Contents

Tables  .................................................. 5
Figures .................................................. 7
Preface  .................................................... 9
About This Book .......................................... 10
Adobe Portable Document Format .................. 12
Documentation Conventions ......................... 14
Candle Customer Support and Satisfaction .......... 16
What’s New ............................................... 17

Chapter 1. API Introduction ............................. 19
Application Programming Interfaces ................. 20
Invoking the APIs ...................................... 23
Specifying API Server Data Provider Variables ..... 24
Specifying Metafiles to the API Server Data Provider . 26
Problem Determination .................................. 28

Chapter 2. API Descriptions ......................... 29
Using the Universal Agent API Functions ............ 31
dp_AcceptRequest ..................................... 36
dp_AllocateBuffer ..................................... 38
dp_AllocateHandle ...................................... 40
dp_BeginInput .......................................... 42
dp_BeginSample ........................................ 45
dp_ClearBuffer ......................................... 47
dp_CloseSession ........................................ 49
dp_Define ................................................. 51
dp_EndInput .............................................. 53
dp_EndSample ........................................... 55
dp_FormatBufferData ................................... 57
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Symbols in Command Syntax</td>
<td>15</td>
</tr>
<tr>
<td>Table 2</td>
<td>API Client Package</td>
<td>22</td>
</tr>
<tr>
<td>Table 3</td>
<td>Name and Location of Variables File by Platform</td>
<td>24</td>
</tr>
<tr>
<td>Table 4</td>
<td>Universal Agent APIs</td>
<td>34</td>
</tr>
<tr>
<td>Table 5</td>
<td>Universal Agent API Console Commands</td>
<td>87</td>
</tr>
</tbody>
</table>
## Figures

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIGURE 1</td>
<td>Components of the Universal Agent API</td>
<td>21</td>
</tr>
<tr>
<td>FIGURE 2</td>
<td>Implementation of API Server Data Provider</td>
<td>25</td>
</tr>
<tr>
<td>FIGURE 3</td>
<td>Sample Program for Inputting Batch Data</td>
<td>64</td>
</tr>
<tr>
<td>FIGURE 4</td>
<td>The metafile <code>RTstat.mdl</code></td>
<td>107</td>
</tr>
<tr>
<td>FIGURE 5</td>
<td>The metafile <code>NWDrives.mdl</code></td>
<td>114</td>
</tr>
</tbody>
</table>
The Universal Agent application programming interfaces (APIs) enable you to develop programs that send data to the API Server Data Provider for monitoring by OMEGAMON XE. The *Universal Agent API/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.
About This Book

Who should read this book

This guide is designed for system application programmers. It assumes that you

- have installed CandleNet Portal™
- have installed Universal Agent
- are familiar with the contents of the Universal Agent User’s Guide
- are familiar with basic CandleNet Portal concepts, tasks, and features

If you are unfamiliar with these concepts and tasks, please review Administering OMEGAMON Products: CandleNet Portal. You should also review the online CandleNet Portal Tour to become familiar with the product’s features and capabilities.

Documentation set information

Candle® provides a complete set of documentation for the Universal Agent. Each manual in this documentation set contains a specific type of information to help you use the product. Relevant manuals are the

- Universal Agent User’s Guide, V410, UM54-6769
  Introduces the features, workspaces, attributes, predefined situations, and take action commands for Universal Agent.

- Universal Agent SNMP Data Provider User’s Guide, V410, UM54-6770
  Introduces the SNMP Data Provider, enabling you to use the monitoring and automation capabilities of OMEGAMON XE to manage your network resources.

- Universal Agent API/Command Programming Reference Guide, V410, UM53-6771
  Explains the procedures for implementing the Universal Agent APIs and provides descriptions, syntax, and return status codes for the API calls and command line interface commands.
the documentation for CandleNet Portal that came with your Universal Agent product
Provides instructions for using the CandleNet Portal interface to monitor the enterprise.

the documentation for OMEGAMON XE that came with your Universal Agent product
Provides instructions for setting up the Candle Management Server® (CMS™).

the installation documentation that came with your Universal Agent product
Provides instructions for installing the product and the framework components.

Where to look for more information
For more information related to this product and other related products, please see the
- technical documentation CD-ROM that came with your product
- technical documentation information available on the Candle web site at www.candle.com
- online help provided with this and the other related products

Ordering additional documentation
To order additional product manuals, contact your Candle Support Services representative.

We would like to hear from you
Candle welcomes your comments and suggestions for changes or additions to the documentation set. A user comment form, located at the back of each manual, provides simple instructions for communicating with the Candle Information Development department. You can also send email to UserDoc@candle.com. Please include "Universal Agent API/Command Programming Reference Guide, V410" in the subject line.
About This Book

Adobe Portable Document Format

Printing this book

Candle supplies documentation in the Adobe Portable Document Format (PDF). The Adobe Acrobat Reader will print PDF documents with the fonts, formatting, and graphics in the original document. To print a Candle document, do the following:

1. Specify the print options for your system. From the Acrobat Reader Menu bar, select **File > Page Setup…** and make your selections. A setting of 300 dpi is highly recommended as is duplex printing if your printer supports this option.

