GMI Command, API

Application Interface, GMI Command Programs
General Management Interface Foundation

http://www.gmi-foundation.org

Application Program Interface, Call Library Description

This document provides a description of all functions in the "GMI-Cmd32.lib" library, which permits software developers to access GMI Agent programs through C/C++ programs, or scripting languages such as PHP and Perl. This permits easy creation of programs that communicate with one or more GMI Agents executing on remote network platforms.

Using the "GMI-Cmd32.lib" call library, and the documentation herein, programmers can create client programs that connect, set, get, upload, install, uninstall, and perform other functions available through the "GMI-cmd.exe" executable program, greatly extending the utility of GMI agents to include sophisticated control and monitor functions.

As a prerequisite to using this documentation and call library, programmers should become familiar with the "GMI-cmd.exe" program, which is a standard command line interface for GMI Agents. The "GMI-cmd.exe" program is furnished with the GMI-Agent "quick start" package, and uses exactly the same functions of the call library described here.

The "GMI-cmd.exe" program will provide a great introduction to functions and capabilities available to programmers. The "GMI-cmd.exe" program is very easy to use, and permits interactive browsing of agent programs. A good understanding of this program's precepts and operation will facilitate any programming activity, and is essential to those unfamiliar with the GMI system.

Note that the information here applies only to GMI client programs. For a description of how to create new GMI Applications (which are uploaded to agents) see the GMI Agent API documentation, provided elsewhere. For a discussion of Agent features, start with the "GMI System Overview" documentation, available within a separate manual. For a discussion of the GMI-Cmd.exe program, see the "GMI-Cmd.exe Reference Manual."

The document herein will be of interest to application developers, system managers, and other people wanting to provide special end-point functions for any site running the GMI Agent program, by creating their own management applications that access agent programs.

GMIC Command Program Development Concepts

All library functions found in the "GMI-Cmd32.lib" library begin with a "GMIC_" prefix. Each function requires specific augments described in detail herein.

All library functions of the "GMI-Cmd32.lib" library return zero on success, or non-zero on failure. The reason for any failure (or the status of any successful operation) can be found by calling the "GMIC_get_last_message()" function, which returns a one line text description of a message associated with the last function call.

Note that, unlike GMI "Agent" applications (which must be "Certified" before they can be installed and used at an agent), GMI "Command" applications can be constructed and used without any certification or special considerations. To use a GMI Command program, the user needs knowledge of the agent "PassKey" (if configured at the agent.)

A GMI Command program typically operates as follows:

1. The program calls "GMIC_init()" as a necessary first step, before calling any other GMI function. Any useful GMI command program must call this function once.

2. The program optionally calls "GMIC_set_passkey()" to set any passkey required for the particular agent. The passkey is provided in clear text (so the storage and encryption of the passkey is left completely to the programmer.)

3. The program calls "GMIC_connect()" to connect to an agent program. If the connection fails, the reason for the failure can be determined by calling "GMIC_get_last_message()", which returns a text description of the operation's success or failure.

4. Once the program is connected, the program can call other GMI Command functions, such as "GMIC_get_agent()" to get an agent value, or "GMIC_set_agent()" to set an agent value. These functions require as an argument the "Agent Path" (herein the "apath") which is simply the pathname to an agent folder.

5. To disconnect from the agent, the program can simply terminate, or may call "GMIC_disconnect()", or can call "GMIC_connect()" to connect to another agent program (which automatically disconnects and reconnects to the new agent.

The above program flow will be typical of virtually all GMI command programs, whether programmed in C, or developed using PHP, Perl, or other scripting languages. Each of the above functions is described in detail within this document.

Application Development

The GMI Command API is intended to be as simple, flexible, and comprehensive as possible. GMI Foundation recognizes that the biggest obstacle to any development activity is the sheer complexity of the activity. Rather than define thousands of obscure functions (as has been a traditional solution of frameworks) GMI supports application development through a small set of commands. This greatly simplifies the development activity.

To create a working application, the programmer simply creates software that accesses the GMI command calls to perform a particular activity. This can range from a simple command line executable needed to query a set of GMI values, or more sophisticated functions that permit user interaction through a GUI or web screen.

Various applications are typical, including the following:

- **Data Reduction Applications.** One of the central reasons for using the GMI application interface is to either derive or reduce the data available at the agent. For example, a GMI command application may periodically poll a value at one or more agents to determine a particular set of metrics on agent or platform performance.

- **Reporting And Alerting Applications.** A common reason for constructing a GMI command application is to access data for reporting purposes. For example, a GMI application may periodically poll values to create a performance graph on a platform activity, or a bar chart showing the relative values across a set of similar agents. The acquired data can compared to thresholds, and alerts generated when values are not as expected.

