Remote Automation COM Server v. 13.5.4

User Guide and Reference

Probe Software, Inc
885 Crest Dr
Eugene, OR, 97405
1-541-343-3400

Contents

Index

Introduction To Remote

Description

Remote is an Active X executable server that may be utilized for automating an electron microprobe from any application client that supports Active X OLE Automation.

Requirements

Applications that meet this criteria include most Microsoft products such as Word, Excel, Access and Explorer and development environments such as Visual Basic 6.0 and higher and C++.

Note that you must click the "Enable Macros" button when opening any Excel spreadsheets that utilize Visual Basic for Applications macros.

Note on Excel spreadsheet versions:

.XLS Excel 2003 (version 11) (Office 2003) and earlier

.XLSX Excel 2007 (version 12) (Office 2007) and later (NOTE NO MACRO SUPPORT)

.XLSM Excel 2007 (version 12) (Office 2007) and later (MACRO SUPPORT)

Installation

Run the Remote.msi installer from the distribution to install the Remote Automation Interface to your computer. Once properly installed you can start programming to the Remote Automation Interface.

To complete the installation of the Remote active-X server you *must* run the TestRemote.exe application the *first time* using the "as admin" right click option. This is necessary to properly register the class objects in the system registry.

Note also that the JEOL 8230/8530/iSP100/iHP200F instruments require that the JEOL EIKS application files be copied from the Probe for EPMA application folder to the C:\Program Files (x86)\Common Files\Probe Software folder.

These JEOL EIKS files are listed here:

jeoleiks.dll

JeolEIKs.ini	
mfc100.dll	(used only by JEOL eiksJSample.exe)
msvcr100.dll	(used only by JEOL eiksJSample.exe)
eiksJSample.exe	(JEOL EIKS test application)

The Remote Automation application files are installed to the C:\Program Files (x86)\Probe Software\Remote Automation folder. The files installed are documented here:

Deadtime.xls	Example Excel macro for calculating deadtime
Remote.pdf	PDF document with Remote interface reference
TestDeadtime.xls	Example Excel macro for measuring deadtime
TestRemote.bas	Visual Basic (VB5) test program source file
TestRemote.exe	Visual Basic (VB5) test program executable
TestRemote.frm	Visual Basic (VB5) test program source file
TestRemote.frx	Visual Basic (VB5) test program source file
TestRemote.vbp	Visual Basic (VB5) test program source file
TestRemote.vbw	Visual Basic (VB5) test program source file
TestRemote.xls	Example Excel macro for testing installation
TestReproduce.xls	Example Excel macro for spectrometer reproducibility
TestStability.xls	Example Excel macro for spectrometer stability

Note that the Remote Automation Interface (Remote.exe, Remote.hlp and Remote.ini) are always installed to the Windows\System32 (32 bit OS) or the Windows\SysWOW64 (64 bit OS) folder so that the Remote Automation Interface is available to all applications. However, the Remote Automation Interface utilizes your existing Probe for EPMA configuration files to initialize properly.

Therefore if your Probe for EPMA software is installed to a folder other than the default (C:\Program Files (x86)\Probe Software\Probe for EPMA\") then you must edit the Remote.ini file(in the C:\Windows\System32 (32 bit OS) or C:\Windows\SysWOW64 (64 bit OS) folder) for the correct path to Probe for EPMA

Comment out lines that you want the program to ignore using a semi-colon. For example, in the following sample REMOTE.INI file the first line (ProgramPath keyword) is ignored and the second line is actually used by the remote interface to locate the Probe for EPMA configuration files.

[Software]

;ProgramPath="C:\Program Files (x86)\Probe Software\Probe for EPMA\"

ProgramPath="D:\Program Files (x86)\Probe Software\Probe for EPMA\"

Update

Simply download and run the latest Remote.msi installer to update Remote.

Excel Troubleshooting

When accessing the Remote Automation Interface from Excel you may receive the error message “Excel does not recognize the RealTime object”.

To correct this error, make sure the Visual Basic toolbar is visible in Excel by clicking the View | Toolbars | Visual Basic menu, then click the Visual Basic Editor icon in the Visual Basic toolbar and then simply go to the Visual Basic Tools menu

and click on the References menu (this may vary with the Excel version) and scroll down until you see “Remote Automation Interface”. Check this box and click OK.

You should be able to run the Excel macros now.

Getting Started

General Coding Suggestions

Option Explicit

Always utilize the statement "Option Explicit" for each code module. The statement will require that all variables be explicitly declared. This greatly reduces errors due to typos.

Variable Types

Variables in Visual Basic and Visual Basic for Applications are declared as follows:

Type	Length	Identifier
Boolean	(one byte, 0 or –1)	
Byte	(one byte, 0-255)	
Integer	(two bytes)	%
Long	(four bytes)	&
Single	(four bytes)	!
Double	(eight bytes)	#
String	(up to 64K)	$

Error Handlers

Always handle error returned to your client application from the Remote Automation Interface by creating an explicit error handler. This is easily done using the "On Error GoTo ErrHandler: " statement.

Declare the Remote Automation Object

To begin, use the statement:

```
Dim Remote as Realtime
```

To declare your Remote Automation object. To load the Remote Automation object use the statement:

```
Set Remote as Realtime
```

To unload your Remote Automation object use the statement:

```
Set Remote = Nothing
```

That is all. All realtime interface initialization and de-initialization is automatically handled by the Remote Automation object using the current Probe for EPMA configuration as specified by the .INI and .DAT files.

Note: Always remember to preface each property or method by the declared object name. For example if the object is declared as above (Remote), then all properties and methods begin with the qualifier "Remote."

Procedure Calls Versus Function Calls

Note that Remote Automation class come in two types, first procedure calls which return a void argument (and usually have a parameter to return values), and function calls which return the value from the function.

Note that all parameters are passed by reference (memory address). This is the default for Visual Basic (VB) and Visual Basic for Applications (VBA).

For example, here is a procedure call:

```
Public Sub RemoteGetFaradayState(tBeamOnFlag As Boolean)
```

Where the procedure returns no value (void), but returns the state of the faraday cup as a parameter passed by reference.

Here is a function call:

```
Public Function RemoteGetConditionKilovolts() As Single
```

Where no parameters are passed, but the kilovolts value is returned by the function call. See the Detailed Description of Properties for more information.

Remote Automation Properties

List of Properties

Public Property Get RemoteInterfaceType() As Integer
Public Property Get RemoteImageInterfaceType() As Integer
Public Property Get RemoteJEOLEOSInterfaceType() As Long
Public Property Get RemoteImageInterfaceImageIxIy() As Single

Public Property Get RemoteNumberofTunableSpecs() As Integer
Public Property Get RemoteNumberofStageMotors() As Integer

Public Property Get RemoteMotHiLimits(motor As Integer) As Single
Public Property Get RemoteMotLoLimits(motor As Integer) As Single

Public Property Get RemoteNumberofCrystals(scal As Integer) As Integer
Public Property Get RemoteCrystalNames(n As Integer, scal As Integer) As String
Public Property Get RemoteCrystalFlipTypes(scal As Integer) As Integer
Public Property Get RemoteCrystalFlipPositions(scal As Integer) As Single
Public Property Get RemoteAutoFocusPresent as Boolean
Public Property Get RemoteMoveAllStageMotorsHardwarePresent as Boolean

