accept report and situation requests. A value of KUMP_API_FALSE indicates that the caller wants to stop accepting inbound requests from the data provider.

int ReplyWaitTime

Specifies the maximum elapsed time, in seconds, that the data provider should wait for the client program to send back the request execution result using the dp_SendRequestResult API call. If the ReplyWaitTime is zero, the default wait time of 60 seconds is assumed. If the program does not support or does not want to send back any request execution result, use the value KUMP_API_PROVIDE_NO_RESULT.

*int **Status

A pointer to an integer storage variable that dp_AcceptRequest uses for storing the API return status code.

Return status code

dp_AcceptRequest returns TRUE if the function is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_AcceptRequest calls:

KUMP_API_OK

dp_AcceptRequest completed successfully.

KUMP_API_Invalid_Handle

The input handle pointer is not valid or is not a valid handle pointer allocated by the `dp_AllocateHandle` API call.

dp_AllocateBuffer

Description

dp_AllocateBuffer allocates a data buffer and associates it with the input handle. The client program must use this API call to obtain a buffer for data output. A client program can use the same handle to allocate as many buffers as needed. The allocated buffers are owned by that handle and cannot be shared among other handles.

The caller should not manipulate buffer contents directly. It must use dp_FormatBufferData to move data from program storage into the allocated buffer.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_Allocatebuffer(dp_handle_t InHandle, char **UserBuffer, int RequestSize, int *Status)
```

Input parameters

The following are required dp_AllocateBuffer input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the dp_AllocateHandle API call.

*char ***UserBuffer

A pointer to a pointer storage variable that contains the address of the allocated buffer, if dp_AllocateBuffer is successful. Otherwise, this pointer variable content is set to NULL.

int RequestSize

Required buffer size needed to accommodate the expected data. If a size of zero is given, dp_AllocateBuffer allocates a default buffer size of 2048 bytes for use by the client program.

*int **Status

A pointer to an integer storage variable that dp_AllocateBuffer uses for storing the API return status code.

Return status code

dp_AllocateBuffer returns TRUE if the buffer is allocated successfully. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_AllocateBuffer calls:

KUMP_API_OK

dp_AllocateBuffer completed successfully.

KUMP_API_AllocBuffer_Input_Parm_Invalid

*char ***UserBuffer is NULL or contains an address that is not valid.

KUMP_API_Invalid_Handle

The input handle pointer is not valid or it is not a valid handle pointer allocated by the dp_AllocateHandle API call.

KUMP_API_AllocBuffer_No_Storage

Not enough memory is available for the buffer storage. The RequestSize parameter is not valid, or insufficient heap storage remains for buffer allocation.

dp_AllocateHandle

Description

A data provider handle represents an application execution instance to the Tivoli Universal Agent data provider. `dp_BeginInput` associates a handle with an application and attribute group. All data provider session, data format, and data transfer API calls require handle ownership. Therefore, `dp_AllocateHandle` must usually be the first API call by the calling program.

A handle is valid for the duration of a program emitting data to the data provider on behalf of an application. A program can have many handles, since it can support multiple applications or multiple attribute groups within an application. However, the caller must be aware of the handle's association with the application, since handles cannot be shared or exchanged among applications and their subordinate attribute groups.

Use the `dp_ResetHandle` call to clear the handle and application relationship before reusing it for another application or attribute group.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
dp_handle_t dp_AllocateHandle(int *Status)
```

Input parameters

`dp_AllocateHandle` requires one input parameter:

int *Status

A pointer to an integer storage variable that `dp_AllocateHandle` uses for storing the API return status code.

Return status code

`dp_AllocateHandle` returns a pointer if the handle is allocated successfully. Otherwise, it returns NULL.

The following are possible status codes returned by `dp_AllocateHandle` calls:

KUMP_API_OK

`dp_AllocateHandle` completed successfully.

KUMP_API_Environment_Init_Failed

API initialization failed because of incomplete or incorrect product installation.

KUMP_API_HostName_Unresolve

Although TCP/IP API transport is selected, the local host name cannot be resolved to an IP address. Either the host name is not configured correctly, or the DNS/HOSTS table is not set up correctly for name resolution.

KUMP_API_Main_Storage_Unavailable

Not enough system storage is available for allocating the data provider API anchor control block.

KUMP_API_AllocHandle_No_Storage

Not enough system storage is available for allocating the data provider API handle.

dp_BeginInput

Description

dp_BeginInput signals the data provider that the caller is ready for data input for a specific application and attribute group. It binds the handle to an application and attribute group until the dp_EndInput call.

After the API Server Data Provider receives the dp_BeginInput request, it registers the caller and the Tivoli Universal Agent notifies the Tivoli Enterprise Monitoring Server that the data source is online.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_BeginInput(dp_handle_t InHandle, char *ApplName, char *AttrGroup, int *Status)
```

Input parameters

The following are required dp_BeginInput input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the dp_AllocateHandle API call.

*char **ApplName

A NULL terminated string of the application name. The name is case-sensitive and must be identical to the name specified on the //APPL statement of a data definition metafile.

*char **AttrGroup

A NULL terminated string of the attribute group name. The name is case-sensitive and must be identical to the name specified on the //NAME statement under the //APPL statement of a data definition metafile.

*int **Status

A pointer to an integer storage variable that dp_BeginInput uses for storing the API return status code.

Return status code

dp_BeginInput returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_BeginInput calls:

KUMP_API_OK

dp_BeginInput completed successfully.

KUMP_API_BgnInput_ApplGroup_Required

The ApplName and/or AttrGroup input parameter is NULL or not valid.

KUMP_API_Invalid_Handle

The input handle pointer is not valid, or it is not a valid handle pointer allocated by the dp_AllocateHandle API call.

KUMP_API_BgnInput_Application_Not_Found

The application specified is not valid, or the defining metafile is not defined to the data provider API server process. If the name is correct, use dp_Define first to define the metafile to the API Server Data Provider or add the metafile name to the data provider configuration file.

KUMP_API_BgnInput_AttrGroup_Not_Found

The attribute group specified is not valid, or the defining metafile is not defined to the data provider

API server process. If the name is correct, use `dp_Define` first to define the metafile to the API Server Data Provider or add the metafile name to the data provider configuration file.

KUMP_API_Server_Source_No_Storage

The API Server Data Provider is unable to obtain enough storage to process this request.

KUMP_API_BgnInput_Server_App_InitFailed

The API Server Data Provider encountered an error while initializing the processing environment for supporting the data input. Retry is not possible.

KUMP_API_BgnInput_Server_Register_Failed

The API Server Data Provider received a unrecoverable error return code while performing application registration. Retry is not possible.

KUMP_API_Server_State_Logic_Error

The `dp_InputData` is called out of sequence or using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

dp_BeginSample

Description

Signals the API Server Data Provider of the beginning of a multiple row data sample. This API call should be used in conjunction with the dp_EndSample API call.

Note: When the dp_BeginSample and dp_EndSample APIs are used for a particular attribute group, you must continue to use those APIs for the life of the program execution. You cannot switch input modes between dp_BeginSample/dp_EndSample and dp_BeginInput/dp_EndInput.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_BeginSample(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required dp_BeginSample input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the dp_AllocateHandle API call and bound to an application by the dp_BeginInput call.

*int **Status

A pointer to an integer storage variable that dp_BeginSample uses for storing the API return status code.

Return status code

dp_BeginSample returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_BeginSample calls:

KUMP_API_OK

dp_BeginSample completed successfully.

KUMP_API_Required_Parm_Missing

The input handle pointer is not valid.

KUMP_API_Invalid_Handle

The handle is not a valid handle pointer allocated by the dp_AllocateHandle API call, or it is not bound to an application.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active data provider.

dp_CancelRequest

Description

dp_CancelRequest is similar to dp_ReturnData in that it responds to a demand-driven report or situation request for data rows as initiated by a dp_ReceiveRequest API. If the API client program does not have any new data rows to return with dp_ReturnData, then use the dp_CancelRequest to tell the API Server Data Provider to cancel the outstanding request for data.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_CancelRequest(dp_handle_t InHandle, dp_ReqID_t *ReqID, int *Status)
```

Input parameters

The following are required dp_CancelRequest input parameters:

dp_handle_t InHandle

Identifies the attribute group of the inbound request. A dp_AcceptRequest API call must already be completed for this handle.

dp_ReqID_t *ReqID

A storage variable of type dp_ReqID_t used to store the request identifier. The ReqID associates the request result to the delivered request. It must be the same as the ReqID returned by the dp_ReceiveRequest API call.

int *Status

A pointer to an integer storage variable that dp_CancelRequest uses for storing the API return status code.

Return status code

dp_CancelRequest returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_CancelRequest calls:

KUMP_API_OK

dp_CancelRequest completed successfully.

KUMP_API_Invalid_Handle

The input handle pointer is not a valid handle pointer allocated by the dp_AllocateHandle API call.

KUMP_API_Buffer_Storage_Unavailable

Not enough memory is available for the buffer storage.

dp_ClearBuffer

Description

dp_ClearBuffer resets the user data portion of the buffer to binary zero and the buffer control fields. The calling program must call dp_ClearBuffer before reusing the same buffer for the next data input. The dp_InputData API does not reset the buffer contents after data transmission.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ClearBuffer(char *UserBuffer, int *Status)
```

Input parameters

The following are required dp_ClearBuffer input parameters:

*char * UserBuffer*

A pointer to the data buffer to be cleared. The buffer must be allocated by the dp_AllocateBuffer API call.

*int *Status*

A pointer to an integer storage variable that dp_ClearBuffer uses for storing the API return status code.

Return status code

dp_ClearBuffer returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_ClearBuffer calls:

KUMP_API_OK

dp_BeginSample completed successfully.

KUMP_API_Buffer_Invalid_Buffer_Pointer

The UserBuffer parameter is not a valid pointer.

KUMP_API_Buffer_Unalloc_Buffer

The UserBuffer parameter contains an address that is not a buffer address allocated by the dp_AllocateBuffer API call.

dp_ClearPriorData

Description

dp_ClearPriorData sends a request to the API Server Data Provider to purge all data rows that have previously been received and cached for the attribute group that is associated with the input handle. If the input handle is valid but there are currently no cached data rows to purge, the dp_ClearPriorData API still returns KUMP_API_OK even though there was no clean-up necessary.

Every attribute group has a Time-To-Live (TTL) value that determines how long collected data rows are cached in the Tivoli Universal Agent process. If you want to clear older data rows for an API application attribute group so that the rows are no longer displayed in the Tivoli Enterprise Portal workspace, call the dp_ClearPriorData API. You typically call this API before sending a new set of data with dp_InputData or dp_ReturnData.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ClearPriorData(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required dp_ClearPriorData input parameters:

dp_handle_t InHandle

A handle obtained by the dp_AllocateHandle API call.

*int **Status

A pointer to an integer storage variable that dp_ClearPriorData uses for storing the API return status code.

Return status code

dp_ClearPriorData returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_ClearPriorData calls:

KUMP_API_OK

dp_ClearPriorData completed successfully.

KUMP_API_Invalid_Handle

The input handle is not valid.

KUMP_API_Buffer_Storage_Unavailable

The API client cannot obtain sufficient communication buffer storage to send the dp_ClearPriorData request.

KUMP_API_Server_Data_Format_Error

The API Server Data Provider detected a data record that is incorrect, incomplete, or contains unexpected fields.

dp_CloseSession

Description

Ends the connection established between the client program and the API Server Data Provider by `dp_OpenSession` call. If the `dp_CloseSession` call is omitted, the connection is automatically terminated by the `dp_FreeHandle` API call.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_CloseSession(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required `dp_CloseSession` input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the `dp_AllocateHandle` API call.

int *Status

A pointer to an integer storage variable that `dp_CloseSession` uses for storing the API return status code.

Return status code

`dp_CloseSession` returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by `dp_CloseSession` calls:

KUMP_API_OK

`dp_CloseSession` completed successfully.

KUMP_API_Invalid_Handle

The handle is not a valid handle pointer allocated by the `dp_AllocateHandle` API call.

dp_Define

Description

dp_Define requests the data provider API server to load the specified data definition metafile and prepare it for data input. The metafile must exist and be accessible to the API server.

You need to call dp_Define only once. You can omit the dp_Define call altogether if the application data definition is loaded at startup of the API Server Data Provider.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_Define(char *MetaFileName, int AcceptOK, int *Status)
```