- **Control Applications.** A main type of GMI command application is to apply control to a platform. For example, an application can automate and simplify various actions associated with a particular GMI App, such as the "rshell" GMI application. The command application provides the logic needed to set values, launch programs, acquire output in a manner that is more straightforward than using the GMI-cmd.exe program.

In general, the GMI-cmd32.lib functions permit a level of automation that would be cumbersome without a complex process of the type described above.

Description of Functions

The remaining pages of this manual discuss the above functions, and additional ancillary (less critical) library functions that are available to the programmer. When reviewing these functions, it may be useful to the developer to study the example programs herein and also the online open source projects that exist for GMI applications. Guidance by GMI Foundation is available on request.

Function: GMIC_init()

The "GMIC_init()" function must be called prior to any other GMIC functions, and creates working data for the GMIC application. If the function is not called, all subsequent GMIC functions will fail, and the "GMIC_get_last_message()" function returns the text string: "ERROR – not initialized, call GMIC_init() first."

Passed Arguments:

None.

Return Status:

The "GMIC_init()" function returns zero on success, and non-zero on error. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage:

```
#include "gmi-cmd.h"

/* Initialize the GMI command program. */

char error_text[GMI_MAX_STRING];

if (GMIC_init() != 0)
{
   GMIC_get_last_message(error_text);
   printf("%s", error_text);
}
else
{
   /* The command program is successfully initialized. */
}
```

Comments:

This function initializes the winsock interface for the interpreter. If the winsock interface has already been initialized, this will be indicated by the GMIC_get_last_message() call, and the program can continue without special consideration.

It is harmless to call this function more than once. The second time the function is called there is no change to the system.

Function GMIC_get_last_message()

The "GMIC_get_last_message(): function returns the last message returned by the system. It is similar to the traditional "GetLastError()" function found on most systems, except the return value can indicate both a successful and an error condition.

Passed Arguments:

The function requires a single string value, which will contain the last message of the system. This string value should be dimensioned as GMI_MAX_STRING (1000) bytes for safety.

Return Status:

The "GMIC_get_last_message()" function returns zero on success, and non-zero on error. Generally, this function will never fail, hence will always return a zero status.

Example Usage:

```
#include "gmi-cmd.h"

/* Get the last system status or error message. */

char error_text[GMI_MAX_STRING];

/* Get the last success or error message. */

GMIC_get_last_message(error_text);

/* Print the message. Note that the message will be */
/* delimited by a newline, hence another newline does */
/* not have to be included below. */

printf("%s", error_text);
```

Comments:

If the last GMIC function did not return an error, the return string will always contain a prefix of "OK –". The return string may contain an error message, or may contain a success indication. Note that this function should be called whenever any GMIC program returns a status other than zero.

The text string, returned by this function, is always delimited by a newline character, and then a null character.

Function: GMIC_set_passkey ()

The "GMIC_set_passkey()" function is called to set the passkey (password) for an agent program, if required. It is only necessary to call this function when connecting to an agent where the "/security/SecurityMode" setting of the agent is set to "1" or "3". The passkey will then apply to all future GMIC operations. Generally, this function is called prior to any GMIC_connect() function call.

Passed Arguments:

The "GMIC_set_passkey()" function requires a single argument, a text string, which is the passkey of the agent that will be connected to. The string value must be GMI_MAX_STRING (1000) characters or less.

Return Status:

The "GMIC_set_passkey()" function returns zero on success, and non-zero on error. Generally, this function will never fail, hence will always return a zero status.

Example Usage:

```
#include "gmi-cmd.h"

/* Set the passkey, for use with subsequent GMIC calls. */

GMIC_set_passkey("MyPassword");

/* The passkey has now been set. */
```

Comments:

This function will generally be near the start of any GMI command application, to handle the case where a password is required to access the agent.

If a password is required, and has not been supplied by this function, then any GMIC function that connects or accesses the agent will fail and the "GMIC_get_last_message()" function will indicate: "Invalid Passkey. Reconnect with valid passkey."

The passkey is generally supplied via a prompt, or via a command line argument to the GMI Command program, or some scheme will be constructed by the programmer to store passwords in encrypted form on the disk.

Function: GMIC_connect()

The "GMIC_connect()" function connects to an agent program in a fashion similar to the "connect" command of the GMI-cmd.exe program. This function establishes communications between the GMI command program and the agent program. Subsequently, the program can get and set values using the connected agent.

Passed Arguments:

The function requires a single string value, a text string, which will be the IP address of the agent program in standard dot notation. Only IPv4 addresses are currently supported by this command.

