# MetroPro™ Remote Access

**OMP-0476F**

zygo®

Zygo Corporation
Laurel Brook Road
P.O. Box 448
Middlefield, Connecticut 06455

Telephone: (860) 347-8506
E-mail: inquire@zygo.com
Website: www.zygo.com

## ZYGO CUSTOMER SUPPORT

For help within North America, please use the contacts listed below. For help in other countries, please contact your local ZYGO representative. Be sure to supply the instrument model and serial number, and the software version.

| | |
|---|---|
| ***PHONE*** | Monday - Friday, 8 a.m. - 8 p.m. (Eastern Standard Time)<br>(800) ZYGO NOW (800) 994-6669<br>*or* (860) 704-5191 |
| ***FAX*** | (860) 346-4188 |
| ***INTERNET*** | www.zygo.com<br>support@zygo.com |
| ***WRITE*** | Zygo Corporation<br>Laurel Brook Road<br>P.O. Box 448<br>Middlefield, CT 06455-0448<br>Attn: Customer Support |

## MANUAL REVISION INFORMATION

The document (OMP) number and the applicable revision letter for this manual appear on the title page. The publication date appears below.

| *Revision* | *Publication Date* | *Software Version* |
|---|---|---|
| A | February 2003 | 7.11.0 |
| B | July 2003 | 7.12.0 |
| C | August 2004 | 8.0.1 |
| D | May 2005 | 8.1.0 |
| E | September 2005 | 8.1.4 |
| F | October 2005 | 8.1.4 |

## MANUAL NOTATIONS

**Warning!**
Denotes a hazard that could cause injury to personnel, and can also cause damage to the equipment.

Note, provides helpful information.

**NOTICE:** The descriptions, drawings, and specifications contained herein are subject to change. ZYGO is not responsible for errors or omissions herein or for incidental damages in connection with the furnishing or use of this information. This document shall not be reproduced, photocopied, or duplicated, in whole or in part, without prior written approval of Zygo Corporation.

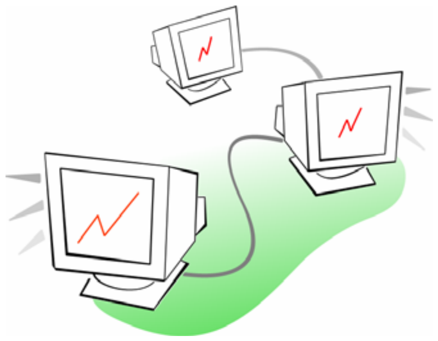
# CONTENTS

Chapter

1

# Introduction

## MetroPro Remote Access



This document explains the use of the *MetroProRemoteAccess.dll* (Dynamic Link Library). This library provides the ability for one computer to control and access other computers running MetroPro software. The computers must be connected through a local area network (LAN)[1]. The remote access dynamic link library functions are accessed with a user-written software program in C, C++, or Visual Basic.

For example, one computer can send MetroScript commands, such as Measure or Analyze, effectively taking control of MetroPro running on another computer. MetroScript commands as well as MetroScript files, pattern files, mask files, and data files can be exchanged. Results and outputs obtained on one computer can be viewed on a second computer.

The remote access function is primarily for the manufacturing environment where one computer can control up to 32 other computers and interferometers running MetroPro software.

## Assumptions

In this document, “Client” refers to the remote controlling computer or process. “Server” refers to MetroPro.

This manual makes the following assumptions:

- you are familiar with MetroPro
- you are familiar with MetroScript
- you have used the text editing programs included with the computer
- you have an understanding of programming concepts
- you have familiarity with C, C++, or Microsoft Visual Basic

[1] TCP/IP should be used as the primary network protocol in both server and client computers. The use of NetBEUI and NWLink as network protocols even though valid, is not recommended. Remote access for MetroPro has been tested extensively with TCP/IP only.

## Software and Hardware Requirements

### Remote Access Using a Local Area Network (Named Pipe Interface)

- Microsoft Windows NT/2000 operating system.
- MetroPro 7.11.0 or later.
- MetroProRemoteAccess.dll, mraClient.cfg, and mraServer.cfg files.
- Connection to a Local Area Network that supports Named Pipes.

## MetroPro Remote Access Files

All of the Remote access files (MetroProRemoteAccess.dll, mraClient.cfg, and mraSserver.cfg) are installed into the Metropro\cfg directory.

### MetroProRemoteAccess.dll

This file is a DLL that can be used by an application written in C, C++, or Visual Basic for controlling MetroPro remotely. The following functions are included in the DLL. See Chapter 3 for a detailed description of each of these functions.

- Connectivity Functions

```
int   mraInitialize ( char errBuf[BUFSIZ],
          char username[MAX_USERNAME_LENGTH],
          char password[MAX_PASSWORD_LENGTH],
          int server_gui_msgs,
          bool_t log_operations )

int   mraClose ( char errBuf[BUFSIZ] )

int   mraIsClientOnline ( )

int   mraGetServerStatus ( )

int   mraSetConnectinPolling ( int flag, int interval )
```

- Remote Control of Multiple Interferometers

```
int   mraSetCurrentServer ( char errBuf[BUFSIZ],
          int newServer )

int   mraGetCurrentServer ( )
```

- Control of MetroPro Functions

```
int   mraSendMetroScriptCommand ( char * metroScriptCommand,
          char errBuf[BUFSIZ] )
int   mraLoadScript ( char scriptFileName[BUFSIZ],
          char errBuf[BUFSIZ] )
int   mraRunScript ( char scriptFileName[BUFSIZ],
          char errBuf[BUFSIZ] )
int   mraIsScriptDone ( char errBuf[BUFSIZ] )
int   mraWaitScriptDone ( char errBuf[BUFSIZ] )
int   mraGetScriptResults ( char mpErrMsg[BUFSIZ],
          char mpOutput[BUFSIZ],
          char errBuf[BUFSIZ] )
```

int mraGetScriptStatus ( *bool poll,*
*char errBuf[BUFSIZ]* )

int mraKillMetroProServer ( *char errBuf[BUFSIZ]* )

int mraStopMetroProProcessing ( *char errBuf[BUFSIZ]* )

int mraTurnOnOffGuiMessages ( *char errBuf[BUFSIZ],*
*bool_t messages* )

int mraSuspendServer ( *char errBuff[BUFSIZ]* )

int mraSetSuspendPermissions ( *int permission,*
*char errBuf[BUFSIZ]* )

- Data Transfer Functions

int mraPutFile ( *char source_fn[BUFSIZ],*
*char destination_fn[BUFSIZ],*
*char errBuf[BUFSIZ]* )

int mraGetFile ( *char source_fn[BUFSIZ],*
*char destination_fn[BUFSIZ],*
*char errBuf[BUFSIZ]* )

- Diagnostics Functions

int mraGetClientAccessInterface ( )

int mraRunLoopBackTest ( *char errBuf[BUFSIZ]* )

int mraFlushPipe ( *char errBuf[BUFSIZ*] )

int mraTurnClientLogOnOff ( *bool_t log* )

int mraSetLoggingParameters ( *bool_t setPath,*
*char path[BUFSIZ],*
*bool_t overwriteLogFile,*
*bool_t createNewLogFile,*
*bool_t enableLogging*,
*char errBuf[BUFSIZ]* )

void mraClearLastError ( )

int mraGetLastError ( )

char mraGetAbortFlag ( )

char mraGetLocalAbortFlag ( )

char mraGetRemoteAbortFlag ( )

HANDLE mraGetClientHandle ( )

HWND mraGetClientWindow ( )

double mraGetTotalBytesReceived ( )

double mraGetTotalBytesSent ( )

- Abort Functions

int mraEnableAbortCheck ( )

int mraDisableAbortCheck ( )

int mraAbortCheck ( *char errBuf[BUFSIZ]* )

int mraSetClientWindow ( *HWND clientProcessWindow* )

int mraSetAbortFlag ( *int value* )

## mraClient.cfg

This configuration file contains all of the settings needed for establishing a valid connection with MetroPro through a Local Area Network. This file is read by *MetroProRemoteAccess.dll*. The parameter settings in this must match the settings in *mraServer.cfg* except for the Connection Timeout value.

## mraServer.cfg

This file contains all of the settings needed for establishing a valid connection with MetroPro through a Local Area Network. This file is read by MetroPro. The parameter settings in this must match the settings in *mraClient.cfg* except for the Connection Timeout value.

# Getting Started

Chapter 2

## Overview

To put MetroPro into Remote Access Mode, create a **Remote Access** button with the New Button → Miscellaneous → Remote Access menu.  When this button is pressed, MetroPro becomes a Named Pipe Server based on the configuration parameters established in *mraServer.cfg*.  The **Remote Access** button remains depressed until the connection closes or an abort occurs.

The *mraClient.cfg* file contains the settings for Remote Access for the Client side.  After the access interface has been determined, the Server will wait for a Client to connect.  Once in Remote Access Mode, MetroPro cannot be used interactively (with the mouse or keyboard) unless terminated or suspended.  The only operations that can be done on the Server are Abort and Suspend.  Abort is accomplished by pressing <ESCAPE>.  Suspend is accomplished by pressing F11.  When the Remote Access is aborted on the server side, the server notifies the Client of the abort, the remote connection is closed, and MetroPro is returned to interactive mode.

All information transferred between MetroPro and the Client application is tested with a Cyclic Redundancy Check (CRC).  This ensures data integrity and avoids many inter-process communications problems.

The Client computer must be visible from the MetroPro Server computer in order for the Client process to connect to MetroPro if using a Local Area Network.

### MetroPro (Server) Instructions

1. Modify file mraServer.cfg to your preferences if needed.
2. Make sure that all settings in mraClient.cfg are the same; only the ConnectionTimeout parameter may differ.
3. Start up MetroPro on the Server computer.  Click on the Remote Access button (New Button → Miscellaneous → Remote Access) to put MetroPro in Remote Access Mode.  MetroPro will wait for client connections and commands.

## Client Process Instructions

1. Make sure that all of the relevant settings in mraClient.cfg match those in mraServer.cfg; only the Connection Timeout parameter may differ. Copy (not move) mraClient.cfg and *MetroProRemoteAccess.dll* from MetroPro\cfg to the same directory as the client application.
2. From the Client application, establish a communications link with MetroPro by calling function *mraInitialize*. If successful, the Client may now do any of the following:
    a. **Send a MetroScript script file**
    Call function *mraPutFile*. MetroPro will receive the MetroScript file.

    b. **Load a MetroScript script file**
    Call function *mraLoadScript*. This function is used to load a MetroScript file that had previously been transferred from the client computer into MetroPro, making the script available for execution.

    c. **Run a MetroScript script**
    Call function *mraRunScript*. MetroPro will load and execute the specified MetroScript file. The file must have been transferred to the Server prior to calling this.
    Call function *mraWaitScriptDone* to wait for MetroPro to finish executing the script.
    Call *mraIsScriptDone* to query whether or not MetroPro has finished executing the script.
    Call function *mraGetScriptResults* to receive the results from the execution of the script. This call is required to close out the script execution cycle.

    MetroPro will not be able to receive new data, process a new run script request, or receive a single MetroScript command until the script execution has been completed and all outbound data produced during the current script execution has been successfully transferred to the client process.

    d. **Send a single MetroScript Command**
    Call function *mraSendMetroScriptCommand*. This function will transfer and execute a string containing a single MetroScript command.
    Call function *mraWaitScriptDone* to wait for MetroPro to finish executing the script command.
    Call *mraIsScriptDone* to query whether or not MetroPro has finished executing the script command.
    Call function *mraGetScriptResults* to receive the results from the execution of the script command. This call is required to close out the command execution cycle.

MetroPro will not be able to receive new data, process a new run script request, or receive a single MetroScript command until the script execution has been completed and all outbound data produced during the current script execution has been successfully transferred to the client process.

e. **Send a data file**
Call function *mraPutFile*. MetroPro will receive the MetroScript file.

f. **Send a pattern or mask file**
Call function *mraPutFile*. MetroPro will receive the MetroScript file.

g. **Abort Check**
Call function *mraAbortcheck.* This function will check for an Escape key press if abort check is enabled.

h. **Enable abort check**
Call function *mraEnableAbortCheck*. This will allow background processing to capture any press of the Escape key.

i. **Disable abort check**
Call function *mraDisableAbortCheck*. This causes the Escape key to be ignored.

j. **Stop MetroPro Processing**
Call function *mraStopMetroProProcressing*. This function will stop the execution of a script in MetroPro.

k. **Kill MetroPro Server**
Call function *mraKillMetroProServer*. This function will kill the MetroPro server and pop up the Remote Access button in MetroPro. You will have to restart the server in MetroPro and connect again to control MetroPro.

3. Close the communications link with MetroPro by calling function *mraClose*.

## Remote Control of Multiple ZYGO Instruments

MetroPro Remote Access DLL provides functionality for remote controlling of more than one ZYGO instrument in real time. Up to 32 interferometers can be controlled simultaneously. The following functions are available for this purpose.

- *mraSetCurrentServer*
- *mraGetCurrentServer*

With these functions, a Client application can switch between different links to different MetroPro Servers.  To run simultaneous measurements on two separate ZYGO instruments, follow these steps.

1. Connect to Server instrument 1 running MetroPro.
   Use function *mraInitialize.*
2. Send Measure MetroScript command.
   Use function *mraSendMetroScriptCommand.*
3. Switch link to Server instrument 2 running MetroPro.
   Use function *mraSetCurrentServer.*
4. Connect to Server instrument 2 running MetroPro.
   Use function *mraInitialize.*
5. Send Measure MetroScript command.
   Use function *mraSendMetroScriptCommand.*
6. Switch link to Server instrument 1.
   Use function *mraSetCurrentServer.*
7. Query for Measure command completion.
   Use function *mraIsScriptDone.*

8 Switch link to Server instrument 2.
   Use function *mraSetCurrentServer.*

9. Query for Measure completion.
   Use function *mraIsScriptDone.*
10. Repeat steps 6 – 9 until both measurements are completed.
    Use a loop.
11. Switch link to Server instrument 1.
    Use function *mraSetCurrentServer.*
12. Get measurements results.
    Use function *mraGetScriptResults.*
13. Switch link to Server instrument 2.
    Use function *mraSetCurrentServer.*
14. Get measurements results.
    Use function *mraGetScriptResults.*
15. Switch link to Server instrument 1.
    Use function *mraSetCurrentServer.*
16. Close connection to Server instrument 1.
    Use function *mraClose.*
17. Switch link to Server instrument 2.
    Use function *MraSetCurrentServer.*
18. Close connection to Server instrument 2.
    Use function *mraClose.*

## Remote Access Server Suspend and Resume

A suspend/resume mechanism allows local interaction with MetroPro while the remote access server is running without having to terminate the server or close a remote connection with a client.