Input parameters

The following are required dp_Define input parameters:

*char ** MetaFileName

A NULL terminated string of the name of the data definition metafile. The metafile must exist and be available to the API Server Data Provider.

int AcceptOK

1 or 0 (Yes or No) to accept the existing application definition known to the agent if the application is already defined, possibly from previous use or from the definition of the same application reported from another data provider.

*int ** Status

A pointer to an integer storage variable that dp_Define uses for storing the API return status code.

Return status code

dp_Define returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_Define calls:

KUMP_API_OK

dp_Define completed successfully.

KUMP_API_Invalid_Parameter

The input MetaFileName parameter contains an address that is not valid.

KUMP_API_DP_Inactive

The data provider API server is not running.

KUMP_API_Define_Appl_Exist

The metafile specified is already defined to the API Server Data Provider. To refresh the data definition, use the dp_Redefine API call.

KUMP_API_Define_Appl_NotFound

The metafile does not exist or cannot be located by the API Server Data Provider.

KUMP_API_Define_Spec_Error

The API Server Data Provider detected a metafile file syntax or specification error. The metafile definition is rejected.

dp_EndInput

Description

dp_EndInput signals the data provider that the caller has completed data input for a specific application and attribute group.

After the API Server Data Provider receives the dp_EndInput request, it unregisters the caller and the Tivoli Universal Agent notifies the Tivoli Enterprise Monitoring Server that the data source is offline.

Do not call dp_EndInput if a client program detects a lost connection to the API Server Data Provider. dp_EndInput requires a valid connection to complete successfully. If a client program detects a lost connection and is preparing to enter a wait/retry loop until the API Server Data Provider is online again, ensure that the client program calls APIs that perform only local cleanup functions, such as dp_FreeBuffer and dp_FreeHandle, before calling dp_Ping to status the API Server Data Provider.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_EndInput(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required dp_EndInput input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained the by dp_AllocateHandle API call and bound to an application by the dp_BeginInput call.

int *Status

A pointer to an integer storage variable that dp_EndInput uses for storing the API return status code.

Return status code

dp_EndInput returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_EndInput calls:

KUMP_API_OK

dp_EndInput completed successfully.

KUMP_API_Invalid_Handle

The handle is not a valid handle pointer allocated by the dp_AllocateHandle API call, or it is not bound to an application.

KUMP_API_EndInput_Application_Not_Active

The dp_EndInput is rejected because the application is no longer active to the API Server Data Provider. The handle probably is obsolete, or a successful dp_EndInput has already completed.

KUMP_API_EndInput_Server_Unregister_Failed

The API Server Data Provider received an unrecoverable error return code while performing application unregistration. Retry is not possible.

KUMP_API_Server_State_Logic_Error

The dp_EndInput is called out of sequence or using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active data provider.

dp_EndSample

Description

Signals the API Server Data Provider of the completion of a multiple row data sample. This call must be used in conjunction with `dp_BeginSample`.

Do not call `dp_EndSample` if a client program detects a lost connection to the API Server Data Provider. `dp_EndSample` requires a valid connection to complete successfully. If a client program detects a lost connection and is preparing to enter a wait/retry loop until the API Server Data Provider is online again, ensure that the client program calls APIs that perform only local cleanup functions, such as `dp_FreeBuffer` and `dp_FreeHandle`, before calling `dp_Ping` to status the API Server Data Provider.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_EndSample(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required `dp_EndSample` input parameters:

dp_handle_t InHandle

An allocated API handle obtained by the `dp_AllocateHandle` API call and bound to an application by the `dp_BeginInput` call.

int *Status

A pointer to an integer storage variable that `dp_EndSample` uses for storing the API return status code.

Return status code

`dp_EndSample` returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by `dp_EndSample` calls:

KUMP_API_OK

`dp_EndSample` completed successfully.

KUMP_API_Required_Parm_Missing

The input handle pointer is not valid.

KUMP_API_Invalid_Handle

The handle is not a valid handle pointer allocated by the `dp_AllocateHandle` API call, or it is not bound to an application.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active data provider.

dp_FormatBufferData

Description

dp_FormatBufferData moves and formats the input application data into the specified data buffer. All data delimiters defined in the data definition metafile are supplied accordingly on behalf of the caller by this API. You must call dp_FormatBufferData once for every attribute value that is being collected in preparation for a dp_InputData or dp_ReturnData API call. Note that you do not need to make a separate dp_FormatBufferData call to append your delimiter characters because the API automatically handles that for you when it formats an attribute value.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_FormatBufferData(char *UserBuffer, void *ApplData, int DataSize, int DataType, int *Status)
```

Input parameters

The following are required dp_FormatBufferData input parameters:

*char *UserBuffer*

A pointer to the data provider data buffer for the formatted application.

*void *ApplData*

A pointer to the application data to be formatted.

int DataSize

Size of the application data.

int DataType

Type of application data. It must be numeric or character, as defined in the kumpapi.h header file (TypelsNumeric and TypelsCharacter). Character data type is assumed if the input DataType parameter is invalid.

*int *Status*

A pointer to an integer storage variable that dp_FormatBufferData uses for storing the API return status code.

Return status code

dp_FormatBufferData returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_FormatBufferData calls:

KUMP_API_OK

dp_FormatBufferData completed successfully.

KUMP_API_Buffer_Invalid_Buffer_Pointer

The UserBuffer parameter is not a valid pointer.

KUMP_API_Buffer_Unalloc_Buffer

The UserBuffer parameter contains an address that is not a dp_AllocateBuffer buffer address, or it is a free buffer that has not been allocated.

KUMP_API_Buffer_Invalid_Ownership

The UserBuffer parameter contains a buffer address that is not valid, or the buffer is not owned by any handle.

KUMP_API_FmtBuffer_No_Data

The ApplData parameter is NULL or points to a NULL string.

KUMP_API_FmtBuffer_Invalid_Size

The DataSize parameter has value of zero or less than.

KUMP_API_FmtBuffer_No_Room

The UserBuffer parameter does not have enough remaining space to contain the formatted application data. Retry using the dp_AllocateBuffer with a larger request size.

dp_FreeBuffer

Description

dp_FreeBuffer releases the storage allocated by dp_AllocateBuffer. If dp_FreeBuffer is omitted, all buffers are freed when the owning handle is freed by the dp_FreeHandle API call.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_FreeBuffer(char *UserBuffer, int *Status)
```

Input parameters

The following are required dp_FreeBuffer input parameters:

*char *UserBuffer*

A pointer to the address of the buffer to be freed. The buffer must be allocated by the dp_AllocateBuffer API call.

*int *Status*

A pointer to an integer storage variable that dp_FreeBuffer uses for storing the API return status code.

Return status code

dp_FreeBuffer returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_FreeBuffer calls:

KUMP_API_OK

dp_FreeBuffer completed successfully.

KUMP_API_Buffer_Invalid_Buffer_Pointer

The UserBuffer parameter is not valid or contains an address that is not valid.

KUMP_API_Buffer_Unalloc_Buffer

The UserBuffer parameter contains an address that is not a dp_AllocateBuffer buffer address, or it is a free buffer that has not been allocated.

KUMP_API_Buffer_Invalid_Ownership

The UserBuffer parameter contains a buffer address that is not valid, or the buffer is not owned by any handle.

dp_FreeHandle

Description

dp_FreeHandle releases allocated handle storage and all of the owning resources such as allocated buffer and connection.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_FreeHandle(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required dp_FreeHandle input parameters:

dp_handle_t InHandle

A handle obtained by the dp_AllocateHandle API call.

int *Status

A pointer to an integer storage variable that dp_FreeHandle uses for storing the API return status code.

Return status code

dp_FreeHandle returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_FreeHandle calls:

KUMP_API_OK

dp_FreeHandle completed successfully.

KUMP_API_Invalid_Handle

The InHandle parameter is not valid or contains a handle address that is not valid.

dp_InputData

Description

dp_InputData sends the data buffer contents to the API Server Data Provider. The buffer must be allocated using the dp_AllocateBuffer API call and owned by a valid data provider handle.

The buffer contents remain unchanged. It is the responsibility of the caller to call dp_ClearBuffer before reusing the same buffer for another data transfer operation.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_InputData(char *UserBuffer, int *Status)
```

Input parameters

The following are required dp_InputData input parameters:

*char *UserBuffer*

A pointer to the address of the data buffer that contains application data to be sent to the API Server Data Provider. The buffer must be allocated by the dp_AllocateBuffer API call.

*int *Status*

A pointer to an integer storage variable that dp_InputData uses for storing the API return status code.

Return status code

dp_InputData returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_InputData calls:

KUMP_API_OK

dp_InputData completed successfully.

KUMP_API_Input_NULL_Buffer

The UserBuffer parameter is not valid.

KUMP_API_Input_Invalid_Buffer

The UserBuffer parameter does not point to a valid data provider buffer address.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider because of incomplete or incorrect local configuration.

KUMP_API_Server_Data_Emit_Error

The API Server Data Provider received an irrecoverable error when sending data. Retry is possible.

KUMP_API_Server_Data_Format_Error

The API Server Data Provider detected data specification errors in the application data. The input data does not match the data definition specified in the application metafile.

KUMP_API_Server_State_Logic_Error

The dp_InputData is called out of sequence or is using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active data provider.

Batch data input

Multiple application data can be combined in batches for including in a single `dp_InputData` API call to the Tivoli Universal Agent. The individual data records must be separated by a line feed character. Combining multiple application data in batches reduces turnaround time and improves overall response time and efficiency. The example in Figure 2 on page 30 demonstrates a sample programming method.

In this example, each application data record consists of four data fields: `PaketReceived` (`PKTrcvd`), `PacketRouted` (`PKTrout`), `PacketDropped` (`PKTdrop`), and `AveragePacketSize` (`PKTsize`). The program batches the records 20 at a time, separated by a line feed character, and calls `dp_InputData` once to send them to the Tivoli Universal Agent.

You must use the `dp_FormatBufferData` API call to insert the line feed character as shown the example above. Do not include the line feed character in the data.