Return Status:

The "GMIC_connect()" function returns zero on success, and non-zero on error. On failure, the "GMIC_get_last_message()" indicates the reason the function call failed, which can be any number of reasons ranging from a badly formed IP address to timeout while connecting to the agent.

Example Usage:

```
#include "gmi-cmd.h"

int status;

/* Connect to an agent program */

status = GMIC_connect("192.168.1.1");

/* Get the last success or error message. */

if (status != 0)
{
   char error_text[GMI_MAX_STRING];

   GMIC_get_last_message(error_text);
   printf("%s", error_text);
}
else
{
   /* We are connected. */
}
```

Function: GMIC_connect() (continued)

Comments:

The "GMIC_connect()" function (preceded by the "GMIC_init()" command) must be executed at least once prior to using any other GMIC function calls. The function can fail for various reasons, including a malformed IP address, but also an attempt to connect to an agent platform when the agent is not running, or attempt to connect to an agent without the proper passkey.

In all cases, the GMIC_get_last_message() function will return the reason for failure. If the failure is due to a bad passkey, the programmer should first call the "GMIC_set_passkey()" prior to calling the "GMIC_connect()" function.

Note that GMIC_connect() automatically disconnects any previous connection. Therefore, the programmer does not specifically have to call the "GMIC_disconnect()" function (described below) prior to calling this function.

An attempt to execute a GMIC function without first connecting will cause the GMIC function to fail, and the result of the "GMIC_get_last_message()" function will return with a value of "Not connected. Connect first."

After connection, the agent directory path will always be set back to the root directory of the agent, the "/" directory.

See Also:

GMIC_init()

GMIC_set_passkey()

GMIC_disconnect();

GMIC_get_connect_status();

Function: GMIC_disconnect()

The "GMIC_disconnect()" function disconnects the currently connected agent. This command is provided for completeness. Generally, there is no need to disconnect from a connected agent (since the next call to "GMI_connect()" automatically disconnects from the current agent prior to establishing a new connection. This function directly corresponds to the "GMI-Cmd.exe" "disconnect" command.

Passed Arguments:

None.

Return Status:

The "GMIC_init()" function returns zero on success, and non-zero on error. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage:

```
#include "gmi-cmd.h"

int status;

/* Disconnect from the current agent. */

status = GMIC_disconnect();

if (status != 0)
{
   /* Connection failed. Display the reason. */

   char error_text[GMI_MAX_STRING];

   GMIC_get_last_message(error_text);
   printf("%s", error_text);
}
```

Comments:

The "GMIC_disconnect()" function will disconnect the command program from the agent. Any attempt to communicate with the agent will fail, and the "GMIC_get_last_message()" function will return the text string "Not connected." If the command program is not currently connected, calling this function does nothing.

Function: GMIC_get_connect_status()

The "GMIC_get_connect_status()" function returns the status of the current connection if any. The function accepts a text string of GMI_MAX_STRING (1000) characters.

Passed Arguments:

The "GMIC_get_connect_status()" function requires a single argument, a pointer to a character string of GMI_MAX_STRING (1000) characters or less. On successful execution, the string contains the value "Not connected." if the program is not currently connected, or "Connected to: (address)" if a connection exists.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage:

```
#include "gmi-cmd.h"

char current_status[GMI_MAX_STRING];

/* Get the connection status. */

GMI_get_connect_status(current_status);

/* Print the current status. */

printf("%s", current_status);

if (strstr(current_status, "Connected to:"))
{
   /* We are currently connected. */
}
```

Comments:

The "GMIC_get_connect_status()" function can be used to determine whether the command program is connected to an agent. The function status should always be zero. The programmer should allocate a string, pass this string to the function, and then test the value of the string using a function such as "strnicmp()" or "strstr()". The text string may also be printed out, and will be delimited by newline followed by a null character.

Function: GMIC_get()

The "GMIC_get()" function gets a value from a currently connected agent. The function is similar to the GMI-Cmd.exe "ls" command, and fetches any single value from the agent. Note that the function returns the value as a text string, and only one value can be returned. (To return multiple values, the programmer should use the "GMIC_list()" function documented elsewhere.)

Passed Arguments:

The "GMIC_get ()" function requires three arguments:

(1) The first argument, a text string, is the agent path value in standard agent path notation.
(2) The second argument, a text string, contains a keyword (or parse spec) that must match the return value. This argument can be NULL, or a zero length string, to match the first (perhaps the only) value;
(3) The third argument is a pointer to a text string that will return the first matched value.