Public Property Get RemoteCalculatePosition(method As Integer, mode As Integer, motor As Integer, xtal As String, syme As String, symx As String, order As Integer) As Single

Public Property Get RemoteDefaultROMPeakingType() As Integer
Public Property Let RemoteSetDefaultROMPeakingType(ptype As Integer)

Public Property Get RemoteROMPeakingParabolicThresholdFraction() As Single
Public Property Get RemoteROMPeakingMaximaThresholdFraction() As Single
Public Property Get RemoteROMPeakingGaussianThresholdFraction() As Single
Public Property Let RemoteSetROMPeakingParabolicThresholdFraction(thresh As Single)
Public Property Let RemoteSetROMPeakingMaximaThresholdFraction(thresh As Single)
Public Property Let RemoteSetROMPeakingGaussianThresholdFraction(thresh As Single)

Note that the spectrometer motors are numbered from 1 to RemoteNumberofTunableSpecs() while the stage motors are designated as follows:

Xmotor% = RemoteNumberofTunableSpecs() + 1
Ymotor% = RemoteNumberofTunableSpecs() + 2
Zmotor% = RemoteNumberofTunableSpecs() + 3
Wmotor% = RemoteNumberofTunableSpecs() + 4 (if present)

Detailed Description of Properties

Public Property Get RemoteInterfaceType() As Integer
Returns (read-only) the instrument interface type. 0=Demo, 1=Unused, 2=JEOL 8900/8200/8500/8x30, 3=Unused, 4=Unused, 5=SX100, 6=Axioscope

Usage:
```
Dim tInterfaceType as Integer

tInterfaceType% = RemoteInterfaceType%
```

Public Property Get RemoteImageInterfaceType() As Integer
Returns (read-only) the image interface type. 0=Demo, 1=Unused, 2=Unused, 3=Unused, 4=JEOL, 5=SX100, 6=SX100 Video, 7=Unused, 8=Unused, 9=Bruker, 10=Thermo

Usage:
```
Dim tImageInterfaceType as Integer

tImageInterfaceType% = RemoteImageInterfaceType%
```

Public Property Get RemoteJEOLEOSInterfaceType() As Long
Returns (read-only) the JEOL EOSINterfaceType. 1 = 8200/8500, 2 = 8900, 3 = 8230/8530

Usage:
```
Dim tJEOLEOSInterfaceType& as Integer

tJEOLEOSInterfaceType& = RemoteJEOLEOSInterfaceType&
```

Public Property Get RemoteImageInterfaceImageIxIy() As Single
Returns (read-only) the imaging aspect ratio. Returns 1.0 for square aspect ratio or 1.333 for 4:3 aspect ratio.

Usage:
```
Dim tImageInterfaceImageIxIy as Single

tImageInterfaceImageIxIy! =
RemoteImageInterfaceImageIxIy!
```

Public Property Get RemoteNumberofTunableSpecs() As Integer
Returns (read-only) the number of tunable spectrometers in the microprobe. Spectrometers 1 to RemoteNumberofTunableSpecs are tunable spectrometers.

Usage:
```
Dim maxtunable as Integer

maxtunable% = Remote.RemoteNumberofTunableSpecs%
```

Public Property Get RemoteNumberofStageMotors() As Integer
Returns (read-only) the number of stage motors in the microprobe. Stage motors are numbered from RemoteNumberofTunableSpecs + 1 to RemoteNumberofTunableSpecs + RemoteNumberofStageMotors.

Usage:

```
Dim maxstage as Integer

maxstage% = Remote.RemoteNumberofStageMotors%
```

Public Property Get RemoteMotHiLimits(motor As Integer) As Single
Returns (read-only) the motor axis high limits in spectrometer or stage units for the specified motor. This information is obtained from the MOTORS.DAT file. Again, spectrometer motors are numbered 1 to RemoteNumberofTunableSpecs, followed by the stage motors (if any).

Usage:

```
Dim hilimits() as Single
ReDim hilimits(1 to maxtunable% + maxstage%) as Single

motor% = Val(FormMAIN.TextMotor.Text)
hilimits!(motor%) = Remote.RemoteMotHiLimits!(motor%)
```

Public Property Get RemoteMotLoLimits(motor As Integer) As Single
Returns (read-only) the motor axis low limits in spectrometer or stage units for the specified motor. This information is obtained from the MOTORS.DAT file. Again, spectrometer motors are numbered 1 to RemoteNumberofTunableSpecs, followed by the stage motors (if any).

Usage:

```
Dim lolimits() as Single
ReDim lolimits(1 to maxtunable% + maxstage%) as Single

motor% = Val(FormMAIN.TextMotor.Text)
lolimits!(motor%) = Remote.RemoteMotLoLimits!(motor%)
```

Public Property Get RemoteNumberofCrystals(scal As Integer) As Integer

Returns (read-only) the number of crystals on each scaler (usually RemoteNumberofTunableSpecs%) and is generally a number between 1 and 6.

Usage:

```
Dim numberofcrystals as integer

scal% = Val(FormMAIN.TextScal.Text)
numberofcrystals% = Remote.RemoteNumberofCrystals%(scal%)
```

Public Property Get RemoteCrystalNames(n As Integer, scal As Integer) As String
Returns (read-only) the crystal name for the specified crystal index and scaler number.

Usage:

```
Dim n as integer
Dim scal as integer
ReDim crystalnames$(1 to maxtunable%) as String

n% = Val(FormMAIN.TextIndex.Text)
scal% = Val(FormMAIN.TextScal.Text)
crystalname$ = Remote.RemoteCrystalNames(n%, scal%)
```

Public Property Get RemoteCrystalFlipTypes(scal As Integer) As Integer

Returns (read-only) the crystal flip flag for the spectrometer (normally applies only to scanning spectrometers). The values are as follows:

Usage:

```
Dim crystalfliptype as integer

scal% = Val(FormMAIN.TextScal.Text)
crystalfliptype% = Remote.RemoteCrystalFlipTypes%(scal%)
```

Public Property Get RemoteCrystalFlipPositions(scal As Integer) As Single

Returns (read-only) the crystal flip position (if it applies) for the given spectrometer (again normally applies only to scanning spectrometers). The spectrometer must be in the crystal flip position or range if required before the RemoteChangeCrystal procedure is called.

Usage:

```
Dim crystalflipposition as single

scal% = Val(FormMAIN.TextScal.Text)
crystalflipposition% = Remote.RemoteCrystalFlipPositions!(scal%)
```

Public Property Get RemoteAutoFocusPresent as Boolean

Returns (read-only) the auto focus present flag. It returns 0 (False) is not present and a –1 (True) if present.

Usage:

```
Dim tautofocuspresent as Boolean

tautofocuspresent = Remote.RemoteAutoFocusPresent
```

Public Property Get RemoteMoveAllStageMotorsHardwarePresent as Boolean
This property returns the current value of the MoveAllStageMotorsHardwarePresent Flag from the PROBEWIN.INI file. This flag indicates that the stage must be moved X, Y and Z at the same time. For example the Jeol 8900 and 8200 instruments.

Before moving the stage, check this flag and if it is set then you should always call the RealTimeMoveStageMotors procedure to move the stage. To write code that will run properly on any instrument use the following code as an example:

```
If MoveAllStageMotorsHardwarePresent And motor% = XMotor% Then
Remote.RemoteMoveStageMotors(Int(1), pos!(XMotor%), pos!(YMotor%),
pos!(ZMotor%), pos!(WMotor%))
If ierror Then Exit Sub
ElseIf Not MoveAllStageMotorsHardwarePresent Or
(MoveAllStageMotorsHardwarePresent And motor% < XMotor%) Then
Remote.RemoteMoveMotor(motor%, pos!(motor%))
If ierror Then Exit Sub
End If
```

Returns (read-only) the move all stage hardware present flag. It returns 0 (False) is not present and a –1 (True) if present.