```
int i,j;
char LineFeed '\n';
char *WorkBuffer;
. . . .
. . . .
. . . .
dp_ClearBuffer(WorkBuffer, &API_Status);
j = 0;
do {
dp_FormatBufferData(WorkBuffer, &PKTrcvd, sizeof(int), TypelsNumeric,
&API_Status);
dp_FormatBufferData(WorkBuffer, &PKTrout, sizeof(int), TypelsNumeric,
&API_Status);
dp_FormatBufferData(WorkBuffer, &PKTdrop, sizeof(int), TypelsNumeric,
&API_Status);
dp_FormatBufferData(WorkBuffer, &PKTsize, sizeof(int), TypelsNumeric,
&API_Status);
dp_FormatBufferData(WorkBuffer, &LineFeed, 1, TypelsCharacter, &API_Status);
} while(++j < 20); /* Batch 20 at a time */
dp_InputData(WorkBuffer, &API_Status);
```

Figure 2. Sample program for combining multiple application data in batches

dp_OpenSession

Description

dp_OpenSession establishes a logical connection to the API Server Data Provider. The input handle is bound to the connection for all subsequent data exchange. If the caller does not invoke dp_OpenSession before initiating data exchange with the server (for example, dp_BeginInput is not preceded by dp_OpenSession), the connectionless data exchange procedure is used.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_OpenSession(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are required dp_OpenSession input parameters:

dp_handle_t InHandle

A handle obtained by the dp_AllocateHandle API call.

int *Status

A pointer to an integer storage variable that dp_OpenSession uses for storing the API return status code.

Return status code

dp_OpenSession returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_OpenSession calls:

KUMP_API_OK

dp_OpenSession completed successfully.

KUMP_API_Invalid_Handle

The input handle is not valid.

KUMP_API_Open_Session_Exist

An active connection is already bound to the input handle.

KUMP_API_Configuration_Error

The data provider API package is not completely installed, or the environment variable KUMP_API_TRANSPORT specified a transport type that is not valid.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider because of incomplete or incorrect local configuration.

KUMP_API_Open_Session_Connect_Failed

The API connection to the API Server Data Provider failed. Retry is not possible.

dp_Ping

Description

dp_Ping checks the operational status of the API Server Data Provider. Although it behaves in a manner similar to the TCP/IP ping command, it does not use the ping command for its operation.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_Ping(int *Status)
```

Input parameters

dp_Ping requires only one parameter:

*int *Status*

A pointer to an integer storage variable that dp_Ping uses for storing the API return status code.

Return status code

dp_Ping returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_Ping calls:

KUMP_API_OK

dp_Ping completed successfully.

KUMP_API_Request_TimedOut

dp_Ping failed. The request timed out.

KUMP_API_Configuration_Error

The data provider API package installation is incomplete, or the environment variable KUMP_API_TRANSPORT specified a transport type that is not valid.

dp_ReceiveRequest

Description

dp_ReceiveRequest queries the API Server Data Provider for any outstanding requests targeted for the attribute group identified by the data provider handle. If a request is available, it is received into the buffer storage provided by the caller.

The received request is stored in the caller buffer storage in one of several formats, depending on the type of request. No interpretation or formatting is done on behalf of the caller to either the application request or the corresponding parameters, if any.

The types of requests received are determined by the EnableRequest parameter that you specified on the preceding dp_AcceptRequest API call. If you specified KUMP_API_TRUE, your API client program receives the following:

- Notifications about situation starts and situation stops
- Automation action requests

If you specified KUMP_API_REPORT, your API client program also receives the following:

- Report requests whenever a Tivoli Enterprise Portal workspace open or refresh occurs
- Situation requests whenever a situation sampling interval occurs

If you specified KUMP_API_FALSE, you will not receive any requests.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ReceiveRequest(dp_handle_t InHandle, int SyncRequest, char *UserBuffer, int UserBufferSize,
dp_ReqID_t *ReqID, int *ReceiveReqSize, int *Call_Status)
```

Input parameters

The following are required dp_ReceiveRequest input parameters:

dp_handle_t InHandle

Identifies the attribute group of the inbound request. A dp_AcceptRequest API call must already be completed for this handle.

int SyncRequest

Selects synchronous or asynchronous behavior for the dp_ReceiveRequest API call. A value of KUMP_API_SYNC_REQ indicates that the calling program waits indefinitely until an inbound request becomes available and received. A value of KUMP_API_ASYNC_REQ polls the data provider for outstanding requests. If an inbound request is available, it is received; otherwise, the API call returns to the caller an appropriate return status code.

*char **UserBuffer

A pointer to the storage location where the received request is saved. The buffer can be any storage allocated by the calling program or a buffer allocated by calling dp_AllocateBuffer API call. See “Return status code” on page 25 for the format of the data received by the buffer.

int UserBufferSize

The size of the storage identified by UserBuffer parameter.

*dp_ReqID_t **ReqID

A storage variable of type dp_ReqID_t to be used to store the request identifier. The ReqID

associates the request result to the delivered request. If the ReqID is a NULL pointer, the request identifier is not saved. In this case, no request result can be sent to the data provider using a dp_SendRequestResult API call.

int *ReceiveReqSize

A pointer to an integer storage variable that dp_ReceiveRequest uses for storing the received request data size, if the API call is successful; otherwise, its content is unchanged.

int *Status

A pointer to an integer storage variable that dp_ReceiveRequest uses for storing the API return status code.

Return status code

dp_ReceiveRequest returns TRUE if the function is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_ReceiveRequest calls:

KUMP_API_OK

dp_ReceiveRequest completed successfully.

KUMP_API_Invalid_Handle

The input handle pointer is not valid or is not a valid handle pointer allocated by the dp_AllocateHandle API call.

KUMP_API_Required_Parm_Missing

The required input parameter UserBuffer, ReceiveReqSize, or ReqID is not valid.

KUMP_API_ReceiveRequest_Invalid_Buffer_Size

The UserBufferSize input parameter value is zero or negative.

KUMP_API_ReceiveRequest_Data_Truncated

The received request data was larger than the UserBuffer storage and it has been truncated to the size specified by the input UserBuferSize parameter.

KUMP_API_ReceiveRequest_NoReq_Outstanding

The API call returns in asynchronous receive mode because no inbound request is currently outstanding.

Data Formats

The user buffer contains received data in one of the following formats.

Situation predicate information

Add filter request

```
<Request_ID> FILTER(ADD) PARM(INTERVAL=<situation_interval>
APPLNAME=<application_name > ATTRGROUP=<attribute_group>
ATTRIBUTES=((<attribute_name operator value>)(...) ...)
SITNAME(<situation_name>)
```

where:

Request_ID

Specifies the integer assigned by the Tivoli Universal Agent to uniquely identify this particular request.

INTERVAL

Specifies the sampling interval for the situation, in seconds. INTERVAL is omitted for situations that monitor event type data.

APPLNAME

Specifies the name of the Tivoli Universal Agent application, as defined in the APPL statement of the corresponding API metafile, that is associated with the situation.

ATTRGROUP

Specifies the name of the attribute group as defined in the NAME statement of the corresponding API metafile that is associated with the situation.

ATTRIBUTES

Specifies one or more situation predicate triplets in the form *attribute_name*, *logical_operator*, *value*. Each triplet is enclosed in parentheses.

SITNAME

Specifies the name of the situation that was started.

The received ReqID uniquely identifies this situation filter.

Delete filter request

```
<Request_ID> FILTER(DELETE) SITNAME(<situation_name>)
```

where:

Request_ID

Specifies the integer assigned by the Tivoli Universal Agent to uniquely identify this particular request.

SITNAME

Specifies the name of the situation that was stopped.

IBM Tivoli Monitoring automation action request

```
<Request_ID> REQ(<command>) PARM(<parameter_list>)
```

where:

Request_ID

Specifies the integer assigned by the Tivoli Universal Agent to uniquely identify this particular request.

REQ Specifies the command or request to be performed by the receiving API client program. The REQ field typically contains the command as defined in the System Command edit control in the Tivoli Enterprise Portal Situation Editor, for example, the name of a script or program to execute as a reflex action when a situation triggers. The receiving program should know the meaning and behavior of the command or request.

PARM Specifies any optional parameters to pass to the command. For example, the PARM field contains run-time attribute values as substituted during reflex action processing. This field is omitted if you do not supply any arguments.

Tivoli Enterprise Portal report request

```
<Request_ID> REPORT(REQUEST) PARM(ATTRGROUP=<attribute_group>
WHERE(<attribute_name operator value> OR/AND ...))
```

where:

Request_ID

Specifies the integer assigned by the Tivoli Universal Agent to uniquely identify this particular request.

ATTRGROUP

Specifies the name of the attribute group, as defined in the NAME statement of the corresponding API metafile, that is associated with the Tivoli Enterprise Portal workspace for which the report request was issued.

WHERE

Specifies any optional WHERE clause filtering that you defined in the Tivoli Enterprise Portal workspace query. If the default workspace query, which selects all columns from the attribute group, is being used, the WHERE parameter is omitted. Your API client program typically uses the WHERE parameter as a means of limiting the number of returned rows to only those that match the criteria. This can help reduce the amount of report data being sent back to the API Server Data Provider.

IBM Tivoli Monitoring situation request

```
<Request_ID> SITUATION(REQUEST) PARM(SITNAME=<situation_name>  
ATTRGROUP=<attribute_group> WHERE(<attribute_name operator value>  
OR/AND...))
```

where:

Request_ID

Specifies the integer assigned by the Tivoli Universal Agent to uniquely identify this particular request.

SITNAME

Specifies the name of the situation that is being evaluated for data that matches the situation threshold criteria.

ATTRGROUP

Specifies the name of the attribute group, as defined in the NAME statement of the corresponding API metafile, that is associated with the situation attributes.

WHERE

Specifies WHERE clause filtering in the situation, which equates to the situation predicates. Your API client program typically uses the WHERE parameter as a means of limiting the number of returned rows to only those that match the situation criteria.

dp_Redefine

Description

dp_Redefine refreshes the active application data definition with the current contents of the metafile specification. The application must be active to the API Server Data Provider.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_Redefine(char *MetaFileName, int AcceptOK, int *Status)
```

Input parameters

The following are required dp_Redefine input parameters:

*char * MetaFileName*

A NULL terminated string of the name of the metafile. The metafile must exist and be available to the API Server Data Provider.

int AcceptOK

1 or 0 (Yes or No) to accept an existing application definition known to the agent if the application is already defined, possibly from previous use or from a definition of the same application reported from another data provider.