Return Status:

The function returns zero on success, and the value of the third argument to the function will contain the acquired value. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example #1 Usage

```
#include "gmi-cmd.h"

char return_value[GMI_MAX_STRING];
char label [GMI_MAX_STRING];
int  intval;

/* Get the system description of the currently */
/* connected agent program. */

/* The second argument is NULL, because we will */
/* get the first and only value of the sysdescr */
/* agent object. */

GMIC_get("/system/sysdescr", NULL, return_value);

/* Print the value. */

printf("System Description: %s\n", return_value);
```

Function: GMIC_get() (continued)

Example #2 Usage:

```
#include "gmi-cmd.h"

char return_value[GMI_MAX_STRING];
char label [GMI_MAX_STRING];
int  intval, status;

/* Get the value of bytes received contained in the */
/* "/perf/netstat/eth" agent folder. We specify a */
/* match value of "Bytes_Received" because the agent */
/* path returns multiple values. */

status = GMIC_get(
   "/perf/netstat/eth", "bytes_received", return_value);

/* Get and print the label and value. */

if (status == 0)
{
   sscanf(return_value, "%s %d", label, &intval);
   printf("Label: %s – Value: %d\n", label, intval);
}
```

Comments:

The "GMIC_get ()" function provides one of the several ways of getting values, other ways being the GMIC_list(), and GMIC_download() functions.

The function specifies an agent folder path, and retrieves the value of that path, optionally selecting one of the lines of the output (if the path contains multiple lines.) This allows the function to easily fetch path values that consist of multiple values.

The function returns the value of the first match to the second argument. If the second argument is NULL or a zero length string, then the first non-blank value of the agent folder is returned. The match value (the second argument) can be a keyword, or a "GMI parser specification" as documented in the "GMI Parser Reference Manual". If the value is not matched by the second argument, the "GMIC_get_last_message()" function will indicate there was no match for the value.

See Also:

GMIC_list()

GMIC_download()

Function: GMIC_set()

The "GMIC_set()" function sets a value at the currently connected agent. The function is similar to the GMI-Cmd.exe "set" command, and allows the operator to set a single value at the currently connected agent. (To set multiple values, the programmer should use the "GMIC_upload()" function documented elsewhere.)

Passed Arguments:

The "GMIC_set()" function requires two arguments:

(1) The first argument, a text string, is the agent path value in standard agent path notation. The path must be settable.
(2) The second argument, a text string, contains the value to be set at the agent. The value must be GMI_MAX_STRING (1000) characters or less.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

int status;
char error_msg[GMI_MAX_STRING];

/* Set the value of SysContact at the agent. */

status = GMIC_set("/system/syscontact, "Local Admin");

if (! status)
{
   /* Display the reason for failure. */

   GMIC_get_last_message(error_msg);
   printf("Function failed. Reason: %s", error_msg);
}
```

Comments:

The "GMIC_set ()" function provides one of two different ways of setting values, other ways being the GMIC_upload() function. If the object is not settable, then the GMIC_get_last_message() function indicates the reason for the failure. Up to GMI_MAX_STRING (1000) characters can be set at the agent program.

Function: GMIC_list()

The "GMIC_list()" function acquires multiple values from the currently connected agent, listing these values to a user specified file. The function is similar to the GMI-Cmd.exe "ls" command, and allows the operator to get one or more values from the agent path.

Passed Arguments:

The "GMIC_list()" function requires two arguments:

(1) The first argument, a text string, is the agent path value in standard agent path notation.
(2) The second argument, a text string, contains the name of a local disk file that will receive the list information. The specified path will be created or overwritten.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

/* Get the value of SysInfo at the agent. */

if (GMIC_list("/system/sysinfo, "tempfile.txt") != 0)
{
   /* Get the reason for failure. */

   char error_msg[GMI_MAX_STRING];
   GMIC_get_last_message(error_msg);
   printf("Function failed. Reason: %s", error_msg);
}
```

Comments:

The "GMIC_list()" function furnishes a general purpose method of getting text objects to a disk file. The specified disk file (the second argument to the function) will be created or overwritten, and must be a valid file.

This function is slightly more difficult to use than the "GMIC_get()" function, but permits any full listing of values to be acquired from an agent. To acquire a non-textual listing, such as a file, the programmer should use the "GMIC_download()" function, documented elsewhere.

Function: GMIC_list_all_agent_paths()

The "GMIC_list_all_agent_paths()" function acquires all agent paths for the currently connected agent to a local file. This provides a simple mechanism for determining the current packages and capabilities of the connected agent. The function operates in a fashion similar to the GMI-Cmd.exe "ls –r /" command.