1. **Suspending the Remote Access Server**
   When the remote access server is running and there are no clients connected, the server can be suspended by pressing the F11 key. Control returns to the local user. This allows local interaction without having to terminate the remote access server. Remote client connection attempts will fail until the remote access server is resumed.

   When a client is connected, the F11 key will only work when the server is idle and waiting for remote commands. If the server is busy, pressing the F11 key will be ignored. While the server is suspended, all remote client commands are also ignored.

   By default, the local user will be asked to verify the suspension of the server after pressing F11 (i.e. interactive suspend/resume). The remote client, however, can change his default behavior by calling mraSetSuspendPermissions(). The client can deny suspend/resume capabilities, meaning that F11 key presses will be ignored. The client can also set the suspend/resume permissions to be non-interactive, so that the user is not prompted when F11 is pressed.
2. **Terminating the Remote Access Server**
   If suspended, pressing the F11 key again clicking on the Remote Access button, will resume the server, and control returns to the remote client.
3. **Terminating the Remote Access Server**
   The remote access server cannot be terminated while suspended. The server must be resumed and then the <Escape> key pressed to terminate. If a client is connected, press the <Escape> key twice to first close the connection with the client and then to terminate the server.
4. **Disconnecting a Remote Client**
   A remote client cannot be disconnected while the remote access server is suspended. The server must be resumed, and then the <Escape> key pressed once to disconnect the remote client.
5. **Interactive Function Key Exceptions**
   Some of the F keys defined in MetroPro do not require the remote access server to be suspended for the local user to use them. The keys F1—Measure, F2—Analyze, F4—Light Level dialog, F5—Auto LLC, and F9—Live Display, will work as if MetroPro were not in remote access mode. However, if a client is connected, these function keys will not work if the remote client has denied suspend/resume permissions.

6. **MetroPro Interface**
   The pop up messages that appeared in MetroPro while in remote access mode have been replaced by a server information string layer attached to the title in an application. This allows faster interaction with the client and faster execution of remote scripts. The messages that appear on the title bare include:

   a. **Remote Access Offline—**This message momentarily appears after the remote access server has been created. After the communication channel has been initialized on the server side, this message is deleted.

   b. **Remote Access Control –Idle -No Host—**This message appears after the communication channel has been initialized and the server is waiting for client connections.

   c. **Remote Access Control – Idle- Controlled by "host"—**This message appears after a client has successfully connected to the server and the server is waiting for remote client commands. Host is the client computer name.

   d. **Remote Access Control – Busy – Controlled by "host"—**This message appears when the server is processing remote client commands. These commands can either be scripts, data transfer, or diagnostics. Host is the client computer name.

   e. **Remote Access Control – Suspended**—This message appears when the server has been either locally or remotely suspended. It will remain displayed until the server is resumed.

   f. **Remote Access Control – Resumed—**This message will momentarily appear after the server has been resumed. It will then be deleted and a message corresponding to the state of the server will be displayed instead. This new message will either be Remote Access Control – Idle No Host or Remote Access Control – Idle –Controlled by host.

   g. **Remote Access Control – Unable to Create—**This message appears when MetroPro is unable to create the Remote access server. Ensure that the configuration file exists.

7. **Remote MetroPro Suspend**
   New functions that have been added to the MetroPro Remote Access.dll allow remote suspending of the remote access server in MetroPro. These functions include:

   a. **Remotely Suspend the Remote Access Server**—Use function *mraSuspendServer* to remotely suspend the remote access server in MetroPro. After suspended, the server will not be able to process remote client commands.

   b. **Set Local User Suspend Permissions**—Use function *mraSetSuspendPermissions* to set local client suspend and resume permissions.

      Only the local user can resume the remote access server.

   c. **Get Remote Access Server Status**—Use function mraGetServerStatus to obtain the current server status.

## Configuration Settings

The parameter options for each mode of remote access are described below. The settings for all parameters must be the same in both *mraClient.cfg* and *mraServer.cfg*.

### Local Area Network Settings (Named Pipe Interface)

- INTERFACE

  Set to LAN to use a Named Pipe Interface to control MetroPro remotely. This is the default setting.

- IO_MODE

  Set to NON_BLOCKING to use communications operations asynchronously (overlapped). Asynchronous read or write operations will not wait for completion and will return immediately after being called. This mode offers more flexibility for integration with other software. This is the default setting.

  Set to BLOCKING to use communications operations synchronously (non-overlapped). Synchronous read or write operations will block the application until they are finished or a time-out occurs. This mode is less flexible and should be used with single threaded applications only.

- MACHINE

  Set to the name of the computer running MetroPro. This name can be obtained by clicking on Control Panel –> Network Icon –> Identification Tab on the computer where MetroPro is located. This parameter is blank by default.

- READ TIMEOUT

  Set to the maximum amount of time (in seconds) that a read operation will wait for a response before failing. This parameter must be set to a value greater than 1. The default timeout is 60 seconds.

- PACKET TIMEOUT

  Set to the maximum amount of time (in seconds) that the application will wait for beginning an internal data transfer between the client and MetroPro. This parameter must be set to a value greater than 1. The default timeout is 60 seconds.

- CONNECTION TIMEOUT

  Set to the maximum amount of time (in seconds) that the application will wait for a named pipe server to become available. The default setting for *mraServer.cfg* is INFINITE. The default setting for *mraClient.cfg* is 30.

- SCRIPTS ALLOWED

  Set to ALL for running all scripts passed by remote client.

  Set to COMPATIBLE_ONLY if prohibiting scripts that require user interaction from running in MetroPro.

  The default setting is COMPATIBLE_ONLY. This setting is optional. If not present server runs compatible scripts only.

## Default Configuration Files

ZYGO provides default configuration files used by MetroPro and the remote access DLL. *mraClient.cfg* and *mraServer.cfg* are installed into the Metropro\cfg directory and have the same matching default settings. If these files are edited, they must both be edited to match.

### Default mraserver.cfg

```
# Zygo Corporation. All Rights Reserved.
# mraserver.cfg
# Remote access configuration file
# To be used by client process accessing MetroPro.
# Lines that begin with '#' are comments.
# Access Interface Setting.
# INTERFACE = LAN
# IOMODE = BLOCKING or IOMODE = NON_BLOCKING
INTERFACE=LAN
IOMODE=NON_BLOCKING
READ TIMEOUT=60
CONNECTION TIMEOUT=120
PACKET TIMEOUT=60
# Named Pipe Interface Settings
MACHINE=NEW_VIEW1
# Script Settings. Set to ALL or COMPATIBLE_ONLY
SCRIPTS ALLOWED=COMPATIBLE_ONLY
```

### Default mraClient.cfg

```
# Zygo Corporation. All Rights Reserved.
# mraclient.cfg
# Remote access configuration file
# To be used by client process accessing MetroPro.
# Lines that begin with '#' are comments.
# Access Interface Setting.
# INTERFACE = LAN
# IOMODE = BLOCKING or IOMODE = NON_BLOCKING
[SERVER 1]
INTERFACE=LAN
IOMODE=NON_BLOCKING
READ TIMEOUT=60
CONNECTION TIMEOUT=120
PACKET TIMEOUT=60
# Named Pipe Interface Settings
MACHINE=NEW_VIEW1
# To add an additional server, copy lines 8-17
# and paste below this line. Modify server marker and parameters.
```

## Sample mraClient.cfg Edited for Controlling Multiple Instruments

```
# Zygo Corporation. All Rights Reserved.
# mraclient.cfg
# Remote access configuration file
# To Be used by client process accessing MetroPro.
# Lines that begin with '#' are comments.
# Access Interface Setting.
# INTERFACE = LAN
# IOMODE = BLOCKING or IOMODE = NON_BLOCKING
[SERVER 1]
INTERFACE=LAN
IOMODE=NON_BLOCKING
READ TIMEOUT=60
CONNECTION TIMEOUT=30
PACKET TIMEOUT=60
# Named Pipe Interface Settings.
MACHINE=NEW_VIEW1
# To add an additional server, copy lines 8-17
# and paste below this line. Modify server marker and parameters.
[SERVER 2]
INTERFACE=LAN
IOMODE=NON_BLOCKING
READ TIMEOUT=60
CONNECTION TIMEOUT=30
PACKET TIMEOUT=60
# Named Pipe Interface Settings.
MACHINE=NEW_VIEW2
# To add an additional server, copy lines 8-17
# and paste below this line. Modify server marker and parameters.
# END
```

Chapter 3

# DLL Function Reference

This section describes all of the functions available in *MetroProRemoteAccess.dll*. They are listed in alphabetical order. Use function *mraGetLastError* to retrieve and check if any error conditions occurred after any remote access function call that has possible error values.

## int mraAbortCheck ( char errBuf[BUFSIZ] )

*Description*

This function checks for the escape key if function *mraEnableAbortCheck* has been called first. It sets the abort flag and local abort flag to TRUE if the escape key was pressed. If *mraDisableAbortCheck* function has been called, the return value is always 0. The client process will not be able to communicate and control MetroPro while the abort Flag is set to TRUE. Call *mraSetAbortFlag* to reset the abort flag.

*Parameters*

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns 1 if the Escape key was pressed. If there are any errors in the data transmission process or the Escape key was not pressed, this function returns 0.

*Errors*

MRA_ERR_INVALID_PARAMETERS

## void mraClearLastError ( )

*Description*

Function clears the last error produced by any *MetroProRemoteAccess.dll* function call.

*Parameters*

None

*Return Values*

None

*Errors*

Function always returns MRA_ERR_NO_ERR.

## int mraClose ( char errBuf[BUFSIZ] )

*Description*

This function closes the client side of the named pipe. All data located in the input or output queues is purged, and the handle to the communications device is closed. This function fails if there is no open handle to a communications device.

*Parameters*

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR (0) if successful. Otherwise, it returns an error code, and errBuf will contain an error message. *mraGetLastError* can also be used to check the status of this command.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_OPERATION

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

## int mraDisableAbortCheck ( )

*Description*

This function disables the abort check. The client application will ignore the Escape key, and function *mraAbortCheck* will return 0 until the abort check is re-enabled.

*Parameters*

None

*Return Values*

0

*Errors*

Always returns MRA_ERR_NO_ERR.

## int mraEnableAbortCheck ( )

*Description*

This function enables the abort check. The client application will capture the Escape key when function *mraAbortCheck* is called.

*Parameters*

None

*Return Values*

0

*Errors*

Always returns MRA_ERR_NO_ERR.

## int mraFlushPipe ( char errBuf[BUFSIZ] )

*Description*

This function will remove all bytes available for reading in the read buffer of the client side of the named pipe. This function fails if a named pipe client has not been created before.

*Parameters*

errBuf – output
  Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns the number of bytes that were purged from the client side of the named pipe if successful. If there was an error in transmission, this function returns an error code, and errBuf will contain a message explaining the error.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_OPERATION

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_NP_UNABLE_TO_READ_DATA

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## char mraGetAbortFlag ( )

*Description*

This returns the value of the abort flag. The abort flag is set to1 when any of the following conditions occur:

- function *mraEnableAbort* has been called and the Escape key has been pressed;
- when the application has detected that the server has been terminated;
- when application has detected that the communication link has been broken.

*Parameters*

None

*Return Values*

Returns the value of the abort flag.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_NO_ACCESS_INTERFACE

## int mraGetClientAccessInterface ( )

*Description*

This function returns an integer value that represents the current communications device used for controlling MetroPro. If connected, this function pings the server to verify a valid connection. If the ping fails, then the client access interface is set to zero.

*Parameters*

None

*Return Values*

Returns a 1 if at the present time the client process is controlling MetroPro through a LAN. Returns a 0 if a communications link has not been established, if the communication link was previously closed, or if the ping failed.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_NO_ACCESS_INTERFACE

## HWND mraGetClientWindow ( )

*Description*

This function returns the handle to the current window in the client process if the handle was previously set. The Client Window Handle is used to check for the Escape key.

*Parameters*

None

*Return Values*

Returns the Handle to the current window of the client process. Otherwise, the return value is NULL.

## int mraGetCurrentServer ( )

*Description*

This function returns the id of the current server being accessed by *MetroProRemoteAccess.dll*. If there are no communications opened, this function returns a default value of 1.

*Parameters*

None

*Return Values.*

Returns the id of the server currently being controlled by *MetroProRemoteAccess.dll* or the default server id value of 1.

## int mraGetFile ( char source_fn[BUFSIZ],
## char destination_fn[BUFSIZ],
## char errBuf[BUFSIZ] )

*Description*

This function gets a file from the MetroPro computer and saves it on the client computer. The file to get is specified by source_fn, and the file to save is specified by destination_fn. This function fails if there are errors in the data transmission process, the file cannot be found in MetroPro, or if there is not enough disk space in the client computer to save the file. All data transfers are verified by a Cyclic Redundancy Check to ensure data integrity.

*Parameters*

source_fn - input
Null terminated string of type char, which holds the name of the file in MetroPro to be transferred.

destination_fn - input
Null terminated string of type char, which holds the name of the file to be saved in the client computer.

errBuf - output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Returns an error code if function failed and errBuf will contain a message explaining the error.

*Errors*

MRA_ERR_NO_ERR
MRA_ERR_INVALID_PARAMETERS
MRA_ERR_INVALID_HANDLE
MRA_ERR_NP_UNABLE_TO_FIND_SERVER
MRA_ERR_INVALID_OPERATION
MRA_ERR_NP_UNABLE_TO_WRITE_DATA
MRA_ERR_PROCESS_ABORTED
MRA_ERR_SERVER_ERROR
MRA_ERR_DATA_TRANSFER_ERROR
MRA_ERR_NP_UNABLE_TO_READ_DATA
MRA_ERR_UNABLE_TO_FIND_FILE
MRA_ERR_UNABLE_TO_WRITE_TO_DISK
MRA_ERR_UNABLE_TO_PROCESS_CRC
MRA_ERR_CRC_ERROR

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## char mraGetLocalAbortFlag ( )

*Description*

The *local abort flag* is set when the abort check has been enabled by calling function *mraEnableAbortCheck*, and the Escape key has been pressed and not cleared.