Usage:

```
Dim tmoveallstagehardwarepresent as Boolean

tmoveallstagehardwarepresent =
Remote.RemotemoveallstagehardwarePresent
```

See also the RemoteMoveStageMotors method in the next section.

Public Property Get RemoteCalculatePosition(method As Integer, mode As Integer, motor As Integer, xtal As String, syme As String, symx As String, order As Integer) As Single

Returns spectrometer position based on spectrometer, crystal, element, xray, and order.

method = 0 calculate theoretical position
method = 1 calculate actual position based on multiple peak .CAL files

mode = 0 return on peak position
mode = 1 return hi-off peak position
mode = 2 return lo-off peak position
mode = 3 return hi-wavescan position
mode = 4 return lo-wavescan position
mode = 5 return hi-peakscan position
mode = 6 return lo-peakscan position
mode = 7 return hi-quickscan position
mode = 8 return lo-quickscan position
mode = 9 return peaking start size
mode = 10 return peaking stop size

Usage:

```
Dim method as integer, mode as integer, motor as integer
Dim xtal as string, syme as string, symx as string
Dim order as integer
Dim pos as single

method% = 1 ‘ calculate theoretical position
mode% = 0   ‘ calculate on-peak position
motor% = 2  ‘ spectrometer 2
xtal$ = “LIF”
syme$ = “Ti”
symx$ = “Ka”
order% = 1  ‘ first order diffraction

pos! = Remote.RemoteCalculatePosition(method%, mode%, motor%,
xtal$, syme$, symx$, order%)
```

Public Property Get RemoteDefaultROMPeakingType() As Integer
Returns the current ROM peaking method
1=Parabolic
2 = Maxima
3 = Gaussian

Usage:

```
Dim ptype as Integer
ptype% = Remote.RemoteDefaultROMPeakingType
```

Public Property Let RemoteSetDefaultROMPeakingType(ptype As Integer)
Sets the current ROM peaking method
1=Parabolic
2 = Maxima
3 = Gaussian

Usage:
```
Remote.RemoteSetDefaultROMPeakingType = 2 ‘ set to
maxima method
```

Public Property Get RemoteROMPeakingParabolicThresholdFraction() As Single
Returns the current ROM peaking parabolic threshold (0.1 to 0.9)

Usage:
```
Dim thresh as Single
thresh! =
Remote.RemoteROMPeakingParabolicThresholdFraction
```

Public Property Get RemoteROMPeakingMaximaThresholdFraction() As Single
Returns the current ROM peaking maxima threshold (0.1 to 0.9)

Usage:
```
Dim thresh as Single
thresh! = Remote.RemoteROMPeakingMaximaThresholdFraction
```

Public Property Get RemoteROMPeakingGaussianThresholdFraction() As Single
Returns the current ROM peaking gaussian threshold (0.1 to 0.9)

Usage:
```
Dim thresh as Single
thresh!
=Remote.RemoteROMPeakingGaussianThresholdFraction
```

Public Property Let RemoteSetROMPeakingParabolicThresholdFraction(thresh As Single)
Sets the current ROM peaking parabolic threshold (0.1 to 0.9)

Usage:
`Remote.Remote`ROMPeakingParabolicThresholdFraction = `0.6`

Public Property Let RemoteSetROMPeakingMaximaThresholdFraction(thresh As Single)
Sets the current ROM peaking maxima threshold (0.1 to 0.9)

Usage:
```
Remote.RemoteROMPeakingMaximaThresholdFraction = 0.6
```

Public Property Let RemoteSetROMPeakingGaussianThresholdFraction(thresh As Single)
Sets the current ROM peaking gaussian threshold (0.1 to 0.9)

Usage:

```
Remote.RemoteROMPeakingGaussianThresholdFraction = 0.6
```

Remote Automation Methods

List of Methods

Motor Methods
Public Sub RemoteMoveMotor(motor As Integer, pos As Single)
Public Sub RemoteMoveStageMotors(mode as integer, method as integer, xpos As Single, ypos As Single, zpos As Single, wpos As Single)
Public Function RemoteGetMotorStatus(motor As Integer) As Integer
Public Function RemoteGetMotorPosition(motor As Integer) As Single
Public Sub RemoteStopAllMotors()
Public Sub RemoteFreeAllMotors()

Public Sub RemoteFlipCrystalMoveMotor(motor As Integer, crystal As String, pos As Single)

Crystal Methods
Public Sub RemoteChangeCrystal(motor As Integer, crystal As String)
Public Function RemoteGetCrystalPosition(motor As Integer) As String
Public Function RemoteGetCrystalStatus(motor As Integer) As Integer

Faraday Methods
Public Sub RemoteFaraday(mode As Integer)

Public Function RemoteGetAbsorbed(counttime As Single) As Single
Public Function RemoteGetFaraday(counttime As Single) As Single

Column Methods
Public Function RemoteGetConditionKilovolts() As Single
Public Function RemoteGetConditionBeamCurrent() As Single
Public Function RemoteGetConditionBeamSize() As Single

Public Sub RemoteSetConditions(takeoff As Single, kilovolts As Single, beamcurrent As Single, beamsize As Single, columnmethod as integer, columncondition as string)

Public Sub RemoteSetColumnParameters(method As Integer, CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Long, ObjectiveFine As Long, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

Public Sub RealTimeGetColumnParameters(CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Long, ObjectiveFine As Long, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

Count Methods
Public Sub RemoteStartCounts(scal As Integer, counttime As Single, maxcounts As Long)
Public Function RemoteGetCountStatus(scal As Integer) As Integer
Public Function RemoteGetCountCount(scal As Integer) As Single
Public Sub RemoteStopAllCounters()

PHA Methods
Public Function RemoteGetPHAParameterBaseline(iscal As Integer) As Single
Public Function RemoteGetPHAParameterWindow(iscal As Integer) As Single
Public Function RemoteGetPHAParameterGain(iscal As Integer) As Single
Public Function RemoteGetPHAParameterBias(iscal As Integer) As Single
Public Function RemoteGetPHAParameterInteDiffMode(iscal As Integer) As Integer
Public Function RemoteGetPHAParameterDeadtime(iscal As Integer) As Single

Public Sub RemoteSetPHAParameters(iscal As Integer, baseline As Single, window As Single, gain As Single, bias As Single, intediffmode As Integer, deadtime As Single)

Public Function RemoteGetPHADistribution(mode As Integer, iscal As Integer, npts As Integer, counttime As Single, startvolts As Single, stopvolts As Single, xdata() As Single, ydata() As Single)

Miscellaneous Methods
Public Sub RemoteAutoFocus()
Public Sub RemoteBacklash(motor As Integer, pos As Single)
Public Sub RemoteChangeSpeed(motor As Integer, speed As Single)
Public Sub RemoteFilamentStandby(mode As Integer)

Public Function RemoteGetMagnification() As Single
Public Sub RemoteSetMagnification(Magnification As Single)
Public Sub RemoteGetFaradayState(BeamFlagOn As Boolean)