*int *Status*

A pointer to an integer storage variable that dp_Redefine uses for storing the API return status code.

Return status code

dp_Redefine returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_Redefine calls:

KUMP_API_OK

dp_Redefine completed successfully.

KUMP_API_Invalid_Parameter

The input MetaFileName parameter contains an address that is not valid.

KUMP_API_Redefine_Appl_NotDefined

The application being redefined is not previously defined to the API Server Data Provider. Use dp_Define instead of dp_Redefine if the metafile specification is correct.

KUMP_API_Redefine_Appl_NotFound

The metafile does not exist or cannot be located by API Server Data Provider.

KUMP_API_Redefine_Spec_Error

The API Server Data Provider detected a syntax or specification error in the metafile. The metafile definition is rejected.

dp_ResetHandle

Description

dp_ResetHandle clears data provider handle control information so that it can be reused for another application data exchange request. The allocated buffers owned by this handle, however, are not freed and they can be reused or reallocated.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ResetHandle(dp_handle_t InHandle, int *Status)
```

Input parameters

The following are the required dp_ResetHandle input parameters:

dp_handle_t InHandle

The handle to be reset.

*int **Status

A pointer to an integer storage variable that dp_ResetHandle uses for storing the API return status code.

Return status code

dp_ResetHandle returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_ResetHandle calls:

KUMP_API_OK

dp_ResetHandle completed successfully.

KUMP_API_Invalid_Handle

The InHandle parameter does not contain a valid data provider handle address.

dp_ReturnData

Description

dp_ReturnData is similar to dp_InputData in that it sends the contents of a data buffer to the API Server Data Provider. The buffer must be allocated using the dp_AllocateBuffer API call and owned by a valid data provider handle.

Where dp_ReturnData differs is that it only sends a data buffer in response to a report or situation request received by the dp_ReceiveRequest API. The action request must either contain a REPORT(REQUEST) or SITUATION(REQUEST) literal. See “Data Formats” on page 34 for the various data formats returned to the dp_ReceiveRequest API.

The ReqID value supplied by dp_ReceiveRequest is used in the dp_ReturnData API call to tell the API Server Data Provider which report or situation request the data is being returned for.

If your application is returning multiple data rows in response to a single report or situation request, return the first n-1 rows using dp_InputData, and then use dp_ReturnData for the last row.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ReturnData(dp_handle_t InHandle, dp_ReqID_t *ReqID, char *UserBuffer, int *Status)
```

Input parameters

The following are required dp_ReturnData input parameters:

dp_handle_t InHandle

Identifies the attribute group of the inbound request. A dp_AcceptRequest API call must already be completed for this handle.

dp_ReqID_t *ReqID

A storage variable of type dp_ReqID_t to be used to store the request identifier. The ReqID associates the request result to the delivered request. It must be the same as the ReqID returned by the dp_ReceiveRequest API call.

char *UserBuffer

A pointer to the address of the data buffer that contains application data to be sent to the API Server Data Provider. The buffer must be allocated by the dp_AllocateBuffer API call.

int *Status

A pointer to an integer storage variable that dp_ReturnData uses for storing the API return status code.

Return status code

dp_ReturnData returns TRUE if processing is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_ReturnData calls:

KUMP_API_OK

dp_ReturnData completed successfully.

KUMP_API_Input_NULL_Buffer

The UserBuffer parameter is not valid.

KUMP_API_Input_Invalid_Buffer

The UserBuffer parameter does not point to a valid data provider buffer address.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider because of incomplete or incorrect local configuration.

KUMP_API_Server_Data_Emit_Error

The API Server Data Provider received an irrecoverable error when sending data. Retry is possible.

KUMP_API_Server_Data_Format_Error

The API Server Data Provider detected data specification errors in the application data. The input data does not match the data definition specified in the application metafile.

KUMP_API_Server_State_Logic_Error

The dp_ReturnData is called out of sequence or is using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active data provider.

dp_SendRequestResult

Description

dp_SendRequestResult sends the outcome of an automation action request to the API Server Data Provider. The result data is typically a return code, error message, or other success or failure indication after a script or command is executed. The result data must be in character form and it cannot be larger than 256 bytes. You must convert all binary or numeric data to character form. The API Server Data Provider does not interpret or perform data manipulation when it receives the result data.

Use this API only to send results after processing an automation action request that was received by the dp_ReceiveRequest API. If dp_ReceiveRequest receives a report or situation request, use the dp_ReturnData API to send the returned data to the API Server Data Provider.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_SendRequestResult(dp_handle_t InHandle, dp_ReqID_t *ReqID, char * ReqResultBuffer, int ResultSize, int *Call_Status)
```

Parameters

The following are required dp_SendRequestResult input parameters:

dp_handle_t InHandle

Identifies the attribute group of the inbound request. A dp_AcceptRequest API call must already be completed for this handle.

dp_ReqID_t *ReqID

A storage variable of type dp_ReqID_t that contains the request identifier. The ReqID associates the request result with the delivered request. It must be the same as the ReqID returned by the dp_ReceiveRequest API call.

char *ReqResultBuffer

A pointer to the storage location which contains the request result data to be sent to the data provider.

int ResultSize

The data length of the request result data.

int *Status

A pointer to an integer storage variable that dp_SendRequestResult uses for storing the API return status code.

Return status code

dp_SendRequestResult returns TRUE if the function is successful. Otherwise, it returns FALSE.

The following are possible status codes returned by dp_SendRequestResult calls:

KUMP_API_OK

dp_SendRequestResult completed successfully.

KUMP_API_Invalid_Handle

The input handle pointer is not valid or it is not a valid handle pointer allocated by the dp_AllocateHandle API call.

KUMP_API_Required_Parm_Missing

The required input parameter ReqResultBuffer or ResultSize is not valid.

KUMP_API_SendReqResult_Buffer_Truncated

The request result data is larger than the 256-byte limit and is truncated to the maximum limit.

KUMP_API_SendReqResult_Request_Not_Found

The request result data identified by the ReqID does not exist or it is no longer outstanding, possibly because the wait time for the request results specified by dp_AcceptRequest has expired.

dp_SetSourceName

Description

dp_SetSourceName associates an unique source name with an application attribute group for data input. An instance of a Tivoli Universal Agent application is represented as a IBM Tivoli Monitoring managed system. Its name is constructed by concatenating the name of the application program's host workstation, the name of the application, and a version number. For example:

wildcat:CustomerInquiry00

This naming convention means that there can be only one application instance per managed system. If an application program supports multiple instances of the same application concurrently, the application program must identify each application instance uniquely.

For example, you might have a control program obtaining the same type of application data, say queue manager performance status, from Queue Manager A and Queue Manager B, and using a data provider API to import them to IBM Tivoli Monitoring for management. The control program must identify Queue Manager A data and Queue Manager B data separately using the dp_SetSourceName API call.

Note that an application attribute group instance is represented by a handle. To support multiple instances of the same group concurrently, you must obtain multiple handles by calling dp_AllocateHandle and then dp_SetSourceName repeatedly to associate the desired data source name to each handle. Since data buffers are allocated by and related to handles, you format the data from a specific source to the corresponding handle buffer and then call dp_InputData to send application data inbound to IBM Tivoli Monitoring.

dp_SetSourceName must be invoked before dp_BeginInput

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_SetSourceName(dp_handle_t InHandle, char *SourceName, int *Status)
```

Input parameters

The following are the required dp_SetSourceName input parameters:

dp_handle_t InHandle

An allocated data provider API handle obtained by the dp_AllocateHandle API call.

char *SourceName

A NULL terminated string of the data source name. The source name plus the application name must not exceed 29 characters. The source name is displayed to the left of the colon in the managed system name.

int *Status

A pointer to an integer storage variable that dp_SetSourceName uses for storing the API return status code.

Return status code

dp_SetSourceName returns TRUE if processing is successful. Otherwise, it returns FALSE. The following are possible status codes returned by dp_SetSourceName calls:

KUMP_API_OK

dp_SetSourceName completed successfully.

KUMP_API_Invalid_Handle

The input handle pointer is not valid, or it is not a valid handle pointer allocated by the `dp_AllocateHandle` API call.

KUMP_API_SetSourceName_Name_Required

The input source name parameter does not point to a character string.

KUMP_API_SetSource_Name_Invalid

The input source name parameter contains invalid or unsupported characters, such as non-displayable or Unicode characters outside the Latin-1 character set.

KUMP_API_Server_State_Logic_Error

The `dp_SetSourceName` API is called out of sequence or is using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

dp_ShowMessage

Description

dp_ShowMessage displays the description of the API Server Data Provider status code.

Synopsis (C)

```
#include <kumpapi.h>
```

Prototype (C)

```
int dp_ShowMessage(int MsgID, char *MsgBuffer, int BufferSize)
```

Input parameters

The following are required dp_ShowMessage input parameters:

int MsgID

The data provider API message ID to be displayed.

*char ** MsgBuffer

A pointer to a caller-provided storage area for the dp_ShowMessage to use to store the output message text. If the MsgBuffer parameter is NULL, the output message text is not copied but instead is written to the standard output.

int BufferSize

The length of the caller-provided message buffer size. If MsgBuffer is NULL, this parameter is ignored. If the output message text is longer than the input buffer size, the message text is truncated to the maximum buffer size.

Return status code

dp_ShowMessage always returns 0.

Chapter 3. Console commands

A subset of API functions can be invoked by console commands. This chapter documents the syntax, descriptions, and status codes for those commands.

Using API console commands

The Tivoli Universal Agent API console commands enable you to invoke a subset of the API functions from a command line. Console commands are particularly useful when you want to load a new metafile or revise an active application data definition without stopping and restarting the data provider. The commands are summarized in Table 4 on page 47.

Command syntax

This guide documents only descriptions of the Tivoli Universal Agent API commands and their required parameters. It assumes that you are familiar with the command syntax enforced on the operating system you are using. Similarly, it assumes that you are aware of any system-specific limitations on entering data or special characters.

Minimum required command-line input

The minimum required command-line input is to call the KUMPINPT command-line program. The command-line program makes all the necessary programming calls on behalf of the caller.