*Parameters*

None

*Return Values*

Returns the value of the local abort flag.

## int mraGetLastError ( )

*Description*

This function returns the last error produced by the last *MetroProRemoteAccess.dll* function called.

*Parameters*

None

*Return Values*

Returns the last error produced by the last *MetroProRemoteAccess.dll* function called.

*Errors*

None

## int mraGetScriptResults ( char mpErrMsg[BUFSIZ], char mpOutput[BUFSIZ], char errBuf[BUFSIZ] )

*Description*

This function obtains the results after a remote script or remote command has been executed in MetroPro. It **must** be called after *mraIsScriptDone* returns true or *mraWaitScriptDone* returns a zero even if the script had no output. It is used to validate that the script or MetroScript command was run successfully. mpErrMsg stores the error number produced by MetroPro (if any) and the message displayed in the MetroPro GUI if the script execution failed. mpOutput stores the string produced by the MetroScript print command. If a single command was transferred and it contained an assignment operation, then the value will be stored in the output buffer.

If a script requires user interaction in MetroPro, a script incompatible error will be returned unless the server configuration file allows all scripts to be executed.

*Parameters*

mpErrMsg - output
Null terminated string of type char, which holds any error number and error message produced by MetroPro during the script execution.

mpOutput - output
Null terminated string of type char, which holds any output produced by the script execution.

errBuf
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if the function completed successfully.

Returns an error code if the function failed. errBuf buffer will contain a string explaining return value.

mpErrMsg buffer indicates whether a script has failed or not. If empty, script has completed successfully in MetroPro. If not null, script has failed and its contents need to be passed to the user for evaluation.

In addition to script output, the mpOutput buffer may contain script handled error information. Evaluate if script output values are previously known.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_GETTING_RESULTS_DATA_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_NP_UNABLE_TO_FIND_SERVER

MRA_ERR_PROCESS_ABORTED

MRA_ERR_INCOMPATIBLE_SCRIPT

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraGetScriptStatus ( bool poll, char errBuf [BUFSIZ] )

*Description*

This function should only be called after calling *mraRunScript* or *mraSendMetroScriptCommand*.

The primary use of this function should be the same as *mraIsScriptDone*. It should be used inside a loop that periodically polls whether the script has completed execution in MetroPro or not. The poll input parameter should be set to true.

The secondary use of this function should be to determine whether there are script status messages pertaining to a particular operation in the script. These messages are returned by this function as a numeric value greater than 1. If an immediate action must be taken prior to the end of the script in a multi-module client application, then this function must be inside a loop that waits for script completion. The return value must be checked, and if a status message is returned, immediate action must be taken.

Function *mraGetScriptResults* must be called after this function returns a script done status.

As of 08/05/04, only one status message has been defined in MetroPro. When FDA (Frequency Domain Analysis) is about to begin a status message is sent by MetroPro and retrieved by this function. This status message has a numeric value of 2.

*Parameters*

Poll – input

Boolean input parameter that determines whether this function returns immediately or waits for incoming data from the server. Pass true if an immediate return is necessary. Else, pass false (faster throughput).

errBuf

Null terminated string of type char, which holds any error messages that *MetroProRemoteAccessMtd.dll* may produce.

*Return Values*

Returns 0 if the script has not completed execution in MetroPro.

Returns -1 if there was an error. Call function mraGetLastError for additional error information.

Returns 1 (script done status) if the script completed execution.

Returns a value greater than 1 if a script status message has been received.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_GETTING_RESULTS_DATA_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_NP_UNABLE_TO_FIND_SERVER

MRA_ERR_PROCESS_ABORTED

MRA_ERR_INCOMPATIBLE_SCRIPT

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraGetServerStatus ( )

*Description*

This function returns the current status of the server.

*Parameters*

None

*Return Values*

The return value is an integer value indicating the current status of the server.

0 – (SERVER UNKNOWN STATE)

This status indicates that either the connection to the remote server has not been made, or it has been lost.  Given that there is no connection, there is no way to determine what the current server status is.

2 – (WAITING FOR COMMANDS STATE)

This status indicates that there is a valid connection with the remote access server in MetroPro, and that the server is idle and waiting for remote commands to process.

3 – (RUNNING CLIENT SCRIPT STATE)

This status indicates that there is a valid connection with the remote access server in MetroPro, and that the remote access server is running a remote script previously sent by the *mraRunScript( )* MetroPro Remote Access dll function, or *mraSendMetroScriptCommand( )* MetroPro Remote Access dll function.

4 – (PROCESSING COMMANDS STATE)

This status indicates that there is a valid connection with the remote access server in MetroPro, and that the remote access server is processing a remote command other than running a script.

5 – (SERVER SUSPENDED STATE)

This status indicates that there is a valid connection with the remote access server, but the server is currently unavailable because it is suspended.

6 – (SERVER_SCRIPT_DONE_STATE)

This status indicates that the script previously being run has finished its execution, and the server is waiting for script results to be retrieved.

*Errors*

None

## char mraGetRemoteAbortFlag ( )

*Description*

The *remote abort flag* is set when an abort message has been received from MetroPro. This message indicates that a user has aborted the remote access functionality on the server side in MetroPro by pressing the Escape key twice while the Remote Access button was pushed down. Function *mraClose* must be called after getting a remote abort flag set to true. Then, if the server has not been terminated, call function *mraInitialize* to reconnect.

*Parameters*

None

*Return Values*

Returns the value of the *remote abort flag*.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_NO_ACCESS_INTERFACE

## double mraGetTotalBytesReceived ( )

*Description*

This function returns the cumulative total number of bytes received from MetroPro during the current remote access session. The total number of bytes received is reset every time the *mraInitialize* function is called. The return value of this function is a good indicator that data is actually being transferred.

*Parameters*

None

*Return Values*

Returns the cumulative total number of bytes received from MetroPro during the current remote access session. Returns a 0 if a communications link was not previously established.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

## double mraGetTotalBytesSent ( )

*Description*

This function returns the cumulative total number of bytes sent to MetroPro during the current remote access session. The total number of bytes sent is reset every time the *mraInitialize* function is called. The return value of this function is a good indicator that data is actually being transferred. If a communications link was not previously established, the return value of this function is 0.

*Parameters*

None

*Return Values*

Returns the cumulative total number of bytes sent to MetroPro during the current remote access session. Returns a 0 if a communications link was not previously established.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraInitialize (char errBuf[BUFSIZ],
## char username[MAX_USERNAME_LENGTH],
## char password[MAX_PASSWORD_LENGTH],
## int serverGuiMsgs,
## bool_t log_operations)

*Description*

This function must be the first function called when using *MetroProRemoteAccess.dll*. It reads the configuration settings from *mraClient.cfg* and then tries to establish a connection with MetroPro using a LAN, according to the configuration file. If successful, it establishes a communications link with the remote access server in MetroPro.

*Parameters*

errBuf - output

Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

username - input

User name for logging in to the server's computer. Use username and password parameters only if either the server or client computers are not logged in to the same domain in the local area network. The username and password must have been previously defined in the server's computer by using Windows NT's User Manager. If the server and client computers are logged in the same domain in the local network, use NULL for both the username and password. MAX_USERNAME_LENGTH is defined as 20.

password - input

Password is used in combination with username. Do not use if username is not applicable. The username and password must have been previously defined in the server's computer by using Windows NT's User Manager. MAX_PASSWORD_LENGTH is defined as 14.

serverGuiMsgs - input

This parameter indicates whether MetroPro will display server status messages. These messages indicate the current status of MetroPro and hint what is happening in the background. Use any of the following values:

1 – (Messages on. Script output enabled)
2 – (Messages off. Script output disabled)
3 – (Messages on. Script output disabled)
4 – (Messages off. Script output enabled)

Function *mraTurnOnOffGuiMessages* can be used to enable or disable messages after the client has successfully connected to MetroPro.

log_operations - input

This parameter indicates whether a log file for the client usage of *MetroProRemoteAccess.dll* will be kept. Set to TRUE or YES if a log is desired to keep a record of function calls made to the DLL. Set to FALSE or NO if no log is needed. Function *mraTurnClientLogOnOff* can be used to stop logging or enable it.

*Return Values*

Returns MRA_ERR_NO_ERR  if successful. Returns an error code if function fails and errBuf will contain an error string.

*Errors*

MRA_ERR_NO_ERR
MRA_ERR_INVALID_PARAMETERS
MRA_ERR_ALREADY_CONNECTED
MRA_ERR_NO_ERR
MRA_ERR_UNABLE_TO_FIND_CFG_FILE
MRA_ERR_INVALID_OPERATING_SYSTEM
MRA_ERR_NP_UNABLE_TO_FIND_SERVER
MRA_ERR_INVALID_USERNAME_OR_PASSWD
MRA_ERR_NP_UNABLE_TO_FIND_SERVER
MRA_ERR_NO_NP_SERVER_AVAILABLE
MRA_ERR_NP_CONNECTION_TIMEOUT
MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT
MRA_ERR_UNABLE_TO_ESTABLISH_LINK

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraIsClientOnline ( )

*Description*

This function checks the communication link and determines if there is a valid connection with the server or not.

*Parameters*

None

*Return Values*

Returns 1 if client is currently online.  Returns 0 if client is offline.

*Errors*

None

## int mraIsScriptDone ( char errBuf[BUFSIZ] )

*Description*

This function queries MetroPro to know if it has finished executing a script. It returns immediately.

*Parameters*

errBuf - output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR (0) if MetroPro has not finished executing a script. Otherwise, it returns a number greater than zero. It returns a value less than zero if an error in communications has occurred.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_QUERYING_SERVER

MRA_ERR_NP_UNABLE_TO_FIND_SERVER

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraLoadScript ( char scriptFileName[BUFSIZ], char errBuf[BUFSIZ] )

*Description*

This function sends the name of a script indicated by scriptFileName to MetroPro. To have MetroPro run the script loaded by this function, you must call function *mraRunScript*. *mraLoadScript* fails if there is an error in the transmission of the script name or if the script cannot be found in MetroPro.

*Parameters*

scriptFileName – input
Null terminated string of type char, which holds the name of the script file to be loaded by MetroPro.

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if successful.  Otherwise, it returns an error code, and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_NP_UNABLE_TO_FIND_SERVER

MRA_ERR_UNABLE_TO_SEND_COMMAND

MRA_ERR_UNABLE_TO_LOAD_SCRIPT

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraKillMetroProServer ( char errBuf[BUFSIZ] )

*Description*

Use this function to stop the MetroPro remote access server.  This function will transfer a series of commands that will halt any execution in MetroPro and close the remote access connection.  The Remote Access button will pop up.

*Parameters*

errBuf - output

Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if remote access server in MetroPro was stopped and terminated.  Returns an error code if an error occurred and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_UNABLE_TO_KILL_SERVER

## int mraPutFile ( char source_fn[BUFSIZ], char destination_fn[BUFSIZ], char errBuf[BUFSIZ] )

*Description*

Use this function to send a script or other type of file to MetroPro. All information sent is validated through a Cyclical Redundancy Check to ensure data integrity. This function waits for a ready to receive message from MetroPro before starting the data transfer. The file to transfer is specified by source_fn, and the target file is specified by destination_fn. A path can be specified for either source or destination if needed. If no path is specified then the file will be transferred to the MetroPro local directory.

*Parameters*

source_fn - input
Null terminated string of type char, which holds the name of the file to be transferred.

destination_fn - input
Null terminated string of type char, which holds the name of the file to be saved to.

errBuf - output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Function returns MRA_ERR_NO_ERR if successful. Returns an error code if it fails, and errBuf will contain a string error.

*Errors*

MRA_ERR_NO_ERR
MRA_ERR_INVALID_PARAMETERS
MRA_ERR_INVALID_HANDLE
MRA_ERR_INVALID_OPERATION
MRA_ERR_NP_UNABLE_TO_FIND_SERVER
MRA_ERR_UNABLE_TO_PROCESS_CRC
MRA_ERR_UNABLE_TO_FIND_FILE
MRA_ERR_PROCESS_TIMEOUT
MRA_ERR_NP_UNABLE_TO_WRITE_DATA
MRA_ERR_NP_UNABLE_TO_READ_DATA
MRA_ERR_PROCESS_ABORTED
MRA_ERR_DATA_TRANSFER_ERROR
MRA_ERR_CRC_ERROR

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraRunLoopBackTest ( char errBuf[BUFSIZ] )

*Description*

This function runs a loop back test with MetroPro, which is a diagnostic tool to ensure that a good communication link exists between the Client process and MetroPro. The test consists of the generation and transfer of random numbers between the client process and MetroPro.

*Parameters*

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Otherwise, it returns an error code, and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_DATA_TRANSFER_ERR

MRA_ERR_LOOPBACKTEST_ERROR

## int mraRunScript ( char scriptFileName[BUFSIZ], char errBuf[BUFSIZ])

*Description*

This function tells MetroPro to load and run the MetroScript script specified by *scriptFileName*. The script must have been transferred to the MetroPro computer prior to calling this function. If *scriptFileName* is null, MetroPro will run the last script loaded while connected. If no scripts have been loaded before, this function fails.

If a script requires user interaction in MetroPro, a script incompatible error will be returned unless the server configuration file allows all scripts to be executed.

*Parameters*

scriptFileName – input
Null terminated string of type char, which holds the name of the script to be run in MetroPro.

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Otherwise, it returns an error code, and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_NP_UNABLE_TO_FIND_SERVER

MRA_ERR_DATA_TRANSFER_ERR

MRA_ERR_INCOMPATIBLE_SCRIPT

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraSendMetroScriptCommand ( char *metroScriptCommand, char errBuf[BUFSIZ] )

*Description*

Use this function to send a one line, single MetroScript command to MetroPro. MetroPro will execute the script command as soon as it receives it. MetroPro will interpret every remote command as a different script. Call function *mraWaitScriptDone* and *mraGetScriptResults* to obtain the results of the execution of the command.