Public Sub RemoteROMStartScan(motor As Integer, mode As Integer, specpos1 As Single, specpos2 As Single, speed As Single, counttime As Single, npoints as integer)

Public Sub RemoteROMWaitScan(motor As Integer, npoints As Integer, specpos() As Single, speccnt() As Single, done As Integer)

Public Sub RemoteStartPeak(motor As Integer, size As Single, npoints as integer, counttime as single)

Public Sub RemoteWaitPeak(motor As Integer, done As Integer)

Public Sub RemoteReturnROMPHAData(iscal As Integer, npts As Integer, xdata() As Single, ydata() As Single)

Public Sub RemotePeakGetCentroid(mode As Integer, motor As Integer, npts As Integer, xdata() As Single, ydata() As Single, pos As Single, ptob As Single, avgdev As Single, success As Integer)

Imaging Methods
Public Sub RemoteImageInit()
Public Sub RemoteImageStop()
Public Sub RemoteImageClose()
Public Sub RemoteImageSetImageMode(channel As Integer)
Public Sub RemoteImageStart()

Public Sub RemoteImageGet(done As Integer, ntype As Integer, nchannel As Integer, adaverages As Integer, ix As Integer, iy As Integer, sxmin As Single, symin As Single, sxmax As Single, symax As Single, iarray() As Byte, darray() As Long, zmin As Long, zmax As Long)

Public Sub RemoteImageGetBeamMode(beammode as Integer)
Public Sub RemoteImageSetBeamMode(beammode as Integer)

Public Sub RemoteImageBeamDeflection2(xoffset as Single, yoffset as Single)

Public Sub RemoteImageGetImageShift(ix as Single, iy as Single)
Public Sub RemoteImageSetImageShift(ix as Single, iy as Single)

Detailed Description of Methods

Motor Methods

Public Sub RemoteMoveMotor(motor As Integer, pos As Single)

This method will perform a move on the specified motor.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs% + RemoteNumberofStageMotors%.

The position parameter must be between the axis high and low limits as specified by the RemoteMotHiLimit!(motor%) and RemoteMotLoLimits!(motor%) properties.

Usage:

```
Dim motor As Integer
Dim pos As Single
motor% = Val(FormMAIN.TextMoveMotor.Text)
pos! = Val(FormMAIN.TextMovePosition.Text)
Remote.RemoteMoveMotor motor%, pos!
```

Public Sub RemoteMoveStageMotors(mode as integer, method as integer, xpos As Single, ypos As Single, zpos As Single, wpos As Single)

This method will perform a move on all the stage motors.

The mode parameter must be 0 (move all motors, including W motor if available) or 1 (move all motors except W motor). The W motor is available only on some Jeol 733 microprobes.

The method parameter is 0 for do not wait for completion and is 1 to wait for motion completion.

The position parameter must be between the axis high and low limits as specified by the RemoteMotHiLimit!(motor%) and RemoteMotLoLimits!(motor%) properties for the X, Y and Z axes.

Usage:

```
Dim mode As Integer, method as integer
Dim xpos As Single, ypos As Single, zpos As Single, wpos
As Single

mode% = 1   ‘ do not move w motor
method = 1  ‘ wait for completion
xpos! = Val(FormMAIN.TextMovePositionX.Text)
ypos! = Val(FormMAIN.TextMovePositionY.Text)
zpos! = Val(FormMAIN.TextMovePositionZ.Text)
wpos! = 0
Remote.RemoteMoveStageMotors mode%, method%, xpos!,
ypos!, zpos!, wpos!
```

Public Function RemoteGetMotorStatus(motor As Integer) As Integer

This function returns the trajectory completion status of the specified motor. This function is usually called in a loop while waiting for the motor to finish moving. False (zero) is not finished, True (negative one) is finished.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs% + RemoteNumberofStageMotors%.

Usage:

```
Dim motor As Integer, done As Integer

motor% = Val(FormMAIN.TextStatusMotor.Text)
done% = Remote.RemoteGetMotorStatus(motor%)
If done Then
FormMAIN.LabelStatusStatus.Caption = "Done"
Else
FormMAIN.LabelStatusStatus.Caption = "Not Done"
End If
```

Public Function RemoteGetMotorPosition(motor As Integer) As Single

This function returns the current motor position in motor units.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs% + RemoteNumberofStageMotors%.

Usage:

```
Dim motor As Integer
```

```
motor% = Val(FormMAIN.TextPositionMotor.Text)
FormMAIN.LabelPositionPosition.Caption =
Str$(Remote.RemoteGetMotorPosition(motor%))
```

Public Sub RemoteStopAllMotors()

This procedure stops all motors.

Usage:

```
Remote.RemoteStopAllMotors
```

Public Sub RemoteFreeAllMotors()

This procedure releases all motors (only applies to AM interface).

Usage:

```
Remote.RemoteFreeAllMotors
```

Public Sub RemoteFlipCrystalMoveMotor(motor As Integer, crystal As String, pos As Single)

This method will perform a move on the specified spectrometer motor. The wait for status call is implicit in the procedure so only one call is required to flip a spectrometer crystal and move to the specified position. If no crystal flip is required, just pass the current crystal position or variable containing an empty string (“”).

The motor parameter must be between 1 and RemoteNumberofTunableSpecs%. The position parameter must be between the axis high and low limits as specified by the RemoteMotHiLimit!(motor%) and RemoteMotLoLimits!(motor%) properties.

Usage:

```
Dim motor As Integer
Dim crystal as string
Dim pos As Single
motor% = Val(FormMAIN.TextMoveMotor.Text)
crystal$ = Val(FormMAIN.TextFlipCrystal.Text)
pos! = Val(FormMAIN.TextMovePosition.Text)
Remote.RemoteFlipCrystalMoveMotor motor%, crystal$, pos!
```

Crystal Methods

Public Sub RemoteChangeCrystal(motor As Integer, crystal As String)

This function changes the crystal on the specified spectrometer (if crystal flipping is available). Use the RemoteGetCrystalStatus function in a loop to check when the crystal flip is completed.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs%.

The crystal parameter must be a valid crystal string for the specified spectrometer as specified in the SCALERS.DAT file. Use the RemoteGetCrystalPosition function to get the current crystal position (string).

Usage:
```
Remote.RemoteChangeCrystal motor%, crystal$
```

Public Function RemoteGetCrystalPosition(motor As Integer) As String

This function gets the crystal position (string) on the specified spectrometer (if available).

The motor parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:
```
crystal$ = Remote.RemoteGetCrystalPosition motor%
```

Public Function RemoteGetCrystalStatus(motor As Integer) As Integer

This function get the crystal flip status on the specified spectrometer (if crystal flipping is available). Use the RemoteChangeCrystal procedure to start the crystal flip. This function is usually called in a loop while waiting for the crystal to finish flipping. False (zero) is not finished, True (negative one) is finished.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:
```
done% = Remote.RemoteGetCrystalStatus(motor%)
```

Faraday Methods

Public Sub RemoteFaraday(mode As Integer)

This procedure inserts or removes the faraday cup. On instruments with electrostatic beam blankers, the method turns off or turns on the beam.

Usage:
```
Remote.RemoteFaraday(Int(1))  '  insert or blank beam
Remote.RemoteFaraday(Int(2))  '  remove or unblank beam
```

Public Function RemoteGetAbsorbed(counttime As Single) As Single

This function returns the absorbed (sample) current for the instrument. The faraday cup or beam blank state should be set first using the RemoteFaraday procedure.