The first call is:

```
KUMPINPT 'application data record' application-name attribute-group-name
```

Subsequent calls are:

```
KUMPINPT 'application data record'
```

Table 4. Tivoli Universal Agent API console commands

Command	Description
KUMPAREQ (dp_AcceptRequest)	Allows a client program to indicate its ability to accept inbound requests from the data provider that arise as the result of IBM Tivoli Monitoring automation actions, situation starts and stops, and demand-driven report requests.
KUMPBGNI (dp_BeginInput)	Identifies an attribute group for data input
KUMPDEFN (dp_Define)	Defines a metafile to the data provider
KUMPENDI (dp_EndInput)	Indicates the completion of data input for an attribute group
KUMPINPT (dp_InputData)	Sends the data buffer to the API Server Data Provider
KUMPPING (dp_Ping)	Checks the operational status of the API Server Data Provider
KUMPRDFN (dp_Redefine)	Refreshes a metafile
KUMPRREQ (dp_ReceiveRequest)	Queries the API Server Data Provider for any outstanding action requests targeted for this client application.
KUMPRTDA (dp_ReturnData)	Uses the Request ID value obtained by KUMPRREQ and sends response data back to the API Server Data Provider for the originating report or situation request.
KUMPSHOW (dp_ShowMessage)	Displays the text of an API return code status message

KUMPAREQ (dp_AcceptRequest)

Description

KUMPAREQ allows a client program to indicate its ability to accept inbound requests from the data provider that arise as the result of IBM Tivoli Monitoring automation actions, situation starts and stops, and demand-driven report requests. This call is valid only after KUMPBGNI has been successfully completed.

Input parameters

KUMPAREQ requires the following two input parameters:

- p1 The EnableRequest flag, signifying which types of inbound requests to receive. This flag can have one of the following integer values:
- 0 - do not send any action requests
 - 1 - send information about automation actions and situations being stopped and started
 - 2 - send information about automation actions, situations being stopped and started, and demand-driven report requests
- p2 The ReplyWaitTime, signifying the maximum elapsed time, in seconds, that the data provider must wait for the client program to respond to an action request.

Return status code

KUMPAREQ returns the following possible status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_Required_Parm_Missing

Either the EnableRequest flag or ReplyWaitTime value or both were not input.

KUMPBGNI (dp_BeginInput)

Description

KUMPBGNI signals the data provider that the caller is ready for data input for a specific application and attribute group. It binds a handle to an application and attribute group until the KUMPENDI call.

After the API Server Data Provider receives the dp_BeginInput request, it registers the data source with the Tivoli Universal Agent and the Tivoli Universal Agent notifies the Tivoli Enterprise Monitoring Server that the data source is online.

If you are running multiple API client scripts from the same computer, and these scripts are all sending data to the same metafile application, you must supply a unique source name parameter for each script by using the third parameter to KUMPBGNI. The unique source name associated with each script allows the API Server Data Provider to manage the different script input without ambiguities.

Input parameters

KUMPBGNI requires two input parameters and has one optional parameter:

- p1 The application name exactly as defined in the metafile APPL statement
- p2 The attribute group name exactly as defined in the metafile NAME statement.
- p3 (optional) The source name to be used when registering the managed system. This name is used in place of the default source name, which is the local host where the API client script is running.

Return status code

KUMPBGNI returns the following status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_Required_Parm_Missing

Either the application name or the attribute group name, or both, were not input.

KUMP_API_Request_TimedOut

The API Server Data Provider is not active.

KUMP_API_Environment_Init_Failed

API initialization failed because of incomplete or incorrect product installation.

KUMP_API_HostName_Unresolved

TCP/IP API transport is selected, but the local host name cannot be resolved to an IP address. Either the host name is not configured correctly, or the DNS/HOSTS table is not set up correctly for name resolution.

KUMP_API_Main_Storage_Unavailable

Not enough system storage is available for allocating the Tivoli Universal Agent API anchor control block.

KUMP_API_AllocHandle_No_Storage

Not enough system storage is available for allocating the Tivoli Universal Agent API handle.

KUMP_API_BgnInput_Application_Not_Found

The application specified is not valid or the defining metafile is not defined to the API Server Data Provider. If the name is correct, use KUMPDEFN first to define the metafile to the data provider or add the name of the metafile to the startup configuration file.

KUMP_API_BgnInput_AttrGroup_Not_Found

The attribute group specified is not valid or the defining metafile is not defined to the API Server

Data Provider. If the name is correct, use KUMPDEFN first to define the metafile to the data provider or add the metafile name to the startup configuration file.

KUMP_API_Server_Source_No_Storage

The API Server Data Provider is unable to obtain enough storage to process this application request.

KUMP_API_BgnInput_Server_App_InitFailed

The API Server Data Provider encountered an error while initializing the processing environment for supporting the application data input. Retry is not possible.

KUMP_API_BgnInput_Server_Register_Failed

The API Server Data Provider received a unrecoverable error return code from the Tivoli Universal Agent while performing application registration. Retry is not possible.

KUMP_API_Server_State_Logic_Error

The dp_BeginInput is called out of sequence or is using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

KUMPDEFN (dp_Define)

Description

KUMPDEFN requests that the API Server Data Provider load the specified data definition metafile and prepare it for data input.

You need to call KUMPDEFN only once. You can omit the KUMPDEFN call altogether if the metafile is loaded when the API Server Data Provider starts up.

Input parameters

KUMPDEFN requires one input parameter:

- p1** The name of the data definition metafile. The metafile must exist and be available to the API Server Data Provider.

Return status code

KUMPDEFN returns the following possible status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_DP_Inactive

The API Server Data Provider is not running.

KUMP_API_Define_Appl_Exist

The metafile specified is already defined to the API Server Data Provider. To refresh the data definition, use KUMPRDFN.

KUMP_API_Define_Appl_NotFound

The metafile does not exist or cannot be located by the the API Server Data Provider.

KUMP_API_Define_Spec_Error

The API Server Data Provider detected a file syntax or specification error in the metafile. The metafile data definition is rejected.

KUMPENDI (dp_EndInput)

Description

KUMPENDI signals the data provider that the caller has completed data input for a specific application and attribute group.

When the API Server Data Provider receives the dp_EndInput request, it unregisters the caller (data source) and the Tivoli Universal Agent notifies the Tivoli Enterprise Monitoring Server that the data source is offline.

Input parameters

KUMPENDI requires either zero or two input parameters. You do not need to specify any input parameters if there is only one application or one attribute group active:

- p1 The application name exactly as defined in the application metafile APPL statement
- p2 The attribute group name exactly as defined in the application metafile NAME statement

Return status code

KUMPENDI returns the following possible status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_Required_Parm_Missing

Either the application name or the attribute group name, or both, were not input.

KUMP_API_EndInput_Ambiguous

More than one application or attribute group is currently active, and no application name or no attribute group name was specified on input.

KUMP_API_EndInput_Application_Not_Active

The specified application is not active. KUMPBGNI or KUMPINPT was not called or ended unsuccessfully.

KUMP_API_EndInput_Server_Unregister_Failed

The API Server Data Provider received an irrecoverable error return code while performing application unregistration. Retry is not possible.

KUMP_API_Server_State_Logic_Error

dp_EndData is called out of sequence or is using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

KUMPINPT (dp_InputData)

Description

KUMPINPT sends input data to the API Server Data Provider.

Input parameters

KUMPINPT requires one or three input parameters. You do not need to specify application name and attribute group name if there is only one application or one attribute group active.

- p1 The data record to be sent to the API server. The input data must be in the format defined by the metafile, including any delimiters. Character input restrictions enforced by the command interface of the operating system can limit the choice of field delimiters.
- p2 The name of the application exactly as defined in the data definition metafile APPL statement
- p3 The name of the attribute group name exactly as defined in the data definition metafile NAME statement

Return status code

KUMPINPT returns the following possible status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_Required_Parm_Missing

Either the application name or the attribute group name or both were not input.

KUMP_API_Input_Ambiguous

More than one application or attribute group is currently active, and no application name or attribute group name was specified as input.

KUMP_API_Request_TimedOut

The API Server Data Provider is not active.

KUMP_API_Environment_Init_Failed

API initialization failed because of incomplete or incorrect product installation.

KUMP_API_HostName_Unresolve

TCP/IP API transport is selected, but the local host name cannot be resolved to an IP address. Either the host name is not configured correctly, or the DNS/HOSTS table is not set up correctly for name resolution.

KUMP_API_Main_Storage_Unavailable

Not enough system storage is available for allocating the API anchor control block.

KUMP_API_AllocHandle_No_Storage

Not enough system storage is available for allocating the data provider handle.

KUMP_API_BgnInput_Application_Not_Found

The application specified is not valid, or the defining metafile is not defined to the API Server Data Provider. If the name is correct, use the KUMPDEFN command first to define the metafile to the data provider, or add the metafile name to the data provider startup configuration file.

KUMP_API_BgnInput_AttrGroup_Not_Found

The attribute group specified is not valid, or the defining metafile is not defined to the API Server Data Provider process. If the name is correct, use the KUMPDEFN command first to define the metafile to the API server or add the metafile name to the startup configuration file.

KUMP_API_Server_Source_No_Storage

The API Server Data Provider is unable to obtain enough storage to process this application request.

KUMP_API_BgnInput_Server_App_InitFailed

The API Server Data Provider encountered an error while initializing the processing environment for supporting the application data input. Retry is not possible.

KUMP_API_BgnInput_Server_Register_Failed

The API Server Data Provider received an unrecoverable error return code while performing application registration. Retry is not possible.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider because of incomplete or incorrect local configuration.

KUMP_API_Server_Data_Emit_Error

The API Server Data Provider received an unrecoverable error when sending data. Retry is possible.

KUMP_API_Server_Data_Format_Error

The API Server Data Provider detected data specification errors in the application data. The input data does not match the data definition specified in the application metafile.

KUMP_API_Server_State_Logic_Error

dp_InputData is called out of sequence or using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

KUMPPING (dp_Ping)

Description

KUMPPING checks the operational status of the API Server Data Provider. Although it behaves in a manner similar to the TCP/IP ping command, it does not use the ping command for its operation.

Input parameters

KUMPPING requires no parameters.

Return status code

KUMPINPT returns the following possible status codes:

KUMP_API_Ping_Success

The API Server Data Provider is active.

KUMP_API_Request_TimedOut

The API Server Data Provider is not active.

KUMPRDFN (dp_Redefine)

Description

The KUMPRDFN requests that the API Server Data Provider load the specified data definition metafile and prepare it for data input.

You need to call KUMPRDFN only once. You can omit the KUMPRDFN call altogether if the application data definition is loaded at API Server Data Provider startup.

Input parameters

KUMPRDFN requires only the following input parameter:

- p1** The name of the data definition metafile. The metafile must exist and be available to the API Server Data Provider.

Return status code

KUMPRDFN returns the following possible status codes:

KUMP_API_OK
 dp_Redefine completed successfully.

KUMP_API_Invalid_Parameter
 The input MetaFileName parameter is not specified.

KUMP_API_Redefine_Appl_NotDefined
 The redefining application is not defined to the API Server Data Provider. If the metafile specification is correct, use KUMPDEFN (dp_Define) instead of KUMPRDFN.

KUMP_API_Redefine_Appl_NotFound
 The metafile does not exist or cannot be located by the API Server Data Provider.

KUMP_API_Redefine_Spec_Error
 The API Server Data Provider detected a syntax or specification error in the metafile. The metafile definition is rejected.