If a command requires user interaction in MetroPro, a script incompatible error will be returned unless the server configuration file allows all scripts to be executed.

*Parameters*

MetroScriptCommand - input
Null terminated string that holds the MetroScript command to be transferred. The maximum size of the string is 512 characters.

errBuf – output
Null terminated string of type char, which holds any error messages *that MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if successful, otherwise, it returns an error code and errBuf will have an error message in it.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_INVALID_OPERATION

MRA_ERR_UNABLE_TO_SEND_COMMAND

MRA_ERR_STRING_TOO_LONG

MRA_ERR_UNABLE_TO_FIND_FILE

MRA_ERR_INCOMPATIBLE_SCRIPT

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraSetAbortFlag ( int value )

*Description*

This function sets the abort Flag to TRUE if *value* is greater than 0. If *value* is equal or less than 0, the abort Flag is set to FALSE. This function must be used carefully. You will not be able to communicate with MetroPro while the abort Flag is set to TRUE.

*Parameters*

value – input
    Integer parameter used to set the abort Flag.

*Return Values*

Always returns 0.

## int mraSetClientWindow ( HWND clientProcessWindow )

*Description*

This function sets the window handle of the client process to be used by function *mraAbortCheck*.

*Parameters*

ClientProcessWindow - input
    A value of type HWND, which holds the window handle of the client process.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Returns an error code if it fails.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_OPERATION

## int mraSetConnectionPolling ( int flag, int interval )

*Description*

Enables or disables the connection polling from the client to the server (i.e. MetroPro) and sets the polling interval. A background thread is used in *MetroProRemoteAccess.dll* to constantly check the connection with the server. This assures that even though the client may not be communicating with the server constantly, the connection is being verified in a continuous base. It is not recommended that the connection polling be disabled. The default polling interval is 2000 milliseconds.

*Parameters*

flag - input
  Set to 1 to enable connection polling. Set to 0 for disabling connection polling.

interval - input
  Set to the number of milliseconds between connection checks.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Otherwise, returns an error.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_UNABLE_TO_SUSPEND_POLLING

MRA_ERR_UNABLE_TO_RESUME_POLLING

## int mraSetCurrentServer ( char errBuf[BUFSIZ], int serverId )

*Description*

Use this function only when remotely controlling more than one ZYGO interferometers.

This function switches the communication link to the server specified by serverId. This parameter must be a valid id value already defined as a tag in *mraClient.cfg*. *mraSetCurrentServer* checks file *mraClient.cfg* to verify that serverId is defined in the configuration file prior to allowing the switching of the communications link. The default serverId is 1.

After calling *mraSetCurrentServer*, the communications link will continue to point to the specified server until another call to *mraSetCurrentServer* is made or the connection terminates. The maximum number of interferometers that can be controlled simultaneously from *MetroProRemoteAccess.dll* is 32.

*Parameters*

errBuf - output
: Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

serverId - input
: Integer value specifying the id of the server to control. This value must be greater than zero and less than 32.

*Return Value*

Returns MRA_NO_ERR if the communications link was switched successfully to the server specified by serverId. Returns an error otherwise.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_UNABLE_TO_FIND_CFG_FILE

MRA_ERR_UNABLE_TO_FIND_SERVER_TAG

MRA_ERR_INVALID_SERVER_INDEX

## int mraSetLoggingParameters ( bool_t setPath, char path[BUFSIZ], bool_t overwriteLogFile, bool_t createNewLogFile, bool_t enableLogging, char errBuf[BUFSIZ] )

*Description*

This function sets remote access DLL logging parameters. It allows setting a new log file path, overwriting and creating a new log file, and enabling and disabling logging.

*Parameters*

setPath — input
Boolean parameter that indicates whether a new log path is to be used. If set to TRUE, path parameter is used as new log file path. If set to FALSE, path parameter is ignored.

path — input
Null terminated string of type char which holds the new log path to be used.

overwriteLogFile — input
Boolean parameter that indicates whether current log file is to be overwritten. If set to TRUE, current log file's contents are erased. If logging is enabled, subsequent logging will continue to be done in same log file.

createNewLogFile — input
Boolean parameter that indicates whether to start logging in a new log file. If set to TRUE, subsequent logging will be done in a new log file if logging is enabled. Current log file's contents are not erased.

enableLogging — input
Boolean parameter that indicates whether to enable or disable logging. If set to TRUE, MetroPro Remote Access DLL function calls are logged to current log file or it creates a new log file if not previously created. If set to FALSE, it does not log Remote Access DLL function calls. If createNewLogFile parameter is set to TRUE, it continues to log in new log file.

errBuf — output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR (0) if successful. Otherwise, it returns an error code, and errBuf will contain an error message. *mraGetLastError* can also be used to check the status of this command.

*Errors*

MRA_ERR_STRING_TOO_LONG

MRA_ERR_INVALID_OPERATION

## int mraSetSuspendPermissions ( int permission, char errBuf[BUFSIZ] )

*Description*

This function sets permissions allowing or denying local suspend and resume capabilities. A valid connection with the remote access server in MetroPro must have been established prior to calling this function. The remote access server must not already be suspended in order for this function to operate correctly.

*Parameters*

permission — input

Integer value which sets the suspend/resume permissions for the local user in MetroPro. The values can be any of the following:

0 – (DENY LOCAL USER SUSPEND PERMISSION)

This permission denies the local user in MetroPro access to suspend and resume capabilities. The F11 key press will not function.

1 – (ALLOW LOCAL USER SUSPEND PERMISSION INTERACTIVE)

This permission allows the local user in MetroPro access to suspend and resume capabilities, but it prompts with a confirmation dialog first.

2 – (ALLOW LOCAL USUER SUSPEND PERMISSION)

This permission allows the local user in MetroPro access to suspend and resume capabilities. There is no prompt for confirmation dialog.

errBuf — output

Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR (0) if successful. Otherwise, it returns an error code, and errBuf will contain an error message. *mraGetLastError* can also be used to check the status of this command.

*Errors*

MRA_ERR_SERVER_SUSPENDED

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_NACK_RECEIVED

MRA_ERR_NO_ERR

MRA_ERR_UNABLE_TO_SEND_COMMAND

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraStopMetroProProcessing ( char errBuf[BUFSIZ] )

*Description*

Use this function to stop a script executing in MetroPro. This function should only be used after function *mraRunScript( )* has been previously called.

*Parameters*

errBuf - output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if processing in MetroPro was stopped. Returns an error code if an error occurred, and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_QUERYING_SERVER

MRA_ERR_UNABLE_TO_STOP_SERVER

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraSuspendServer ( char errBuff[BUFSIZ] )

*Description*

This function suspends the remote access server in MetroPro if the server is running and idle when this function is called. A valid connection to the remote access server in MetroPro must exist prior to calling this function.

*Parameters*

errBuf – output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR (0) if successful. Otherwise, it returns an error code, and errBuf will contain an error message. *mraGetLastError* can also be used to check the status of this command.

*Errors*

MRA_ERR_INVALID_PARAMETERS
MRA_ERR_INVALID_HANDLE
MRA_ERR_NO_ACCESS_INTERFACE
MRA_ERR_SERVER_SUSPENDED
MRA_ERR_NO_ERR
MRA_ERR_NP_UNABLE_TO_WRITE_DATA
MRA_ERR_SERIAL_UNABLE_TO_WRITE_DATA
MAR_ERR_INVALID_OPERATION
MRA_ERR_NACK_RECEIVED
MRA_ERR_UNABLE_TO_SEND_COMMAND

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraTurnClientLogOnOff ( bool_t log )

*Description*

Use to enable or disable logging function calls made to *MetroProRemoteAccess.dll* once a communication has been established. The log file is stored in the directory where the DLL is located. The log file name has an 'mrac' prefix, which stands for MetroPro Remote Access Client, the current date and time, and a '.log' extension (e.g. mrac040402115002.log). Every time a *MetroProRemoteAccess.dll* function is called, it is stored in the log file with the parameters passed, the return value, error buffer received, and a token indicating whether the function succeeded or failed. Each entry in the log file is given an index number. The date and time the function was called is also recorded.

*Parameters*

log - input

Use TRUE or YES to begin logging function calls or to resume logging activities. If parameter log_operations in *mraInitialize* was set to FALSE, this function creates a new log file and begins recording function calls made. Use FALSE or NO for stopping logging activities. Subsequent function calls made will not be recorded.

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Otherwise an error code is returned.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_OPERATION

## int mraTurnOnOffGuiMessages ( char errBuf[BUFSIZ], int messages )

*Description*

Use this function for turning on or off the server's status messages in MetroPro. This function should only be used after communications have been established with the server. Do not use while waiting for script or single command results. MetroScript's 'print' command will not display information on the screen if the messages are turned off.

*Parameters*

errBuf - output

Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

messages - input

Integer value used for determining server GUI information display. Use any of the following values.

1 – (Messages on. Script output enabled)
2 – (Messages off. Script output disabled)
3 – (Messages on. Script output disabled)
4 – (Messages off. Script output enabled)

*Return Values*

Returns MRA_ERR_NO_ERR if successful. Otherwise an error code is returned.

*Errors*

MRA_ERR_NO_ERR

MRA_ERR_INVALID_PARAMETERS

MRA_ERR_INVALID_HANDLE

MRA_ERR_NO_ACCESS_INTERFACE

MRA_ERR_INVALID_OPERATION

MRA_ERR_UNABLE_TO_SEND_COMMAND

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

## int mraWaitScriptDone ( char errBuf[BUFSIZ] )

*Description*

This function waits until MetroPro has finished executing a script.

*Parameters*

errBuf - output
Null terminated string of type char, which holds any error messages that *MetroProRemoteAccess.dll* may produce.

*Return Values*

Returns MRA_ERR_NO_ERR if script was executed.  Returns an error code if an error occurred, and errBuf will contain an error message.

*Errors*

MRA_ERR_NO_ERR
MRA_ERR_QUERYING_SERVER
MRA_ERR_INVALID_PARAMETERS
MRA_ERR_INVALID_HANDLE
MRA_ERR_INVALID_OPERATION
MRA_ERR_SERVER_ERROR
MRA_ERR_DATA_TRANSFER_ERROR

Use function *mraGetLastError* to retrieve errors produced by MetroPro Remote Access DLL functions.