The counttime parameter is the actual integration time for the AM (MCAPI) interface or the number of A/D averages using other interfaces.

Usage:
```
Dim acurrent as single
Dim ctime as single
ctime! = 1.0
```

```
acurrent! = Remote.RemoteGetAbsorbedCurrent(ctime!)
```

Public Function RemoteGetFaraday(counttime As Single) As Single

This function returns the faraday (beam) current for the instrument. The faraday cup or beam blank state should be set first using the RemoteFaraday procedure.

The counttime parameter is the actual integration time for the AM (MCAPI) interface or the number of A/D averages using other interfaces.

Usage:

```
Dim fcurrent as single
Dim ctime as single
ctime! = 1.0
fcurrent! = Remote.RemoteGetFaradayCurrent(ctime!)
```

Column Methods

Public Function RemoteGetConditionKilovolts() As Single

This function returns the current operating voltage of the instrument in kilovolts (if the Operating Voltage interface is present).

Usage:

```
Dim kev as single
kev! = Remote.RemoteGetConditionKilovolts!
```

Public Function RemoteGetConditionBeamCurrent() As Single

This function returns the current beam current of the instrument in nano-amps (if the Beam Current interface is present).

Usage:

```
Dim bcurrent as single
bcurrent! = Remote.RemoteGetConditionBeamCurrent!
```

Public Function RemoteGetConditionBeamSize() As Single

This function returns the current beam size of the instrument in microns (if the Beam Size interface is present).

Usage:

```
Dim bsize as single
bsize! = Remote.RemoteGetConditionBeamSize!
```

Public Sub RemoteSetConditions(takeoff As Single, kilovolts As Single, beamcurrent As Single, beamsize As Single, columnmethod as integer, columncondition as string)

This procedure sets the actual column conditions for the instrument, including take-off angle (not usually modified), operating voltage (in kilovolts), beam current (in

nano-amps), and beam size (in microns). Normally the columnmethod is set to zero, unless using the columncondition string method described below.

To use a column condition string, for example with the SX50 interface, specifiy a 1 for the columnmethod and the columncondition string, e.g., "hv15".

Usage:

```
Dim takeoff as single, kilovolts as single
Dim beamcurrent as single, beamsize as single
Remote.RemoteSetConditions(takeoff!, kilovolts!,
beamcurrent!, beamsize!, columnmethod%,
columncondition$)
```

Public Sub RemoteSetColumnParameters(method As Integer, CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Long, ObjectiveFine As Long, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

This call will set the specified column parameter. Only the actual value indicated by "method" needs to be loaded, however all arguments must be passed. The "method" argument can take the following values:

method% = 1 change condenser coarse
method% = 2 change condenser fine
method% = 3 change objective coarse
method% = 4 change objective fine
method% = 5 change astigmatism1
method% = 6 change astigmatism2
method% = 7 change rotation
method% = 8 change magnification

For example to set the condenser fine you can call the procedure as follows:

Dim tcondenserfine as Long
tcondenserfine& = 124
Remote.RemoteSetColumnParameters 2, 0, tcondenserfine&, 0, 0, 0, 0, 0, 0

Public Sub RemoteSetColumnParametersExt(method As Integer, CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Double, ObjectiveFine As Double, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

Same as RemoteSetColumnParameters but using double precision variables for ObjectiveCoarse and ObjectiveFine parameters (8x30 instruments). This call will set the specified column parameter. Only the actual value indicated by "method" needs to be loaded, however all arguments must be passed. The "method" argument can take the following values:

method% = 1 change condenser coarse
method% = 2 change condenser fine
method% = 3 change objective coarse
method% = 4 change objective fine
method% = 5 change astigmatism1
method% = 6 change astigmatism2

method% = 7 change rotation
method% = 8 change magnification

For example to set the condenser fine you can call the procedure as follows:

```
Dim tcondenserfine as Long
tcondenserfine& = 124
Remote.RemoteSetColumnParameters 2, 0, tcondenserfine&, 0, 0, 0, 0, 0, 0
```

Public Sub RemoteGetColumnParameters(CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Long, ObjectiveFine As Long, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

This call will get the column parameters.

Usage:

```
Dim CondenserCoarse As Long, CondenserFine As Long
Dim ObjectiveCoarse As Long, ObjectiveFine As Long
Dim Astigmatism1 As Long, Astigmatism2 As Long
Dim Rotation As Single, magnification As Single

Remote.RemoteGetColumnParameters CondenserCoarse&, CondenserFine&,
ObjectiveCoarse&, ObjectiveFine&, Astigmatism1&, Astigmatism2&, rotation!,
magnification!
```

Public Sub RemoteGetColumnParametersExt(CondenserCoarse As Long, CondenserFine As Long, ObjectiveCoarse As Double, ObjectiveFine As Double, Astigmatism1 As Long, Astigmatism2 As Long, rotation As Single, magnification As Single)

Same as RemoteGetColumnParameters but using double precision variables for ObjectiveCoarse and ObjectiveFine parameters (8x30 instruments). This call will get the column parameters.

Usage:

```
Dim CondenserCoarse As Long, CondenserFine As Long
Dim ObjectiveCoarse As Double, ObjectiveFine As Double
Dim Astigmatism1 As Long, Astigmatism2 As Long
Dim Rotation As Single, magnification As Single

Remote.RemoteGetColumnParameters CondenserCoarse&, CondenserFine&,
ObjectiveCoarse#, ObjectiveFine#, Astigmatism1&, Astigmatism2&, rotation!,
magnification!
```

Public Sub RemoteGetGunParameters(Kilovolts As Single, KilovoltStatus as String, FilamentEmission as Single, FilamentCurrent As Single)

This call will get the gun parameters. The KilovoltStatus only applies to JEOL instruments.

Usage:
Dim Kilovolts As Single
Dim KilovoltStatus As String
Dim FilamentEmission As Single, FilamentCurrent As Single

Remote.RemoteGetGunParameters Kilovolts!, KilovoltsStatus$, FilamentEmission!, FilamentCurrent!

Public Sub RemoteSetGunParameters(Method As Integer, Kilovolts As Single, FilamentEmission as Single, FilamentCurrent As Single)

This call will set the gun parameters:
Method% = 1 set kilovolts (in keV)
Method% = 2 set filament emission
Method% = 3 set filament current

Usage:
Dim Method As Integer
Dim Kilovolts As Single
Dim KilovoltStatus As String As Long
Dim FilamentEmission As Single, FilamentCurrent As Single

Method% = 1
Kilovolts! = 15
FilamentEmission! = 0.
FilamentCurrent! = 0.

Remote.RemoteSetGunParameters Kilovolts!, KilovoltsStatus$, FilamentEmission!, FilamentCurrent!

Count Methods

Public Sub RemoteStartCounts(scal As Integer, counttime As Single, maxcounts As Long)

This procedure starts a count cycle on the specified scaler, for the specified count time, with the specified maximum count.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

The counttime parameter must be between the following limits:

Interface Type	Minimum Count Time	Maximum Count Time
Demonstration	0.01	1000000
Jeol 8900/8200/8500	0.01	2147
Jeol 8230/8530	0.01	10000
Cameca SX100/SXFive	0.01	1000000

The maxcounts parameter defines the maximum number of counts required by the count cycle (or counting time, whichever comes first). Usually this is set to an arbitrarily high number such as 10000000 (ten million).

