Initialization Exit Programmer's Guide

Version 70 Release 3
Initialization Exit Programmer's Guide

Version 70 Release 3
## Contents

**About this publication** ........................................ v

**Chapter 1. Optim Exit Overview** ............................ 1

**Chapter 2. Writing Your Own Exit** ........................... 3

- Prerequisites to Signing a User-Supplied Exit ................. 3
- Signing Required After Each Install ........................... 3
- The UserExit Directory ........................................ 4
- What Language can be used to Write an Exit ..................... 4
- What to do with the exit once it is compiled .................. 5
- ASCII and UNICODE Support ................................. 5
- Life of the Exit within the Process and Multithreading ........ 6

**Chapter 3. Signing and Protecting the Exit** ................. 9

- Securing the Exit using System Security ........................ 9
- Parameter Checking .............................................. 10
- Exception Logic .................................................. 10
- Signing an Exit .................................................... 10
  - The pr0sign program ........................................ 11

**Chapter 4. The Exit Entry Point** ............................ 13

- Return Value: Integer ............................................ 13
- Parameter: Integer ExitPoint .................................... 13
- Parameter: typedef Callback ................................. 15

**Chapter 5. The Callback** ...................................... 17

- Return Value: Integer ............................................ 17
- Parameter: int Function ....................................... 17
- Parameter: const char/wchar_t *Info ......................... 18
- Parameter: int InfoIdx ....................................... 18
- Parameter: char/wchar_t *Data ................................ 18
- Parameter: int DataSizeChars ................................. 18

**Chapter 6. Callback Functions** ............................... 19

- Initial Callback Functions ..................................... 19
  - Function: OPMEX_SET_SINGLE_THREAD, single thread calls to the exit ............... 19
  - Function: OPMEX_RETURN_ALL_ERRORS, return fatal errors .......................... 19
- Exit Timeout ....................................................... 20
  - Function: OPMEX_SET_TIMEOUT, override default timeout ............................ 20
  - Function: OPMEX_NOP, no operation ................................ 20
- System Information Callback Functions ......................... 21
  - Function: OPMEX_GET_CALLER, what program is running ......................... 21
  - Function: OPMEX_GET_SYS_ERROR, get last system error .......................... 21
  - Function: OPMEX_IS_CLIENT, running in client or server ....................... 22
  - Function: OPMEX_IS_SERVER, running in client or server ....................... 22
  - Function: OPMEX_IS_DEBUGGER, are we being debugged ....................... 22
  - Function: OPMEX_IS_USERLIKELY, is operator present ....................... 23
  - Function: OPMEX_IS_USERVALID, are credentials valid .......................... 23
  - Function: OPMEX_IS_USERGROUP, is user in a system group ...................... 24
- Variable Support ................................................ 25
  - Function: OPMEX_GET_VAR, get contents of variable .................. 25
  - Function: OPMEX_GETVARMAX, get max array index .................... 25
  - Function: OPMEX_SET_VAR, set variable's value .................... 26
- Global Variable Support ........................................ 26
  - Function: OPMEX_GETGLOBAL, get variable's value .................... 27
  - Function: OPMEX_SETGLOBAL, set variable's value .................... 27
- Client-Server Exit-To-Exit Support ............................ 28
  - Function: OPMEX_CSGETVAR, get client/server variable .................... 28
  - Function: OPMEX_CS_SETVAR, set variable's value .................... 29
  - Function: OPMEX_CS_CLEAR, delete all variables ....................... 29
  - Function: OPMEX_CS_CALL, call the other side .......................... 29
- Trace Support ..................................................... 30
  - Function: OPMEX_TRACE, place text into the trace file .................... 30
- System Log Support ............................................... 30
  - Function: OPMEX_LOG, place text into the system log file ............... 31
- Message Support .................................................. 31
  - Function: OPMEX_MSG, write a message to the operator ................... 32

**Chapter 7. Overriding Credentials** .......................... 33

- Server Credentials .............................................. 33
  - Function: OPMEX_SET_SERVER_CREDENTIALS, override server credentials ........ 33
- ODBC Credentials .................................................. 37
  - Function: OPMEX_SET_ODBC_CREDENTIALS, override ODBC credentials .......... 37
- Ask Operator for Credentials .................................... 38
  - Function: OPMEX_ASK_CREDENTIALS, get credentials from operator ............. 38

**Chapter 8. Variables by Exit Point** ........................ 41

- All exit points ................................................... 41
  - Exit Point OPMEX_EXIT_PROGRAM_START - PR0TOOL ...................... 41
  - Exit Point OPMEX_EXIT_PROGRAM_START - PR0CMND ...................... 42

© Copyright IBM Corp. 1994, 2010
| Exit Point OPMEX_EXIT_PROGRAM_START - PROSVER | 47 |
| Exit Point OPMEX_EXIT_PROGRAM_EXIT | 48 |
| Exit Point OPMEX_EXIT_ACTION_CALL | 48 |
| Exit Point OPMEX_EXIT_ACTION_START | 48 |
| Exit Point OPMEX_EXIT_CS_CALL | 49 |

**Notices** .............. 51
Trademarks .............. 53

**Index** ............... 55
About this publication

This document describes how you can create a custom exit for Optim™. A custom exit allows you to apply an additional layer of security to Optim, beyond the extensive security already included in the product. You might need this additional layer of security to meet any security requirements mandated by your company or government regulations.
Chapter 1. Optim Exit Overview

The IBM® Optim solution includes a mechanism that allows you to use a custom exit to apply an additional layer of security to Optim, beyond the extensive security already included in the product, to meet any security requirements mandated by your company or government regulations.

This additional security layer is accomplished through a client-supplied exit that identifies who can use Optim and which executables each user can run.

Client-supplied exits are called user-supplied exits in Optim to differentiate them from the default exit supplied with Optim. The Optim default exit allows all requests by all users, within the security limitations defined for each user or user group via the security functionality included in Optim.

The default exit is intended for clients who do not need to use a user-supplied exit, although it may also be used as a temporary solution while you create your own, customized exit. If you use the default exit, Optim user security will function as it did prior to release 6.5.

If you implement a user-supplied exit, that exit will augment the extensive security functionality already included in Optim.

Note: A user-supplied exit may also be used do other functions, such as manage user accounts, monitor user activity, force inactive sessions to timeout, audit product use, and override user authorization credentials.

Regardless of which exit you use (i.e., the default exit or your own exit), you must “sign” that exit before you can use Optim. After the exit is assigned, Optim will invoke that exit at initialization and call it at various “exit points” in the program to determine whether Optim should continue with what it was about to do. An exit point is a point within a program at which an exit routine can take control to do some external function. The exit allows you to:

• See what is being done by a given user at various points in a program’s logic,
• Ensure that the user’s request meets your company standards, and
• Change the request, if needed, to pass your company standards or forbid the request altogether.

Optim will call the exit at each exit point to verify that the user’s request meets your company standards, such as verifying that the user has permission to run a given executable. The first exit point occurs when the user launches Optim. If you use the exit to provide external security, that exit point determines whether the user has permission to access the product. If the user has the appropriate permissions, the user can continue; if not, Optim will terminate the user’s session after displaying an appropriate error message.

Beginning with Optim release 6.5, a “signed” exit must exist to use Optim, whether the exit is the Optim default exit or a user-supplied exit. To sign an exit, you must specify the “company credentials” supplied to your organization when you received Optim. Your company credentials consist of your Optim-supplied company ID, Name, and Password. The Optim setup process will automatically request these credentials during installation, so you can sign an exit.

See the IBM Optim Installation and Configuration Manual for detailed instructions on signing an exit.
Chapter 2. Writing Your Own Exit

If you want to employ the additional functionality available via a user-supplied exit, you must write your own exit, as described in this topic.

To write an exit, you must do the following:
1. Determine what you want the exit to do.
2. Determine which Optim exit points that call your exit are suitable for what you want to do.
3. Write the appropriate code to respond to those exit points within your user-supplied exit.

After you create an exit, you must compile, link, and copy the exit to the bin directory in which Optim is installed, before you can sign it. The same is true when you modify an exit. If a signed exit does not exist, you cannot use Optim.

Within your user-supplied exit, you can call the provided callback vector to get information from Optim, test information using Optim, set the information to be used by Optim, or call any other routine outside of Optim.

For each call, the exit will return a continue, stop, fail, or unknown response. An unknown code is returned if an exit point is encountered for which the exit was not coded, in which case Optim will continue.

You can use the make files provided with Optim to build the exit, copy it to the install directory, and finally “sign” it.

Note: If you want IBM Optim to create a customized exit for your company, this can be done via a Professional Services agreement with IBM Optim. Contact your IBM Optim account manager for further information.

Prerequisites to Signing a User-Supplied Exit

There are a number of prerequisites to signing a user-supplied exit, as outlined in this topic.

If you want to use your own, user-supplied exit, the following requirements apply:
• The exit load linked module name must be appropriate for your platform, as shown below:

<table>
<thead>
<tr>
<th>Platform</th>
<th>Linked Module File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Windows</td>
<td>opmexit.dll</td>
</tr>
<tr>
<td>AIX®, Solaris, Linux</td>
<td>libopmexit.so</td>
</tr>
<tr>
<td>HPUX</td>
<td>libopmexit.sl</td>
</tr>
</tbody>
</table>

• You must copy the exit file to the bin directory before you run the Configuration program in Windows or the opmusign script in UNIX. (The opmusign script is used to sign a user-specified exit, while the opmdsign script is used to sign the default Optim exit.)
• The exit file must exist in the bin directory and be signed on every Optim installation. Thus, each time you install Optim in a directory, a signed exit must exist in the bin directory.

Signing Required After Each Install

Unlike the Optim Security feature, which you must initialize once per Optim Directory, you must sign an exit each time you install Optim on a machine.
Moreover, when you upgrade to a new Optim release, you must sign a valid exit for each Optim installation, before you can use Optim. (If you install a new version of Optim over a previous version, you must either re-copy your exit to the `bin` directory and resign it, or sign the default exit.) The same is true if you reinstall Optim.

Anytime you replace a signed exit executable (i.e., opmexit.dll, libopmexit.so, or libopmexit.sl) with another version of that exit, you must sign the updated exit to use Optim. This is true, even if the executable was previously signed (e.g., in another installation or copied from a backup of a signed exit).

**The UserExit Directory**

A “UserExit” directory is included on the distribution CDs, along with the other items outlined in this topic.

The distribution CDs include the following, in addition to the “UserExit” directory:

- A “C” header that contains defines for all of the Optim exit points, callback functions and anything they reference
- The make files for each supported platform

**Note:** An LDAP reference implementation is also available on the CD if you want to create an exit using LDAP.

You can use this programmer’s guide, the sample, and the make files to modify the sample exit to create your own exit, or you can write your exit from “scratch.” After you create your own exit, you must compile, link, and install the exit.

The sample exit contains many examples of the Callback “API,” as well as comments describing why things were coded a given way. The sample exit, however, is not meant to be used “as is.” In fact, it should not be used without substantial modifications.

The makes files are called OPMEXIT.xxx, where “xxx” is the platform. To use these, run the make for the appropriate platform, and then specify where to put the output, as follows:

```
make –f opmexit.win OUTDIR=output-directory
```

Only the Windows version of the make files creates a debug version of the exit. We recommend that you build the exit within directories external to the Optim install directories to keep this information private.

**What Language can be used to Write an Exit**

A great deal of flexibility is allowed in choosing the language in which you write your user-supplied exit.