# Example User Programs

Chapter 4

## Sample C Header File

This is a C header file used with samples in this chapter.  It is referred to as mpClient.h.

```
/*
 * mpClient.h
 */
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <winbase.h>
#include <time.h>
#include <conio.h>
#include <io.h>
#include <ctype.h>
#include <winnt.h>
#include <malloc.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <winnetwk.h>
#include <sys/timeb.h>
#include <mmsystem.h>
#include <winuser.h>

#define MRA_NP_INTERFACE     1
#define MRA_NO_INTERFACE     0

#ifndef _BOOL_T
#   define _BOOL_T
  typedef unsigned char      bool_t;
#endif

#define YES                  ((bool_t) 1)
#define NO                   ((bool_t) 0)
#define ON                   YES
#define OFF                  NO
#define MAX_USERNAME_LENGTH  20
#define MAX_PASSWORD_LENGTH  14

#define CALLBACK             __stdcall
/* Define Error Codes*/
#define MRA_ERR_NO_ERR                         0
```

```
#define MRA_ERR_INVALID_PARAMETERS             1
#define MRA_ERR_PROCESS_ABORTED                2
#define MRA_ERR_INVALID_HANDLE                 3
#define MRA_ERR_NO_ACCESS_INTERFACE            4
#define MRA_ERR_NP_PACKET_TIMEOUT              6
#define MRA_ERR_NP_READ_TIMEOUT                8
#define MRA_ERR_NP_CONNECTION_TIMEOUT          10
#define MRA_ERR_NP_UNABLE_TO_FIND_SERVER       11
#define MRA_ERR_NP_UNABLE_TO_READ_DATA         12
#define MRA_ERR_NP_UNABLE_TO_WRITE_DATA        15
#define MRA_ERR_UNABLE_TO_FIND_CFG_FILE        16
#define MRA_ERR_INVALID_OPERATING_SYSTEM       17
#define MRA_ERR_UNABLE_TO_ESTABLISH_LINK       18
#define MRA_ERR_NO_NP_SERVER_AVAILABLE         19
#define MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT     20
#define MRA_ERR_SERVER_ERROR                   21
#define MRA_ERR_DATA_TRANSFER_ERROR            22
#define MRA_ERR_UNABLE_TO_FIND_FILE            23
#define MRA_ERR_PROCESS_TIMEOUT                24
#define MRA_ERR_UNABLE_TO_PROCESS_CRC          25
#define MRA_ERR_CRC_ERROR                      26
#define MRA_ERR_UNABLE_TO_WRITE_TO_DISK        27
#define MRA_ERR_ALREADY_CONNECTED              28
#define MRA_ERR_LOOPBACKTEST_ERROR             29
#define MRA_ERR_UNABLE_TO_ECHO_DATA            30
#define MRA_ERR_UNABLE_TO_SEND_COMMAND         31
#define MRA_ERR_STRING_TOO_LONG                32
#define MRA_ERR_UNABLE_TO_STOP_SERVER          33
#define MRA_ERR_UNABLE_TO_KILL_SERVER          34
#define MRA_ERR_INVALID_OPERATION              35
#define MRA_ERR_INVALID_USERNAME_OR_PASSWD     36
#define MRA_ERR_INVALID_SERVER_INDEX           37
#define MRA_ERR_UNABLE_TO_FIND_SERVER_TAG      38
#define MRA_ERR_UNABLE_TO_LOAD_SCRIPT          39
#define MRA_ERR_UNABLE_TO_DELETE_FILE          40
#define MRA_ERR_UNABLE_TO_RESUME_POLLING       41
#define MRA_ERR_UNABLE_TO_SUSPEND_POLLING      42
#define MRA_ERR_GETTING_RESULTS_DATA_ERR       -1
#define MRA_ERR_QUERYING_SERVER                -2

/* define function pointers data type*/
typedef  int  (CALLBACK* MRAINITIALIZE)  ( char errBuf[BUFSIZ],
                                            char username[MAX_USERNAME_LENGTH],
                                            char password[MAX_PASSWORD_LENGTH],
                                            int messages,
                                            bool_t log_operations );
typedef  int  (CALLBACK* MRARUNSCRIPT)   ( char scriptFileName[BUFSIZ],
                                            char errBuf[BUFSIZ] );
```

```
typedef  int  (CALLBACK* MRAPUTFILE)     ( char source_fn[BUFSIZ],
                                           char destination_fn[BUFSIZ],
                                           char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAGETFILE)     ( char source_fn[BUFSIZ],
                                           char destination_fn[BUFSIZ],
                                           char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAGETSCRIPTRESULTS)  ( char mpErrMsg[BUFSIZ],
                                                 char mpOutput[BUFSIZ],
                                                 char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRALOADSCRIPT)  ( char scriptFileName[BUFSIZ],
                                           char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRATURNONOFFGUIMESSAGES) ( char errBuf[BUFSIZ], int messages );
typedef  int  (CALLBACK* MRASETCURRENTSERVER)  ( char errBuf[BUFSIZ], int server );
typedef  int  (CALLBACK* MRAWAITSCRIPTDONE)    ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRARUNLOOPBACKTEST)   ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRACLOSE)             ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAABORTCHECK)        ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAFLUSHPIPE)         ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAISSCRIPTDONE)      ( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRASTOPMETROPROPROCESSING)( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAKILLMETROPROPROSERVER)( char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRASETCLIENTWINDOW)   ( HWND clientProcessWindow );
typedef  char (CALLBACK* MRAGETABORTFLAG)      ( );
typedef  int  (CALLBACK* MRASETABORTFLAG)      ( int value );
typedef  char (CALLBACK* MRAGETLOCALABORTFLAG) ( );
typedef  char (CALLBACK* MRAGETREMOTEABORTFLAG)   ( );
typedef  int  (CALLBACK* MRAENABLEABORTCHECK)  ( );
typedef  int  (CALLBACK* MRADISABLEABORTCHECK) ( );
typedef  int  (CALLBACK* MRAGETCURRENTSERVER)  ( );
typedef  HANDLE  (CALLBACK* MRAGETCLIENTHANDLE)   ( );
typedef  HWND (CALLBACK* MRAGETCLIENTWINDOW)   ( );
typedef  double (CALLBACK* MRAGETTOTALBYTESRECEIVED) ( );
typedef  double (CALLBACK* MRAGETTOTALBYTESSENT)  ( );
typedef  int (CALLBACK* MRATURNCLIENTLOGONOFF) ( bool_t log );
typedef  int (CALLBACK* MRASETCONNECTIONPOLLING)  ( int flag, int interval );
typedef  int (CALLBACK* MRAGETLASTERROR )      ( );
typedef  void(CALLBACK* MRACLEARLASTERROR )    ( );
typedef  int  (CALLBACK* MRAISCLIENTONLINE)    ( );
typedef  int  (CALLBACK* MRASENDMETROSCRIPTCOMMAND)  ( char * metroScriptCommand,
                                                       char errBuf[BUFSIZ] );
typedef  int  (CALLBACK* MRAGETCLIENTACCESSINTERFACE)( );

/* declare extern function pointers. */
MRAINITIALIZE                   mra_Initialize;
MRASENDMETROSCRIPTCOMMAND       mra_SendMetroScriptCommand;
MRARUNSCRIPT                    mra_RunScript;
MRAPUTFILE                      mra_PutFile;
MRAGETFILE                      mra_GetFile;
MRALOADSCRIPT                   mra_LoadScript;
MRAWAITSCRIPTDONE               mra_WaitScriptDone;
MRAGETSCRIPTRESULTS             mra_GetScriptResults;
MRARUNLOOPBACKTEST              mra_RunLoopBackTest;
MRACLOSE                        mra_Close;
```

```
    MRAABORTCHECK                   mra_AbortCheck;
    MRAFLUSHPIPE                    mra_FlushPipe;
    MRAISSCRIPTDONE                 mra_IsScriptDone;
    MRASETCLIENTWINDOW              mra_SetClientWindow;
    MRAGETABORTFLAG                 mra_GetAbortFlag;
    MRASETABORTFLAG                 mra_SetAbortFlag;
    MRAGETLOCALABORTFLAG            mra_GetLocalAbortFlag;
    MRAGETREMOTEABORTFLAG           mra_GetRemoteAbortFlag;
    MRAENABLEABORTCHECK             mra_EnableAbortCheck;
    MRADISABLEABORTCHECK            mra_DisableAbortCheck;
    MRAGETCLIENTHANDLE              mra_GetClientHandle;
    MRAGETCLIENTWINDOW              mra_GetClientWindow;
    MRAGETTOTALBYTESRECEIVED        mra_GetTotalBytesReceived;
    MRAGETTOTALBYTESSENT            mra_GetTotalBytesSent;
    MRAGETCLIENTACCESSINTERFACE     mra_GetClientAccessInterface;
    MRASTOPMETROPROPROCESSING       mra_StopMetroProProcessing;
    MRAKILLMETROPROPROSERVER        mra_KillMetroProServer;
    MRATURNONOFFGUIMESSAGES         mra_TurnOnOffGuiMessages;
    MRATURNCLIENTLOGONOFF           mra_TurnClientLogOnOff;
    MRASETCURRENTSERVER             mra_SetCurrentServer;
    MRAGETCURRENTSERVER             mra_GetCurrentServer;
    MRASETCONNECTIONPOLLING         mra_SetConnectionPolling;
    MRAGETLASTERROR                 mra_GetLastError;
    MRACLEARLASTERROR               mra_ClearLastError;
    MRAISCLIENTONLINE               mra_IsClientOnline;


/* End of File*/
```

## Example 1 - Transfer and Run a MetroScript File

This C program uses MetroProRemoteAccess.dll to run a script remotely in MetroPro and wait for the execution to finish.

```
/* runscript.c
 * Algorithm:
 *    1. Load MetroProRemoteAccess.DLL
 *    2. Get Process Addresses for function needed.
 *    3. Verify all functions were loaded.
 *    4. Establish connection with MetroPro.
 *    5. Transfer script file to MetroPro. ( if script file is not in MetroPro computer ).
 *    6. Send Run Script command to MetroPro.
 *    7. Wait for script to be done.
 *    8. Get Script Results.
 *    9. Close Connection.
 *    10.Close DLL.
 */

#include "mpClient.h"
int main()
{
   int      fail        = 0,
            ok          = 0,
            scriptDone  = 0,
            dll_ok      = 0,
            err         = 0;
   char     errBuf[BUFSIZ];
   char     MpErrorMsg[BUFSIZ], MpOutput[BUFSIZ],
            in_script[BUFSIZ], out_script[BUFSIZ],
            username[MAX_USERNAME_LENGTH], password[MAX_PASSWORD_LENGTH];
   int      messages;
   bool_t   log_operations;
   HINSTANCE   mpClientLib;
   memset(errBuf, '\0', BUFSIZ);

   // STEP 1
   // Load DLL.
   mpClientLib = LoadLibrary( "MetroProRemoteAccess.dll" );
   if (mpClientLib != NULL)
   {
      // DLL loaded!
      printf("\nDLL loaded successfully.");

      // STEP 2
      // Get Process Addresses for DLL functions needed.
      mra_Initialize          = (MRAINITIALIZE) GetProcAddress( mpClientLib, "mraInitialize" );
      mra_RunScript           = (MRARUNSCRIPT)  GetProcAddress( mpClientLib, "mraRunScript" );
      mra_WaitScriptDone      = (MRAWAITSCRIPTDONE) \
                                   GetProcAddress( mpClientLib, “mraWaitScriptDone" );
      mra_GetScriptResults    = (MRAGETSCRIPTRESULTS) \
                                       GetProcAddress( mpClientLib, "mraGetScriptResults");
      mra_Close               = (MRACLOSE) GetProcAddress( mpClientLib, "mraClose" );
      mra_PutFile             = (MRAPUTFILE) GetProcAddress( mpClientLib, "mraPutFile" );
      mra_GetLastError        = (MRAGETLASTERROR) GetProcAddress( mpClientLib, "mraGetLastError");
      mra_ClearLastError      = (MRACLEARLASTERROR) \
                                 GetProcAddress( mpClientLib, "mraClearLastError");
      mra_IsClientOnline      = (MRAISCLIENTONLINE) \
                                 GetProcAddress( mpClientLib, "mraIsClientOnline");

      // STEP 3
      // Verify that all functions were loaded.
      if( ! mra_Initialize     || ! mra_RunScript        ||
          ! mra_WaitScriptDone || ! mra_GetScriptResults ||
          ! mra_Close          || ! mra_PutFile          ||
          ! mra_GetLastError   || ! mra_ClearLastError   ||
          ! mra_IsClientOnline )
      {
         // Not all functions were loaded. Check spelling.
         dll_ok = 0;
      }
      else
      {
         // All functions were loaded successfully.
         dll_ok = 1;
      }
   }
   else  // Error loading DLL. Check file.
   {
         printf("\nError. Unable to load MetroProRemoteAccess.dll");
   }

   // Begin Remote Control of MetroPro.
   if( dll_ok )
   {
      // STEP 4
      // Establish Connection with MetroPro.
      // Set username and password.
      strcpy(username, "my_username");
      strcpy(password, "my_pwd");
      // Turn server info messages on.
      messages = 1;
      // Turn client logging on.
      log_operations = TRUE;
      mra_ClearLastError();
      err = mra_Initialize ( errBuf, username, password, messages, log_operations );
      if ( err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch ( err )
         {
            case MRA_ERR_INVALID_PARAMETERS:
               // Make sure parameters have the right data type.
         case MRA_ERR_ALREADY_CONNECTED:
            // Already connected to MetroPro.
         case MRA_ERR_UNABLE_TO_FIND_CFG_FILE:
            // Can't find mraClient.cfg. Make sure it is in the local directory.
         case MRA_ERR_INVALID_OPERATING_SYSTEM:
            // Must use Windows NT.
         case MRA_ERR_NP_UNABLE_TO_FIND_SERVER:
            // Machine with name as specified in mraClient.cfg cannot be found on network.
         case MRA_ERR_INVALID_USERNAME_OR_PASSWD:
            // User could not be authenticated.
         case MRA_ERR_NO_NP_SERVER_AVAILABLE:
            // A Named Pipe server cannot be found on machine specified in mraClient.cfg
         case MRA_ERR_NP_CONNECTION_TIMEOUT:
            // A named pipe connection timeout has occurred.
            // Named pipe server is currently not available.
         case MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT:
            // Internal error. Cannot create named pipe client.
         case MRA_ERR_UNABLE_TO_ESTABLISH_LINK:
            // Unable to perform initial software handshake. Disconnect and reconnect.
            fail = 1;
            break;
      }
   }
   else  // Connection was established successfully.
   {
      fail = 0;
   }
   if( ! fail )
   {
      // STEP 5
      // Transfer script to Metropro.
      mra_ClearLastError();
      strcpy(in_script,  "c:\\remotehost\\zygo\\measure.scr");
      strcpy(out_script, "c:\\metropro\\cur_dir\\measure.scr");
      err = mra_PutFile( in_script, out_script, errBuf );
      if ( err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch ( err )
         {
            case MRA_ERR_INVALID_PARAMETERS:
               // Make sure parameters have the right data type.
            case MRA_ERR_INVALID_HANDLE:
               // The client handle has become invalid. Disconnect and reconnect.
            case MRA_ERR_INVALID_OPERATION:
               // A script is running in MetroPro and its results need to be obtained
               // prior to performing any other operation.
            case MRA_ERR_NP_UNABLE_TO_FIND_SERVER:
               // Connection has been lost. Disconnect and reconnect.
            case MRA_ERR_UNABLE_TO_PROCESS_CRC:
               // Unable to process Cyclic Redundancy Check on data.
               // Verify a valid file is being transferred.
            case MRA_ERR_UNABLE_TO_FIND_FILE:
                   // Unable to find file in provided pathname.
                case MRA_ERR_PROCESS_TIMEOUT:
                   // Error. Disconnect and reconnect.
                case MRA_ERR_NP_UNABLE_TO_WRITE_DATA:
                   // A Named Pipe error occurred. Unable to write data. Disconnect and reconnect.
                case MRA_ERR_NP_UNABLE_TO_READ_DATA:
                   // A Named Pipe error occurred. Unable to read data. Disconnect and reconnect.
                case MRA_ERR_PROCESS_ABORTED:
                   // Process has been aborted by local user.
                case MRA_ERR_DATA_TRANSFER_ERROR:
                   // Unable to transfer all bytes in file. Retry.
                case MRA_ERR_CRC_ERROR:
                   // Data has been corrupted. Unable to verify Cyclic Redundancy Check. Retry.
                   fail = 1;
                   break;
             }
          }
          else
             fail = 0;
          if ( fail )
             goto close;

          // STEP 6
          // Run Script "c:\\metropro\\cur_dir\\measure.scr"
          mra_ClearLastError();
          err = mra_RunScript( out_script, errBuf );
          if ( err != MRA_ERR_NO_ERR )
          {
             err = mra_GetLastError();
             switch ( err )
             {
                case MRA_ERR_INVALID_PARAMETERS :
                   // Make sure parameters have the right data type.
                case MRA_ERR_INVALID_HANDLE :
                   // The client handle has become invalid. Disconnect and reconnect.
                case MRA_ERR_NO_ACCESS_INTERFACE :
                   // There is no communication channel with MetroPro. Connect first.
                case MRA_ERR_INVALID_OPERATION :
                   // A script is running in MetroPro and its results need to be obtained
                   // prior to performing any other operation.
                case MRA_ERR_NP_UNABLE_TO_FIND_SERVER :
                   // Connection has been lost. Disconnect and reconnect.
                case MRA_ERR_DATA_TRANSFER_ERROR :
                   // Unable to transfer data. Retry.
                   fail = 1;
                   break;
             }
          }
          else
             fail = 0;
          if ( fail || errBuf[0] != '\0' )
          {
             // There was an error loading and/or opening the script and/or sending the command.
```

```
   printf("\nError = %s", errBuf);
}
if ( ! fail )
{
   // STEP 7
   // If run script request did not fail and was received by MetroPro successfully,
   // then wait for results.
   mra_ClearLastError();
   err = mra_WaitScriptDone( errBuf );
   if ( err != MRA_ERR_NO_ERR )
   {
      err = mra_GetLastError();
      switch( err )
      {
         case MRA_ERR_QUERYING_SERVER :
            // Fatal Error. Process is unable to get information from server. Make sure
            // functions are being called in the right order.  Retry.
         case MRA_ERR_INVALID_PARAMETERS :
            // Make sure parameters have the right data type.
         case MRA_ERR_INVALID_HANDLE :
            // The client handle has become invalid. Disconnect and reconnect.
         case MRA_ERR_INVALID_OPERATION :
            // A script is running in MetroPro and its results need to be obtained prior
            // to performing any other operation.
         case MRA_ERR_SERVER_ERROR :
            // Disconnect and reconnect.
         case MRA_ERR_DATA_TRANSFER_ERROR :
            // Unable to transfer data. Retry.
            fail = 1;
            break;
      }
   }
   else
   {
      // STEP 8
      mra_ClearLastError();
      err = mra_GetScriptResults( MpErrorMsg, MpOutput, errBuf );
      if ( errBuf[0] != '\0' && err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch( err )
         {
            case MRA_ERR_GETTING_RESULTS_DATA_ERR :
               // Process is unable to get results from script execution.
               // Disconnect, reconnect and retry.
            case MRA_ERR_INVALID_PARAMETERS :
               // Make sure parameters have the right data type.
            case MRA_ERR_INVALID_HANDLE :
               // The client handle has become invalid. Disconnect and reconnect.
            case MRA_ERR_NO_ACCESS_INTERFACE :
               // There is no communication channel with MetroPro. Connect first.
            case MRA_ERR_INVALID_OPERATION :
               // A script must be completed in MetroPro in order for this function to
```

```
                  // work properly. Functions mraRunScript and mraWaitScriptDone or
                  // mraIsScriptDone must have been called first in order to obtain
                  // script results.
               case MRA_ERR_NP_UNABLE_TO_FIND_SERVER :
                  // Connection has been lost.
               case MRA_ERR_PROCESS_ABORTED :
                  // Process has been aborted by user.
                  fail = 1;
                  break;
            }
         }
         else
         {
            // Script execution was successful.
            if ( MpErrorMsg[0] == '\0' )
            {
               printf("\nCommand execution was successful!");
               if ( MpOutput[0] != '\0' )
               {
                  // If output expected from script execution.
                  printf ("\nMetroPro output = %s", MpOutput);
               }
            }

            // In case that MetroPro doesn't report an error code.
            if ( MpErrorMsg[0] != '\0' )
               printf("\nReceived Error = %s", MpErrorMsg);
            if ( err > 0 )
            {
               // Script execution failed.
               if ( MpErrorMsg[0] != '\0' )
               {
                  // There were errors in script execution.
                  printf ("\nMetroPro error = %s", MpErrorMsg);
               }
            }
         }
      }
   }
}

// Step 9
// Close connection.
mra_ClearLastError();
err = mra_Close( errBuf );
if ( err != MRA_ERR_NO_ERR )
{
   err = mra_GetLastError();
   switch ( err )
   {
      case MRA_ERR_INVALID_PARAMETERS:
         // Make sure parameters have the right data type.
      case MRA_ERR_INVALID_OPERATION:
            // Connection already closed.
         case MRA_ERR_INVALID_HANDLE:
            // Unable to close connection. Invalid handle.
         case MRA_ERR_NO_ACCESS_INTERFACE:
            // There is no connection to close.
            break;
      }
   }
}  // end of “if( dll_ok )”

close:
   // STEP 10
   // Close DLL.
   FreeLibrary( mpClientLib );
   mpClientLib          = NULL;
   mra_Initialize       = NULL;
   mra_RunScript        = NULL;
   mra_WaitScriptDone   = NULL;
   mra_GetScriptResults = NULL;
   mra_Close            = NULL;
   mra_PutFile          = NULL;

return 0;
}
```