2. To start printing, select **File > Print…** on the Acrobat Reader Menu bar.

3. On the Print pop-up, select one of the **Print Range** options for
   - All
   - Current page
   - Pages from: [ ] to: [ ]

4. (Optional). Select the Shrink to Fit option if you need to fit oversize pages to the paper size currently loaded on your printer.

Printing problems?

The print quality of your output is ultimately determined by your printer. Sometimes printing problems can occur. If you experience printing problems, potential areas to check are:

- settings for your printer and printer driver. (The dpi settings for both your driver and printer should be the same. A setting of 300 dpi is recommended.)
- the printer driver you are using. (You may need a different printer driver or the Universal Printer driver from Adobe. This free printer driver is available at www.adobe.com.)
- the halftone/graphics color adjustment for printing color on black and white printers (check the printer properties under **Start > Settings > Printer**). For more information, see the online help for the Acrobat Reader.
- the amount of available memory in your printer. (Insufficient memory can cause a document or graphics to fail to print.)

For additional information on printing problems, refer to the documentation for your printer or contact your printer manufacturer.
Contacting Adobe

If additional information is needed about Adobe Acrobat Reader or printing problems, see the Readme.pdf file that ships with Adobe Acrobat Reader or contact Adobe at www.adobe.com.

Adding annotations to PDF files

If you have purchased the Adobe Acrobat application, you can add annotations to Candle documentation in .PDF format. See the Adobe product for instructions on using the Acrobat annotations tool and its features.
Documentation Conventions

Introduction
Candle documentation adheres to accepted typographical conventions for command syntax. Conventions specific to Candle documentation are discussed in the following sections.

Panels and figures
The panels and figures in this document are representations. Actual product panels may differ.

Required blanks
The slashed-b (ёт) character in examples represents a required blank. The following example illustrates the location of two required blanks.

ёёёBA*ServiceMonitorё0990221161551000

Revision bars
Revision bars (|) may appear in the left margin to identify new or updated material.

Variables and literals
In examples of command syntax, uppercase letters are actual values (literals) that the user should type; lowercase letters are used for variables that represent data supplied by the user. Default values are underscored.

LOGON APPLID (cccccccc)

In the above example, you type LOGON APPLID followed by an application identifier (represented by cccccccc) within parentheses.

Note: In ordinary text, variable names appear in italics.
Symbols

The following symbols may appear in command syntax:

**Table 1. Symbols in Command Syntax**

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Usage</th>
</tr>
</thead>
</table>
| | The “or” symbol is used to denote a choice. Either the argument on the left or the argument on the right may be used. Example:  
  \textbf{YES I NO}  
  In this example, YES or NO may be specified. |
| \[ \] | Denotes optional arguments. Those arguments not enclosed in square brackets are required. Example:  
  \textbf{APPLDEST DEST [ALTDEST]}  
  In this example, DEST is a required argument and ALTDEST is optional. |
| \{ \} | Some documents use braces to denote required arguments, or to group arguments for clarity. Example:  
  \textbf{COMPARE \{workload\} -}  
  \textbf{REPORT=\{SUMMARY | HISTOGRAM\}}  
  The \textit{workload} variable is required. The REPORT keyword must be specified with a value of SUMMARY or HISTOGRAM. |
| _ | Default values are underscored. Example:  
  \textbf{COPY infile outfile - \{COMPRESS=\{YES I NO\}\}}  
  In this example, the COMPRESS keyword is optional. If specified, the only valid values are YES or NO. If omitted, the default is YES. |
Candle Customer Support and Satisfaction

Background
To assist you in making effective use of our products, Candle offers a variety of easy-to-use online support resources. The Candle Web site provides direct links to a variety of support tools that include a range of services. For example, you can find information about training, maintenance plans, consulting and services, and other useful support resources. Refer to the Candle Web site at www.candle.com for detailed customer service information.

Candle Customer Service and Satisfaction contacts
You will find the most current information about how to contact Candle Customer Service and Satisfaction by telephone or e-mail on the Candle Web site. Go to www.candle.com support section and choose the link to Support Contacts to locate your regional support center.
Overview

In Version 410, a variety of features improve the usability of the Universal Agent and make it possible to monitor data from additional sources. A new Data Provider monitors data from ODBC-compliant databases. Universal Agent also allows you to automatically generate ODBC metafiles, to collect SNMP MIB data from non-standard ports, and to collect demand-driven data from the API Server Data Provider.

ODBC Data Provider

Open Database Connectivity (ODBC) is a standard application programming interface for accessing data in relational data sources. The Universal Agent ODBC Data Provider, only available on Windows platforms, allows you to collect data from ODBC-compliant databases using SQL Select statements and stored procedures supported by the particular ODBC source that is being monitored.