Passed Arguments:

The "GMIC_list_all_agent_paths()" function requires a single argument, a pointer to a character string specifying the local file that will be created.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

/* Get all agent paths to the listfile.txt file. */

if (GMIC_list_all_agent_paths("/temp/listfile.txt"))
{
   /* Function failed. Get the reason for failure. */

   char error_msg[GMI_MAX_STRING];
   GMIC_get_last_message(error_msg);
   printf("Function failed. Reason: %s", error_msg);
}
```

Comments:

The "GMIC_list_all_agent_paths()" function furnishes a general purpose method of getting a description of the agent capabilities. The function will list all the pathnames for the agent, including the type, access, and pathname values.

The listing is identical to the result of the "GMI-Cmd.exe" function "ls –r ./", except that only the pathnames are listed (and the number of entries is not summarized at the bottom of the program.

The output of the program should be easily parsed by simply opening the file and reading values with an "fscanf()" function. The listing contains three columns, where the first column is the path type, the second column is the path access, and the third column is the full path name.

Function: GMIC_upload()

The "GMIC_upload()" function sets a value at the currently connected agent, where the value can be a single value, or a text or binary file. The function is similar to the GMI-Cmd.exe "upload" command, and allows the operator to set or upload entire files to an "uploadable" agent path.

Passed Arguments:

The "GMIC_upload()" function requires two arguments:

(1) The first argument, a text string, is the agent path value in standard agent path notation. The path must be uploadable,.
(2) The second argument, a text string, contains the name of a text or binary file to be uploaded to the agent.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

int status;
char error_msg[GMI_MAX_STRING];

/* Upload "myfile.txt" to the SysInfo agent folder. */

status = GMIC_upload("/system/sysinfo, "myfile.txt");

if (! status)
{
   /* Display the reason for failure. */

   GMIC_get_last_message(error_msg);
   printf("Function failed. Reason: %s", error_msg);
}
```

Comments:

The "GMIC_upload()" function provides one of two different ways of setting values, other ways being the GMIC_set() function. If the object is not uploadable, then the GMIC_get_last_message() function indicates the reason for the failure. Note that not all agent paths can be uploaded; the specified path must have "upload" access.

Function: GMIC_download()

The "GMIC_download()" function downloads a value at the currently connected agent to a local file. The function is similar to the "GMIC_list()" function, except the resulting value can be any value that was previously uploaded using the "GMIC_upload()" function, including: (1) A text file uploaded via the "GMIC_upload()" function, and / or; (2) A binary file uploaded via the GMIC_upload()" function. Note that the "GMIC_download()" allows the operator to acquire data that has been uploaded to the agent, including binary data, and is the only method available for acquiring binary data objects from the agent.

Passed Arguments:

The "GMIC_download()" function requires two arguments:

(1) The first argument, a text string, is the agent path value in standard agent path notation. The path must be readable.
(2) The second argument, a text string, contains the name of a local file that will receive the downloaded information.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

int status;
char error_msg[GMI_MAX_STRING];

/* Download from the "SysInfo" path to the local */
/* disk file "myfile.txt". The file is created or */
/* overwritten. */

status = GMIC_download("/system/sysinfo, "myfile.txt");

if (! status)
{
   /* Display the reason for failure. */

   GMIC_get_last_message(error_msg);
   printf("Function failed. Reason: %s", error_msg);
}
```

Function: GMIC_download() (continued)

Comments:

The "GMIC_download()" function retrieves the value of an agent path that has been uploaded via the "GMIC_upload()" function. If the value was not uploaded, the "GMIC_get_last_message()" function returns "ERROR: Agent path is not downloadable."

Note that an agent value can be settable, uploadable, or both.

If the object is settable, then it can accept a single value via the "GMIC_set()" function, and that object can be fetched via the "GMIC_list()" function and / or the "GMIC_get()" function.

If an object is uploadable, then it can accept single or multiple values, or arbitrary binary data, via the "GMIC_upload()" command. The value can be fetched via the "GMIC_get()" command, or "GMIC_list()" command. However, if the uploaded object value is binary data, then the only the size of the object (and object type) is fetched by these functions, and not the actual object value.

The "GMI_download()" will return the value that was uploaded to the agent, regardless of whether the object is a text value, a text file, or a binary file. For example, the user can upload a PDF file to the agent, and then download the file at a later time using this function.

If the programmer is not sure whether an agent path was "set" or "uploaded", the programmer can first execute the "GMI_download()" function, and if it fails, attempt to execute the "GMI_list()" function. This provides a simple way of acquiring values that were either uploaded or set at the agent.

This function can fail for a variety of reasons. The "GMIC_get_last_message()" function can return multiple values indicating the path is incorrect, the specified file is not writeable, the agent timed out, the agent path is not readable, or the connection is not valid. agent is not valid.

See Also:

GMIC_set()

GMIC_upload()

GMIC_get();

GMIC_list();

Function: GMIC_change_agent_directory()

The "GMIC_change_agent_directory()" function provides two purposes; (1) The function can be used to set the path of the agent, where all subsequent agent paths are relative to the specified path, and; (2) The function is useful for testing to see whether a specified agent path is valid. The function performs an action similar to the "GMI-Cmd.exe" program "cd" command.

Passed Arguments:

The function accepts a single argument, a text string, specifying the value of the agent path that is to be set.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function. On failure, the current agent path does not change.

Example Usage