Usage:

```
Dim scal as integer
Dim counttime as single
Dim maxcounts as long
Remote.RemoteStartCounts scal%, counttime!, maxcounts&
```

Public Function RemoteGetCountStatus(scal As Integer) As Integer

This function returns the counter status for the specified scaler number. This function is usually called in a loop while waiting for the count to finish flipping. False (zero) is not finished, True (negative one) is finished.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim done as integer
Dim scaler as integer
scaler% = 1
done% = Remote.RemoteGetCountStatus(scaler%)
```

Public Function RemoteGetCountCount(scal As Integer) As Single

This function returns the actual counts (in counts per second) for the specified scaler number.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim tunablespecs as integer
tunablespecs% = Remote.RemoteNumberofTunableSpecs%
Dim scaler as integer
ReDim counts(1 to tunablespecs%) as single
For scaler% = 1 to tunablespecs%
Counts!(scaler%) = Remote.RemoteGetCountCount(scaler%)
Next scaler%
```

Public Sub RemoteStopAllCounters()

This procedure stops the counting on all counters.

Usage:
Remote.RemoteStopAllCounters

PHA Methods

Public Function RemoteGetPHAParameterBaseline(iscal As Integer) As Single

This function returns the current PHA baseline setting for the specified scaler number.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim baseline as single
Dim iscal as integer
iscal% = 1
baseline! = Remote.RemoteGetPHAParameterBaseline(iscal%)
```

Public Function RemoteGetPHAParameterWindow(iscal As Integer) As Single

This function returns the current PHA window setting for the specified scaler number.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim window as single
Dim iscal as integer
iscal% = 1
window! = Remote.RemoteGetPHAParameterWindow(iscal%)
```

Public Function RemoteGetPHAParameterGain(iscal As Integer) As Single

This function returns the current PHA detector gain setting for the specified scaler number.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim gain as single
Dim iscal as integer
iscal% = 1
gain! = Remote.RemoteGetPHAParameterGain(iscal%)
```

Public Function RemoteGetPHAParameterBias(iscal As Integer) As Single

This function returns the current PHA detector bias setting for the specified scaler number.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim bias as single
Dim iscal as integer
```

```
iscal% = 1
bias! = Remote.RemoteGetPHAParameterBias(iscal%)
```

Public Function RemoteGetPHAParameterInteDiffMode(iscal As Integer) As Integer

This function returns the current PHA Integral/Differential mode setting for the specified scaler number. If true (negative one) then differential mode is set, if false (zero) then integral mode is set.

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim intediff as integer
Dim iscal as integer
iscal% = 1
intediff% =
Remote.RemoteGetPHAParameterInteDiffMode(iscal%)
```

Public Function RemoteGetPHAParameterDeadtime(iscal As Integer) As Single

This function returns the current PHA deadtime setting for the specified scaler number in seconds (1 microsecond = 0.000001 seconds)

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim deadtime as single
Dim iscal as integer
iscal% = 1
deadtime! = Remote.RemoteGetPHAParameterDeadtime(iscal%)
```

Public Sub RemoteSetPHAParameters(iscal As Integer, baseline As Single, window As Single, gain As Single, bias As Single, intediffmode As Integer, deadtime As Single)

This procedure sets the current PHA setting for the specified scaler number including baseline (in volts), window (in volts), gain (unitless), bias (in volts), integral/differential mode (false or zero = integral mode and true or negative one = differential mode) and deadtime (in seconds, where 1 microsecond = 0.000001 seconds).

The scal parameter must be between 1 and RemoteNumberofTunableSpecs%.

Usage:

```
Dim baseline as single, window as single
Dim gain as single, bias as single
Dim intediffmode as integer
Dim deadtime as single
Dim iscal as integer
iscal% = 1
baseline! = 0.5
window! = 9.0
```

```
gain! = 32.0
bias! = 1750.0
intediffmode% = 0       ' integral mode
deadtime! = 0.000002    ' 2 microseconds
Remote.RemoteSetPHAParameters iscal%, baseline!,
window!, gain!, bias!, intediffmode%, deadtime!
```

Public Function RemoteGetPHADistribution(mode As Integer, iscal As Integer, npts As Integer, counttime As Single, startvolts As Single, stopvolts As Single, xdata() As Single, ydata() As Single)

Perform a PHA distribution of the type specified by the mode parameter. Note that the xdata (voltage or gain values) and the ydata (x-ray intensities) arrays must be properly dimensioned before calling the procedure.

' mode = 0 PHA scan
' mode = 1 bias scan
' mode = 2 gain scan

Usage:

```
Dim mode as integer, iscal as integer, i as integer
Dim npts as integer, counttime as single
Dim startvolts as single, stopvolts as single
Dim xdata() as single, ydata() as single
Dim tmsg as string

Mode% = 2         ‘ perform a bias scan
iscal% = 3        ‘ spectrometer 3
npts% = 40        ‘ measure 40 point scan
counttime! = .5   ‘ 0.5 seconds per point
startvolts! = 1500      ‘ starting voltage or gain
stopvolts! = 1800       ‘ stopping voltage or gain

ReDim xdata(1 to npts%) as single
ReDim ydata(1 to npts%) as single

Remote.RemoteGetPHADistribution mode%, iscal%, npts%,
counttime!, startvolts!, stopvolts!, xdata!(), ydata!()
tmsg$ = ""
For i% = 1 To npts%
tmsg$ = tmsg$ & Str$(xdata!(i%)) & Str$(ydata!(i%))
Next i%
```

Miscellaneous Methods

Public Sub RemoteSetMagnification(Magnification As Single)

This procedure sets the current beam scan magnification for the instrument.

Usage:

```
Dim magnification as single
magnification! = 400.0
```

```
Remote.RemoteSetMagnification(magnification!)
```

Public Function RemoteGetMagnification() As Single

This procedure gets the current beam scan magnification for the instrument.

Usage:

```
Dim magnification as single
magnification! = Remote.RemoteGetMagnification
```

Public Sub RemoteAutoFocus()

This procedure performs an auto-focus on the instrument (if supported).

Usage:

```
Remote.RemoteAutoFocus
```

Public Sub RemoteBacklash(motor As Integer, pos As Single)

This procedure performs a motor backlash (jog) for the specified spectrometer or stage motor based on the given position.

The motor parameter must be between 1 and RemoteNumberofTunableSpecs% + RemoteNumberofStageMotors%.

The pos parameter must be between the motor high limit and motor low limit as given by the RemoteMotHiLimits and RemoteMotLoLimits properties.

Usage:

```
Dim motor as integer
Dim pos as single
motor% = 1
' Get the current position
pos! =  Remote.RemoteGetMotorPosition(motor%)
' Jog the motor
Remote.RemoteBacklash motor%, pos!
```

Public Sub RemoteChangeSpeed(motor As Integer, speed As Single)

This procedure sets the motor speed for the specified spectrometer or stage motor axis (if supported by the interface).

The motor parameter must be between 1 and RemoteNumberofTunableSpecs% + RemoteNumberofStageMotors%.

The speed parameter is given in percent of normal speed. Usually between 1 and 100 percent.

Usage:

```
Dim speed as single
speed! = 50.0           ' set speed to 50% of normal
Remote.RemoteChangeSpeed motor%, speed!
speed! = 100.0          ' set speed to 100% of normal
```

```
Remote.RemoteChangeSpeed motor%, speed!
```

Public Sub RemoteFilamentStandby(mode As Integer)

This procedure places the instrument in filament standby mode (if Filament Standby interface is present).