Automatic Generation of ODBC Metafiles

The GENERATE command automatically builds a complete and syntactically correct ODBC metafile when given a data source name as input. This new command supports full generation of all tables defined to the specified data source. You can also limit which tables will be generated by selecting user tables, system tables, or views (or some combination of the three) and by specifying a beginning string of characters to pattern match against any of the three table types.
SNMP MIB Data Collection from Non-standard Ports

Certain vendor SNMP agents may feature special requirements, such as using ports other than the default monitoring port. Universal Agent’s SNMP Data Provider, which acts as an SNMP Manager that polls SNMP agents for MIB data, now supports configuration and polling of SNMP agents using ports other than 161 (the default port).

Demand-Driven Data Collection for the API Server Data Provider

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. The API Server Data Provider now supports interval-driven and demand-driven data collection. Collecting data "on demand" offers fresher data and potentially reduces unnecessary processing overhead.
Introduction

This chapter provides an introduction to the Universal Agent application programming interfaces (APIs). It discusses

- the implementation of the APIs
- invoking the APIs
- specifying the transport mode between client and server
- defining metafiles to an API Server Data Provider
- problem determination

Chapter contents

Application Programming Interfaces ......................................... 20
Invoking the APIs ................................................................. 23
Specifying API Server Data Provider Variables ........................... 24
Specifying Metafiles to the API Server Data Provider ................. 26
Problem Determination ............................................................. 28
Application Programming Interfaces

Overview

The Universal Agent Application Programming Interfaces (API) provide the basic tools for developing programs that send user-defined data to the Universal Agent.

This chapter provides a brief description of how the APIs are implemented. It also discusses how to invoke the API functions, how to specify the transport mode to be used between a client program and the API Server Data Provider, and how to specify to a Data Provider which metafiles it should load.

Implementation of the Universal Agent APIs

C/C++ calling programs using the Universal Agent API functions rely upon services provided by the Universal Agent runtime dynamic library. The implementation of the 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) which exports API functions. Client programs may reside on the same host as the API Server Data Provider or they may execute at a remote location. Figure 1 illustrates the relationships among the components.

The Universal Agent also supports a command line interface which 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. The API Server Data Provider supports interval-driven and demand-driven data collection.

You can use the API client package to build simple or complex programs that collect and send data to the API Data Provider. The API Data Provider acts as a server daemon inside the UA process that listens for client connections and responds to client requests. Collecting data "on demand" offers fresher data and potentially reduces unnecessary processing overhead.
The API client package

The API client package was developed in standard C language and requires only a common C runtime environment and TCP/IP with a socket interface. The client package consists of

- 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 as the client program, as well as on the machine where you compile the program if they are not the same.

To install the client package:

1. Navigate to the `\cma` or `/bin` directory where Universal Agent is installed.
2. Copy the files listed in Table 2 to the target machine.
### Table 2. API Client Package

<table>
<thead>
<tr>
<th>Part Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KUMPAPI.h</td>
<td>API C header file</td>
</tr>
<tr>
<td>KUMPAPI.lib</td>
<td>API function export library file</td>
</tr>
<tr>
<td>KUMPAPI.dll or libkumpapi</td>
<td>API function runtime dynamic link library</td>
</tr>
<tr>
<td>KUMPAREQ</td>
<td>dp_AcceptRequest console command</td>
</tr>
<tr>
<td>KUMPBGINI</td>
<td>dp_BeginInput console command</td>
</tr>
<tr>
<td>KUMPDEFN</td>
<td>dp_Define console command</td>
</tr>
<tr>
<td>KUMPENDI</td>
<td>dp_EndInput console command</td>
</tr>
<tr>
<td>KUMPINPT</td>
<td>dp_InputData console command</td>
</tr>
<tr>
<td>KUMPPING</td>
<td>dp_Ping console command</td>
</tr>
<tr>
<td>KUMPRDFN</td>
<td>dp_Redefine console command</td>
</tr>
<tr>
<td>KUMPRREQ</td>
<td>dp_ReceiveRequest console command</td>
</tr>
<tr>
<td>KUMPRTDA</td>
<td>dp_ReturnData console command</td>
</tr>
<tr>
<td>KUMPSHOW</td>
<td>dp_ShowMessage console command</td>
</tr>
</tbody>
</table>
Invoking the APIs

Overview
You can invoke the 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 routine processing procedures. However, the API command line interface implements only a subset of the API suite due to the discontinuous nature of the command interface. These limitations are discussed in detail later in this document.

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 would enter an API command with text. This causes the OMEGAMON XE situation to become true and the on-duty action team to be notified.
Specifying API Server Data Provider Variables

Name and location of variables file

Environment variables for the API Server Data Provider all begin with the string KUMP_API_DP_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 platform.