## Example 2 – Transfer, Run, and Query a MetroScript File

This C program uses MetroProRemoteAccess.dll to run a script in MetroPro remotely and query MetroPro to see if the script execution has finished.  This program is more flexible than Example 1.

```
/* runscript2.c
 * Algorithm:
 *    1. Load MetroProRemoteAccess.DLL.
 *    2. Get Process Addresses for function needed.
 *    3. Verify all functions were loaded.
 *    4. Establish connection with MetroPro.
 *    5. Transfer script file to MetroPro.
 *       ( if script file is not in MetroPro computer ).
 *    6. Send Run Script command to MetroPro.
 *    7. Query MetroPro to see if script is done.
 *    8. Get Script Results.
 *    9. Close Connection.
 *    10.Close DLL.
 */
#include "mpClient.h"
int main()
{
   int   fail        = 0,
         ok          = 0,
         scriptDone  = 0,
         dll_ok      = 0,
         err         = 0,
         nbytes      = 0;
   char  errBuf[BUFSIZ];
   char  MpErrorMsg[BUFSIZ],
         MpOutput[BUFSIZ],
         in_script[BUFSIZ],
         out_script[BUFSIZ],
         username[MAX_USERNAME_LENGTH],
         password[MAX_PASSWORD_LENGTH];
   int         messages;
   bool_t      log_operations;
   HINSTANCE   mpClientLib;
   memset(errBuf, '\0', BUFSIZ);

   // STEP 1
   // Load DLL.
   mpClientLib = LoadLibrary( "MetroProRemoteAccess.dll" );
   if (mpClientLib != NULL)
   {
      // DLL loaded
      printf("\nDLL loaded successfully.");
      // STEP 2
      // Get Process Addresses for DLL functions needed.
      mra_Initialize          = (MRAINITIALIZE) GetProcAddress( mpClientLib, "mraInitialize" );
      mra_RunScript           = (MRARUNSCRIPT)  GetProcAddress( mpClientLib, "mraRunScript" );
      mra_IsScriptDone        = (MRAISSCRIPTDONE) GetProcAddress( mpClientLib, "mraIsScriptDone");
      mra_GetScriptResults    = (MRAGETSCRIPTRESULTS) \
                                GetProcAddress( mpClientLib, "mraGetScriptResults");
      mra_Close               = (MRACLOSE) GetProcAddress( mpClientLib, "mraClose" );
      mra_PutFile             = (MRAPUTFILE) GetProcAddress( mpClientLib, "mraPutFile" );
      mra_GetLastError        = (MRAGETLASTERROR) GetProcAddress( mpClientLib, "mraGetLastError");
      mra_ClearLastError      = (MRACLEARLASTERROR) \
                                 GetProcAddress( mpClientLib, "mraClearLastError");
      mra_IsClientOnline      = (MRAISCLIENTONLINE) \
                                 GetProcAddress( mpClientLib, "mraIsClientOnline");

      // STEP 3
      // Verify that all functions were loaded.
      if( ! mra_Initialize     || ! mra_RunScript        ||
          ! mra_IsScriptDone   || ! mra_GetScriptResults ||
          ! mra_Close          || ! mra_PutFile          ||
          ! mra_GetLastError   || ! mra_ClearLastError   ||
          ! mra_IsClientOnline )
      {
         // Not all functions were loaded. Check spelling.
         dll_ok = 0;
      }
      else
      {
         // All functions were loaded successfully.
         dll_ok = 1;
      }
   }
   else
   {
      // Error loading DLL. Check file.
      printf("\nError. Unable to load MetroProRemoteAccess.dll");
   }
   // Begin Remote Control of MetroPro.
   if( dll_ok )
   {
      // STEP 4
      // Establish Connection with MetroPro.
      // Set username and password.
      strcpy(username, "my_username");
      strcpy(password, "my_pwd");
      // Turn server info messages on.
      messages = 1;
      // Turn client logging on.
      log_operations = TRUE;
      mra_ClearLastError();
      err = mra_Initialize ( errBuf, username, password, messages, log_operations );
      if ( err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch ( err )
         {
            case MRA_ERR_INVALID_PARAMETERS:
               // Make sure parameters have the right data type.
```

```
        case MRA_ERR_ALREADY_CONNECTED:
           // Already connected to MetroPro.
        case MRA_ERR_UNABLE_TO_FIND_CFG_FILE:
           // Can't find mraClient.cfg. Make sure it is in the local directory
        case MRA_ERR_INVALID_OPERATING_SYSTEM:
           // Must use Windows NT.
        case MRA_ERR_NP_UNABLE_TO_FIND_SERVER:
           // Machine with name as specified in mraClient.cfg cannot be found in network.
        case MRA_ERR_INVALID_USERNAME_OR_PASSWD:
           // User could not be authenticated.
        case MRA_ERR_NO_NP_SERVER_AVAILABLE:
           // A Named Pipe server cannot be found on machine specified in mraClient.cfg
        case MRA_ERR_NP_CONNECTION_TIMEOUT:
           // A named pipe connection timeout has occurred.
           // Named pipe server is currently not available .
        case MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT:
           // Internal error. Cannot create named pipe client.
        case MRA_ERR_UNABLE_TO_ESTABLISH_LINK:
           // Unable to perform initial software handshake. Disconnect and reconnect.
           fail = 1;
           break;
     }
  }
  else
  {
     // Connection was established successfully.
     fail = 0;
  }
  if( ! fail )
  {
     // STEP 5
     // Transfer script to Metropro.
     mra_ClearLastError();
     strcpy(in_script, "c:\\remotehost\\zygo\\measure.scr");
     strcpy(out_script, "c:\\metropro\\cur_dir\\measure.scr");
     err = mra_PutFile( in_script, out_script, errBuf );
     if ( err != MRA_ERR_NO_ERR )
     {
        err = mra_GetLastError();
        switch ( err )
        {
           case MRA_ERR_INVALID_PARAMETERS:
              // Make sure parameters have the right data type.
           case MRA_ERR_INVALID_HANDLE:
              // The client handle has become invalid. Disconnect and reconnect.
           case MRA_ERR_INVALID_OPERATION:
              // A script is running in MetroPro and its results
              // need to be obtained prior to performing any other operation.
```

```
      case MRA_ERR_NP_UNABLE_TO_FIND_SERVER:
         // Connection has been lost. Disconnect and reconnect.
      case MRA_ERR_UNABLE_TO_PROCESS_CRC:
         // Unable to process Cyclic Redundancy Check on data.
         // Verify a valid file is being transferred.
      case MRA_ERR_UNABLE_TO_FIND_FILE:
         // Unable to find file in provided pathname.
      case MRA_ERR_PROCESS_TIMEOUT:
         // Error. Disconnect and reconnect.
      case MRA_ERR_NP_UNABLE_TO_WRITE_DATA:
         // A Named Pipe error occurred. Unable to write data. Disconnect and reconnect.
      case MRA_ERR_NP_UNABLE_TO_READ_DATA:
         // A Named Pipe error occurred. Unable to read data. Disconnect and reconnect.
      case MRA_ERR_PROCESS_ABORTED:
         // Process has been aborted by local user.
      case MRA_ERR_DATA_TRANSFER_ERROR:
         // Unable to transfer all bytes in file. Retry.
      case MRA_ERR_CRC_ERROR:
         // Data has been corrupted. Unable to verify Cyclic Redundancy Check. Retry.
         fail = 1;
         break;
   }
}
else
   fail = 0;
if ( fail )
   goto close;

// STEP 6
// Run Script "c:\\metropro\\cur_dir\\measure.scr"
mra_ClearLastError();
err  = mra_RunScript( out_script, errBuf );
if ( err != MRA_ERR_NO_ERR )
{
   err = mra_GetLastError();
   switch ( err )
   {
      case MRA_ERR_INVALID_PARAMETERS :
         // Make sure parameters have the right data type.
      case MRA_ERR_INVALID_HANDLE :
         // The client handle has become invalid. Disconnect and reconnect.
      case MRA_ERR_NO_ACCESS_INTERFACE :
         // There is no communication channel with MetroPro. Connect first.
      case MRA_ERR_INVALID_OPERATION :
         // A script is running in MetroPro and its results
         // need to be obtained prior to performing any other operation.
      case MRA_ERR_NP_UNABLE_TO_FIND_SERVER :
         // Connection has been lost. Disconnect and reconnect.
         case MRA_ERR_DATA_TRANSFER_ERROR :
            // Unable to transfer data. Retry.
            fail = 1;
            break;
      }
   }
   else
      fail = 0;

   if ( fail || errBuf[0] != '\0' )
   {
      // There was an error loading and/or opening the script and/or sending
      // the runscript command
      printf("\nError = %s", errBuf);
   }
   if ( ! fail )
      {
         // STEP 7
         // If run script request did not fail and was received by MetroPro
         // successfully, then query for results.
         while ( nbytes <= 0 )
         {
            mra_ClearLastError();
            nbytes = mra_IsScriptDone( errBuf );
            if ( nbytes < 0 )
            {
               err = mra_GetLastError();
               if ( err != MRA_ERR_NO_ERR )
                  switch( err )
                  {
                     case MRA_ERR_QUERYING_SERVER :
                        // Fatal Error. Process is unable to get information from server.
                        // Make sure functions are being called in the right order. Retry.
                     case MRA_ERR_INVALID_PARAMETERS :
                        // Make sure parameters have the right data type.
                     case MRA_ERR_INVALID_HANDLE :
                        // The client handle has become invalid. Disconnect and reconnect.
                     case MRA_ERR_INVALID_OPERATION :
                        // A script is running in MetroPro and its results
                        // need to be obtained prior to performing any other operation.
                     case MRA_ERR_SERVER_ERROR :
                        // Disconnect and reconnect.
                     case MRA_ERR_DATA_TRANSFER_ERROR :
                        // Unable to transfer data. Retry.
                        fail = 1;
                        break;
                  }
            }
            // DO OTHER CLIENT SOFTWARE PROCESSING WHILE
            // WAITING FOR METROPRO TO FINISH EXECUTING
            // SCRIPT.
         }
```

```
if ( nbytes > 0 )
{
      // STEP 8
      mra_ClearLastError();
      err = mra_GetScriptResults (  MpErrorMsg, MpOutput, errBuf );
      if ( errBuf[0] != '\0' && err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch( err )
         {
            case MRA_ERR_GETTING_RESULTS_DATA_ERR :
               // Fatal Error. Process is unable to get results from script
               // execution.  Disconnect, reconnect and retry.
            case MRA_ERR_INVALID_PARAMETERS :
               // Make sure parameters have the right data type.
            case MRA_ERR_INVALID_HANDLE :
               // The client handle has become invalid. Disconnect and reconnect.
            case MRA_ERR_NO_ACCESS_INTERFACE :
               // There is no communication channel with MetroPro. Connect first.
            case MRA_ERR_INVALID_OPERATION :
               // A script must be completed in MetroPro in order for this function
               // to work properly. Functions mraRunScript and mraWaitScriptDone
               // or mraIsScriptDone must have been called first in order
               // to obtain script results.
            case MRA_ERR_NP_UNABLE_TO_FIND_SERVER :
               // Connection has been lost.
            case MRA_ERR_PROCESS_ABORTED :
               // Process has been aborted by user.
               fail = 1;
               break;
         }
      }
      else
      {
         // Script execution was successful.
         if ( MpErrorMsg[0] == '\0' )
         {
            printf("\nCommand execution was successful!");
            if ( MpOutput[0] != '\0' )
            {
               // If output expected from script execution.
               printf ("\nMetroPro output = %s", MpOutput);
            }
         }
         // In case that MetroPro doesn't report an error code.
         if ( MpErrorMsg[0] != '\0' )
            printf("\nReceived Error = %s", MpErrorMsg);
         if ( err > 0 )
         {
            // Script execution failed.
            if ( MpErrorMsg[0] != '\0' )
            {
               // There were errors in script execution.
```

```
                              printf ("\nMetroPro error = %s", MpErrorMsg);

                           }
                        }
                     }
               }
            }
      }

      // Step 9
      // Close connection.
      mra_ClearLastError();
      err = mra_Close( errBuf );
      if ( err != MRA_ERR_NO_ERR )
      {
         err = mra_GetLastError();
         switch ( err )
         {
            case MRA_ERR_INVALID_PARAMETERS:
               // Make sure parameters have the right data type.
            case MRA_ERR_INVALID_OPERATION:
               // Connection already closed.
            case MRA_ERR_INVALID_HANDLE:
               // Unable to close connection. Invalid handle.
            case MRA_ERR_NO_ACCESS_INTERFACE:
               // There is no connection to close.
               break;
         }
      }
   }

   close:
      // STEP 10
      // Close DLL.
      FreeLibrary( mpClientLib );
      mpClientLib          = NULL;
      mra_Initialize       = NULL;
      mra_RunScript        = NULL;
      mra_IsScriptDone     = NULL;
      mra_GetScriptResults = NULL;
      mra_Close            = NULL;
      mra_PutFile          = NULL;

   return 0;
}
```