Usage:

```
Remote.RemoteFilamentStandby(Int(1))      ' filament off
Remote.RemoteFilamentStandby(Int(2))      ' filament on
```

Public Sub RemoteGetFaradayState(tbeamonflag As Boolean)

This procedure returns the current state of the faraday cup. True if beam is on (unblanked) or false if beam is off (blanked).

```
Usage:
Dim tbeamonflag as Boolean

Remote.RemoteGetFaradayState tbeamonflag
```

Public Sub RemoteROMStartScan(motor As Integer, mode As Integer, specpos1 As Single, specpos2 As Single, speed As Single, counttime As Single, npoints as Integer)

This procedure will start a ROM based spectrometer scan if the hardware supports it. Be sure not to call the RemoteROMStartScan procedure more time than necessary for the available data by checking the done flag.

Note there are two modes to call the routine. First with a zero npoints parameter in which case the program will use the speed parameter to calculate the step size. Or with a non-zero npoints parameter in which case the program will use the npoints parameter to calculate the step size.

Note that the mode parameter is no longer supported. Simply pass a zero parameter for backward compatibility with bthe ActiveX interface.

```
Usage:
Dim motor as Integer    ‘ spectrometer number
Dim mode As Integer     ‘ no longer supported
Dim specpos1 As Single  ‘ spectrometer start position
Dim specpos2 As Single  ‘ spectrometer stop position
Dim speed As Single     ‘ spectro speed (0-100 percent)
Dim counttime As Single ‘ counttime per point (in secs)
Dim npoints As Integer  ‘ number of points in scan

Remote.RemoteROMStartScan motor%, CInt(0), specpos1!,
specpos2!, speed!, counttime!, npoints%
```

Public Sub RemoteROMWaitScan(motor As Integer, npoints As Integer, specpos() As Single, speccnt() As Single, done As Integer)

This procedure retrieves the data for a ROM based spectrometer scan. Call it after the above RemoteROMStartScan procedure. Note that specpos and speccnt are arrays dimensioned by the npoints parameter.

```
Usage:
Dim motor As Integer    ‘ spectrometer number
Dim npoints As Integer  ‘ number of points returned
Dim specpos() As Single ‘ spectrometer position array
Dim speccnt() As Single ‘ spectrometer intensity array
Dim done As Integer     ‘ scan finished flag

Remote.RemoteROMWaitScan motor%, npoints%, specpos!(),
speccnt!(), done%
```

Public Sub RemoteStartPeak(motor As Integer, size As Single, npoints as Integer, counttime as Single)

Start a "ROM" based spectrometer peak center at the current position. Performs a fine scan and if that fails (bad P/B, fit deviation, centeredness, etc.) the procedure performs a coarse scan and then a second fine scan. Moves the spectrometer to the centroid to the fit or the original position if it fails.

The peaking size parameter is based on the LIF peaking start size from the SCALERS.DAT and normally adjusted (larger) for the actual 2d and spectrometer position of the current spectrometer. That value is further modified by the following internal code:

```
temp! = Abs(MotHiLimits!(motor%) - MotLoLimits!(motor%))
/ ScalPeakScanSizeFactors!(motor%) ' basic scan width
temp! = temp! * size! /ScalLiFPeakingStartSizes!(motor%)
temp! = temp! * 2#      ' fudge factor for proper width

startpos! = RealTimeMotorPositions!(motor%) - 0.5 *
temp!
stoppos! = RealTimeMotorPositions!(motor%) + 0.5 * temp!
```

Note that the size parameter is used by all interfaces except the Jeol 8900/8200. The Jeol interface uses the npoints and count time parameters for the ROM based peaking scan sizes.

```
Usage:
Dim motor as Integer          ‘ spectometer number
Dim size as Single            ‘ peaking size parameter
Dim npoints as Integer        ‘ number of points
Dim counttime as Single       ‘ count time (in sec)

Remote.RemoteStartPeak(motor%, size!, npoints%,
counttime!)
```

Public Sub RemoteWaitPeak(motor As Integer, done As Integer)

Wait for a "ROM" based spectrometer peak procedure. The function should be called until the done flag or an error is returned True (-1).

```
Usage:
Dim motor as Integer    ‘ spectrometer number
Dim done as Integer     ‘ returned flag for peak
completion

Remote.RemoteWaitPeak(motor%, done%)
```

Public Sub RemoteReturnROMPHAData(iscal As Integer, npts As Integer, xdata() As Single, ydata() As Single)

Get the ROM peak or PHA data that was acquired using RemoteROMStartScan and RemoteROMWaitScan or RemoteStartPeak and RemoteWaitPeak.

Note that the xdata and ydata arrays must be dimensioned properly before calling this method! If the size of the returned array is larger than that allocated then an error will occur.

Usage:

```
Dim iscal as Integer    (passed)
Dim npts as Integer     (returned)
Dim xdata() as Single   (returned)
Dim ydata() as Single   (returned)

iscal% = 2
npts% = 24  ‘ overwritten by actual values when called
ReDim xdata(1 to npts%) as single
ReDim ydata(1 to npts%) as single
Remote.RemoteReturnROMPHAData(iscal%, npts%, xdata!(),
ydata!())
```

Public Sub RemotePeakGetCentroid(mode As Integer, motor As Integer, npts As Integer, xdata() As Single, ydata() As Single, pos As Single, ptob As Single, avgdev As Single, success As Integer)

Get the centroid for the ROM peak data. Uses the current ROM peaking method and threshold values. See above to “get” and “let” these values.

Usage:

```
Dim motor as integer    (passed)
Dim mode as Integer     (passed)
Dim npts as Integer     (passed)
Dim xdata() as Single   (passed)
Dim ydata() as Single   (passed)
Dim pos as Single       (returned)
Dim ptob as Single      (returned)
Dim avgdev as Single    (returned)
Dim success as integer  (returned)

mode% = 1   ‘ 1 = fine, 2 = coarse, 3 = 2nd fine
```

```
motor% = 2
npts% = 24
ReDim xdata(1 to npts%) as single
ReDim ydata(1 to npts%) as single

‘ Fill x and y data arrays…

Remote.RemotePeakGetCentroid(mode%, motor%, npts%,
xdata!(), ydata!(), pos!, ptob!, avgdev!, success%)
```

Public Sub RemoteSetReflectedLight(lightonoff as Integer, lightintensity as Integer)

Set reflected light mode (0 = off, 1 = on), and light intensity (0 to 64 for Cameca, 0 to 127 for JEOL).

```
Usage:
‘ Set reflected light on, and brightness to mid range
(for JEOL)
Remote.RemoteSetReflectedLight(Int(1), Int(63))
```

Public Sub RemoteSetTransmittedLight(lightonoff as Integer, lightintensity as Integer)

Set transmitted light mode (0 = off, 1 = on), and light intensity (0 to 64 for Cameca, 0 to 127 for JEOL).

```
Usage:
‘ Set transmitted light on, and brightness to mid range
(for JEOL)
Remote.RemoteSetTransmittedLight(Int(1), Int(63))
```

Imaging Methods

Public Sub RemoteImageInit()

Initialize an electron image acquisition (call once first to be sure the imaging interface is initialized). This includes calls to get or set the beam mode (spot or scan mode).

```
Usage:
Remote.RemoteImageInit
```