You can write your user-supplied exit in any language that can:

1. Publish an external / exported symbol that can be located using the system API (dlsym and GetProcAddress, respectively).
2. Call a “cdecl” calling convention function.
3. Produce a shared object in UNIX, or a dynamic link library in Windows.

You also can write a stub in almost any language to do the above.

The load module must be named as follows for each supported platform:

<table>
<thead>
<tr>
<th>Load Module Name</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>opmexit.dll</td>
<td>Microsoft Windows</td>
</tr>
</tbody>
</table>
There is no facility to use any other naming convention (the make files ensure this).

**What to do with the exit once it is compiled**

The resulting load module must be copied into the bin directory where Optim is installed. It must then be “signed” in that directory using the **prosign** program, which you can do interactively via the **Optim Configuration** program in Windows.

**Note:** See Appendix A in the IBM Optim Installation and Configuration Manual for detailed instructions on signing a user-supplied exit in UNIX.

If there is a coding error in the exit, Optim may become inoperable for anyone using that exit (i.e., anyone using that install directory). Therefore, it is suggested you first test your exit on a test installation.

It is easy to revert to the Optim default exit, however, if your user-supplied exit fails in production, although any checks included in your user-supplied exit will be bypassed during the time the default exit is in use.

**ASCII and UNICODE Support**

The exit is loaded during startup of every Optim program, and its signing is “challenged” to ensure it is the exit you signed; then one of two external/exported entry points is found within the exit's load module: “OptimExitW” or “OptimExitA”.

- “OptimExitW” is the UNICODE (wide) entry point
- “OptimExitA” is the ASCII (multibyte) entry point

Each of these entry points has two parameters: a code depicting the exit point that is calling the exit, and the address of a callback you can use to get, set, or test information using Optim from your code. The differences between the two entry points is that the callback provided to the ASCII (multibyte) uses “char” parameters, while the UNICODE (wide) uses “wchar_t” parameters.

The wide characters are platform specific: some platforms use 16 bit UTF-16 with surrogates (Windows, I86 Linux, AIX), while others use 32 bit UTF-32 encodings (Solaris and HPUX). Unicode is also subject to “Endian” bit order within the numeric Unicode characters. In all cases, Optim uses the “native” form of Unicode.

Multibyte depends on code pages and the “locale” of the process. It can be single-byte ASCII, such as the code page 1252 (English United States) in Windows, ISO 8859 in UNIX, or the built-in “C” code page in Windows or UNIX, or it can be multibyte, such as UTF-8 or SJIS.

You can use the utility program “prolocl” (supplied with Optim) to determine whether the local you want to use is valid for Optim.

The “OptimExitW” (Unicode/Wide) entry point is addressed first. If it is found, it is used and the ASCII/multibyte version is ignored, if also present. If the Unicode/Wide entry point is not found, the “OptimExitA” (ASCII/multibyte) entry point is addressed. If neither is found, the exit is considered “corrupt” and the Optim program that is loading it will exit and issue the appropriate error messages to the operator.
The Optim exit load module is validated to ensure it contains one of these entry points when it is signed, prior to being used in Optim.

Internally, Optim is written to use Unicode. For it to convert to or from Unicode, it uses the appropriate system API for multibyte to wide (or wide to multibyte) conversions, which use the locale of the process to determine the translation.

This can result in lost characters if the data being passed to or from the exit was put there from a system using a locale other then the one that is active in the process.

That is, while Unicode encompasses all code pages and all characters in all languages, locales refer to one or a few code pages and the characters within them. (There are exceptions to how Unicode is handled on some platforms, but this is basically why you may have unknown characters.)

In our case, we substitute a default character for the invalid character when converting from Unicode (determined by the code pages we support, usually the equivalent of a “?”).

If the conversion fails for any reason other than an unknown character, it is considered a fatal error and an exception is raised.

In this regard, Optim will run a remote request from a client on an Optim server using the same locale that is active in the client process at the time the action is started (we translate between platforms).

---

### Life of the Exit within the Process and Multithreading

The exit is loaded when the Optim program loads, and it remains in memory until the program terminates or a fatal error occurs in the exit, in which case the exit is immediately unloaded. If the exit is unloaded, it is never reloaded in the same process.

There is one exception to this rule in the Windows Configuration program. When you click Options → Sign Optim Exit from the Configuration menu to re-sign an exit, the existing exit is unloaded, prior to signing the new one. The newly signed exit is then loaded, and the program start exit point for Configuration is called after the signing is done.

Since the life of the exit is the life of the operating system’s process, you can use static variables inside the exit to hold things between calls to the exit.

There is an initial exit point that indicates the exit was just loaded (OPMEX_EXIT_INITIAL), and one indicating the program is exiting and the exit is about to be unloaded (OPMEX_EXIT_PROGRAM_EXIT). You can use these two exit points to handle the initialization and termination of any external linkages you must maintain.

**Note:** If the program aborts (or is killed / terminated), the OPMEX_EXIT_PROGRAM_EXIT exit point may not be called.

The exit will be multithreaded if you do not specify that it should be single-threaded during the initial exit point. While all callback functions are thread-safe, static variables or other code in your exit or called by your exit may not be. Therefore, you must provide for the simultaneous access for multiple threads.

“Single-threaded” means the exit will be called by only one thread at a time, and not necessarily the same thread. While we try to reuse the same physical system thread, that thread may timeout from disuse or have an exception that requires it to be terminated, in which case the next exit call will create a different thread.

“Multithreaded” means a pool of threads is used to call your exit, but each thread has a disuse timeout that may expire, resulting in the thread being terminated to free system resources.
To reiterate, the same thread may not be used across all exit calls, so you should not use thread storage on the assumption that the same thread will be used for every call.

**Note:** You must call the callback on the same thread in which your exit was entered. Failure to do so will cause the exit to terminate and unload. If you must use other threads, use thread synchronization techniques, so the calling thread is used to call back into Optim.
Chapter 3. Signing and Protecting the Exit

Since the exit facility is used by some clients to provide additional security, it must be protected from unauthorized access and manipulation. Therefore, the following security features are incorporated in the exit logic to help ensure this:

1. When an exit is signed, a challenge is issued to validate the signing user's credentials. After the user's credentials are verified, the new exit is loaded. (If a previously signed exit is being replaced, that exit is called to ensure that the user signing the new exit has the authority to do so.)

2. The user who signs the exit must know the company Id, Name, and Password assigned to your company when you received Optim. You should keep these entries confidential (i.e., only divulged on a need-to-know basis) and store them in a secure location for future use. (The Name assigned to your company may not match the spelling or punctuation used in your company’s actual name.)

3. You must have system authority to write to the Optim install directory to sign an exit. You may also need permission from your system administrator to write to the install directory, depending on where that directory resides.

4. If a signed exit does not exist, you cannot use Optim, so you must sign a valid exit to use Optim. You can sign the default exit supplied with Optim (which does not provide any additional user security), or a user-supplied exit of your own creation that includes additional security features.

5. The Optim default exit is delivered unsigned to ensure:
   • it is signed by a user with the appropriate company credentials, and
   • the person signing the default exit is authorized to make the decision to use that exit, as opposed to a user-supplied exit. This is important because the default exit returns a “continue” code at every exit point. Thus, if the default exit was delivered signed, it could be used to bypass any security checks and additional functionality included your user-supplied exit (assuming you already created one).

It is advantageous to statically link as much as possible into the exit load module, so it is taken into consideration during signing (i.e., to ensure no one usurps any checks in the exit by changing a system “PATH” statement to override a link).

If you do not need this level of security for your exit, you still need to sign your exit and be bound by the other requirements.

Securing the Exit using System Security

All supported platforms allow you to restrict who can read, write, or execute a file (program). To protect your installation, we suggest you do the following or something even more restrictive:

• Protect the install bin directory by setting its permissions to a specific system group (such as an OPTIM system group), and give the group execute permission within that directory, but not read or write access to its contents. Then create one user id with full read-write access to the bin directory. No other permissions (or ACLs in Windows) should be set for that directory.

• Assign all Optim users to the same system group, such as an OPTIM system group. (The work volume for your IT staff will vary based on the number of users who can access Optim.)

• The sole user assigned read-write access to the directory is the only one allowed to install the product and sign exits in that directory. (That user does not need to know what is in the exit.)

The above procedures ensure that:

1. Only members of the OPTIM system group can execute programs in the install directory.
2. No member of the OPTIM system group or any other user group can copy the executables in the install directory to create their own environment.

3. No user, other than the one assigned read-write access to the install directory, can write to that directory or sign an exit.

In a UNIX environment, members of the OPTIM system group should be assigned read-only access to the “etc” directory that holds the configuration files. In addition, one user should be assigned read-write access to that directory to update the configuration files.

While this allows anyone to read the configuration files (including the programs used to set up the environment), you can use the “pr0pass” program to encrypt any passwords in the configurations file.

If you put the bin (and “etc”, etc.) directories on a file share, make sure you specify local directories for the Temp, Data, Archive, Archive Index, Archive Browse, and Scheduling (Windows only) directories using Personal Options or the Optim Server Applet in Windows, and pstserv.cfg or pstlocal.cfg in UNIX. PSTHOME for UNIX would point to the file share (mount point).

Parameter Checking

All parameters to the callback are validated. If any parameter is in error, an exception occurs. The details of these are explained below for each of the parameters passed into the callback.

Each call into the exit and from the exit via the callback is wrapped in “try-catch” blocks to catch exceptions, so that errors in the exit will not bring down Optim.

Exception Logic

If any abnormal situation happens, an “internal exception” occurs within Optim, resulting in the following:

1. A trace entry is made in the appropriate trace file for use by Optim support personnel.
2. A message is sent to the operator. For actions, this is put into the results of an action or within the error message bar.
3. If there is no action and an operator is likely involved, a pop-up dialog displays.
4. In UNIX, a message is written to “syslog.” In Windows, an event is recorded in the Event Log. These items carry a severity of “fatal.” Moreover, if Administrator Notification is turned on, an email will be sent to the proper administrators to address the failure.
5. For most occurrences, the program is terminated.

Signing an Exit

The method of signing an exit in a Windows environment differs from the method used in a UNIX environment.

• In a Windows environment, the Optim Configuration program allows you to sign either the Optim default exit or a user-supplied exit via a Sign Optim Exit dialog. Optim will automatically display that dialog anytime you execute the Configuration program if a signed exit does not exist. (After Optim is installed, you can redisplay the Sign Optim Exit dialog at any time to sign a new exit by clicking Options • Sign Optim Exit from the Configuration main window. You also can use a “pr0sign.exe” program to sign a new exit.)

• In a UNIX environment, you can only sign the default exit during Optim installation. During installation, the Optim Setup program will ask if you want to “sign” the default exit:
  – If you answer “Yes” to this question and provide the appropriate credentials, the default exit is “signed,” and you can begin using Optim.
If you answer "No" to this question, you will be prevented from executing any Optim program (other than the signing program), until you sign the default exit or a user-supplied exit.

Following setup, you can sign the Optim default exit or a user-supplied exit using one of these methods:

- Run the **opmdsign** script file, located in the rt/sbin directory, to sign the Optim default exit. The letter “d” in the script’s name indicates that it is used to sign the default exit.
- Run the **opmusign** script file, located in the rt/sbin directory, to sign a user-supplied exit of your own creation. The letter “u” in the script’s name indicates that it is used to sign a user-specified exit. (See Appendix A in the IBM Optim Installation and Configuration Manual for further information on signing an exit in UNIX using the opmdsign or opmusign script file.)
- Use the **pr0sign** program to sign either the default exit or a user-supplied exit. However, before you run the **pr0sign** program, you must ensure that the environment is set up, as noted below.