KUMPRREQ (dp_ReceiveRequest)

Description

KUMPRREQ queries the API Server Data Provider for any outstanding action requests targeted for this client application. This call is valid only after KUMPAREQ has been successfully completed.

Input parameters

KUMPRREQ requires the following two input parameters:

- p1 The SyncRequest flag to select synchronous or asynchronous behavior. This flag can have one of the following integer values:
 - 0 - asynchronous, which means the data provider will be polled for outstanding requests. If an inbound request is available, it is received.
 - 1 - synchronous, which means the calling program is willing to wait indefinitely until an inbound request becomes available and is received.
- p2 The OutputRequestBuffer flag to indicate whether or not the received request buffer should be stored in a local file. This flag can have one of the following integer values:
 - 0 - do not store the received request buffer in a local file.
 - 1 - store the received request buffer in a local file with the name API_REQUEST#nnn, where nnn is the Request ID value in the return code.

Return status code

KUMPRREQ returns either an integer > 0, representing the Request ID value that can then be passed to KUMPRTDA, or 0, which means that KUMPRREQ call returned without receiving an action request.

KUMPRTDA (dp_ReturnData)

Description

KUMPRTDA uses the Request ID value obtained by KUMPRREQ and sends response data back to the API Server Data Provider for the originating report or situation request.

Input parameters

KUMPRTDA requires the following four input parameters:

- p1 The Request ID for which the returned data is associated. This must be an integer > 0 and it can be obtained from a previous KUMPRREQ call.
- p2 The data record to be returned to the API server. The data must be in the format defined by the metafile, including any delimiters. Character input restrictions enforced by the command interface of the operating system can limit the choice of field delimiters.
- p3 The name of the application exactly as defined in the data definition metafile APPL statement
- p4 The name of the attribute group exactly as defined in the data definition metafile NAME statement

Return status code

KUMPRTDA returns the following possible status codes:

KUMP_API_OK

Command completed successfully.

KUMP_API_Required_Parm_Missing

One of the required parameters was not input.

KUMP_API_Input_Ambiguous

More than one application or attribute group is currently active, and no application name or attribute group name was specified as input.

KUMP_API_Request_TimedOut

The API Server Data Provider is not active.

KUMP_API_Environment_Init_Failed

API initialization failed because of incomplete or incorrect product installation.

KUMP_API_HostName_Unresolve

TCP/IP API transport is selected, but the local host name cannot be resolved to an IP address. Either the host name is not configured correctly, or the DNS/HOSTS table is not set up correctly for name resolution.

KUMP_API_Main_Storage_Unavailable

Not enough system storage is available for allocating the API anchor control block.

KUMP_API_AllocHandle_No_Storage

Not enough system storage is available for allocating the data provider handle.

KUMP_API_BgnInput_Application_Not_Found

The application specified is not valid, or the defining metafile is not defined to the API Server Data Provider. If the name is correct, use the KUMPDEFN command first to define the metafile to the data provider, or add the metafile name to the data provider startup configuration file.

KUMP_API_BgnInput_AttrGroup_Not_Found

The attribute group specified is not valid, or the defining metafile is not defined to the API Server Data Provider process. If the name is correct, use the KUMPDEFN command first to define the metafile to the API server or add the metafile name to the startup configuration file.

KUMP_API_Server_Source_No_Storage

The API Server Data Provider is unable to obtain enough storage to process this application request.

KUMP_API_BgnInput_Server_App_InitFailed

The API Server Data Provider encountered an error while initializing the processing environment for supporting the application data input. Retry is not possible.

KUMP_API_BgnInput_Server_Register_Failed

The API Server Data Provider received an unrecoverable error return code while performing application registration. Retry is not possible.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider because of incomplete or incorrect local configuration.

KUMP_API_Server_Data_Emit_Error

The API Server Data Provider received an unrecoverable error when sending data. Retry is possible.

KUMP_API_Server_Data_Format_Error

The API Server Data Provider detected data specification errors in the application data. The input data does not match the data definition specified in the application metafile.

KUMP_API_Server_State_Logic_Error

dp_InputData is called out of sequence or using an obsolete data provider handle.

KUMP_API_Server_Incompatible_Version

The API client and the API Server Data Provider are incompatible, or the current API call is not supported by the active API Server Data Provider.

KUMPSHOW (dp_ShowMessage)

Description

KUMPSHOW displays a description of the data provider API message on the command console.

Input parameters

KUMPSHOW requires only the following parameter:

p1 The message status code to be displayed.

Return status code

KUMPSHOW always returns a status code of zero.

Appendix A. Sample programs

This appendix contains the following sample programs:

- A C program, RoutStat, which demonstrates the use of the Tivoli Universal Agent application programming interfaces (APIs)
- A shell script that demonstrates the use of API console commands on a UNIX platform
- A Visual Basic script that demonstrates demand-driven data collection

About the sample C program

The RoutStat sample C program demonstrates how to use the Tivoli Universal Agent data provider API set. This program inputs data to the data provider based on the metafile RTstat.mdl, shown in Figure 3 on page 61.

After the sample program is initialized and contacts the API Server Data Provider, it randomly selects a router name from a list of five. It then proceeds to generate the remaining four attribute values. This input process is repeated for a number of iterations based on program call input parameters or the default of 1000 times. A delay interval between each sample input ranging from 15 to 120 seconds is also randomly determined. At the end of input iteration, the program calls dp_EndInput and exits.

APIs called in sample program

The RoutStat program exercises most of the APIs. The following calls were made in the sample program:

```
dp_OpenSession
dp_CloseSession
dp_Define
dp_Redefine
dp_AllocateBuffer
dp_FreeBuffer
dp_ClearBuffer
dp_FormatBufferData
dp_AllocateHandle
dp_FreeHandle
dp_BeginInput
dp_EndInput
dp_InputData
dp_Ping
dp_ShowMessage
dp_AcceptRequest
dp_ReceiveRequest
```

```
APPL RTstatus
NAME RTnetIO p 120
ATTRIBUTES ';'
RouterName      D 32
PktReceivedPerSec C 1000000
PktRoutedPerSec  C 1000000
PktDiscardedPerSec C 1000000
AvgPktSize      C 100000
```

Figure 3. The metafile RTstat.mdl

The RoutStat sample C program

```
#if 0
```

```
/******
```

```
IBM Confidential.
```

```
OCO Source Materials
```

```
5724-K10
```

```
Copyright IBM Corp. 2005
```

```
The source code for this program is not published or otherwise  
divested of its trade secrets, irrespective of what has  
been deposited with the U.S. Copyright Office.
```

```
Candle Copyright (c) 1997, 2004. All rights reserved.
```

```
*****
```

```
Tivoli Universal Agent Sample Test Program--RoutStat
```

This sample program demonstrates the basic usage of Tivoli Universal Agent APIs.

The program uses a random number generator to randomly select one of five statically defined RouterNames. It then randomly creates router I/O statistics and forwards the data record to the API Server Data Provider. A record interval between 15 to 120 seconds is also randomly chosen.

The total number of iterations can be controlled by the first input parameter. If it is not specified, a default of 1000 is used.

The second input parameter controls the use of connection-oriented or connectionless communication methodology, that is, whether the APIs `dp_OpenSession` and `dp_CloseSession` are used (Yes or No). The default is connectionless (No). If the first input parameter is omitted, the second input parameter assumes the first input parameter position.

```
*****/
```

```
#endif
```

```
#include <stdio.h>
```

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

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

```
#include <time.h>
```

```
#ifdef WIN32
```

```
#include <windows.h>
```

```
void sleep(int wait)
```

```
{
```

```
    Sleep(wait);
```

```
    return;
```

```
}
```

```
#endif
```

```
#include "kumpapi.h"
```

```
#define ConnectionMode 'Y'
```

```
int main(argc, argv)
```

```
int argc;
```

```
char *argv[];
```

```
{
```

```
    int API_Status, InputCount, MaxInputCount = 100, WaitTime = 30;
```

```
    int NameIndex, PKTrcvd, PKTrout, PKTdrop, PKTsize, DataInterval;
```

```
    int RequestReplyWait = KUMP_API_PROVIDE_NO_RESULT; /* Assumes 60 second wait */
```

```
    int RequestDataSize;
```

```
    char RequestBuffer[512];
```

```
    char CommunicationMode = ConnectionMode;
```

```
    char ApplicationName[] = "RTstatus";
```

```
    char AttrGroupName[] = "RTnetIO";
```

```
    char *RouterName[10] = {"Router-NYC", "Router-LAX", "Router-SFO",
```

```

"Router-ORD","Router-DFW", "Router-DFW",
"Router-ORD","Router-SFO","Router-LAX","Router-NYC"};
char *WorkBuffer;
dp_ReqID_t ReqID;
dp_handle_t dpHandle;
struct tm *NOW;
time_t ltime;

#if !defined(UNIX)
WaitTime *= 1000;
#endif

/*-----*/
/* Check input parameters */
/*-----*/
if (argv[1])
{
if (isdigit(*argv[1]))
{
MaxInputCount = atoi(argv[1]);
printf("MaxInputCount set to %d\n", MaxInputCount);
if (argv[2])
{
CommunicationMode = toupper(*argv[2]);
printf("CommunicationMode set to %c\n",communication mode);
}
else ;
}
else
CommunicationMode = toupper(*argv[1]);
}
else ;
/*-----*/
/* Check that the DP is active. If not, wait until it is ready. */
/*-----*/
while(!dp_Ping(&API_Status))
{
dp_ShowMessage(API_Status,NULL,0);
sleep(WaitTime);
}

/*-----*/
/* The DP is now operational. Define application data definitions. */
/* Omit this if the metafile is defined to DP through other means. */
/*-----*/
do
{
char MetaFileName[] = "c:\\candle\\cma\\metafiles\\rtstatus.mdl";
if (dp_Define(MetaFileName, 1, &API_Status)) ||
(API_Status == KUMP_API_Define_Appl_Exist))
/*-----*/
/* Successful metafile definition. Continue processing. */
/*-----*/
continue;

/*-----*/
/* Metafile definition error. Show message and exit. */
/*-----*/
dp_ShowMessage(API_Status, NULL, 0);
return(API_Status);

} while(0);
/*-----*/
/* Do DP application initialization procedures */
/*-----*/
do {
dpHandle = dp_AllocateHandle(&API_Status);