```
#include "gmi-cmd.h"

/* Change working directories at the agent. */

if (GMIC_change_agent_directory("/perf/netststat"))
{
   /* Specified path does not exist at the agent. */

   printf("Invalid path.\n");
}
else
{
   /* Specified path is valid. Reset the path. */

   GMIC_change_agent_directory("/");
}
```

Comments:

The "GMIC_change_directory()" function will set the working directory from the default "/" directory, to some other valid agent directory. If the agent does not support the specified directory, the function fails with a non-zero return value, and the reason for the failure will be "Invalid Path". This provides a simple method of determining whether a particular path exists at the connected agent.

Function: GMIC_get_agent_directory()

The "GMIC_get_agent_directory()" function returns the current working directory of the agent, and is similar "GMI-Cmd.exe" program "pwd" command. Unless the programmer has executed the "GMIC_change_agent_directory()" function, this function will return a value of "/", indicating the default root directory of the currently connected agent.

Passed Arguments:

The function accepts a single argument, a pointer to a text string, which will contain the current pathname at the connected agent, typically GMI_MAX_STRING (1000) bytes in length.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

char current_working_directory[GMI_MAX_STRING];

/* Get the current agent working directory. */

GMIC_get_agent_directory(current_working_directory);

/* Print the directory value. */

printf("%s\n", current_working_directory);
```

Comments:

The "GMIC_get_agent_directory()" function will return the current agent working directory. The only reason that this directory may not be valid is that a GMI application has been uninstalled from the agent subsequent to the last "GMIC_change_agent_ directory()" call.

This function can be used, along with the "GMIC_change_agent_directory()" function to create a function to test whether a directory is valid; (1) the programmer gets the current working directory using the "GMIC_get_agent_directory()" function, then; (2) the programmer changes working directories to the directory to test for, recording the return status of the "GMIC_change_agent_directory()" function, then; (3) Restores the original working directory.

Function: GMIC_install_at_agent()

The "GMIC_install_at_agent()" function installs a certified GMI package at the agent program. This function is similar to the "GMI-Cmd.exe" program "install" command, useful for updating an agent with new functionality..

Passed Arguments:

The function accepts a single argument, a text string, which is the local disk pathname to a certified GMI package.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

int status;
char errtxt[MAX_GMI_STRING];

status = GMIC_install_at_agent(".\netstat.gmi");

if (status != 0)
{
   /* GMI package installation failed. */

   GMIC_get_last_message(errtxt);
   printf("%s", errtxt);
}
```

Comments:

The "GMIC_install_at_agent()" function provides a simple way of installing a particular GMI package at the agent. The package must be certified, and any existing package at the agent will be uninstalled prior to installing the package. The function can fail for various reasons, including that the pathname argument does reference a valid path or GMI package, or the connection timed out, or the program is not currently connected to a GMI agent.

Function: GMIC_remove_from_agent()

The "GMIC_remove_from_agent()" function removes a GMI package at the agent program. This function is similar to the "GMI-Cmd.exe" program "uninstall" command, useful for removing unwanted functions from an agent..

Passed Arguments:

The function accepts a single argument, a text string, which is a pathname to an agent folder that references an installed package.

Return Status:

The function returns zero on success. The reason for failure can be acquired by calling the "GMIC_get_last_message()" function.

Example Usage

```
#include "gmi-cmd.h"

int status;
char errtxt[MAX_GMI_STRING];

/* Remove the netstat.gmi package from the agent. */

status = GMIC_remove_from_agent("/perf/netstat");

if (status != 0)
{
   /* GMI package uninstall failed. */

   GMIC_get_last_message(errtxt);
   printf("%s", errtxt);
}
```

Comments:

The "GMIC_remove_from_agent()" function provides a method of removing a particular GMI package at the agent. The function can fail for various reasons, including that the pathname argument does reference a valid agent path, or the connection timed out, or the program is not currently connected to a GMI agent.

GMI-Cmd32.lib Function Call Reference

This section briefly lists all the functions of the preceding section, as a quick reference guide.

To use any function that contains the GMIC prefix requires that the programmer link with the "GMI-cmd32.lib" or "gmi-cmd.obj" file. These files are available as part of the GMI Foundation, and are small files of basic utility described herein. The functions are documented in the "gmi-cmd.h" include file, and further documented in this section to provide additional explanation.

The "GMI-Cmd32.lib" is available for Windows 32 bit applications, and is linked with the other programmer files (including the wsock32.lib standard WinSock library) to create the final application.

Function: GMIC_init()

This function should be called prior to any other function listed herein. If the function is not called, all other GMIC commands will fail.