### The **pr0sign** program

The **pr0sign** program exists in the **bin** directory. You must ensure it is in the PATH in Windows and UNIX (and LIBPATH / SHLIB_PATH / LD_LIBRARY_PATH in UNIX for the shared objects loaded from that directory).

**Note:** If you use either the **opmdsign** or **opmusign** script file to sign an exit, the script file will automatically set up the environment and call **pr0sign**.

To sign the default exit, use the following command line:

```
pr0sign -d <company-id> "<company name>" company-password
```

To sign a user-supplied exit after you have copied it to the **bin** directory, use the following command line:

```
pr0sign -s <company-id> "<company name>" company-password
```

To see who signed the exit, use the following command line:

```
pr0sign -i <company-id> "<company name>" company-password
```

If you do not specify your company’s id, name and password, you will be prompted for that information via a dialog in Windows or the console in UNIX. The advantage to using the dialog or console is that you can edit your entries, and your password will not be displayed when you type it.

The company name must be entered in quotes. In UNIX, if the name contains spaces, you have to "escape" the quotes in some shells, as shown below:

```
pr0sign -d <company-id> \"<company name>\" company-password
```

Alternatively, you can omit the company credentials from the command line and specify that information when you are prompted for it by the Windows dialog or UNIX console. If you do that, you do not need to enclose the company name in quotes.

You must know your company’s credentials to use the “-i” parameter to find out who signed the current exit, because the output from that action indicates who has authority to sign an exit, as well as who has write access to the bin directory. This information could be used by someone to impersonate that user, which is the reason your company credentials are required to view that information.

**Note:** See the IBM Optim Installation and Configuration Manual for further information on signing an exit in Windows or UNIX.
Chapter 4. The Exit Entry Point

Use one of the following protocols to code one of the two entry points in the exit:

**C++, UNIX**

```c
extern "C" int OptimExitA(int ExitPoint tOptimExitCallbackA Callback);
extern "C" int OptimExitW(int ExitPoint tOptimExitCallbackW Callback);
```

**C++, Windows**

```c
extern "C" int __declspec(dllexport) OptimExitA(int ExitPoint tOptimExitCallbackA Callback);
extern "C" int __declspec(dllexport) OptimExitW(int ExitPoint tOptimExitCallbackW Callback);
```

**C, UNIX**

```c
extern int OptimExitA(int ExitPoint tOptimExitCallbackA Callback);
extern int OptimExitW(int ExitPoint tOptimExitCallbackW Callback);
```

**C, Windows**

```c
int __declspec(dllexport) OptimExitA(int ExitPoint tOptimExitCallbackA Callback);
int __declspec(dllexport) OptimExitW(int ExitPoint tOptimExitCallbackW Callback);
```

Both `tOptimExitCallbackA` and `tOptimExitCallbackW` are defined in the OPMEXIT.H header for your use.

These declarations will cause the entry point to be externalized so Optim can find them in the shared object / dynamic link library.

**Return Value: Integer**

This enumerated value, defined in OPMEXIT.H, indicates what Optim should do when the exit returns.

<table>
<thead>
<tr>
<th>Define in OPMEXIT.H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEX_RC_FAILURE</td>
<td>A fatal error occurred. Optim will unload the exit and notify the operator; in some cases, this will also terminate the program.</td>
</tr>
<tr>
<td>OPMEX_RC_CONTINUE</td>
<td>Optim should continue.</td>
</tr>
<tr>
<td>OPMEX_RC_STOP</td>
<td>Optim should continue, but will not perform the function for which the exit point was called. If the exit was called to indicate a program start, the program is terminated, if it was called because an Action is about to be started, the action is not started.</td>
</tr>
<tr>
<td>OPMEX_RC_UNKNOWN</td>
<td>The exit point is unknown to the exit, so Optim will continue with whatever default is best. This is the preferred logic for any exit point the exit does not want to process or for a default return code.</td>
</tr>
</tbody>
</table>

If any other value is returned, it is considered a fatal error and an exception is raised.

**Parameter: Integer ExitPoint**

This enumerated value, defined in OPMEXIT.H, indicates which Optim exit point is calling the exit.

<table>
<thead>
<tr>
<th>Define in OPMEXIT.H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEXIT_EXIT_INITIAL</td>
<td>The exit module has just been loaded. This exit point is always called single-threaded.</td>
</tr>
<tr>
<td>Define in OPMEXIT.H</td>
<td>Meaning</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_PROGRAM_START</td>
<td>A program is starting up; variables contain the identifier of the program and its arguments.</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_PROGRAM_EXIT</td>
<td>The program is terminating, and the exit code is provided as a variable. The exit is unloaded shortly after this call.</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_ACTION_CALL</td>
<td>An action is preparing to start. In a client-server request, this occurs on the client.</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_ACTION_START</td>
<td>An action is being started; in a client-server request, this occurs on the server side.</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_CS_CALL</td>
<td>Your exit on the other side of a client-server connection is calling you.</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_ODBC_CONNECT</td>
<td>An ODBC connection is about to be performed using either ODM or the Optim ODBC server (PR0COMS).</td>
</tr>
<tr>
<td>OPMEXIT_EXIT_ODBC_DISCONNECT</td>
<td>An ODBC connection is about to be disconnected using either ODM or the Optim ODBC server (PR0COMS).</td>
</tr>
</tbody>
</table>

Some explanation of the flow of control is required to understand when this exit point is called.

- **PR0TOOL** (Toolbox, Windows)
  OPMEXIT_EXIT_PROGRAM_START is called when the program is loaded, and
  OPMEXIT_EXIT_PROGRAM_EXIT is called before the program closes the trace file and returns to the operating system.
  When you use *File > Run* for an action, the OPMEXIT_EXIT_ACTION_CALL exit point is called. If it is allowed to continue, it will start a Request Process (PR0SVER), perform the action locally, or start a connection to the specified Optim server.

- **PR0SVER** (Request Process)
  OPMEXIT_EXIT_PROGRAM_START is called when the program is loaded, and
  OPMEXIT_EXIT_PROGRAM_EXIT is called before the program closes the trace file and returns to the operating system.
  Communications to the client is not possible during exit processing.
  Before the Request Process starts the action, it calls the OPMEXIT_EXIT_ACTION_START exit point.

  **Note:** To validate values on the server side from the various editors or actions, a “mirror” process may be created on the server to do some things, such as validate files. These processes have a component identifier of OPMEX_COMP_XAPS. The component (editor or action) that called it will be in the variable that contains the parent component identifier (see the OPMEX_COMP_* in OPMEXIT.H).

- **PR0CMND** (Command Line)
  OPMEXIT_EXIT_PROGRAM_START is called when the program is loaded, and
  OPMEXIT_EXIT_PROGRAM_EXIT is called before the program closes the trace file and returns to the operating system.
  OPMEXIT_EXIT_ACTION_CALL is called after the parameters are validated and just prior to each request starting. If the command line has a “@parameter” file with multiple requests, this is called for each request.
  Command line does not use the PR0SVER Request Process program; it performs the actions internally, one after another. It will call OPMEXIT_EXIT_ACTION_START, however, prior to starting each of the actions.

- **PR0SMON** (Scheduling Monitor)
  OPMEXIT_EXIT_PROGRAM_START is called when the program is loaded, and
  OPMEXIT_EXIT_PROGRAM_EXIT is called before the program closes the trace file and returns to the operating system.
As each schedule job step is started, OPMEX_EXIT_ACTION_CALL is called. If it is allowed to continue, it will start a Request Process (PR0SVER) to perform the action.

- **PR0SVCE** (Server Service/Daemon)
- **PR0ASAP** (Window’s Server Applet)
- **PR0CNFG** (Windows Configuration)

OPMEX_EXIT_PROGRAM_START is called when the program is loaded, and OPMEX_EXIT_PROGRAM_EXIT is called before the program closes the trace file and returns to the operating system.

**Parameter: typedef Callback**

This parameter identifies the address of the callback routine in Optim that is called to get, set, or test information.

The address does not change, regardless of the threading model. If you want to use this in multiple places within your code, you can copy this to a static variable.
Chapter 5. The Callback

The exit is designed so you do not have to change your exit when new exit points or features are added to Optim. To do that, the exit is passed very little information to the entry point and instead provides the address of a callback to get, set, and test information.

The protocols for the exits are in the OPMEXIT.H header:

```c
typedef long (*tOptimExitCallbackA) (int Function,
const char *Info,
long InfoIdx,
char *Data,
int DataSizeChars);
```

```c
typedef long (*tOptimExitCallbackW) (int Function,
const wchar_t *Info,
wchar_t *Data,
ínt DataSizeChars);
```

Return Value: Integer

The return value of the callback is determined by the function you call. Some functions return values and some return success/failure codes.

See the description for each callback function, listed below, to determine what is used.

<table>
<thead>
<tr>
<th>Define in OPMEXIT.H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEX_OK</td>
<td>The callback has no errors.</td>
</tr>
<tr>
<td>OPMEX_ERR_TOOLONG</td>
<td>The Data area is not large enough to hold the returned data. You will get this return code only if you called the OPMEX_RETURN_ALL_ERRORS callback function; if not, this is an exception.</td>
</tr>
<tr>
<td>OPMEX_ERR_NOT_FOUND</td>
<td>The specified name was not found.</td>
</tr>
<tr>
<td>OPMEX_ERR_NOT_ALLOWED</td>
<td>This is a valid function, but you are not allowed to do it at this time.</td>
</tr>
<tr>
<td>OPMEX_ERR_ACCESS_DENIED</td>
<td>You are not allowed to modify something you wanted changed.</td>
</tr>
<tr>
<td>OPMEX_ERR_INVALID</td>
<td>One or more pieces of information are missing or invalid in the arguments you passed the exit. You will receive this return code only if you called the OPMEX_RETURN_ALL_ERRORS callback function; if not, this is an exception.</td>
</tr>
<tr>
<td>OPMEX_TRUE</td>
<td>For functions returning a Boolean result, this is returned for TRUE.</td>
</tr>
<tr>
<td>OPMEX_FALSE</td>
<td>For functions returning a Boolean result, this is returned for FALSE.</td>
</tr>
<tr>
<td>Other</td>
<td>Some functions return the results as the return value, instead of in the Data parameter.</td>
</tr>
</tbody>
</table>

Parameter: int Function

This enumerated value depicts what the callback will do on behalf of the exit.
There are currently 28 functions that may be called, and each one is explained in its own section, below. If an unknown function is passed, an exception occurs.

**Parameter: const char/wchar_t *Info**

This parameter is used to pass information to the callback to determine what it should do. For instance, if you want to retrieve the value for a variable, this would be the variable name.

The `Info` parameter can be NULL, if not used. If used, it must be a null-terminated “C” string, not to exceed 8,192 characters, or an exception will occur.

If the format of the data in the `Info` parameter is invalid, an exception will occur (e.g., in some functions, multiple parameters are passed, separated by commas).

If accessing the data in `Info` results in a system exception, an internal exception is also raised (addressability issues, read protection, etc.).

**Parameter: int InfoIdx**

This parameter is used to modify the `Info` parameter. It usually contains the index (relative to zero) of an element within a variable that is an array.