```

```

if (API_Status == KUMP_API_OK)
{
if (CommunicationMode == ConnectionMode)
if (!dp_OpenSession(dpHandle,&API_Status))
{
dp_ShowMessage(API_Status, NULL, 0);
return(API_Status);
}
else ;
else ;
/*-----*/
/* Allocate DP buffer. Assumes 80 bytes maximum user data.*/
/*-----*/
dp_AllocateBuffer(dpHandle, &WorkBuffer, 80, &API_Status);
if (API_Status == KUMP_API_OK)
{
dp_BeginInput(dpHandle,ApplicationName,AttrGroupName,&API_Status);
if (API_Status == KUMP_API_OK)
/*-----*/
/*Initialization complete.Begin data input loop.*/
/*-----*/
continue;
else ;
}
else ;
}
else ;

dp_ShowMessage(API_Status, NULL, 0);
return(API_Status);

} while(0);
/*-----*/
/* Indicate willingness to accept automation requests */
/*-----*/
if (!dp_AcceptRequest(dpHandle,KUMP_API_TRUE,RequestReplyWait,&API_Status))
{
dp_ShowMessage(API_Status, NULL, 0);
return(API_Status);
}
/*-----*/
/* DP data input loop */
/*-----*/
InputCount = 1;
while(InputCount <= MaxInputCount)
{
dp_ClearBuffer(WorkBuffer, &API_Status); /* ignore status */
NameIndex = abs((rand() * 9) / RAND_MAX);
dp_FormatBufferData(WorkBuffer,
RouterName[NameIndex],
strlen(RouterName[NameIndex]),
TypeIsCharacter,
&API_Status);
PKTrcvd = abs((rand() * 1000000) / RAND_MAX);
PKTrout = abs((rand() * PKTrcvd) / RAND_MAX);
PKTdrop = PKTrcvd - PKTrout;
PKTsize = (rand() * 100000) / RAND_MAX;
PKTsize = (PKTsize < 20) ? 20 : PKTsize;
dp_FormatBufferData(WorkBuffer, &PKTrcvd, sizeof(int),
TypeIsNumeric, &API_Status);
dp_FormatBufferData(WorkBuffer, &PKTrout, sizeof(int),
TypeIsNumeric, &API_Status);
dp_FormatBufferData(WorkBuffer, &PKTdrop, sizeof(int),
TypeIsNumeric, &API_Status);
dp_FormatBufferData(WorkBuffer, &PKTsize, sizeof(int),
TypeIsNumeric, &API_Status);
printf("Send data %04.4d ->%s\n",InputCount, WorkBuffer);

```

```

dp_InputData(WorkBuffer,&API_Status);
if (API_Status == KUMP_API_OK)
{
    InputCount++;
    DataInterval = (rand() / RAND_MAX) * 120;
    DataInterval = (DataInterval < 15) ? 15 : DataInterval;
    #if !defined(UNIX)
    sleep(DataInterval*1000);
    #else
    sleep(DataInterval);
    #endif
}
else
{
    dp_ShowMessage(API_Status, NULL, 0);
    break;
}
/*-----*/
/* Periodically poll for any action request */
/*-----*/
if ((InputCount % 3) == 0)
{
    while (dp_ReceiveRequest(dpHandle,
        KUMP_API_ASYNC_REQ, RequestBuffer, sizeof(RequestBuffer),
        &ReqID, &RequestDataSize, &API_Status))
    {
        int rc, j=0;
        char *Cptr;
        printf("Action Request Received! ID %d, Data
            Size %d, [%s]\n",ReqID,RequestDataSize,RequestBuffer);
        if (memcmp(RequestBuffer,"REQ(",4) == 0)
        {
            strcpy(RequestBuffer,RequestBuffer+4);
            Cptr = strchr(RequestBuffer,');
            memset(Cptr++, ' ',1);
            if (memcmp(Cptr,"PARM(",5) == 0)
            {
                strcpy(Cptr,Cptr+5);
                Cptr = strchr(Cptr,');
                if (Cptr)
                    memset(Cptr,'\0',1);
            }
            printf("Command->%s\n",RequestBuffer);
            rc = system(RequestBuffer);
            time(&lt;time);
            NOW = localtime(&lt;time);
            j += sprintf(RequestBuffer+j,"Request executed! ");
            j += sprintf(RequestBuffer+j,"Return code %d, ",rc);
            j += sprintf(RequestBuffer+j,"Local time %s",asctime(NOW));
            printf("Send request result-> ID %d %s",ReqID,RequestBuffer);
            if (!dp_SendRequestResult(dpHandle, &ReqID, RequestBuffer, strlen(RequestBuffer)-1, &API_Status))
                dp_ShowMessage(API_Status, NULL, 0);
        }
    }
    dp_ShowMessage(API_Status, NULL, 0);
}
/* end of while (InputCount <= MaxInputCount) */
/*-----*/
/* End data input and free allocated storage */
/*-----*/
dp_EndInput(dpHandle,&API_Status);
dp_FreeBuffer(WorkBuffer, &API_Status);

if (CommunicationMode == ConnectionMode)
    dp_CloseSession(dpHandle, &API_Status);

```

```

else ;

dp_FreeHandle(dpHandle, &API_Status);

return(0);
}

```

About the sample shell script

The Rout_Cmd shell script illustrates the use of API console commands on a UNIX platform. The IBM Tivoli Universal Agent binaries are installed into two different directories on UNIX platforms. Therefore, it is necessary to set and export environment variables before calling the API console commands.

The Rout_Cmd shell script uses the same RTSTATUS application as the previous sample C program. The script inputs data to the data provider based on the metafile RTstat.mdl, shown in the following figure.

```

APPL RTstatus
NAME RTnetIO p 120
ATTRIBUTES ','
RouterName      D 32
PktReceivedPerSec C 1000000
PktRoutedPerSec  C 1000000
PktDiscardedPerSec C 1000000
AvgPktSize      C 100000

```

Figure 4. The metafile RTstat.mdl

The Rout_Cmd shell script

```

#!/bin/sh
# This sample script demonstrates how to call UA API commands on Unix.
#
# Three required arguments are:
# 1 - install_dir directory
# 2 - platform architecture (for example: aix51, li6243, so1283)
# 3 - host name where IBM Tivoli Universal Agent is running
#
#####

CANDLHOME=/${1};          export install_dir
LD_LIBRARY_PATH=${install_dir}/tmaitm6/${2}/lib:${LD_LIBRARY_PATH}; export LD_LIBRARY_PATH
LIBPATH=${install_dir}/tmaitm6/${2}/lib:${LIBPATH};          export LIBPATH
SHLIB_PATH=${install_dir}/tmaitm6/${2}/lib:${SHLIB_PATH}      export SHLIB_PATH
KUMP_API_DPAPI_HOST=${3};          export KUMP_API_DPAPI_HOST
KUMP_API_VERBOSE=N;

# *** Change to the directory where the API command binaries are located
cd ${install_dir}/${2}/um/bin

# *** Check if API Server is listening at given host
./kumpping

# *** Register /import metafile
# *** Required only if metafile has not already been imported
./kumpdefn RTstatus

# *** Indicate beginning of data for attribute group RTnetIO
./kumpbgni RTstatus RTnetIO

# *** Send sample data
./kumpinpt "Westlake_router;3745;36495;28475;36491; RTstatus RTnetIO"
sleep 1

```

```

./kumpinput "Atlanta_router;3857;3983;29465;2810; RTstatus RTnetIO"
sleep 1
./kumpinput "Chicago_router;3726;9182;8475;26172; RTstatus RTnetIO"
sleep 1
./kumpinput "NewYork_router;3319;387;102;94857; RTstatus RTnetIO"
sleep 1
./kumpinput "Dallas_router;2984;9188;9163;1625; RTstatus RTnetIO"
./kumpendi RTstatus RTnetIO

```

About the sample VBScript

This VBScript illustrates the use of demand-driven data collection. The script waits for report requests from the Tivoli Universal Agent API Server Data Provider. When a request is received, the script uses Windows Management Instrumentation (WMI) to query disk data for network drive information and returns the data to the Tivoli Universal Agent. This script inputs data to the data provider based on the metafile NWDrives.mdl, shown in the following figure.

```

//APPL NWDrives
//NAME NetworkDrives K 300
//ATTRIBUTES ','
Name      D 32 Key
FileSystem      D 32
VolumeSerial      D 32
Size      D 32
FreeSpace      D 32
PercentFree      D 32

```

Figure 5. The metafile NWDrives.mdl

The NetworkDrives sample VBScript

```

NetworkDrives.vbs
'-----
' *** Definitions ***
'-----
dim disk
dim disks
dim diskSize
dim diskFreeSpace
dim diskFreeSpacePercent
dim SrvrName
dim requestID
dim requestEnv
dim rc
dim rowCounter

'-----
' *** Main Script ***
'-----
' Request the name of the system to gather data from
do
    SrvrName = "Localhost"
loop until SrvrName <> ""

set oshell = wscript.CreateObject("wscript.shell")

' Note: The "0" parameter in the following oshell.run commands is used to hide the
'       shell-created DOS command window. The "true" parameter means to halt script
'       execution until the called program finishes and returns control.

rc = oshell.run("kumpping", 0, true)
if rc <> 0 then
    wscript.echo "Status from kumpping is " & rc

```

```

        wscript.echo "Verify that the Universal Agent API Data Provider is active. Exiting..."
        wscript.quit
    end if

' Note: The NWDrives application name and NetworkDrives attribute group name should be
'       converted to runtime parameters and not hard-coded in this script. The kumpdefn
' call is required only if the metafile has not already been imported.
rc = oshell.run("kumpdefn NWDrives", 0, true)
if rc <> 0 then
    wscript.echo "Status from kumpdefn is " & rc
    wscript.echo "Verify that the application metafile is known to the Universal Agent API DP.
Exiting..."
    wscript.quit
end if

rc = oshell.run("kumpendi NWDrives NetworkDrives", 0, true)
if rc > 1 then
    wscript.echo "Status from kumpendi is " & rc
end if

' Note: case matters in the kumpbgni ApplName and AttrGroup parameters.
rc = oshell.run("kumpbgni NWDrives NetworkDrives", 0, true)
if rc = 0 then
    wscript.echo "dp_BeginInput failed. Verify that the application
name and attribute group name are defined to the Universal Agent API DP.
Exiting..."
    wscript.quit
end if

' Note: First parameter is a flag integer defined in kumpapi.h. Can be either:
'       KUMP_API_FALSE 0
'       KUMP_API_TRUE  1
'       KUMP_API_REPORT 2
'       We're using KUMP_API_REPORT to specify demand-driven data collection.
'       The second parameter is the maximum elapsed time, in seconds, that the API DP
'       should wait for this program to send back the request execution result.

rc = oshell.run("kumpareq 2 30", 0, true)
if rc = 0 then
    wscript.echo "dp_AcceptRequest failed. Verify that the required in-
teger parameters have been specified. Exiting..."
    wscript.quit
end if

' Initialize the requestID value before entering the kumprreq loop
requestID = 0

do
    ' Block on the dp_ReceiveRequest call
    ' Note: First parameter is a flag integer defined in kumpapi.h. Can be either:
    '       KUMP_API_ASYNC_REQ 0
    '       KUMP_API_SYNC_REQ  1

    '       Second parameter is a flag integer defined in kumpapi.h. Can be either:
    '       KUMP_API_OUTPUT_REQUEST_BUFFER_FALSE 0
    '       KUMP_API_OUTPUT_REQUEST_BUFFER_TRUE  1

    requestID = oshell.run("kumprreq 1 1", 0, true)
    if requestID = 0 then
        wscript.echo "dp_ReceiveRequest failed. Exiting..."
        wscript.quit
    end if

    ' Call the subroutine that collects and returns the network drive data
    GetDriveInfo(requestID)

    ' Note: the following until condition is superfluous because if requestID