```
int GMIC_init
(
void
);
```

Function: GMIC_get_last_message()

This function provides general utility in returning the last message from the GMIC library, either a reason for failure, or a success indication. The function accepts as a single argument a pointer to a string of GMI_MAX_STRING (1000 characters) size.

```
int GMIC_get_last_message
(
char *string
);
```

Function: GMIC_set_passkey()

This function allows the programmer to set the "passkey" to the agent (if any) that is used in subsequent GMIC calls. The function accepts a single text string argument, which is the passkey to the agent program.

```
int GMIC_set_passkey
(
char *passkey
);
```

GMI-Cmd32.lib Function Call Reference (Continued)

Function: GMIC_connect()

This function connects to an agent, permitting other GMIC calls to operate on the agent. If the agent requires a passkey, the programmer must first call GMIC_set_passkey() as documented above. The function accepts a single text string argument, which is the IP address of the agent program.

```
int GMIC_connect
(
char *ipaddr
);
```

Function: GMIC_get_connect_status()

This function provides general utility in determining the current connection status to an agent (if any.) The function accepts as a single argument a pointer to a text string of GMI_MAX_STRING (1000 characters) size. The function returns the connect status as either "Connected to:" or "Not connected."

```
int GMIC_get_connect_status
(
char *connect_status
);
```

Function: GMIC_get()

This function allows the programmer to fetch a single value from the agent. The function accepts three arguments; (1) a text string containing the agent path to get; (2) a match qualifier that matches the first value returned by the agent (which may be NULL to match any value), and; (3) a pointer to a text string of GMI_MAX_STRING (1000 character) that will contain the fetched value. The match value permits the user to select the first matched line when there is multi-line values for the object. The match value can be a partial match, and is not case-sensitive.

```
int GMIC_get
(
char *agent_path,
char *match_value,
char *return_value
);
```

GMI-Cmd32.lib Function Call Reference (Continued)

Function: GMIC_set()

This function allows the programmer to set a single value at the agent. The function accepts two arguments; (1) a text string containing the agent path to set, and; (2) a pointer to a text string of GMI_MAX_STRING (1000 character) specifying the value to set. This function provides the utility associated with the "GMI-cmd.exe" program's "set" directive, used to set agent values.

```
int GMIC_set
(
char *agent_path,
char *set_value
);
```

Function: GMIC_list()

This function allows the programmer to acquire any readable object's value to a file. The function accepts two arguments; (1) a text string containing the agent path to fetch, and; (2) a text string specifying the location of a local file that will contain the listing for the agent path. This function provides the utility associated with the "GMI-cmd.exe" program's "ls" directive, used to list agent values.

```
int GMIC_list
(
char *agent_path,
char *local_file_path
);
```

Function: GMIC_list_all_agent_paths()

This function allows the programmer to acquire the full description of all paths supported by the agent program. The function accepts a single argument, which is a text string specifying the location of a local file that will contain the listing for the agent path. This function provides the utility associated with the "GMI-cmd.exe" program's "ls -r" directive, used to recursively list all agent folders and paths.

```
int GMIC_list_all_agent_paths
(
char *local_file_path
);
```

GMI-Cmd32.lib Function Call Reference (Continued)

Function: GMIC_upload()

This function allows the programmer to upload files to agent paths that support this feature. The function accepts two arguments: (1) a text string that specifies the pathname to the uploadable object at the agent, and; (2) a single argument, which is a text string specifying the location of a local file that will be uploaded. This function provides the utility associated with the "GMI-cmd.exe" program's "upload" directive, used to upload files to the agent.

```
int GMIC_upload
(
char *agent_path,
char *local_file_path
);
```

Function: GMIC_download()

This function allows the programmer to download files from agent paths that support this feature. The function accepts two arguments: (1) a text string that specifies the pathname to the downloadable object at the agent, and; (2) a single argument, which is a text string specifying the location of a local file that will contain the download. This function provides the utility associated with the "GMI-cmd.exe" program's "download" directive, used to download files from the agent.