**Parameter: char/wchar_t *Data**

This parameter is used to pass data back from the callback, but it may also be used to pass information into the callback.

The `Data` parameter can be NULL, if not used. If `Data` is used to pass data to the callback, the string must not exceed the “DataSizeChars” value or an exception will occur. Moreover, the format of the data within it must be valid or an exception will occur.

When copying data back into the exit using `Data`, the string being copied (including the null terminated “C” string character) must fit within “DataSizeChars” value or an exception will occur.

If accessing the data in `Data` results in a system exception, an internal exception is also raised (addressability issues, read/write protection, etc.).

**Parameter: int DataSizeChars**

This parameter indicates how large the `Data` parameter string array is.

This value is represented in characters, not bytes, to handle differences between Unicode and ASCII, as well as the different wchar_t encoding sizes. The size specified should also include the trailing null character for “C” style strings.

Macro “OPMEX_STRSIZE(str)” in OPMEXIT.H will return the proper character size for any kind of string array (but not pointers).
Chapter 6. Callback Functions

This chapter describes the various callback functions.

Initial Callback Functions

These callback functions are valid only during the OPMEX_EXIT_INITIAL call; calling these at any other time will result in an OPMEX_NOT_ALLOWED return code, but will not result in an exception.

Function: OPMEX_SET_SINGLE_THREAD, single thread calls to the exit

This topic describes the callback function OPMEX_SET_SINGLE_THREAD (single thread calls to the exit).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>Always OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Call this function within the initial exit point to indicate that your exit should be called single-threaded or, more precisely, on only one thread at a time.

If possible, a single thread is used throughout the life of the exit. However, if the thread is not used within the disuse timeout period or an error occurs on the thread, it will be terminated, in which case another thread will be created for any future exit calls.

For multi-threaded exits, a pool of threads is created and reused. However, if they are not used within the disuse timeout period, the thread will exit to free resources and another one will be created, when needed.

Function: OPMEX_RETURN_ALL_ERRORS, return fatal errors

This topic describes the callback function OPMEX_RETURN_ALL_ERRORS (return fatal errors).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>Always OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Call this function within the initial exit point if you want some errors that are normally considered fatal to be returned to your exit, instead. Normally, there is nothing you can do about these errors, so an exception as a default action is reasonable.
<table>
<thead>
<tr>
<th>Define in OPMEXIT.H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEX_ERR_TOOLONG</td>
<td>The <strong>Data</strong> area is not large enough to hold the returned data.</td>
</tr>
<tr>
<td>OPMEX_ERR_INVALID</td>
<td>One or more pieces of information are missing or invalid in the arguments you passed the exit.</td>
</tr>
</tbody>
</table>

**Exit Timeout**

Every call into the exit must either return or call the callback within a specified timeout period (60 seconds is the default value, if not overridden). If the exit does not do one of these within the timeout period, Optim will assume a hang or runaway condition has occurred and will terminate the exit.

You can override the timeout for the duration of a single call into the exit or periodically perform a “do nothing” callback to satisfy the timeout condition.

This setting is not persistent: you must reset it on each call into your exit, but it is valid for the duration of that exit point call.

**Function: OPMEX_SET_TIMEOUT, override default timeout**

This topic describes the callback function OPMEX_SET_TIMEOUT (override default timeout).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>New timeout value in seconds</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

This function sets a new timeout value for the duration of the call into the exit. The value specified for **Infoldx** can be from 10 seconds to 600 seconds (i.e., 10 minutes). If a value of less than 10 (seconds) is specified, 10 seconds is assumed; similarly, if a value of more than 600 (seconds) is specified, 600 seconds is assumed.

If you are changing the timeout to display a dialog for credentials, it is better to use the OPMEX_ASK_CREDENTIALS callback function, which does not have a timeout feature.

**Function: OPMEX_NOP, no operation**

This topic describes the callback function OPMEX_NOP (no operation). This function resets the timeout timer.

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>
System Information Callback Functions

These callback functions are valid at all times and return the status of the overall program or the system-related information.

Function: OPMEX_GET_CALLER, what program is running

This topic describes the callback function OPMEX_GET_CALLER (what program is running).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>Identifier of the calling program</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

Currently, one of the following enumerated values from OPMEXIT.H will be returned.

<table>
<thead>
<tr>
<th>Define in OPMEXIT.H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEX_CALLER_TOOL</td>
<td>PROTOOL (Windows, Toolbox)</td>
</tr>
<tr>
<td>OPMEX_CALLER_CMND</td>
<td>PROCMD (Command Line)</td>
</tr>
<tr>
<td>OPMEX_CALLER_SMON</td>
<td>PROSMON (Scheduling Monitor)</td>
</tr>
<tr>
<td>OPMEX_CALLER_SVER</td>
<td>PROSVER (Request Process)</td>
</tr>
<tr>
<td>OPMEX_CALLER_SVCE</td>
<td>PROSVCE (Server Daemon/Service)</td>
</tr>
<tr>
<td>OPMEX_CALLER_CONFIG</td>
<td>PROCNFG (Windows, Configuration)</td>
</tr>
<tr>
<td>OPMEX_CALLER_ASAP</td>
<td>PROASAP (Windows, Server Applet)</td>
</tr>
</tbody>
</table>

Function: OPMEX_GET_SYS_ERROR, get last system error

This topic describes the callback function OPMEX_GET_SYS_ERROR (get last system error).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>Last known system error code</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Text for the error (Optional)</td>
</tr>
</tbody>
</table>

This function returns the last system error that occurred from a callback [GetLastError() in Windows or errno in UNIX]. These values are cached from each callback and may return different results than if you called the system API yourself.

A callback may result in multiple errors, as well as non-system errors. If there are multiple errors, the first system type error is returned. If there are no system type errors, zero (0) is returned.
If you provided a Data area, the text for the error is returned. If there are multiple errors, they will all be listed within this text, separated by new lines ("\n"), including non-system errors. If there are no errors, Data will contain a null string. It may also contain non-system error text, even if this function returns zero.

OPMEX_ERR_TOOLONG is not raised for this call, so as much text as can fit in the area provided will be returned from the last error, followed by a “C” null-termination string character.

**Function: OPMEX_IS_CLIENT, running in client or server**

This topic describes the callback function OPMEX_IS_CLIENT (running in client or server).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

The exit is being called from the client side (or if no server connection is established). This is the inverse of OPMEX_IS_SERVER.

**Function: OPMEX_IS_SERVER, running in client or server**

This topic describes the callback function OPMEX_IS_SERVER (running in client or server).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

The exit is being called from the server side (PR0SVCE Optim server itself, or PR0SVER Request Process, under a PR0SVCE Optim server). This is the inverse of OPMEX_IS_CLIENT.

**Function: OPMEX_IS_DEBUGGER, are we being debugged**

This topic describes the callback function OPMEX_IS_DEBUGGER (are we being debugged).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

OPMEX_TRUE is returned by this function if:
• a recognized debugger is present on the chain of parent processes to this process (in UNIX), or
• the system API indicates the process is currently running under a debugger (in Windows).

Use this function to determine whether you should enter a hard breakpoint during debugging. To “hacker-proof” an exit, you may want to return a failure code if a debugger is found.

**Function: OPMEX_IS_USERLIKELY, is operator present**
This topic describes the callback function OPMEX_IS_USERLIKELY (is operator present).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>Not used</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not used</td>
</tr>
</tbody>
</table>

OPMEX_TRUE is returned by this function if an operator is likely available to answer a question. A user/operator is likely present if the client is:
• PR0TOOL (toolbox)
• PR0ASAP (Windows Server Applet)
• PR0CMND (command line) without the /QUIET command line option
• PR0CNFG (Configuration), unless run from the command line.

Since the Optim scheduler is an interactive program, OPMEX_TRUE is returned if the exit is being called by the scheduler (PR0SMON), even though the scheduler usually runs unattended.

A TRUE response is no guarantee that an operator is actually present. For example, the operator may have stepped away from the workstation after executing an action.

If this is being called from the server side (OPMEX_IS_SERVER is TRUE), the check is made on the client side. You would then use the Client-Server Exit-to-Exit facility to communicate to the client to ask any questions, as explained below.

**Function: OPMEX_IS_USERVALID, are credentials valid**
This topic describes the callback function OPMEX_IS_USERVALID (are credentials valid).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>PAM Service Id (Optional)</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>USER,PASSWORD[,DOMAIN]</td>
</tr>
</tbody>
</table>

OPMEX_TRUE is returned by this function if the specified credentials are valid, according to the system API.
UNIX Only:

**Info** can contain a PAM (Pluggable Authentication Module) service id to validate these credentials. The default Optim “rt4s” PAM service id is used if an empty string is provided or NULL is passed and PAM is used on the UNIX computer. If PAM is not installed, this field is ignored.

PAM Service Ids are used to determine which set of PAM Security Modules (shared objects) are used and, thus, what technique is used to do the validation. For instance, if you have a Service Id called “ldap” that uses the pam_ldap module or your own version of that module, LDAP will be used to authenticate the user.

To secure the credentials in the **Data** parameter from hackers, the password is immediately encrypted and the entire **Data** buffer is cleared to zeros, so do not expect **Data** to contain anything useful after the call.

The format of **Data** is as follows: **USERID, PASSWORD [,DOMAIN]**

(the commas are mandatory, but you may omit the last one if a Domain is not specified).

<table>
<thead>
<tr>
<th>Argument</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>The user to be validated.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>The password to validate against the specified user id.</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>In a UNIX environment that uses PAM to validate users, the specified domain is used with the PAM_RHOST attribute to access PAM on a remote host. (If a domain is not specified, the Optim “rt4s” default service id is used.)</td>
</tr>
<tr>
<td></td>
<td>In a UNIX environment that does not use PAM, this setting is ignored (e.g., LAM in AIX).</td>
</tr>
<tr>
<td></td>
<td>In a Windows environment, this setting identifies the domain used to validate the user. If a computer name is specified, Windows will usually fail, unless the computer is the local computer. (The current computer is usually used when a domain is not specified in Windows).</td>
</tr>
</tbody>
</table>

If the USERID, PASSWORD, or DOMAIN contains spaces, enclose the argument in double quotes (“”). If the argument contains double quotes, enclose the string in single quotes (’’). If the argument contains double and single quotes, enclose the string in single quotes (’’) and change the imbedded single quotes to double quotes (i.e., from ’’ to “’’).

When OPMEX_FALSE is returned, the credentials are not specified, per the security guidelines.

**Function: OPMEX_IS_USER_GROUP, is user in a system group**

This topic describes the callback function OPMEX_IS_USER_GROUP (is user in a system group).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>System group name</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Credentials</td>
</tr>
</tbody>
</table>

This function returns OPMEX_TRUE if the specified credentials represent a user in the specified system group.

The format of **Data** is: **USERID [,DOMAIN]**.
### Variable Support

In this context, variables are named pieces of information that each exit point sets, prior to calling the exit.

These variables are usually unique to the exit point and are created to allow you to determine whether you want the exit point to continue. See Chapter 8, “Variables by Exit Point,” on page 41 for a complete list of the exit points used, and the variables they set.

### Function: OPMEX_GET_VAR, get contents of variable

This topic describes the callback function OPMEX_GET_VAR (get contents of variable).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK, OPMEX_ERR_NOT_FOUND, OPMEX_ERR_TOOLONG, or length of variable (see below)</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Index of variable if array (relative to zero)</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Value of the variable</td>
</tr>
</tbody>
</table>

This function returns the value of a variable that the exit point set for your use.