Table 3 contains the name and location of the variables file on each of the supported platforms. Consult Appendix D of the *Universal Agent User’s Guide* for further information on Universal Agent variables.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Location</th>
<th>Name/Member</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIX</td>
<td>Candle/config/</td>
<td>UM.CONFIG</td>
</tr>
<tr>
<td>Windows</td>
<td>Candle\cma\</td>
<td>KUMENV</td>
</tr>
</tbody>
</table>

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_DP_API_HOST to the host name of the API Server Data Provider. You may set them in the program or login script of the client host, or as part of the machine’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_DP_API_PORT to the preferred port. If you set this variable for the API Server Data Provider, you must set the same variable on the API client side.
Specifying API Server Data Provider Variables

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 DP's UAGENT DPLOG report. You would then specify that value in the KUMP_API_DPAPI_PORT environment variable on the API client side.

FIGURE 2. Implementation of API Server Data Provider
Specifying Metafiles to the API Server Data Provider

Overview
The Data Provider must load the appropriate data definition metafiles in order to support application data. There are 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 an optional configuration file named KUMPCNFG. KUMPCNFG is a free-form text file which contains a list of metafile names. (On case-sensitive platforms, the file name KUMPCNFG must be all upper case.)

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 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 Universal Agent console interface command IMPORT to load a new metafile or the command REFRESH to revise an existing metafile. If there are multiple Data Providers active at the same time, be sure to select the ASFS or APIS Data Provider as the target of the console command.
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.
Problem Determination

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 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 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_APIVERBOSE=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.

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 once 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.
Introduction

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

Chapter contents

Using the Universal Agent API Functions ........................................... 31
dp_AcceptRequest ................................................................. 36
dp_AllocateBuffer ............................................................... 38
dp_AllocateHandle ............................................................... 40
dp_BeginInput ................................................................. 42
dp_BeginSample ............................................................... 45
dp_ClearBuffer ................................................................. 47
dp_CloseSession ............................................................... 49
dp_Define ................................................................. 51
dp_EndInput ............................................................... 53
dp_EndSample ............................................................... 55
dp_FormatBufferData .......................................................... 57
dp_FreeBuffer ............................................................... 59
dp_FreeHandle ............................................................... 61
dp_InputData ............................................................... 62
dp_OpenSession ............................................................. 65
dp_Ping ............................................................... 67
dp_ReceiveRequest ...................................................... 68
dp_Redefine .............................................................. 72
dp_ResetHandle .......................................................... 74
dp_ReturnData ............................................................. 76
dp_SendRequestResult ..................................................... 79
dp_SetSourceName ......................................................... 81
dp_ShowMessages ......................................................... 84
Using the Universal Agent API Functions

Overview

This section discusses the programming requirements for contacting a Data Provider and formatting and sending data. The API calls are summarized in Table 4 on page 34.

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 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_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.
Using the Universal Agent API Functions

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 may consist of multiple data records. The calls dp_BeginSample and dp_EndSample act as brackets for the 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.
Using the Universal Agent API Functions

Table 4. Universal Agent APIs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Definition</strong></td>
<td></td>
</tr>
<tr>
<td>dp_Define</td>
<td>Defines a metafile to the Data Provider</td>
</tr>
<tr>
<td>dp_Redefine</td>
<td>Refreshes a metafile</td>
</tr>
<tr>
<td>dp_SetSourceName</td>
<td>Associates an unique source name with an application attribute group for data input</td>
</tr>
<tr>
<td><strong>Session</strong></td>
<td></td>
</tr>
<tr>
<td>dp_OpenSession</td>
<td>Initiates a connection between a client program and the API Server Data Provider</td>
</tr>
<tr>
<td>dp_CloseSession</td>
<td>Ends the connection between a client program and the API Server Data Provider</td>
</tr>
<tr>
<td><strong>Data Formatting</strong></td>
<td></td>
</tr>
<tr>
<td>dp_AllocateBuffer</td>
<td>Allocates a data transport buffer</td>
</tr>
<tr>
<td>dp_FreeBuffer</td>
<td>Releases an allocated data transport buffer</td>
</tr>
<tr>
<td>dp_ClearBuffer</td>
<td>Clears an allocated buffer of application data and control information</td>
</tr>
<tr>
<td>dp_FormatBufferData</td>
<td>Formats a data field in an allocated buffer for transmission</td>
</tr>
<tr>
<td>dp_BeginSample</td>
<td>Indicates the beginning of a multiple row data sample</td>
</tr>
<tr>
<td>dp_EndSample</td>
<td>Indicates the completion of a multiple row data sample</td>
</tr>
<tr>
<td><strong>Data Transfer</strong></td>
<td></td>
</tr>
<tr>
<td>dp_AllocateHandle</td>
<td>Allocates a handle for data exchange with a Data Provider</td>
</tr>
<tr>
<td>dp_FreeHandle</td>
<td>Frees a previously allocated handle</td>
</tr>
<tr>
<td>dp_ResetHandle</td>
<td>Resets the status of an allocated handle for reuse</td>
</tr>
<tr>
<td>dp_BeginInput</td>
<td>Identifies an attribute group for data input</td>
</tr>
<tr>
<td>dp_EndInput</td>
<td>Indicates the completion of data input for an attribute group</td>
</tr>
</tbody>
</table>
### Table 4. Universal Agent APIs (Continued)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dp_InputData</td>
<td>Sends the data buffer to the API Server Data Provider</td>
</tr>
<tr>
<td>dp_ReturnData</td>
<td>Sends the data buffer to the API Server Data Provider in response to an action request received by the dp_ReceiveRequest API.</td>
</tr>
<tr>
<td>Status</td>
<td></td>
</tr>
<tr>
<td>dp_Ping</td>
<td>Checks the operational status of the API Server Data Provider</td>
</tr>
<tr>
<td>dp_ShowMessage</td>
<td>Displays the text of an API return code status message</td>
</tr>
<tr>
<td>Request Activity</td>
<td></td>
</tr>
<tr>
<td>dp_AcceptRequest</td>
<td>Indicates that an application program is willing and able to accept requests from a Data Provider as the result of OMEGAMON XE automation actions or activities</td>
</tr>
<tr>
<td>dp_ReceiveRequest</td>
<td>Receives request from the Data Provider</td>
</tr>
<tr>
<td>dp_SendRequestResult</td>
<td>Sends result and status of request execution to the Data Provider</td>
</tr>
</tbody>
</table>
dp_AcceptRequest