## Example 3 – Transfer, Run and Query a MetroScript Command

This C program uses MetroProRemoteAccess.dll to run a single MetroScript command remotely in MetroPro and query MetroPro to establish if the command execution has finished.

```
/* runcommand.c
* Algorithm:
 *    1. Load MetroProRemoteAccess.DLL.
 *    2. Get Process Addresses for function needed.
 *    3. Verify all functions were loaded.
 *    4. Establish connection with MetroPro.
 *    5. Send Run Script command to MetroPro.
 *    6. Query MetroPro to see if script is done.
 *    7. Get Script Results.
 *    8. Close Connection
 *    9. Unload DLL.
 *
 *    Short sample: Obtain the value of the peak to valley result in Micro.app
 *    x = getval( getid ( ("surface wavefront map / results / peak valley / pv " ), "um")
 */

#include "mpClient.h"
int main()
{
   int   fail        = 0,
         ok          = 0,
         scriptDone  = 0,
         dll_ok      = 0,
         err         = 0,
         nbytes      = 0;
   char  errBuf[BUFSIZ];
   char  MpErrorMsg[BUFSIZ],
         MpOutput[BUFSIZ],
         script_cmd[BUFSIZ],
         username[MAX_USERNAME_LENGTH],
         password[MAX_PASSWORD_LENGTH];
   int      messages;
   bool_t   log_operations;
   HINSTANCE   mpClientLib;

   memset(errBuf, '\0', BUFSIZ);
```

```
// STEP 1
// Load DLL.
mpClientLib = LoadLibrary( "MetroProRemoteAccess.dll" );
if (mpClientLib != NULL)
{
   // DLL loaded.
   printf("\nDLL loaded successfully.");

   // STEP 2
   // Get Process Addresses for DLL functions needed.
   mra_Initialize       = (MRAINITIALIZE) GetProcAddress( mpClientLib, "mraInitialize" );
   mra_IsScriptDone     = (MRAISSCRIPTDONE) GetProcAddress( mpClientLib, "mraIsScriptDone");
   mra_GetScriptResults = (MRAGETSCRIPTRESULTS) \
                           GetProcAddress( mpClientLib, "mraGetScriptResults");
   mra_Close            = (MRACLOSE) GetProcAddress( mpClientLib, "mraClose" );
   mra_GetLastError     = (MRAGETLASTERROR) GetProcAddress( mpClientLib, "mraGetLastError");
   mra_ClearLastError   = (MRACLEARLASTERROR) GetProcAddress( mpClientLib, "mraClearLastError");
   mra_IsClientOnline   = (MRAISCLIENTONLINE) GetProcAddress( mpClientLib, "mraIsClientOnline");
   mra_SendMetroScriptCommand = (MRARUNSCRIPT) \
                                 GetProcAddress( mpClientLib, "mraSendMetroScriptCommand" );

   // STEP 3
   // Verify that all functions were loaded.
   if( ! mra_Initialize     || ! mra_RunScript        ||
       ! mra_IsScriptDone   || ! mra_GetScriptResults ||
       ! mra_Close          || ! mra_GetLastError     ||
       ! mra_ClearLastError || ! mra_IsClientOnline   ||
       ! mra_SendMetroScriptCommand )
   {
      // Not all functions were loaded. Check spelling.
      dll_ok = 0;
   }
   else
   {
      // All functions were loaded successfully.
      dll_ok = 1;
   }
}
else
{
   // Error loading DLL. Check file.
   printf("\nError. Unable to load MetroProRemoteAccess.dll");
}
```

```
// Begin Remote Control of MetroPro.
if( dll_ok )
{
   // STEP 4
   // Establish Connection with MetroPro. Set username and password.
   strcpy(username, "my_username");
   strcpy(password, "my_pwd");
   // Turn server info messages on.
   messages    = 1;
   // Turn client logging on.
   log_operations = TRUE;
   mra_ClearLastError();
   err = mra_Initialize ( errBuf, username, password, messages, log_operations );
   if ( err != MRA_ERR_NO_ERR )
   {
      err = mra_GetLastError();
      switch ( err )
      {
         case MRA_ERR_INVALID_PARAMETERS:
            // Make sure parameters have the right data type.
         case MRA_ERR_ALREADY_CONNECTED:
            // Already connected to MetroPro.
         case MRA_ERR_UNABLE_TO_FIND_CFG_FILE:
            // Can't find mraClient.cfg. Make sure it's in local directory
         case MRA_ERR_INVALID_OPERATING_SYSTEM:
            // Must use Windows NT.
         case MRA_ERR_NP_UNABLE_TO_FIND_SERVER:
            // Machine with name as specified in mraClient.cfg cannot be found on network.
         case MRA_ERR_INVALID_USERNAME_OR_PASSWD:
            // User could not be authenticated.
         case MRA_ERR_NO_NP_SERVER_AVAILABLE:
            // A Named Pipe server cannot be found on machine specified in mraClient.cfg
         case MRA_ERR_NP_CONNECTION_TIMEOUT:
            // A named pipe connection timeout has occurred.
            // Named pipe server is currently not available
         case MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT:
            // Internal error. Cannot create named pipe client.
         case MRA_ERR_UNABLE_TO_ESTABLISH_LINK:
            // Unable to perform initial software handshake. Disconnect and reconnect.
            fail = 1;
            break;
      }
   }
```

```
else
{
   // Connection was established successfully.
   fail = 0;
}
if( ! fail )
{
   // STEP 5
   mra_ClearLastError();
   strcpy(script_cmd, "measure");
   err = mra_SendMetroScriptCommand( script_cmd, errBuf );
   if ( err != MRA_ERR_NO_ERR )
   {
      err = mra_GetLastError();
      switch ( err )
      {
         case MRA_ERR_INVALID_PARAMETERS :
            // Make sure parameters have the right data type.
         case MRA_ERR_INVALID_HANDLE :
            // The client handle has become invalid. Disconnect and reconnect.
         case MRA_ERR_INVALID_OPERATION :
            // A script is running in MetroPro and its results
            // need to be obtained prior to performing any other operation.
         case MRA_ERR_UNABLE_TO_SEND_COMMAND :
            // General Error. Disconnect and reconnect.
         case MRA_ERR_STRING_TOO_LONG  :
            // Command string should be less than 512 characters.
         case MRA_ERR_UNABLE_TO_FIND_FILE :
            // Unable to find tmp file.
            fail = 1;
            break;
      }
   }
   else
      fail = 0;
   if ( fail || errBuf[0] != '\0' )
   {
      // There was an error loading and/or opening the script and/or sending
      // the runscript command
      printf("\nError = %s", errBuf);
   }

   if ( ! fail )
```

```
{
   // STEP 6
   // If run script request did not fail and was received by MetroPro successfully, then
   // query for results.
   while ( nbytes <= 0 )
   {
      mra_ClearLastError();
      nbytes = mra_IsScriptDone( errBuf );
      if ( nbytes < 0 )
      {
         err = mra_GetLastError();
         if ( err != MRA_ERR_NO_ERR )
            switch( err )
            {
               case MRA_ERR_QUERYING_SERVER :
                  // Fatal Error. Process is unable to get information from server.
                  // Make sure functions are being called in the right order.  Retry.
               case MRA_ERR_INVALID_PARAMETERS :
                  // Make sure parameters have the right data type.
               case MRA_ERR_INVALID_HANDLE :
                  // The client handle has become invalid. Disconnect and reconnect.
               case MRA_ERR_INVALID_OPERATION :
                  // A script is running in MetroPro and its results need to be
                  // obtained prior to performing any other operation.
               case MRA_ERR_SERVER_ERROR :
                  // Disconnect and reconnect.
               case MRA_ERR_DATA_TRANSFER_ERROR :
                  // Unable to transfer data. Retry.
                  fail = 1;
                  break;
            }
      }
      // DO OTHER CLIENT SOFTWARE PROCESSING WHILE
      // WAITING FOR METROPRO TO FINISH EXECUTING
      // SCRIPT.
   }
```

```
        if ( nbytes > 0 )
        {
           // STEP 7
           mra_ClearLastError();
           err = mra_GetScriptResults ( MpErrorMsg,
                                  MpOutput,
                                  errBuf );
           if ( errBuf[0] != '\0' && err != MRA_ERR_NO_ERR )
           {
              err = mra_GetLastError();
              switch( err )
              {
                 case MRA_ERR_GETTING_RESULTS_DATA_ERR :
                    // Fatal Error. Process is unable to get results from script execution.
                    // Disconnect, reconnect and retry.
                 case MRA_ERR_INVALID_PARAMETERS :
                    // Make sure parameters have the right data type.
                 case MRA_ERR_INVALID_HANDLE :
                    // The client handle has become invalid. Disconnect and reconnect.
                 case MRA_ERR_NO_ACCESS_INTERFACE :
                    // There is no communication channel with MetroPro. Connect first.
                 case MRA_ERR_INVALID_OPERATION :
                    // A script must be completed in MetroPro in order for this function to
                    // work properly. Functions mraRunScript and mraWaitScriptDone or
                    // mraIsScriptDone must have been called first in order to obtain
                    // script results.
                 case MRA_ERR_NP_UNABLE_TO_FIND_SERVER :
                    // Connection has been lost.
                 case MRA_ERR_PROCESS_ABORTED :
                    // Process has been aborted by user.
                    fail = 1;
                    break;
              }
           }
           else
           {
              // Script execution was successful.
              if ( MpErrorMsg[0] == '\0' )
              {
                 printf("\nCommand execution was successful!");
                 if ( MpOutput[0] != '\0' )
                 {
                     // If output expected from script execution
                     printf ("\nMetroPro output = %s", MpOutput);
                  }
               }
               // In case that MetroPro doesn't report an error code.
               if ( MpErrorMsg[0] != '\0' )
                  printf("\nReceived Error = %s", MpErrorMsg);
               if ( err > 0 )
               {
                  // Script execution failed.
                  if ( MpErrorMsg[0] != '\0' )
                  {
                     // There were errors in script execution.
                     printf ("\nMetroPro error = %s", MpErrorMsg);

                  }
               }
            }
         }
      }
   }

   // Step 8
   // Close connection.
   mra_ClearLastError();
   err = mra_Close( errBuf );
   if ( err != MRA_ERR_NO_ERR )
   {
      err = mra_GetLastError();
      switch ( err )
      {
         case MRA_ERR_INVALID_PARAMETERS:
            // Make sure parameters have the right data type.
         case MRA_ERR_INVALID_OPERATION:
            // Connection already closed.
         case MRA_ERR_INVALID_HANDLE:
            // Unable to close connection. Invalid handle.
         case MRA_ERR_NO_ACCESS_INTERFACE:
            // There is no connection to close.
            break;
      }
   }
}
```

```
   close:
      // STEP 9
      // Close DLL.
      FreeLibrary( mpClientLib );
      mpClientLib          = NULL;
      mra_Initialize       = NULL;
      mra_RunScript        = NULL;
      mra_IsScriptDone     = NULL;
      mra_GetScriptResults = NULL;
      mra_Close            = NULL;

   return 0;
}
```

Appendix A

# Error Codes

This appendix lists the error codes and descriptions produced by the *MetroProRemoteAccess.dll*.