If the variable name is not found or the index is an array that is out of bounds, OPMEX_ERR_NOT_FOUND is returned, and Data is set to an empty string.

If the data area is too small to hold the value and you called the OPMEX_RETURN_ALL_ERRORS callback during the initial call, OPMEX_ERR_TOOLONG is returned. If you did not ask to receive fatal errors, an exception is raised.

To get the length of the variable without receiving its value, pass NULL as the Data variable. The length of the variable is then returned by the call in characters, without the trailing null end of the string. If the variable is empty (“”), zero is returned. If the variable is not found, -1 is returned.

**Note:** Arrays for these variables are “sparse” (not all indexes may be present).

### Function: OPMEX_GET_VARMAX, get max array index

This topic describes the callback function OPMEX_GET_VARMAX (get max array index).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>The user to be validated.</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>In UNIX, this is ignored. The “getgrpent()” API is used to validate that the user is in the group, and that API only uses the local /etc/group file. In Windows, this is the domain used to enumerate the user groups and determine whether the user belongs to a specified group. If omitted, Windows usually uses the local computer.</td>
</tr>
</tbody>
</table>
This function either returns the highest index of the specified variable (relative to zero), or -1 if the variable name is not found.

Arrays are “sparse,” so you may get back a “5” indicating NAME(5) is the highest index, but be aware that NAME(0) through NAME(4) may not even exist.

**Function: OPMEX_SET_VAR, set variable’s value**

This topic describes the callback function OPMEX_SET_VAR (set variable’s value).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_ERR_NOT_FOUND,</td>
</tr>
<tr>
<td></td>
<td>OPMEX_ERR_ACCESS_DENIED,</td>
</tr>
<tr>
<td></td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Index of variable if array (relative to zero)</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>New value for variable</td>
</tr>
</tbody>
</table>

When implemented, this function will replace the value of Variable Name at Index if it exists and you are allowed to write to it. Passing NULL in **Data** is the same as setting the variable to an empty (“”) string.

**Note:** This function is designed for future use. Currently, no exit point allows variables to be changed.

Some variables cannot be set by the exit; each exit point determines whether you can modify a given variable. If you can modify the variable, you can use this callback to update the variable’s value.

You cannot, however, create your own variables: the variables must already exist, since the exit points only recognize the variables they set.

**Global Variable Support**

Global variables are created by the **exit**, unlike the variables described in the previous section, which are created by the exit points.

In Windows, “global” variables are maintained within the registry in the Optim section of the Software section of the proper hive. In UNIX, the emulated registry is used. The global variables are saved encrypted, so you can keep sensitive data in them, such as passwords.

Since the data is maintained in the registry, it can be deleted or modified by anyone who has the authority to update the registry. Thus, it is possible the data may not be there on subsequent calls, even though it is encrypted. (In UNIX, the entire emulated registry is a file that can be deleted.)
Since the data is maintained in the registry, there is no protection against it being simultaneously updated by multiple threads or even multiple processes, unless the registry is designed to prevent this and will not be corrupted (last in overlays what is there).

The specified global variable name may only consist of alphanumeric (ASCII) characters and the following characters: the underscore (_), minus sign (-), pound sign (#), and at sign (@). Spaces are not allowed. This is done to allow retrieval of the variable from different locales. (These restrictions are even enforced in Unicode-based exits that do not use locales.)

The contents of the variable, however, are up to you, but you should make sure the variable can be used between locales if Optim runs under different locales (for instance, if the remote action process runs under the same locale as the client, we translate between platforms, so it depends on the locale of the clients that connect to the server).

If the name is invalid and you requested that all errors be returned, OPMEX_ERR_INVALID is returned; otherwise, an exception occurs.

**Function: OPMEX_GETLOBAL, get variable's value**

This topic describes the callback function OPMEX_GETLOBAL (get variable's value).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK, OPMEX_ERR_NOT_FOUND, OPMEX_ERR_TOOLONG, or length of variable (see below)</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Index of variable if array (relative to zero)</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Value of the variable</td>
</tr>
</tbody>
</table>

If the variable name is not found or the index is an array that is out of bounds, OPMEX_ERR_NOT_FOUND is returned, and Data is set to an empty string.

If the data area is too small to hold the value and you called the OPMEX_RETURN_ALL_ERRORS callback during the initial call, OPMEX_ERR_TOOLONG is returned. If you did not ask to receive fatal errors, an exception is raised.

To get the length of the variable without receiving its value, pass NULL as the Data variable. The length of the variable is then returned by the call in characters, without the trailing null end of the string. If the variable is empty (""), zero is returned. If the variable is not found, -1 is returned.

**Note:** Arrays for these variables are “sparse” (not all indexes may be present).

**Function: OPMEX_SETLOBAL, set variable's value**

This topic describes the callback function OPMEX_SETLOBAL (set variable's value).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
</tbody>
</table>
This function updates or adds a global variable. If an index is used, it need not be consecutive (indexed variables are stored sparse).

To delete a variable, pass NULL as the **Data** address; deleting a variable ignores the index and all elements of an array are deleted.

**Note:** A new line ("\n") must not exist within the global variable's value; if one does, it will be removed prior to saving it to the registry.

### Client-Server Exit-To-Exit Support

When running on the server side, there is no one to ask a question if you are missing information and want to prompt for that information, and if you need information unique to the client side, it would be too late to gather it.

Requesting the data on the client side will not help you if you need the server environment to determine what to request. You can use this client-server facility to format a call to your exit on the “other” side of the conversation (i.e., from the server it will run on the client, from the client it will run on whatever server is associated with the request).

You pass data to and from the client or server side using client-server variable support. The sender uses OPMEX_CS_SETVAR to set one or more client-server variables to a value, and then calls OPMEX_CS_CALL to call the other side. The other side can use OPMEX_CS_GETVAR to get these variables and update them or add new ones using OPMEX_CS_SETVAR.

On return from the OPMEX_CS_CALL, you can get the updated variables using the OPMEX_CS_GETVAR callback function.

The other side also gets all the “normal” variables (available using OPMEX_GET_VAR); these contain the value from the “other” side during the call.

### Function: OPMEX_CS_GETVAR, get client/server variable

This topic describes the callback function OPMEX_CS_GETVAR (get client/server variable).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK, OPMEX_ERR_NOT_FOUND, OPMEX_ERR_TOOLONG, or length of variable</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Index of variable if array (relative to zero)</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Value of the variable</td>
</tr>
</tbody>
</table>

If the variable name is not found or the index is an array that is out of bounds, OPMEX_ERR_NOT_FOUND is returned, and **Data** is set to an empty string.
If the data area is too small to hold the value and you called the OPMEX_RETURN_ALL_ERRORS callback during the initial call, OPMEX_ERR_TOOLONG is returned. If you did not ask to receive fatal errors, an exception is raised.

To get the length of the variable without receiving its value, pass NULL as the `Data` variable. The length of the variable is then returned by the call in characters, without the trailing null end of the string. If the variable is empty (""), zero is returned. If the variable is not found, -1 is returned.

**Note:** Arrays for these variables are “sparse” (not all indexes may be present).

**Function: OPMEX_CS_SETVAR, set variable's value**

This topic describes the callback function OPMEX_CS_SETVAR (set variable's value).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>PMEX_OK, OPMEX_ERR_NOT_FOUND (delete)</td>
</tr>
<tr>
<td>Info</td>
<td>Variable Name</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Index of variable if array (relative to zero)</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>New value for variable</td>
</tr>
</tbody>
</table>

This function is used to update or add a client-server variable.

To delete a variable, pass NULL as the `Data` address. Unlike global variables, only the indexed element is deleted.

**Function: OPMEX_CS_CLEAR, delete all variables**

This topic describes the callback function OPMEX_CS_CLEAR (delete all variables).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Not Used</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Not Used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

If you are sending multiple requests to the other side's exit, any variables you defined will be passed to and sent back by Optim.

If you do not delete the variables (using OPMEX_CS_SETVAR with `Data` set to NULL), the variables will be resent with each request, even though they may not be used.

This callback will delete all client-server variables.

**Function: OPMEX_CS_CALL, call the other side**

This topic describes the callback function OPMEX_CS_CALL (call the other side).

The parameters and possible values for this function are listed below.
This function calls your exit on the other side of the client-server call; the other side will receive an OPMEX_EXIT_CS_CALL.

If an “other side” does not exist, OPMEX_ERR_NOT_FOUND is returned; otherwise, the value returned from the OPMEX_EXIT_CS_CALL exit point on the other side is returned.

### Trace Support

All Optim programs create a trace file in the TEMP directory (as defined in the Product Options, Server Applet, pstserv, or pstlocal configuration files in UNIX).

You can add text to the trace file from your exit, if needed.

**Function: OPMEX_TRACE, place text into the trace file**

This topic describes the callback function OPMEX_TRACE (place text into the trace file).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_ERR_NOT_FOUND, OPMEX_RC_*</td>
</tr>
<tr>
<td>Info</td>
<td>Not Used</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not Used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

This function writes the text in **Info** to the trace file. **Info** may contain a new line (“\n”) to create a multi-line trace entry in the file.

There is an internal limit to the size of a trace entry, but the callback logic will truncate the string to fit the available space, so an exception is not raised.

### System Log Support

In Windows, the Event Log is used for this purpose. In UNIX, the “syslog” API is used to write this data to long-term logs.

If this is called from a request running on an Optim server and you turned on “Admin Email Notification,” an email will be sent to those indicated in the notification settings, based on the severity of the issue.
UNIX Only:

In UNIX, the platform determines where the system log files reside.

The openlog() API is used to prepare the output for the system log; the log's prefix will be the same as the program's name in lowercase letters (i.e., pr0svce, pr0cmnd, etc.). The prefix can be used to determine where (or even if) log entries are written to disk.

If you cannot find the syslog (message) file, notify your system administrator because it may be turned off or only recording certain prefixes.

Function: OPMEX_LOG, place text into the system log file

This topic describes the callback function OPMEX_LOG (place text into the system log file).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK</td>
</tr>
<tr>
<td>Info</td>
<td>Severity</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not Used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Text to put in the log</td>
</tr>
</tbody>
</table>

Info contains the severity code and can be one of the following enumerated values defined in OPMEXIT.H:

<table>
<thead>
<tr>
<th>OPMEXIT.H define</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPMEX_LOG_INFO</td>
<td>Information</td>
</tr>
<tr>
<td>OPMEX_LOG_WARN</td>
<td>Warning</td>
</tr>
<tr>
<td>OPMEX_LOG_ERROR</td>
<td>Error</td>
</tr>
<tr>
<td>OPMEX_LOG_CRITICAL</td>
<td>Critical</td>
</tr>
</tbody>
</table>

If none of these values is appropriate and you indicated you wanted all errors to be returned to the exit, OPMEX_ERR_INVALID is returned; otherwise, an exception is raised.

Data contains the text to be put in the log file.

Message Support

The status of all Optim actions (Extract, Archive, etc.) is recorded in a request file that you can display in several ways.

You can display the status of all Optim actions using one of the following methods:

- the File menu of the request's editor
- the /output parameter on the command line
- the request dialog of the scheduling monitor.

If the exit is called outside of an action where there is no request output, the output will appear in the Error Message Bar at the bottom of the request.
Finally, if all of the above are unavailable and an operator is likely present, a pop-up message is
displayed. (In UNIX, the message is written to the console.) If an operator is not likely present, only a
trace file entry is written.