Description

dp_AcceptRequest allows a client program to indicate its willingness to accept inbound requests from the Data Provider that arise as the result of OMEGAMON XE 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 supports 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, or dp_SendRequestResult API calls.

Synopsis (C)

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

Prototype (C)

```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 DP 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.
**dp_AcceptRequest**

**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 codes**

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 invalid 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)**

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

**Prototype (C)**

```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 DP 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.
**dp_AllocateBuffer**

```c
int *Status
```

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

**Return status codes**

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 invalid address.

- **KUMP_API_Invalid_Handle**
  - The input handle pointer is invalid 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 invalid, or insufficient heap storage remains for buffer allocation.
**dp_AllocateHandle**

**Description**

A Data Provider handle represents an application execution instance to the 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 may 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)**

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

**Prototype (C)**

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

**Input parameters**

`dp_AllocateHandle` requires one input parameter:

```c
int *Status
```

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

**Return status codes**

`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 due to 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.

Once the API Server Data Provider receives the `dp_BeginInput` request, it registers the caller and the Universal Agent notifies the Candle Management Server that the data source is online.

**Synopsis (C)**

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

**Prototype (C)**

```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 DP 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.
dp_BeginInput

int *Status

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

Return status codes

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 invalid.

KUMP_API_Invalid_Handle

The input handle pointer is invalid, 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 invalid, or the defining metafile is not defined to the DP 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 invalid, or the defining metafile is not defined to the DP 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.
dp_BeginInput

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.

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 DP 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 codes
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 invalid.
**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_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 codes

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.
dp_ClearBuffer

KUMP_API_Buffer_Invalid_Buffer_Pointer
   The UserBuffer parameter is an invalid 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_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 codes
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.


dp_CloseSession

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 DP 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 existing application definition known to the agent if the application is already defined, perhaps 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.
**dp_Define**

**Return status codes**

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 invalid address.

- **KUMP_API_DP_Inactive**
  
  The DP 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.

Once the API Server Data Provider receives the dp_EndInput request, it unregisters the caller and the Universal Agent notifies the CMS that the data source is offline.

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 DP 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 codes

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.
dp_EndInput

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.

**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 codes**
- `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 invalid.
*dp_EndSample*

**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.

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 binary, numeric, or character, as defined in the kumpapi.h header file. 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 codes

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 invalid, 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)**

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

**Prototype (C)**

```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 codes**

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 invalid or contains an invalid address.
dp_FreeBuffer

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 invalid, 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)**

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

**Prototype (C)**

```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 codes**

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 invalid or contains an invalid handle address.
**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 DP 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)**

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

**Prototype (C)**

```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 codes**

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

The following are possible status codes returned by dp_InputData calls:
**dp_InputData**

KUMP_API_OK

dp_InputData completed successfully.

KUMP_API_Input_NULL_Buffer

The UserBuffer parameter is invalid.

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 due to 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_Lockup_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 may be batched together for inputting to the Universal Agent in a single dp_InputData API call. The individual data records must be separated by a line feed character. This reduces turnaround time and improves overall response time and efficiency. The example in Figure 3 on page 64 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 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.

**FIGURE 3. Sample Program for Inputting Batch Data**

```c
int i,j;
char LineFeed '\n';
char *WorkBuffer;
... ...
... ...
... ...
dp_ClearBuffer(WorkBuffer, &API_Status);
j = 0;
do {
    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);
    dp_FormatBufferData(WorkBuffer, &LineFeed, 1,
        TypeIsCharacter, &API_Status);
} while(++j < 20); /* Batch 20 at a time */
dp_InputData(WorkBuffer,&API_Status);
```
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 codes
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
dp_OpenSession

The input handle is invalid.