| ***Code*** | ***Name/Description*** |
|---|---|
| **0** | MRA_ERR_NO_ERR<br>The operation completed successfully. |
| **1** | MRA_ERR_INVALID_PARAMETERS<br>Invalid parameters have been passed to the function. Look at function description to check for parameter data types. |
| **2** | MRA_ERR_PROCESS_ABORTED<br>The client application was previously aborted. If connected, you must close the connection and reconnect in order to control MetroPro remotely. |
| **3** | MRA_ERR_INVALID_HANDLE<br>The handle to the communications device is in an invalid state. Either there is no connection or a fatal error has occurred while connected. If this error occurs while transferring data or running a script, disconnect and reconnect to MetroPro. Do not attempt to do anything else. |
| **4** | MRA_ERR_NO_ACCESS_INTERFACE<br>There is no connection with MetroPro at the present time. If this error occurred while connected, disconnect and reconnect. The client application must have either a Named Pipe Interface to control MetroPro. |
| **5** | MRA_ERR_SERIAL_PACKET_TIMEOUT (*obsolete*)<br>The maximum amount of time allowed in mraClient.cfg for receiving data has elapsed. This error occurs if using a serial communications interface and no incoming data is detected while expected. Make sure null the modem cable is properly connected. Retry operation. If it fails again, disconnect and reconnect to MetroPro. |
| **6** | MRA_ERR_NP_PACKET_TIMEOUT<br>The maximum amount of time allowed in mraClient.cfg for receiving data has elapsed. This error occurs if using TCP/IP (named pipes) for controlling MetroPro and no incoming data is detected while expected. Make sure the data cable is properly connected and that the server is running properly. Use function mraRunLoopBackTest to verify that data can be transmitted back and forth. Retry the operation. If it fails again, disconnect and reconnect to MetroPro. |
| **7** | MRA_ERR_SERIAL_READ_TIMEOUT (*obsolete*)<br>The maximum amount of time allowed in mraClient.cfg for reading data has elapsed. This error occurs if using a serial communications interface and unable to read all the data expected. Make sure null modem cable is connected properly. Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. |

| *Code* | *Name/Description* |
|---|---|
| **8** | MRA_ERR_NP_READ_TIMEOUT<br>The maximum amount of time allowed in mraClient.cfg for reading data has elapsed. This error occurs if using TCP/IP (named pipes) interface and unable to read all the data expected. Make sure data cable is connected properly. Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. |
| **9** | MRA_ERR_SERIAL_CONNECTION_TIMEOUT (*obsolete*)<br>The maximum amount of time allowed in mraClient.cfg for connecting to MetroPro using a null modem cable has elapsed. Make sure cable is connected properly and that the Remote Access Server in MetroPro is running and expecting a client connection. |
| **10** | MRA_ERR_NP_CONNECTION_TIMEOUT<br>The maximum amount of time allowed in mraClient.cfg for connecting to MetroPro using TCP/IP (named pipes) has elapsed. Make sure data cable is connected properly and that the Remote Access Server in MetroPro is running and expecting a client connection. |
| **11** | MRA_ERR_NP_UNABLE_TO_FIND_SERVER<br>Unable to find the server (MetroPro) computer as indicated in mraClient.cfg. Make sure that the name of the computer where MetroPro is located has not been misspelled and that the computer where MetroPro is located can be accessed by other computers from the network. |
| **12** | MRA_ERR_NP_UNABLE_TO_READ_DATA<br>Unable to read data using TCP/IP. Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. If this fails, disconnect and reconnect to MetroPro. |
| **13** | MRA_ERR_SERIAL_UNABLE_TO_READ_DATA (*obsolete*)<br>Unable to read data using serial communications. Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. If this fails, disconnect and reconnect to MetroPro. |
| **14** | MRA_ERR_SERIAL_UNABLE_TO_WRITE_DATA (*obsolete*)<br>Client is unable to send data to MetroPro using serial communications. Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. Check cable. Disconnect and reconnect to MetroPro. |
| **15** | MRA_ERR_NP_UNABLE_TO_WRITE_DATA<br>Client is unable to send data to MetroPro using TCP/IP (named pipes). Use function mraRunLoopBackTest to verify that data can be transferred between MetroPro and client process. Make sure data cable is connected properly. If this fails, disconnect and reconnect to MetroPro. |
| **16** | MRA_ERR_UNABLE_TO_FIND_CFG_FILE<br>Unable to find mraClient.cfg file. File must be located in the same directory with MetroProRemoteAccess.dll and the client application. This configuration file is required to establish a communications interface with MetroPro. |

| *Code* | *Name/Description* |
|---|---|
| **17** | MRA_ERR_INVALID_OPERATING_SYSTEM<br>MetroPro Remote Access only works under Windows NT 4.0 SP1 (or higher) or Windows 2000. Attempts to run it under Windows 95, 98, ME, XP or any other operating systems will fail. |
| **18** | MRA_ERR_UNABLE_TO_ESTABLISH_LINK<br>Unable to perform initial message exchange with MetroPro after opening communications device. Use mraRunLoopbackTest to verify that data can be transferred to MetroPro. If this fails, do not attempt to do anything else. Disconnect and reconnect to MetroPro. |
| **19** | MRA_ERR_NO_NP_SERVER_AVAILABLE<br>When using TCP/IP to connect to MetroPro, a named pipe server (in MetroPro) is detected but not available for connections. This error occurs when a connection is attempted to MetroPro, but there is another client already connected to the server. Try again later when MetroPro is available. |
| **20** | MRA_ERR_UNABLE_TO_CREATE_NP_CLIENT<br>This error occurs when an internal fatal error does not allow MetroProRemoteAccess.dll to create a named pipe client even though a named pipe server is available when trying to establish a connection through TCP/IP. |
| **21** | MRA_ERR_SERVER_ERROR<br>This error is received from MetroPro when a fatal error occurs in the server processing of client commands. Do not attempt to perform any other commands. You must disconnect and reconnect to MetroPro. |
| **22** | MRA_ERR_DATA_TRANSFER_ERROR<br>Process is unable to transfer data. This error occurs when not all of the bytes expected are read or written to the communications device. Retry the transfer. |
| **23** | MRA_ERR_UNABLE_TO_FIND_FILE<br>Process is unable to find wanted file for binary data transfer. Check the file path and name and verify that the file does exist. |
| **24** | MRA_ERR_PROCESS_TIMEOUT<br>Process has timed out during the data transfer operation. Try again. |
| **25** | MRA_ERR_UNABLE_TO_PROCESS_CRC<br>Process cannot obtain a CRC (Cyclic Redundancy Check) value for data transfer. Verify file exists in path. |
| **26** | MRA_ERR_CRC_ERROR<br>Process cannot verify CRC (Cyclic Redundancy Check) value for incoming vs. outgoing data. Use function mraRunLoopBackTest to test the data transferring process. Retry transfer. |
| **27** | MRA_ERR_UNABLE_TO_WRITE_TO_DISK<br>The disk is full. Free some space to write received data. |
| **28** | MRA_ERR_ALREADY_CONNECTED<br>A connection is already opened. Do not attempt to connect until after connection has been closed or lost. |

| *Code* | *Name/Description* |
|---|---|
| **29** | MRA_ERR_LOOPBACKTEST_ERROR<br>Unable to perform echo test for data transfers. Do not attempt to perform any other commands. If connected, disconnect and reconnect to MetroPro. |
| **30** | MRA_ERR_UNABLE_TO_ECHO_DATA<br>Process is unable to echo data while performing communications test. See code 29. |
| **31** | MRA_ERR_UNABLE_TO_SEND_COMMAND<br>This error occurs when trying to send a single MetroScript command of length zero. |
| **32** | MRA_ERR_STRING_TOO_LONG<br>This error occurs when trying to send a single MetroScript command of length larger than 512. |
| **33** | MRA_ERR_UNABLE_TO_STOP_SERVER<br>This error occurs when the command to stop MetroPro from running a script cannot be transferred. Use function mraRunLoopBackTest to see if data can be transferred. |
| **34** | MRA_ERR_UNABLE_TO_KILL_SERVER<br>This error occurs when the command to terminate MetroPro's Remote Access Server cannot be transferred. Use function mraRunLoopBackTest to see if data can be transferred. |
| **35** | MRA_ERR_INVALID_OPERATION<br>This error occurs when trying to perform an operation while a script is running in MetroPro. You must make a call to mraGetScriptResults to complete the process of running a script. Do not attempt to put or get a file, run a single command or script until after getting the results of the script execution. |
| **36** | MRA_ERR_INVALID_USERNAME_OR_PASSWD<br>This error occurs when the username and/or password passed to the mraInitialize function are invalid. |
| **37** | MRA_ERR_INVALID_SERVER_INDEX<br>This error occurs when the server tag in mraClient.cfg is not within the valid range of allowed server indexes. The valid range is from 1 to 32. |
| **38** | MRA_ERR_UNABLE_TO_FIND_SERVER_TAG<br>This error occurs when there is no server tag in mraClient.cfg. A server tag ([server n]) is required for establishing the right link to MetroPro. |
| **39** | MRA_ERR_UNABLE_TO_LOAD_SCRIPT<br>This error occurs when a script cannot be found on MetroPro's computer for execution. Transfer the script first using function mraPutFile and retry. |
| **40** | MRA_ERR_UNABLE_TO_DELETE_FILE<br>This error occurs when the file being deleted does not exist or is inaccessible. |
| **41** | MRA_ERR_UNABLE_TO_RESUME_POLLING<br>This error occurs when attempting to resume connection polling between the remote client and MetroPro. You must disconnect and reconnect in order to have predictable client/server model behavior. |
| **42** | MRA_ERR_UNABLE_TO_SUSPEND_POLLING<br>This error occurs when attempting to suspend connection polling between the remote client and MetroPro. Try again. A thread collision may have occurred. |

| *Code* | *Name/Description* |
|---|---|
| **43** | MRA_ERR_UNABLE_TO_CREATE_SERIAL_CLIENT (*obsolete*)<br>This error occurs when function mraInitialize is unable to create a serial client. Verify that right port is indicated in mraClient.cfg. Verify that serial port in MetroPro is available and that the Remote Access Server is running. |
| **44** | MRA_ERR_SERVER_SUSPENDED<br>This error occurs when the server is suspended. Server rejects all incoming requests. Resume the server in MetroPro by pressing F11 and try again. |
| **45** | MRA_ERR_NACK_RECEIVED<br>This error is returned when an error in communications has happened. Try again. |
| **46** | MRA_ERR_UNABLE_TO_TERMINATE_METROPRO<br>This error occurs when the call to mraTerminateMetroPro fails. This may occur because there is no connection or because MetroPro process is unable to end. Verify there is a valid connection by calling mraIsClientOnline and that the server is not suspended by calling mraGetServerStatus. |
| **47** | MRA_ERR_INCOMPATIBLE_SCRIPT<br>This error occurs when the server receives a command to run a script that requires user interaction in MetroPro. This error can be bypassed by modifying the server configuration file by adding an optional script parameter to it. For more information read the configuration settings section of Getting Started in this manual. |

| *Code* | *Name/Description* |
|---|---|
| **-1** | MRA_ERR_GETTING_RESULTS_DATA_ERR<br>Process is unable to get results from script execution. |
| **-2** | MRA_ERR_QUERYING_SERVER<br>Process is unable to get information from server. Make sure functions are being called in the right order. Retry. |

# Glossary

Appendix B

*Application*

A computer program running on the Client side of the Remote Access.

*Asynchronous*

See *Non_Blocking* definition.

*Baudrate*

Transmission rate for a serial port.  Valid baud rates are: 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, and 256000.

*Blocking*

An input / output mode for communications between two processes, which blocks the processes until it has finished reading or writing data or a timeout occurs.  If performing a read operation, the process will wail until it has finished reading all the bytes needed before doing anything else.  If performing a write operation, the process will wait until it has finished writing all of the bytes in the buffer.

*Buffersize*

Maximum amount of data, in bytes, that is transferred between two processes in a single read or write operation.

*Client*

An application, process or program, other than MetroPro, that intends to control MetroPro remotely.

*CRC*

*Cyclic Redundancy Check* (CRC) A common technique for detecting errors in data transmission.  In CRC error checking, the sending device calculates a number based on the data transmitted.  The receiving device repeats the same calculation after transmission.  If both devices obtain the same result, it is assumed the transmission was error-free.  The procedure is known as a redundancy check because each transmission includes not only data but also the additional redundant values for error checking. (1)

*DLL*

A Dynamic Link Library, which contains functions that can be called by a process for a specific purpose.

*Interface*

Method of communication between two computers, processes, or applications.

(1) Definition found in  Microsoft Developer Network version 6.0

*IO_Mode*

Input / Output mode for communications between two or more processes.

*LAN*

Local Area Network. Used by named pipes to transfer data.

*MetroPro*

Copyrighted software provided by Zygo Corporation to handle Zygo’s metrology and interferometry instruments.

*mraClient.cfg*
*mraServer.cfg*

Configuration files provided by Zygo Corporation that contains all the necessary settings to establish a remote communication with MetroPro.

*MetroProRemoteAccess.dll*

Dynamic Link Library provided by ZYGO that contains all of the necessary functions to control MetroPro remotely.

*MetroScript*

Programming language used by MetroPro that can be used to control ZYGO’s metrology instruments.

*Named Pipe Interface*

A communications interface that uses named pipes to transfer data between two or more processes located remotely or in the same computer through a Local Area network.

*Named Pipe*

A communications object used for transferring data between two processes. Named Pipes use the underlying Local Area Network and its protocols to transfer data.

*Named Pipe Client*

Process using *MetroProRemoteAccess.dll* to control MetroPro Remotely.

*Named Pipe Server*

MetroPro.

*Non_Blocking*

An Input/Output mode for communications between two processes. Read and Write operations do not wait for completion. They return immediately. Whether the data was transferred successfully, can be checked later. This mode gives more flexibility to integrating *MetroProRemoteAccess.dll* with other software.

(1) Definition found in Microsoft Developer Network version 6.0

*Non_Overlapped*

See *Blocking* definition.

*Overlapped*

See *Non_Blocking* definition.

*Process*

A computer program.

*Script*

A text file of extension ".scr", which contains MetroScript commands.

*Serial Communications Interface*

A communications interface that uses serial ports (RS-232) to transfer data between two or more processes.

*Serial Port*

An input/output location (channel) that sends and receives data to and from a computer's central processing unit or a communications device one bit at a time. Serial ports are used for serial data communication and as interfaces to peripheral devices, such as mice and printers. (1)

*Server*

MetroPro software provided by Zygo Corporation.

*SIO*

Serial Input Output

*Synchronous*

See *Blocking* definition.

*Timeout*

Maximum amount of time it can pass before a function fails.

(1) Definition found in Microsoft Developer Network version 6.0