**Function: OPMEX_MSG, write a message to the operator**

This topic describes the callback function OPMEX_MSG (write a message to the operator).

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK or OPMEX_ERR_NOT_ALLOWED</td>
</tr>
<tr>
<td>Info</td>
<td>Text of the message</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>OPMEX_TRUE to pop up a message box, if possible</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>Not Used</td>
</tr>
</tbody>
</table>

This function adds a message to the output of the request. Note that there may be messages prior to this
message, as well as others after it. Exit messages are also traced.

If **InfoIdx** is OPMEX_TRUE (or any non-zero value) and an operator is likely present, the message is
displayed in a message box and added to the request’s output.

**OPMEX_ERR_NOT_ALLOWED** is returned if:

- the message could not be written because there are no results to write to
- there is no error message bar, and
- an operator is likely not present.

The trace entry, however, will still exist.
Chapter 7. Overriding Credentials

This chapter discusses overriding credentials, including server credentials and ODBC credentials.

Server Credentials

When an Optim server can execute an Action Request, the credentials are taken from Personal Options Server tab in Windows, or the pstlocal.cfg “server” parameter in UNIX.

The credentials are used only if:
- the server being contacted is running under the pstserv.cfg parameter
- “filelogon” is set to “client” in UNIX, or
- “File Input/Output” is set to “Client” on the Server Applet’s Security tab in Windows.

If the above are set to any other values, the credentials are ignored.

Based on these server settings, the Request Process (program PR0SVER) must run under a login with the specified credentials for three reasons:
- All file permissions are based on the user who executed the process that is opening the file. Therefore, to properly use system file permissions, the user must log on and then execute the entire request under his or her user id.
- On some DBMS that use OS Authentication, the user’s identifier is used to connect to the database.
- Some DBMS functions use the current user's credentials to confirm that the function is allowed.

This is required because the API used to open/create a file does not allow you to override the user; this is true on all platforms. Therefore, the system API is used to log on the user and run the request.

You can use the exit to override the credentials specified in Windows or UNIX (i.e., via the Personal Options Server tab in Windows or the pstlocal.cfg file “server” parameter in UNIX). You can do this at any time because the credentials are maintained in internal tables, but the credentials must be reset by the exit on each invocation because the overrides are not maintained in a persistent storage.

Function: OPMEX_SET_SERVER_CREDENTIALS, override server credentials

This topic describes the function OPMEX_SET_SERVER_CREDENTIALS, which is used to override server credentials.

The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK,</td>
</tr>
<tr>
<td></td>
<td>OPMEX_ERR_NOT_FOUND,</td>
</tr>
<tr>
<td></td>
<td>OPMEX_ERR_NOT_ALLOWED</td>
</tr>
<tr>
<td>Info</td>
<td>Server name to be overridden</td>
</tr>
<tr>
<td>Infoldx</td>
<td>“Ask If Invalid” flag</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>New credentials as USERID, PASSWORD [DOMAIN]</td>
</tr>
</tbody>
</table>
After the specified action is started on the server side, it is too late to change the credentials. In UNIX, the credentials must be changed under the “super user’s” credentials, but the underlying code will not allow this on-the-fly. In Windows, the proper user hive must be registered, before executing the process.

Therefore, this function must be called on the client side. If it is called on the server side, OPMEX_ERR_NOT_ALLOWED is returned and ignored.

**Note:** The credentials supplied here, as well as on Personal Options Server tab and in the pstlocal configuration “server” entries, must be for an actual user on the remote system because those credentials will be used to log on to the remote system using whatever platform API is available. If the login fails, so will the action.

### Info parameter

**Info** is the server name you want to modify:

- You can specify a specific server name, as indicated in the Product Options Server tab in Windows, or the SERVER parameter in the pstlocal.cfg config file in UNIX. This is not the name of the computer; it is the Optim Server Name.
  
  Optim server names are defined in the Servers Applet General tab in Windows, and the “rtservername” parameter of the pstserv.cfg file in UNIX.

- If **Info** is omitted (NULL), empty (“”), or all spaces, the credentials supplied represent the “(Default)” entry.
  
  While UNIX does not include the concept of (Default) credentials (as Personal Options does in Windows), this override exists on both sides. In UNIX, omit the pstlocal.cfg's server parameter’s user id, password, and domain to use these defaults.

- You can specify an * (a single asterisk) to indicate that all servers should use this entry. This entry will override any server credentials or (Default) credentials specified in Windows or UNIX (i.e., via Personal Options in Windows or the pstlocal.cfg file’s “server” parameters in UNIX).

  If you specify an asterisk, the credentials specified in Windows or UNIX are effectively ignored (i.e., the credentials specified via the Personal Options Server tab in Windows, or the pstlocal.cfg file’s “server” parameters in UNIX). However, any override you specified for a specific server via this callback function will be used.

  If you specify an asterisk, a (Default) entry will not be used, even if one is specified via the Personal Options Server tab or this callback function.

  OPMEX_ERR_NOT_FOUND is returned if the server is not listed in the Product Options Server tab or the pstlocal.cfg “server” parameters listing.

### InfoIdx parameter

**InfoIdx** can contain either OPMEX_TRUE or OPMEX_FALSE (or zero or non-zero).

If OPMEX_TRUE (non-zero) is returned and the credentials given in **Data** are not valid on the server side (after the server is contacted), a pop-up will appear on the client to ask for the appropriate credentials, assuming the client is Windows. (In UNIX, the connection is always refused.)

This is the action that occurs without overrides.

If OPMEX_FALSE (zero) is returned, it means the connection should be refused, so a pop-up will not be displayed to ask for new credentials.

This option allows you full control over what credentials are used on a server connection.
**Data parameter**

Data is of the following format:

To secure the credentials in the Data parameter from hackers, the password is immediately encrypted and the entire Data buffer is cleared to zeros, so do not expect Data to contain anything after the call.

Data must contain USERID, PASSWORD [,DOMAIN] (the commas are mandatory).

These credentials are not validated when you set them. If you enter them interactively, use OPMEX_IS_USER_VALID or some other method to validate them.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>The user to be used to login on the server.</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>The password for the specified user id.</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>In a UNIX environment that uses PAM to validate users, the specified domain is used with the PAM_RHOST attribute to access PAM on a remote host. (If a domain is not specified, the Optim “rt4s” default service id is used.)</td>
</tr>
<tr>
<td></td>
<td>In a UNIX environment that does not use PAM, this setting is ignored (e.g., LAM in AIX).</td>
</tr>
<tr>
<td></td>
<td>In a Windows environment, this setting identifies the domain used to validate the user. If a computer name is specified, Windows will usually fail, unless the computer is the local computer. (The current computer is usually used when a domain is not specified in Windows).</td>
</tr>
</tbody>
</table>

**UNIX Only:**

UNIX does not include a “LogonUser()” API call. However, you can change various system process attributes if you have “super user” (root) authority. UNIX leaves it up to the individual programs to validate user ids and passwords.

To do this with Optim, you must have Enterprise Mode installed and you must execute the daemon (PR0SVCE) under root or change the pr0ulog program to run as root (often called “setuid”). In AIX using LAM, you must execute PR0SVCE as root.

PAM is used to validate that the specified credentials are for a valid user; this is done using the default “rt4s” PAM user id against the PAM_RHOST depicted by the domain. If PAM is not installed in AIX, LAM is used, instead.

Once validated, the user and group ids are changed (real and effected), as well as other system settings. The PR0SVER Request Process program is then executed under the new user.

If you use PAM, you can determine which technique will be used to validate a user. For example, if you have a PAM service identifier of “ldap” that references the PAM Service Modules that use OpenLDAP to validate users, LDAP will be used to validate the user.

However, on most systems, the local /etc/passwd and /etc/group files must exist and must contain the proper list of users and groups in which they reside (although some platforms may allow these to be modified on-the-fly).

If DOMAIN is omitted, you can also omit the leading comma, but this is not required.

If the USERID, PASSWORD, or DOMAIN contains spaces, enclose the argument in double quotes (“”). If the argument contains double quotes, enclose the string in single quotes (‘’). If the argument contains both double and single quotes, enclose the string in single quotes (’) and change the imbedded single quotes to double quotes (i.e., from ‘’ to “”).
**Note:** In Windows, if the Personal Option Server tab entry for the user or for (Default) drop down indicates “[X] Always Ask For Password,” it is ignored once you override the server using this function. You must allow or disallow new credentials using the `InfoIdx` parameter of this call.

Data may be NULL (or empty or space), in which case the specified server override, (Default) override, or global override is deleted and reverts back to normal processing.

**Examples:**

These examples assume the server requires credentials to run a request; if not, the credentials are ignored by the server.

As an example, assume the following are set via the Product Options Serve tab in Windows or the `pstlocal.cfg` “server” parameter in UNIX.

<table>
<thead>
<tr>
<th>Server</th>
<th>Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>UserIdA, APassword, ADomain (not in UNIX)</td>
</tr>
<tr>
<td>SERVER1</td>
<td>UserIdX, XPassword, XDomain</td>
</tr>
<tr>
<td>SERVER2</td>
<td>UserIdY, YPassword, YDomain</td>
</tr>
<tr>
<td>SERVER3</td>
<td>Not set</td>
</tr>
</tbody>
</table>

If you run requests on these servers, the following credentials would be used:

<table>
<thead>
<tr>
<th>Server</th>
<th>Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>UserIdA, APassword, ADomain</td>
</tr>
<tr>
<td>SERVER1</td>
<td>UserIdX, XPassword, XDomain</td>
</tr>
<tr>
<td>SERVER2</td>
<td>UserIdY, YPassword, YDomain</td>
</tr>
<tr>
<td>SERVER3</td>
<td>UserIdA, XPassword, ADomain</td>
</tr>
</tbody>
</table>

[If called by Windows, the (Default) server entry is used. If called from a UNIX command line, empty credentials are sent to the server and the request will probably fail.]

Now, say you call `OPMEX_SET_CREDENTIALS` for SERVER1 and specify “USERABC, ABCPassword, ABCDomain” to override the credentials. When you call the servers, the following credential will be used:

<table>
<thead>
<tr>
<th>Server</th>
<th>Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>UserIdA, APassword, ADomain</td>
</tr>
<tr>
<td>SERVER1</td>
<td>UserABC, ABCPassword, ABCDomain</td>
</tr>
<tr>
<td></td>
<td>(because you overrode the credential for this server)</td>
</tr>
<tr>
<td>SERVER2</td>
<td>UserIdY, YPassword, YDomain</td>
</tr>
<tr>
<td>SERVER3</td>
<td>UserIdA, XPassword, ADomain</td>
</tr>
</tbody>
</table>

Now, say you call `OPMEX_SET_CREDENTIALS` with an empty server name (i.e., change the Default) for USERDEF, DEFPassword, and DEFDomain. When you run actions on these servers, the following credential will be used:

<table>
<thead>
<tr>
<th>Server</th>
<th>Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>USERDEF, DEFPassword, DEFDomain</td>
</tr>
</tbody>
</table>
Finally, say you call OPMEX_SET_CREDENTILAS with a server name of ‘*’ (an asterisk for global credentials) with the credentials USERME, MYPASSWORD, MYDOMAIN. When you call the servers, the following credential will be used:

<table>
<thead>
<tr>
<th>Server</th>
<th>Credentials</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Default)</td>
<td>USERDEF, DEFPassword, DEFDomain</td>
</tr>
<tr>
<td></td>
<td>(not used any more)</td>
</tr>
<tr>
<td>SERVER1</td>
<td>UserABC, ABCPassword, ABCDomain</td>
</tr>
<tr>
<td></td>
<td>(because you overrode the credential for this server)</td>
</tr>
<tr>
<td>SERVER2</td>
<td>USERME, MYPASSWORD, MYDOMAIN</td>
</tr>
<tr>
<td></td>
<td>(because you overrode anything that didn’t have a specific server override)</td>
</tr>
<tr>
<td>SERVER3</td>
<td>USERME, MYPASSWORD, MYDOMAIN</td>
</tr>
<tr>
<td></td>
<td>[even though this is supposed to use the (Default), it will not use the default because you overrode anything that didn’t have a specific server override]</td>
</tr>
</tbody>
</table>

You can override the credentials at any time, since they are maintained in the program for the duration of the program; however, you should note the following:

- When you receive a client-side exit call, the client side cannot know whether the server side requires credentials, because there has not been any contact with the server yet. It is not possible, therefore, to know the server’s “FileLogon” or “File Input/Output” settings, so it is not possible to know whether credentials are required.
  Therefore, if you blindly ask for credentials on the client side, you may not need them, which may cause confusion for your operators.
- On the server side, by the time you can check the credentials, the Optim process will have already logged on using the credentials listed in the Personal Options Sever tab or the pslocal.cfg “server” parameter, or the credentials specified via this callback on the client side. You cannot simply validate a new user id at that time, and then change it; it’s too late.