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 installed incompletely, or the environment variable KUMP_API_TRANSPORT specified an invalid transport type.

KUMP_API_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider due to 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)**

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

**Prototype (C)**

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

**Input parameters**

dp_Ping requires only one parameter:

```c
int *Status
```

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

**Return status codes**

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 an invalid transport type.
**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's buffer storage in the following format:

```
REQ(application-request-name)PARM(request-parameter)
```

The PARM() is present only if the application request parameters are received in conjunction with the request. No interpretation or formatting is done on behalf of the caller to either the application request or the corresponding parameters, if any.

**Synopsis (C)**

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

**Prototype (C)**

```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 is willing to wait 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 will be saved. The buffer can be any storage allocated by the calling program or a buffer allocated by calling dp_AllocateBuffer API call. See "Data formats" on page 70 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 codes**

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.
dp_ReceiveRequest

KUMP_API_Invalid_Handle
The input handle pointer is invalid 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 invalid.

KUMP_API_ReceiveRequest_Inavlid_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 UserBufferSize 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 two formats.

- OMEGAMON XE situation predicate information
  - Add filter request
    
    \[\text{FILTER(ADD) PARM(INTERVAL=monitor\_interval APPLNAME=application\_name ATTRGROUP=attribute\_group\_name ATTRIBUTES=((attribute\_name operator value) (...) ...)}}\]
    
    where
    
    INTERVAL specifies the sampling interval for the situation, in seconds. INTERVAL is omitted for situations which monitor event type data.
    
    APPLNAME specifies the name of the Universal Agent application as defined in the APPL statement of the corresponding data definition metafile
ATTRGROUP specifies the name of the attribute group as defined in the NAME statement of the corresponding data definition metafile.

ATTRIBUTES specifies one or more situation predicate triplets in the form attribute name, logical operator, value, each enclosed in parentheses.

The received ReqID uniquely identifies this situation filter.

B. Delete filter request

FILTER(DELETE)

The received ReqID identifies the previously received filter to be deleted.

OMEGAMON XE automation action request

REQ(command) PARM(command)

where:

REQ specifies the command or request to be performed by the receiving program. The meaning and behavior of the command or request should be known by the receiving program.

PARM specifies the command or request arguments, if any. This field is omitted if no arguments are supplied.
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:

cchar  * 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, perhaps 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 codes

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

The following are possible status codes returned by dp_Redefine calls:
dp_Redefine

KUMP_API_OK
    dp_Redefine completed successfully.

KUMP_API_Invalid_Parameter
    The input MetaFileName parameter contains an invalid address.

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)**

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

**Prototype (C)**

```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 codes**

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.
dp_ResetHandle

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 DP handle.

Where dp_ReturnData differs is that it only sends a data buffer in response to an action request received by the dp_ReceiveRequest API. The action request must either be of the form: "REPORT(REQUEST) PARM(request-parameter)" or "SITUATION(REQUEST) PARM(request-parameter)".

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

**Synopsis (C)**

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

**Prototype (C)**

```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.
**dp_ReturnData**

```c
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.

```c
int *Status
```

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

**Return status codes**

`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 invalid.

- **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 due to 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_Lock_Error**
dp_ReturnData

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**

\(dp_{\text{SendRequestResult}}\) sends the outcome of a request to the API Server Data Provider. The result data must be in character form: all binary or numeric data must be converted to character form. No interpretation or data manipulation is performed on the input result data.

**Synopsis (C)**

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

**Prototype (C)**

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

**Parameters**

The following are required \(dp_{\text{SendRequestResult}}\) 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.
**dp_SendRequestResult**

`int *Status`

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

**Return status codes**

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 invalid 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 invalid.

- **KUMP_API_SendReqResult_Buffer_Truncated**
  
  The request result data is larger than the value of KUMP_API_MAX_REQ_RESULT_SIZE and it 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, perhaps 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 Universal Agent application is represented as a OMEGAMON XE managed system. Its name is constructed by concatenating the the name of the application program's host machine, 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 OMEGAMON XE 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 OMEGAMON XE.

```
dp_SetSourceName must be invoked before dp_BeginInput
```

Synopsis (C)

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

**Prototype (C)**

```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 DP 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 should not exceed 29 characters. The source name appears 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 codes**

`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 invalid, 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_Server_StateLogic_Error**
  
  The `dp_SetSourceName` API is called out of sequence or is using an obsolete DP handle.

- **KUMP_API_Server_Incompatible_Version**
  
  The `dp_SetSourceName` API is called out of sequence or is using an obsolete DP handle.
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_ShowMessages

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 codes

dp_ShowMessage returns TRUE if processing is successful. Otherwise, it returns FALSE.
Introduction

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

Appendix contents

Using API Console Commands ............................................. 86
KUMPAREQ (dp_AcceptRequest) ........................................ 88
KUMPBGNI (dp_BeginInput) ................................................. 89
KUMPDEFN (dp_Define) ..................................................... 92
KUMPENDI (dp_EndInput) ................................................... 93
KUMPINPT (dp_InputData) ................................................. 95
KUMPPING (dp_Ping) .......................................................... 98
KUMPRDFN (dp_Redefine) ................................................. 99
KUMPREQ (dp_ReceiveRequest) ......................................... 100
KUMPRTDA (dp_ReturnData) .............................................. 101
KUMPSHOW (dp_ShowMessage) ......................................... 104
Using API Console Commands

Overview
The 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 5 on page 87.

Command syntax
Console command syntax differs among operating systems. This guide documents only descriptions of the 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 platform-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'
```
Using API Console Commands