```
int GMIC_download
(
char *agent_path,
char *local_file_path
);
```

Function: GMIC_change_agent_directory()

This function changes the current working directory of the currently connected agent, useful for either testing for directory existence, or simplifying subsequent agent path references (to be relative paths). The function accepts a single argument, which is the pathname of an existing agent folder. If the specified agent path exists, the return status is zero and the new path becomes the current path. Otherwise, the return status is non-zero, and the current working directory is not changed. This function provides the utility associated with the "GMI-cmd.exe" program's "cd" directive, used to change agent directories.

```
int GMIC_change_agent_directory
(
char *new_agent_path
);
```

GMI-Cmd32.lib Function Call Reference (Continued)

Function: GMIC_get_agent_directory()

This function gets the current working directory of the currently connected agent, The function accepts a single argument, which is a pointer to a text string of GMI_MAX_STRING (1000) characters size. This function provides the utility associated with the "GMI-cmd.exe" program's "pwd" directive, used to print the current agent directory path.

```
int GMIC_get_agent_directory
(
char *current_agent_path
);
```

Function: GMIC_install_at_agent()

This function installs a package at the currently connected agent. The function accepts a single argument, a text string, which is the pathname to a valid and certified GMI application. This function provides the utility associated with the "GMI-cmd.exe" program's "install" directive, used to install new functions and applications at an agent program. The function checks the package, verifies that the package is certified, and then transmits the package to the running agent. If the agent fails to accept the package, the return status is non-zero and the "GMIC_get_last_message()" function will return the reason for the failure.

```
int GMIC_install_at_agent
(
char *local_path_to_package
);
```

Function: GMIC_remove_from_agent()

This function removes a package at the currently connected agent. The function accepts a single argument, a text string, which is the agent pathname to a folder of the installed package. This function provides the utility associated with the "GMI-cmd.exe" program's "uninstall" directive, used to remove packages from an agent program. The program accepts a single argument, which is an agent path pointing to the remote application to uninstall. Note that this pathname is any agent path that is within the installed package (not necessarily the root directory of the package), which may be convenient for programmers unaware of the extent of a package's full directory structure.

```
int GMIC_remove_from_agent
(
char *agent_path
);
```

GMI-Cmd32.lib Function Call Reference (Continued)

Function: GMIC_get_tempfile()

This function provides general convenience in deriving the name of a temporary file that can be used with the "GMIC_list()" and other functions. The function is generally not essential, but may simplify some of the work associated with listing values to a file. The function accepts a single argument, a pointer to a text string that will contain the pathname to a unique temporary file on the system that can be created by the "GMIC_list()" and other functions..

```
int GMIC_get_tempfile()
(
char *tempfile_path,
);
```

Function: GMIC_ipaddr()

This function provides general convenience in converting a hostname to an IP address. The "GMIC_connect()" function requires an IP address, and the function herein will derive the IP address from a hostname using DNS services. The function accepts two arguments: (1) the first argument is a text string containing a valid hostname (or IP address), and; (2) the second argument is a pointer to a text string that will be the IP address for the first argument. If the first argument is already an IP address, the function simply returns that value as the second argument. If the IP address cannot be resolved, the function returns non-zero.

```
int GMIC_ipaddr
(
char *hostname,
char *ipaddr
);
```

Function: GMIC_parse()

This function provides general convenience in parsing an input string, and returning a value corresponding to a particular field, based upon a "parse specification". Parse functions are documented in the "GMI Parser Reference Manual", available from the GMI Foundation website. (See that manual for more information.)

```
int GMIC_parse
(
char *parse_spec,
char *in_string,
char *out_string
);
```

Additional Information

Any programming activity can be frustrating. GMI Foundation is devoted to reducing this frustration by answering programmer questions to the extent feasible in order to promote this technology.

Further information on the Agent API, libraries, programming techniques, code samples, and additional application notes are available at the GMI-Foundation.org website. Application developers are encouraged to contact our organization at the information below, or consult with one of our technical partners. Questions, suggestions, and reasonable criticism are always welcome.

GMI Foundation
http://www.gmi-foundation.org
mailto: info@gmi-foundation.org

About Our Technical Partners

Additional information on the GMI Software, including a selection of useful GMI Apps, as well as professional and support services, is available from various members of the GMI Foundation.

A central clearinghouse for GMI applications is provided by Vallum Software, LLC, which provide support, certification, and development solutions, as well as the "Halo Management System", based on the GMI agent. Visit the link below.

Vallum Software, LLC
http://www.vallumsoftware.com
mailto: info@vallumsoftware.com