Currently, you cannot get around the above scenario without knowing whether the server being contacted requires logon credentials.

**ODBC Credentials**

Whether you use ODM or the Optim ODBC server (PR0COMS), all connections to an Archive/Extract using ODBC require user credentials to allow them to use ODM or ODBC, although you can specify that none are to be used.

Normally, these credentials are retrieved from Attunity Studio or the connection parameters. You may override these credentials in the exit.

**Function: OPMEX_SET_ODBC_CREDENTIALS, override ODBC credentials**

This topic describes the function OPMEX_SET_ODBC_CREDENTIALS, which is used to override ODBC credentials.
The parameters and possible values for this function are listed below.

<table>
<thead>
<tr>
<th>Callback Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_OK, OPMEX_ERR_NOT_FOUND, OPMEX_ERR_NOT_ALLOWED</td>
</tr>
<tr>
<td>Info</td>
<td>Full path to Archive File</td>
</tr>
<tr>
<td>Infoldx</td>
<td>Not used</td>
</tr>
<tr>
<td>Data/DataSizeChars</td>
<td>New credentials as USERID, PASSWORD [,DOMAIN]</td>
</tr>
</tbody>
</table>

**Info** is the archive file to override and may be one of the following:
- An Archive Collection specified as <PSTDIR>:ID.NAME
  
  The Optim Directory, colon, and the Archive Collection identifier and name, as it resides on that Optim Directory.

- A full file system path to the archive file in the form of <SERVER>:<path>.
  
  That is, a server that the file is to be referenced, a colon, and the full path to the archive file (since all Archive requests must run on the server that created the Archive and since the file is relative only to the server computer).

  There is no default directory. If you leave the directory part of the archive file blank, it will not assume the Archive Directory as specified in the Applet/pstserv.cfg files.

  For a (Local) archive, omit the server name (you can also omit the colon, but this is not required).

In any case, the archive file(s) must exist in the Archive Directory of the Optim Directory.

If you are doing this within an OPMEX_EXIT_ODBC_CONNECT or OPMEX_EXIT_ODBC_DISCONNECT exit point, you can leave **Info** empty (or NULL), and it will use the identifier of the file being connected/disconnected.

**Ask Operator for Credentials**

As indicated earlier, every call into an exit must either return or call the callback within a specified timeout period (60 seconds is the default value, if not overridden). If the exit does not do one of these within the timeout period, Optim will assume a hang or runaway condition has occurred and will terminate the exit.

As noted earlier, you can use the OPMEX_SET_TIMEOUT callback to change the timeout period from 60 seconds to as much as 10 minutes, but even 10 minutes may not be sufficient time to respond to a dialog if the operator is not present when the dialog is sent. Therefore, this function allows you to get these credentials using a built-in dialog in Windows or a series of prompts in UNIX.

In UNIX, where the console is used to get the credentials, you can use the following to edit the input as you enter them (one after another):
- Backspace to erase the last character
- ESC or CTRL-C to cancel the input, regardless of where you are in the entry

**Note:** Each entry type has a max length; if you exceed that length, a console “bell” will sound.

**Function: OPMEX_ASK_CREDENTIALS, get credentials from operator**

This topic describes the function OPMEX_ASK_CREDENTIALS, which is used to get credentials from an operator.

The parameters and possible values for this function are listed below.
### Callback Parameter Contents

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>(return)</td>
<td>OPMEX_TRUE or OPMEX_FALSE</td>
</tr>
<tr>
<td>Info</td>
<td>Message to prompt the operator</td>
</tr>
<tr>
<td>InfoIdx</td>
<td>Format to be used</td>
</tr>
<tr>
<td>Data/SizeChars</td>
<td>Input/Output USERID, PASSWORD [,DOMAIN]</td>
</tr>
</tbody>
</table>

**Info** is the message that appears in the Credential Dialog's Error Message Bar in Windows or is printed prior to asking for the credentials in UNIX.

**InfoIdx** can be OPMEX_ASK_NODOMAIN or OPMEX_ASK_DOMAIN, depending on whether you want a domain returned, entered or not.

If you “OR” **InfoIdx** with OPMEX_ASK_DISPLAY, any credentials entered in **Data** will be used to prime the display (in Windows, the edit fields are primed; in UNIX, each line appears with the data primed and pressing Enter will result in the data being data).

**Data** will contain the following credentials on output, and may also contain this information on input:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>USERID</td>
<td>User Id</td>
</tr>
<tr>
<td>PASSWORD</td>
<td>Password (a series of asterisks will be displayed in place of the actual password)</td>
</tr>
<tr>
<td>DOMAIN</td>
<td>The Domain to use.</td>
</tr>
</tbody>
</table>

The credentials are not validated because the callback does not know what the credential will be used for.
Chapter 8. Variables by Exit Point

This chapter discusses variables by exit point.

All exit points

This topic lists all exit points and includes a brief description of each exit point.

**OPMEX_VAR_DESC**
A description of the program.

**OPMEX_VAR_FUNCTION**
A description of the exit point.

**OPMEX_VAR_USER**
The currently logged on user.

**OPMEX_VAR_COMPUTER**
The computer's name.

**OPMEX_VAR_USER_DOMAIN**
The domain of the logged on user.

**OPMEX_VAR_COMPUTER_DOMAIN**
The domain of the computer.

**OPMEX_VAR_PSTDIR**
The default Optim Directory.

**OPMEX_VAR_RELEASE**
The Optim release.

**OPMEX_VAR_BUILD**
The Optim build.

**OPMEX_VAR_SERVER**
The server, if running in server.

**OPMEX_VAR_TEMPDIR**
The Temporary Directory.

**OPMEX_VAR_DATADIR**
The Data Directory.

**OPMEX_VAR_ARCHDIR**
The Archive Directory.

**OPMEX_VAR_ARCHIDXDIR**
The Archive Index Directory.

**OPMEX_VAR_ARCHBROIDXDIR**
The Archive Browse Index Directory.

Exit Point **OPMEX_EXIT_PROGRAM_START - PROTOOL**

The reason PROTOOL was executed appears in the following variable.

**OPMEX_ARG_ACTION**
Why started

The **ACTION** can be one of the following:
Exit Point OPMEX_EXIT_PROGRAM_START - PR0CMND

The reason PR0CMND was executed appears in the following variable.

OPMEX_ARG_ACTION
Why started

The ACTION can be one of the following:

OPMEX_ACTION_CMND_RUN
/RUN
OPMEX_ACTION_CMND_RESTART
/RESTART or /RR
OPMEX_ACTION_CMND_ARCHMAINT
/ARCHMAINT or AM
OPMEX_ACTION_CMND_CREATE
/CREATE or /C
OPMEX_ACTION_CMND_EDITOR
/E
OPMEX_ACTION_CMND_XF
/X
OPMEX_ACTION_CMND_IMPORT
/IMPORT or /I
OPMEX_ACTION_CMND_EXPORT
/EXPORT or /EX
OPMEX_ACTION_CMD_MIGRATE
/MIGRATE or /M

Variables for Command Line /RUN Requests

The variables for Command line /RUN requests are:

OPMEX_ARG_QUIET
/QUIET
OPMEX_ARG_MONITOR
/MONITOR
OPMEX_ARG_SERVER
/SERVER=
OPMEX_ARG_OUTPUT
/OUTPUT=

The command line /RUN can have an @parameter file with multiple requests in it. The number of requests (command line and/or @parameter file) is maintained in the following variable.
Variables for Command Line /RR Requests

The variables for Command line /RR (RestartRetry) requests are:

- **OPMEXARGQUIET**
  
- **OPMEXARGMONITOR**
  
- **OPMEXARGSERVER**
  
- **OPMEXARGOUTPUT**
  
- **OPMEXARGPSTDIR**
  
- **OPMEXARGCOMMIT**
  
- **OPMEXARGDISCARD**
  
- **OPMEXARGCONTROL**

Variables for Command Line /AM Requests

The variables for Command line /AM (ArchiveMaint) requests are:

- **OPMEXARGDELEFILE**
  
- **OPMEXARGOVERWRITE**
  
- **OPMEXARGREPLACEPRI**
OPMEX_ARG_REMOVEDUP
   /REMOVEDUPLICATE

OPMEX_ARG_SWAP
   /SWAP

OPMEX_ARG_AUTOFIX
   /AUTOFIX

OPMEX_ARG_PSTDIR
   /PSTDIR=

OPMEX_ARG_OPER
   /OPERATION=

OPMEX_ARG_AFFILE
   /AFFILE=

OPMEX_ARG_AFXFILE
   /AFXFILE=

OPMEX_ARG_DUPLICATE
   /DUPLICATE=

OPMEX_ARG_SERVER
   /SERVER=

OPMEX_ARG_OUTPUT
   /OUTPUT=

OPMEX_ARG_GUID
   /GUID=

OPMEX_ARG_AFID
   /AFID=

OPMEX_ARG_GROUP
   /GROUP=

OPMEX_ARG_DESC
   /DESC=

OPMEX_ARG_STGPROF
   /PROFILE=

Variables for Command Line /CREATE Requests

The variables for Command line /CREATE requests are:

OPMEX_ARG_AFFILE
   /AFFILE=

OPMEX_ARG_XFFILE
   /XFFILE=

OPMEX_ARG_DESTQUAL
   /DESTQUAL=

OPMEX_ARG_OUTPUT
   /OUTPUT=

Variables for Command Line /E Requests

The variables for Command line /E (Table Editor) requests are:
Variables for Command Line /XF Requests

The variables for Command line /XF (Display Extract/Archive File) requests are:

- **OPMEX_ARG.FILE**
  Archive/Extract file to browse

Variables for Command Line /IMPORT Requests

The variables for Command line /IMPORT requests are:

- **OPMEX_ARG.OVERRIDE**
  /OV=

The /IMPORT command line option can specify one or more DBALIAS to override in the import. The following variable contains the number of these:

- **OPMEX_ARG.DBALIAS_CNT**
  Number of OPMEX_ARG.DBALIAS entries present

- **OPMEX_ARG.DBALIAS**
  DB Alias Override
Variables for Command Line /EXPORT Requests