```

```

'         ever equals zero, the script will exit immediately.

loop until requestID = 0

Sub GetDriveInfo(RequestID)

wscript.echo "Retrieving network drive information..." & vbCrLf

rowCounter = 1

' Note: drivetype "4" means Network Drive

set disks = GetObject("winmgmts:{impersonationLevel=impersonate}!\\&SrvrName&_\"
root\cimv2").ExecQuery("Select * from Win32_LogicalDisk WHERE drivetype = '4'")

for each disk in disks
    diskSize = round(disk.size/1048576, 0) & "MB"
    diskFreeSpace = round(disk.freespace/1048576, 0) & "MB"
    diskFreeSpacePercent = round(disk.freespace/disk.size * 100, 2) & "%"

    ' We should only use kumprtda with the requestID for the last row of data.
    ' Any previous rows should use standard kumpinpt program. So if the current
    ' rowCounter equals the total number of returned disks, we use kumprtda.

    if rowCounter = disks.Count then
        wscript.echo "Calling kumprtda with requestID " & RequestID
        ' Note: The space after the requestID is needed so that
        '         kumprtda will correctly parse the input arguments.
        rc = oshell.run("kumprtda " &
            RequestID & "-" & _
            disk.name & ";" & _
            disk.filesystem & ";" & _
            disk.volumeserialnumber & ";" & _
            diskSize & ";" & _
            diskFreeSpace & ";" & _
            diskFreeSpacePercent & ";" & _
            " NWDrives NetworkDrives", _
            0, true)

        if rc > 1 then
            wscript.echo "Status from kumprtda is " & rc
        end if
    else
        rc = oshell.run("kumpinpt " & _
            disk.name & ";" & _
            disk.filesystem & ";" & _
            disk.volumeserialnumber & ";" & _
            diskSize & ";" & _
            diskFreeSpace & ";" & _
            diskFreeSpacePercent & ";" & _
            " NWDrives NetworkDrives", _
            0, true)

        if rc > 1 then
            wscript.echo "Status from kumpinpt is " & rc
        end if
    end if
    rowCounter = rowCounter + 1
next

rc = oshell.run("kumpendi NWDrives ...")
if rc > 1 then
    wscript.echo "Status from kumpendi is " & rc
end if

End Sub

```

```
'-----  
' *** End Main ***  
'-----
```

Appendix B. Notices

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

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

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

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

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

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

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

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information in softcopy form, the photographs and color illustrations might not appear.

Trademarks

AIX, IBM, the IBM logo, OMEGAMON, Tivoli, the Tivoli logo, and z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.



Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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

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

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

Index

A

- accessibility xi
- API client package 2
- API function calls
 - dp_AcceptRequest 10
 - dp_AllocateBuffer 12
 - dp_AllocateHandle 13
 - dp_BeginInput 14
 - dp_BeginSample 16
 - dp_CancelRequest 17
 - dp_ClearBuffer 18
 - dp_ClearPriorData 19
 - dp_CloseSession 20
 - dp_Define 21
 - dp_EndInput 22
 - dp_EndSample 24
 - dp_FormatBufferData 25
 - dp_FreeBuffer 27
 - dp_FreeHandle 28
 - dp_InputData 29
 - dp_OpenSession 31
 - dp_Ping 32
 - dp_ReceiveRequest 33
 - dp_Redefine 32
 - dp_ResetHandle 38
 - dp_ReturnData 39
 - dp_SendRequestResult 41
 - dp_SetSourceName 43
 - dp_ShowMessage 45
- API Server Data Provider
 - specifying host 4
 - specifying port 4
- application data
 - combining in batches 30

B

- books
 - feedback on ix
 - online ix
 - ordering ix
 - see publications x

C

- C programs
 - RoutStat sample 61, 62
- client package 2
- command-line interface calls 47, 60
 - KUMPAREQ 48
 - KUMPBGNI 49
 - KUMPDEFN 51
 - KUMPENDI 52
 - KUMPINPT 53
 - KUMPPING 55
 - KUMPRDFN 56
 - KUMPRREQ 57

- command-line interface calls (*continued*)
 - KUMPRTDA 58
 - KUMPSHOW 60
- connecting to data provider 7
- console commands
 - minimum input 47
 - syntax 47
- conventions
 - typeface xii

D

- data collection
 - demand-driven 2
 - interval-driven 2
- demand-driven data collection 2
- directory names, notation xii
- dp_AcceptRequest 10
 - description 10
 - parameters 10
 - prototype 10
 - status codes 10
 - synopsis 10
- dp_AllocateBuffer 12
 - description 12
 - input parameters 12
 - status codes 12
 - synopsis 12
- dp_AllocateHandle
 - description 13
 - input parameters 13
 - prototype 13
 - status codes 13
 - synopsis 13
- dp_BeginInput
 - description 14
 - input parameters 14
 - prototype 14
 - status codes 14
 - synopsis 14
- dp_BeginSample
 - description 16
 - input parameters 16
 - prototype 16
 - status codes 16
 - synopsis 16
- dp_CancelRequest
 - description 17
 - input parameters 17
 - prototype 17
 - status codes 17
 - synopsis 17
- dp_ClearBuffer
 - description 18
 - input parameters 18, 19
 - prototype 18
 - status codes 18
 - synopsis 18

- dp_ClearPriorData
 - description 19
 - prototype 19
 - status codes 19
 - synopsis 19
- dp_CloseSession
 - description 20
 - input parameters 20
 - prototype 20
 - status codes 20
 - synopsis 20
- dp_Define
 - description 21
 - input parameters 21
 - prototype 21
 - status codes 21
 - synopsis 21
- dp_EndInput
 - description 22
 - input parameters 22
 - prototype 22
 - status codes 22
 - synopsis 22
- dp_EndSample
 - description 24
 - input parameters 24
 - prototype 24
 - status codes 24
 - synopsis 24
- dp_FormatBufferData
 - description 25
 - input parameters 25
 - prototype 25
 - status codes 25
 - synopsis 25
- dp_FreeBuffer
 - description 27
 - input parameters 27
 - prototype 27
 - status codes 27
 - synopsis 27
- dp_FreeHandle
 - description 28
 - input parameters 28
 - prototype 28
 - status codes 28
 - synopsis 28
- dp_InputData
 - combining multiple application data in batches 30
 - description 29
 - input parameters 29
 - prototype 29
 - status codes 29
 - synopsis 29
- dp_OpenSession
 - description 31
 - input parameters 31
 - prototype 31
 - status codes 31
 - synopsis 31

- dp_Ping
 - description 32
 - input parameters 32
 - prototype 32
 - status codes 32
 - synopsis 32
- dp_ReceiveRequest 33
 - description 33
 - input parameters 33
 - prototype 33
 - return status codes 34
 - synopsis 33
- dp_Redefine
 - description 37
 - input parameters 37
 - prototype 37
 - status codes 37
 - synopsis 37
- dp_ResetHandle
 - description 38
 - input parameters 38
 - prototype 38
 - status codes 38
 - synopsis 38
- dp_ReturnData 39
 - description 39
 - input parameters 39
 - prototype 39
 - return status codes 39
 - synopsis 39
- dp_SendRequestResult 41
 - description 41
 - prototype 41
 - return status codes 41
 - synopsis 41
- dp_SetSourceName 43
 - description 43
 - input parameters 43
 - prototype 43
 - status codes 43
 - synopsis 43
- dp_ShowMessage
 - description 45
 - input parameters 45
 - prototype 45
 - status codes 45
 - synopsis 45

E

- education
 - see Tivoli technical training xi
- environment variables, notation xii
- examples
 - combining multiple application data in batches 30
 - verbose trace option 5

I

- invoking API
 - manual commands 3

invoking API (*continued*)
 program function calls 3
 script file calls 3

K

KUMP_API_BYPASS_VAL 5
KUMP_API_DPAPI_HOST 4
KUMP_API_DPAPI_PORT 4
KUMP_API_REPORT 10
KUMPAREQ 48
 description 48
 input parameters 48
 status codes 48
KUMPBGNI 49
 description 49
 input parameters 49
 status codes 49
KUMPCNFG file 4
KUMPDEFN 51
 description 51
 input parameters 51
 status codes 51
KUMPENDI 52
 description 52
 input parameters 52
 status codes 52
KUMPINPT 53
 description 53
 input parameters 53
 status codes 53
KUMPPING 55
 description 55
 input parameters 55
 status codes 55
KUMPRDFN 56
 description 56
 input parameters 56
 status codes 56
KUMPRREQ 57
 description 57
 input parameters 57
 status codes 57
KUMPRTDA 58
 description 58
 input parameters 58
 status codes 58
KUMPSHOW 60
 description 60
 input parameters 60
 status codes 60

L

legal notices 71

M

manuals
 feedback on ix
 online ix

manuals (*continued*)
 ordering ix
 see publications x
metafiles
 NWDrives.mdl 67
multiple attribute groups 7
multiple row data 8

N

NetworkDrives VBScript sample 67
notation
 environment variables xii
 path names xii
 typeface xii
NWDrives.mdl metafile 67

O

online publications
 accessing x
OPAL
 See Open Process Automation Library
Open Process Automation Library (OPAL) x
options, verbose trace 5
ordering publications ix, xi

P

path names, notation xii
problem determination
 bypassing parameter validation 5
 status of API Server Data Provider 5
 verbose trace 5
programming requirements
 multiple attribute groups 7
 multiple row data 8
publications
 accessing online x
 feedback on ix
 online ix
 ordering ix, xi

R

Rout_Cmd shell script sample 66
RoutStat sample C program 61, 62
RTstat.mdl 61, 66

S

samples
 C program APIs 61, 62
 combining multiple application data in batches 30
 NetworkDrives VBScript 67
 Rout_Cmd shell script 66
 RoutStat C program 61
 VBScript 67
scripts
 sample NetworkDrives VBScript 67

- scripts (*continued*)
 - sample Rout_Cmd shell 66
 - sample VBScript 67
- specifying metafiles
 - API call 4
 - console command 4
 - KUMPCNFG file 4

T

- Tivoli software information center x
- Tivoli technical training xi
- trace, verbose option 5
- trademarks 72
- training, Tivoli technical xi
- typeface conventions xii

U

- Universal Agent
 - client package 2
 - connecting to data provider 7
- Universal Agent API 1
 - handling multiple attribute groups 7
 - implementation 1
 - invoking 3
 - minimum program requirements 7
 - multiple row data 8

V

- variables, notation for xii
- VBScript sample 67
- verbose trace option example 5



Printed in USA

SC32-9461-01