### Table 5. Universal Agent API Console Commands

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

**Description**

KUMPAREQ allows a client program to indicate its willingness to accept inbound requests from the Data Provider that arise as the result of OMEGAMON XE automation actions, situation starts/stops, and demand-driven report requests. This call is valid only after KUMPBGNI has been successfully completed.

**Input parameters**

KUMPAREQ requires 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 should wait for the client program to respond to an action request.

**Return status codes**

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.

Once the API Server Data Provider receives the dp_BeginInput request, it registers the data source with the Universal Agent and the Universal Agent notifies the Candle Management Server that the data source is online.

Input parameters

KUMPBGNI requires two input parameters:

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.

Return status codes

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_BGNI (dp_BeginInput)

KUMP_API_Environment_Init_Failed
API initialization failed due to 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 Universal Agent API anchor control block.

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

KUMP_API_BgnInput_Application_Not_Found
The application specified is invalid 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 invalid 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.
KUMPBGINI (dp_BeginInput)

KUMP_API_BgnInput_Server_Register_Failed

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

KUMP_API_Server_StateLogic_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.
**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 codes**

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 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 Universal Agent notifies the CMS 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 codes**

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.
KUMPENDI (dp_EndInput)

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 may 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 codes
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.
KUMPINPT (dp_InputData)

KUMP_API_Environment_Init_Failed
   API initialization failed due to 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 invalid, 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 invalid, 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 a 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 due to 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 codes**
KUMPPING 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 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 codes

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 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 codes
KUMPRREQ returns either an integer > 0, representing the Request ID value which 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 DP for the originating report or situation request.

Input parameters
KUMPRTDA requires 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 may 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 codes
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.
KUMPRTDA (dp_ReturnData)

KUMP_API_Request_TimedOut
The API Server Data Provider is not active.

KUMP_API_Environment_Init_Failed
API initialization failed due to 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 invalid, 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 invalid, 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.
KUMPRTDA (dp_ReturnData)

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_Input_Local_Transport_Setup_Error

The API client is unable to start communication with the API Server Data Provider due to 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
KUMP SHOW displays a description of the Data Provider API message on the command console.

Input parameters
KUMP SHOW requires one parameter:
p1 the message status code to be displayed.

Return status codes
KUMP SHOW always returns a status code of zero.
Sample Programs

Introduction

This appendix contains a sample C program, RoutStat, which demonstrates the use of the Universal Agent application programming interfaces (APIs) and a sample Visual Basic script, NetworkDrives, which illustrates demand-driven data collection.

Appendix contents

About the Sample C Program .............................................. 106
The RoutStat Sample C Program ............................. 108
About the Sample VBScript .................................................. 114
The NetworkDrives Sample VBScript .............................. 115
About the Sample C Program

Overview

The RoutStat sample C program demonstrates the usage of the Universal Agent Data Provider API set. This program inputs data to the Data Provider based on the metafile RTstat.mdl, shown in Figure 4 on page 107.

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
FIGURE 4. The metafile RTstat.mdl

APPL RTstatus
NAME RTnetIO p 120
ATTRIBUTES ';'
  RouterName D 32
  PktReceivedPerSec C 1000000
  PktRoutedPerSec C 1000000
  PktDiscardedPerSec C 1000000
  AvgPktSize C 100000
The RoutStat Sample C Program

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#ifdef WIN32
    #include <windows.h>
#endif
#include "kumpapi.h"

The RoutStat Sample C Program

#if 0

/**************************************************************************
 Copyright (c) 1997 an unpublished work by
 Candle Corporation.
 2425 Olympic Boulevard
 Santa Monica, California 90404
 (310) 829-5800
**************************************************************************/
Candle Universal Agent Sample Test Program--RoutStat

This sample program demonstrates the basic usage of 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 sta-
tistics 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 param-
eter. If it is not specified, a default of 1000 is used.