The variables for Command line /EXPORT requests are:

- **OPMEX_ARG_SUBORD**
  `/SUBORDINATES`

- **OPMEX_ARG_CONTONERR**
  `/CONTINUEONERR`

- **OPMEX_ARG_TYPE**
  `/TYPE=`

- **OPMEX_ARG_FILE**
  `/FILE=`

- **OPMEX_ARG_OUTPUT**
  `/OUTPUT=`

Variables for Command Line /MIGRATE Requests

The variables for Command line /MIGRATE requests are:

- **OPMEX_ARG_OVERWRITE**
  `/OVERWRITE`

- **OPMEX_ARG_REPDIR**
  `/REPLACEDIR`

- **OPMEX_ARG_ALLOWORPHAN**
  `/ALLOWORPHAN`

- **OPMEX_ARG_COMPRESS**
  `/COMPRESS`

- **OPMEX_ARG_RETAINGUID**
  `/RETAINGUID`

- **OPMEX_ARG_REGISTER**
  `/REGISTER`

- **OPMEX_ARG_COPYCRIT**
  `/COPYCRIT`

- **OPMEX_ARG_PRIVACY**
  `/PRIVACY`

- **OPMEX_ARG_PSTDIR**
  `/PSTDIR=`

- **OPMEX_ARG_IAF**
  `/IAF=`

- **OPMEX_ARG_OAF**
  `/OAF=`

- **OPMEX_ARG_OUTPUT**
  `/OUTPUT=`

- **OPMEX_ARG_SERVER**
  `/SERVER=`

- **OPMEX_ARG_FAD**
  `/FAD=`
Exit Point OPMEX_EXIT_PROGRAM_START - PR0SVER

This topic lists the variables for exit point OPMEX_EXIT_PROGRAM_START - PR0SVER.

OPMEX_ACT_SERVER
The server's Optim name.

OPMEX_ACT_PSTDIR
The Optim Directory in which the request resides.

OPMEX_ACT_COMPID
Optim Component id (X* server component, see OPMEX_COMP_* defines).

OPMEX_ACT_REQID
Request's ID.NAME.

OPMEX_ACT_USERID
User id from Personal Options, pstlocal.cfg, or an override.

OPMEX_ACT_DOMAIN
Domain from Personal Options, pstlocal.cfg, or an override.

OPMEX_ACT_SILENT
Is client running silent mode.

OPMEX_ACT_PARENT
Parent component running server component (see OPMEX_COMP_* defines).

OPMEX_ACT_EXE
Executable on client side (PR0xxx file name only).

OPMEX_ACT_USER
Logged on user id on client side.

OPMEX_ACT_COMPUTER
Client's computer name.

OPMEX_ACT_PLATFORM
Client's platform (see OPMEX_PLATFORM_* defines).

OPMEX_ACT_OPSYS
Client's operating system (see OPMEX_OPSYS_* defines).

OPMEX_ACT_SVCE
Running under SVCE (Server), “T” or “F”.
**Exit Point OPMEX_EXIT_PROGRAM_EXIT**

This topic lists the variables for exit point OPMEX_EXIT_PROGRAM_EXIT.

**OPMEX_VAR_EXITCODE**
Exit code returned to the operating system (ERRORLEVEL in Windows; $? in UNIX).

**Exit Point OPMEX_EXIT_ACTION_CALL**

This topic lists the variables for exit point OPMEX_EXIT_ACTION_CALL.

**OPMEX_ACT_SERVER**
Server that will run the request or (Local, OPMEX_LOCAL).

**OPMEX_ACT_PSTDIR**
The Optim Directory in which the request resides.

**OPMEX_ACT_COMPID**
Optim Component id (X* server component, see OPMEX_COMP_* defines).

**OPMEX_ACT_REQID**
Request's ID.NAME.

**OPMEX_ACT_USERID**
User id from Personal Options, pstlocal.cfg, or from an override.

**OPMEX_ACT_DOMAIN**
Domain from Personal Options, pstlocal.cfg, or from an override.

**OPMEX_ACT_DEFAULT**
User and domain are set from defaults.

**OPMEX_ACT_SILENT**
Will server run in silent mode “T”/“F”.

**OPMEX_ACT_PARENT**
Parent component running server component (see OPMEX_COMP_* defines).

**OPMEX_ACT_NETADDR**
Network address of server.

**OPMEX_ACT_ENDPT**
Endpoint on server for connection.

**OPMEX_ACT_PROTOCOL**
Protocol used to contact server.

**OPMEX_ACT_SVCE**
Running under SVCE (Server). (Currently always “F” on this exit point).

**Exit Point OPMEX_EXIT_ACTION_START**

This topic lists the variables for exit point OPMEX_EXIT_ACTION_START.

**OPMEX_ACT_SERVER**
This server's Optim name or (Local, OPMEX_LOCAL).

**OPMEX_ACT_PSTDIR**
The Optim Directory in which the request resides.

**OPMEX_ACT_COMPID**
Optim Component id (X* server component, see OPMEX_COMP_* defines).

**OPMEX_ACT_REQID**
Request's ID.NAME.
OPMEX_ACT_USERID
   User id from Personal Options, pstlocal.cfg, or from an override.

OPMEX_ACT_DOMAIN
   Domain from Personal Options, pstlocal.cfg, or from an override.

OPMEX_ACT_SILENT
   Is client running in silent mode.

OPMEX_ACT_PARENT
   Parent component running server component.

OPMEX_ACT_EXE
   Executable on client side (PR0xxxx file name only).

OPMEX_ACT_USER
   Logged on user id on client side.

OPMEX_ACT_COMPUTER
   Client's computer name.

OPMEX_ACT_PLATFORM
   Client's platform (see OPMEX_PLATFORM_* defines).

OPMEX_ACT_OPSYS
   Client's operating system (see OPMEX_OPSYS_* defines).

OPMEX_ACT_SVCE
   Running under SVCE (Server).

---

**Exit Point OPMEX_EXIT_CS_CALL**

This topic lists the variables for exit point OPMEX_EXIT_CS_CALL.

OPMEX_REM_COMPID
   Component being executed.

OPMEX_REM_REQID
   Request's ID.NAME.

OPMEX_REM_PSTDIR
   The Optim Directory in which the request resides.

OPMEX_REM_SILENT
   Is other side w/o operator?

OPMEX_REM_EXE
   Executable from other side.

OPMEX_REM_USER
   User on other side.

OPMEX_REM_WHERE
   See OPMEX_WHERE_* defines.

OPMEX_REM_COMPUTER
   Computer on other side.

OPMEX_REM_SERVER
   Server on other side.

OPMEX_REM_PARENT
   Parent component id.

OPMEX_REM_PLATFORM
   Platform on other side (see OPMEX_PLATFORM_* defines).
OPMEX_REM_OPSYS
Operating system on other side (see OPMEX_OPSYS_* defines).
Notices

This information was developed for products and services offered in the U.S.A.

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

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

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

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

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

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

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

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

COPYRIGHT LICENSE:

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

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.
If you are viewing this information softcopy, the photographs and color illustrations may not appear.

**Trademarks**

IBM, AIX, Optim, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

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

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

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

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

A
ASCII support 5
ask operator for credentials 38

C
Callback Functions
are credentials valid 23
are we being debugged 22
call the other side 29
delete all variables 29
get client/server variable 28
get contents of variable 25
get last system error 21
get max array index 25
get variable’s value 27
is operator present 23
is user in a system group 24
no operation 20
override default timeout 20
place text into system log file 31
place text into the trace file 30
return fatal errors 19
running in client or server 22
set variable’s value 26, 27, 29
single thread calls to the exit 19
what program is running 21
write a message to the operator 32

char/wchar_t *Data parameter 18
const char/wchar_t *Info parameter 18

D
default exit 1

E
exception logic 10
Exit Overview
after it’s compiled 5
ASCII support 5
default exit 1
life of the exit 6
protecting the exit 9
signing an exit 10
system security 9
UNICODE support 5
user exit languages 4
user-supplied exit prerequisites 3
user-supplied exits 1
writing a user exit 3

Exit Point 4

Exit Point (continued)
OPMEX_EXIT_ACTION_CALL 48
OPMEX_EXIT_ACTION_START - PR0CMND 42
OPMEX_EXIT_PROGRAM_EXIT 48
OPMEX_EXIT_PROGRAM_START - PR0CMND 42
OPMEX_EXIT_PROGRAM_START - PR0TOOL 41
OPMEX_ARG_AFID 44
OPMEX_ARG_AFXFILE 44
OPMEX_ARG_ALLOWORPHAN 46
OPMEX_ARG.AutoFIX 44
OPMEX_ARG.COMMIT 43
OPMEX_ARG.COMPRESS 46
OPMEX_ARG.CONTONERR 45, 46
OPMEX_ARG.CONTROL 43
OPMEX_ARG.COPIE 46
OPMEX_ARG.COPIE 46
OPMEX_ARG.DBALIAS 45, 47
OPMEX_ARG.DBALIAS_CNT 45
OPMEX_ARG.DELFILE 43
OPMEX_ARG.DESC 44, 47
OPMEX_ARG.DESTQUAL 44
OPMEX_ARG.DISCARD 43
OPMEX_ARG.DUPPLICATE 44
OPMEX_ARG.FAD 46
OPMEX_ARG.FILE 45, 46
OPMEX_ARG.FULL 45
OPMEX_ARG.GROUP 44, 47
OPMEX_ARG.GUID 44
OPMEX_ARG.IAF 46
OPMEX_ARG.INPLT 45
OPMEX_ARG.JOB 43
OPMEX_ARG.JOIN 45
OPMEX_ARG.MDESC 47
OPMEX_ARG.MONITOR 42, 43
OPMEX_ARG.NUMREQ 43
OPMEX_ARG.OAF 46
OPMEX_ARG.OPER 44
OPMEX_ARG.OUTPUT 42, 43, 44, 45, 46
OPMEX_ARG.OVERRIDE 43, 45
OPMEX_ARG.OVERWRITE 43, 45, 46
OPMEX_ARG.PRIVACY 46
OPMEX_ARG.PSIDIR 43, 44, 45, 46
OPMEX_ARG.QUIET 42, 43
OPMEX_ARG.SUPERUSER 42, 43
OPMEX_ARG.REMOVEDUP 44
OPMEX_ARG.REPDIR 46
OPMEX_ARG.REPLACEPRI 43
OPMEX_ARG.REQID 43, 45
OPMEX_ARG.RETAINGUID 46
OPMEX_ARG.SAVE 45
OPMEX_ARG.SERVER 42, 43, 44, 46
OPMEX_ARG.STGPROF 44, 47
OPMEX_ARG.STORE 43
OPMEX_ARG.SUBORD 46
OPMEX_ARG.SWAP 44
OPMEX_ARG.TABLE 45
OPMEX_ARG.TYPE 43, 46
OPMEX_ARG.XFILE 44
OPMEX_ARG.CREDENTIALS 20, 39
OPMEX_CALLER_ASP 21
OPMEX_CALLER_CMND 21
OPMEX_CALLER_CNFG 21
OPMEX_CALLER_MON 21
OPMEX_CALLER_SMON 21
OPMEX_CALLER_SVC 21
OPMEX_CALLER_SVS 21
OPMEX_CALLER_TOOL 21
OPMEX_CS_CALL 29
OPMEX_CS_CLEAR 29