The second input parameter controls the use of connection-oriented or con-
nectionless 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>
#endif
#include "kumpapi.h"

void sleep(int wait)
{
    Sleep(wait);
    return;
}
#endif
#include "kumpapi.h"
The RoutStat Sample C Program

#define  ConnectionMode  'Y'

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

int  API_Status, InputCount, MaxInputCount = 100, WaitTime = 30;
int  NameIndex, PKTrcvd, PKTout, 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
   CommunicationMode = toupper(*argv[1]);
} else ;
The RoutStat Sample C Program

/* 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)
The RoutStat Sample C Program

```c
{
    dp_BeginInput(dpHandle, ApplicationName, AttrGroupName, &API_Status);
    if (API_Status == KUMP_API_OK)
        /*----------------------------------------------*/
        /* Initialization complete. Begin data input loop. */
        /*----------------------------------------------*/
        continue;
    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),
```
The RoutStat Sample C Program

TypeIsNumeric, &API_Status);
    dp_FormatBufferDataData(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);
            }
The RoutStat Sample C Program

time(&ltime);
NOW = localtime(&ltime);
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, &Re-
quID, RequestBuffer, strlen(RequestBuffer)-1, &API_Status))
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 VBScript

Overview

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

FIGURE 5. The metafile NWDrives.mdl

```
//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
```
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.
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

' Note: The following call to kumpendi is only necessary in support of multiple invocations of this script
The NetworkDrives Sample VBScript

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 kumprdta 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 kumprdta.

if rowCounter = disks.Count then
    wscript.echo "Calling kumprdta with requestID " & RequestID
    ' Note: The space after the requestID is needed so that kumprdta will correctly parse the input arguments.
    rc = oshell.run("kumprdta " & _
        RequestID & " " & _
        disk.name & ";" & _
        disk.filesystem & ";" & _
        disk.volumeserialnumber & ";" & _
        diskSize & ";" & _
        diskFreeSpace & ";" & _
        diskFreeSpacePercent & ";" & _
        " NWDrives NetworkDrives", _ 0, true)
    if rc > 1 then
        wscript.echo "Status from kumprdta is " & rc
    end if
else
    rc = oshell.run("kumpinpt " & _
        disk.name & ";" & _

Sample Programs    117
The NetworkDrives Sample VBScript

```vbnet
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
next
rowCounter = rowCounter + 1

End Sub

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

118   Universal Agent API/Command Programming Reference Guide, V410
Index

A
Adobe portable document format 12
API client package 21
API function calls
dp_AcceptRequest 36
dp_AllocateBuffer 38
dp_AllocateHandle 40
dp_BeginInput 42
dp_BeginSample 45
dp_ClearBuffer 47
dp_CloseSession 49
dp_Define 51
dp_EndInput 53
dp_EndSample 55
dp_FormatBufferData 57
dp_FreeBuffer 59
dp_FreeHandle 61
dp_InputData 62
dp_OpenSession 65
dp_Ping 67
dp_ReceiveRequest 68
dp_Redefine 67
dp_ResetHandle 74
dp_ReturnData 76
dp_SendRequestResult 79
dp_SetSourceName 81
dp_ShowMessages 84
API Server Data Provider
specifying host 24
specifying port 24

C
Candle Customer Support 16
Candle web site 11
client package 21
command line interface calls 85–104
KUMPAREQ 88

KUMPBGNI 89
KUMPDEFN 92
KUMPENDI 93
KUMPINPT 95
KUMPPING 98
KUMPRDFN 99
KUMPRREQ 100
KUMPRRTDA 101
KUMPSHOW 104
connecting to Data Provider 31
console commands
minimum input 86
overview 86
syntax 86

D
data collection
demand-driven 20
interval-driven 20
documentation set information 10
dp_AcceptRequest
description 36
parameters 36
prototype 36
status codes 37
synopsis 36
dp_AllocateBuffer 38
description 38
input parameters 38
status codes 39
synopsis 38
dp_AllocateHandle
description 40
input parameters 40
prototype 40
status codes 40
dp_BeginInput
  description 42
  input parameters 42
  prototype 42
  status codes 43
  synopsis 42

dp_BeginSample
  description 45
  input parameters 45
  prototype 45
  status codes 45
  synopsis 45

dp_ClearBuffer
  description 47
  input parameters 47
  prototype 47
  status codes 47
  synopsis 47

dp_CloseSession
  description 49
  input parameters 49
  prototype 49
  status codes 49
  synopsis 49

dp_Define
  description 51
  input parameters 51
  prototype 51
  status codes 52
  synopsis 51

dp_EndInput
  description 53
  input parameters 53
  prototype 53
  status codes 53
  synopsis 53

dp_EndSample
  description 55
  input parameters 55
  prototype 55
  status codes 55

dp_FormatBufferData
  description 57
  input parameters 57
  prototype 57
  status codes 58
  synopsis 57

dp_FreeBuffer
  description 59
  input parameters 59
  prototype 59
  status codes 59
  synopsis 59

dp_FreeHandle
  description 61
  input parameters 61
  prototype 61
  status codes 61
  synopsis 61

dp_InputData
  description 62
  input parameters 62
  prototype 62
  status codes 62
  synopsis 62

dp_OpenSession
  description 65
  input parameters 65
  prototype 65
  status codes 65
  synopsis 65

dp_Ping
  description 67
  input parameters 67
  prototype 67
  status codes 67
  synopsis 67

dp_ReceiveRequest
  description 68
  input parameters 68
  prototype 68
  return status codes 69