Public Sub RemoteImageStop()

Stop an electron image acquisition

```
Usage:
Remote.RemoteImageStop
```

Public Sub RemoteImageClose()

' Close an electron image acquisition

```
Usage:
```

```
Remote.RemoteImageClose
```

Public Sub RemoteImageSetImageMode(channel As Integer)

Set Image Mode for an electron image acquisition (call before starting image on JEOL instruments)
channel = 1 = SEI
channel = 2 = BSE
channel = 3 = AUX (usually CL)

```
Usage:
Dim channel as Integer
Remote.RemoteImageSetImageMode(channel%)
```

Public Sub RemoteImageStart()

' Start an electron image acquisition (call RemoteImageGet to check if image is done)

```
Usage:
Remote.RemoteImageStart
```

Public Sub RemoteImageGet(done As Integer, ntype…

Get electron image data and status
ntype = image type (1 = electron, 2 = xray)
nchannel = channel number (e.g., 1 = SE, 2 = BSE or 1 = spec 1, 2 = spec 3)
adaverages = number of pixel averages
ix = x pixels (usually 64, 128, 256, 512, or 1024 or 2048, etc)
iy = y pixels
sxmin, symin, sxmax, symax = stage coordinate of image corners in stage units
iarray() = byte array containing image data (array must be dimensioned first)
darray() = long array containing image data
zmin = minimum signal value in darray
zmax = maximum signal value in darray

```
Usage:
Dim done% (returned, true (-1) if image complete)
Dim ntype as Integer (passed)
Dim nchannel as Integer (passed)
Dim adaverages as Integer (passed)
Dim ix as Integer, iy as Integer (passed)
Dim sxmin as Single, symin as Single (returned)
Dim sxmax as Single, symax as Single (returned)
Dim iarray(1 to ix%, 1 to iy%) as Byte (returned)
Dim darray(1 to ix%, 1 to iy%) as Long (returned)
Dim zmin as Long, zmax as Long (returned)

Remote.RemoteImageGet(done%, ntype%, nchannel%,
adaverages%, ix%, iy%, sxmin!, symin!, sxmax!, symax!,
iarray(), darray&(), zmin&, zmax&)
```

Note that this function should be called in a loop approximately every 500 msec or so until the completion status is true (done = -1).

Public Sub RemoteImageGetBeamMode(beammode as Integer)

Public Sub RemoteImageSetBeamMode(beammode as Integer)

Get and set beam modes (0 = analog spot, 1 = analog scan, 2 = digital spot). Note that the RealTimeImageInit procedure MUST be called once before any get or set beam mode calls are performed.

```
Usage:
Dim beammode as Integer

‘ Get the beam mode
Remote.RemoteImageGetBeamMode(beammode%)

‘ Set spot mode
Remote.RemoteImageGetBeamMode(Int(0))

‘ Set scan mode
Remote.RemoteImageGetBeamMode(Int(1))
```

Public Sub RemoteImageBeamDeflection2(xoffset as Single, yoffset as Single)

Deflect the beam where xoffset and yoffset are the beam deflection positions in device independent units (+/- 0.5, where 0,0 is centered). The beam must be in spot mode for this function. Note also that the RealTimeImageInit procedure MUST be called once before any get or set beam mode calls are performed.

```
Usage:
Dim xoffset as Single, yoffset as Single

‘ Set the beam spot to the center
xoffset! = 0.0
yoffset! = 0.0
Remote.RemoteImageBeamDeflection(xoffset!, yoffset!)

‘ Set the beam spot to the corner
xoffset! = 0.5
yoffset! = 0.5
Remote.RemoteImageBeamDeflection(xoffset!, yoffset!)
```

Public Sub RemoteImageGetImageShift(ix as Single, iy as Single)

Public Sub RemoteImageSetImageShift(ix as Single, iy as Single)

Get or set the image shift where ix and iy are in micon units. Note that the set micron range is limited by the current magnification. The beam must be in scan mode for this function. Note also that the RealTimeImageInit procedure MUST be called once before any get or set beam mode calls are performed.

```
Usage:
```

```
Dim ix as Single, iy as Single

‘ Get the current image shift in microns
Remote.RemoteImageGetImageShift(ix!, iy!)

‘ Set the current image shift in microns
ix! = 1.0
iy! = 1.0
Remote.RemoteImageSetImageShift(ix!, iy!)
```

Excel Code Examples

Excel- Coding Suggestions

To create a Visual Basic for Applications Excel project to access the Remote Automation Interface first start Excel and create a new Workbook.

Then click the Tools | Customize menu and check both the Visual Basic and Control Toolbox items. Drag the new toolboxes to the top of the application and dock them next to the other toolbars.

Now click the Tools | Macro | Visual Basic Editor (or simply click the Visual Basic Editor button on the newly visible Visual Basic toolbar) and the Visual basic design mode window will become visible.

Now click the Tools | References menu in the newly visible Visual Basic window and scroll down to the Remote Automation Interface and check it.

To add code. Click the Insert | Module menu. To add controls switch back to the spreadsheet and select a control (button, etc) from the toolbar and click and drag it on the spreadsheet.

For examples, see the supplied Excel spreadsheets: TestRemote.xls, TestReproduce.xls and TestDeadtime.xls.

Excel- Sample Code

This following Excel (Visual Basic for Applications) code performs a simple get motor position on all the motors and writes the positions to the spreadsheet.

```
' (c) Copyright 1995-2000 by John J. Donovan
Option Explicit

Dim ierror As Integer

Sub TestRemoteGetMotorPositions()
' Example code for PFW Remote Automation Interface

On Error GoTo TestRemoteGetMotorPositionsError

Dim motor As Integer
Dim tunablespecs As Integer, stagemotors As Integer
```

```
Dim excelrow As Integer, excelcol As Integer

' Create the Excel object
Dim xlsheet As Excel.Worksheet
Set xlsheet = TestRemote.Sheet1

' Create the Remote Automation Interface object
Dim remote As Realtime
Set remote = New Realtime

excelrow% = 5
excelcol% = 0

' Get number of spectrometers and stage motors from
remote
tunablespecs% = remote.RemoteNumberofTunableSpecs
stagemotors% = remote.RemoteNumberofStageMotors

' Dimension position array
ReDim positions(1 To tunablespecs% + stagemotors%) As
Single

' Write column labels to spreadsheet
For motor% = 1 To tunablespecs% + stagemotors%
xlsheet.Cells(excelrow%, excelcol% + (motor%)) = "Motor
" & Format$(motor%)
Next motor%
excelrow% = excelrow% + 1

' Get positions
For motor% = 1 To tunablespecs% + stagemotors%
positions!(motor%) =
remote.RemoteGetMotorPosition(motor%)
Next motor%

' Save current positions to spreadsheet
For motor% = 1 To tunablespecs% + stagemotors%
xlsheet.Cells(excelrow%, excelcol% + motor%) =
Format$(positions!(motor%))
Next motor%

' Release the object variable
Set xlsheet = Nothing
Exit Sub

' Errors
TestRemoteGetMotorPositionsError:
MsgBox Error$, vbOKOnly + vbCritical,
"TestRemoteGetMotorPositions"
Exit Sub

End Sub
```

Excel- Sample Code (Motor Reproducibility)

The following code example performs a spectrometer and stage motor reproducibility test, by cycling the motors between the high and low axis limits and recording the actual positions in a spreadsheet.

```
' (c) Copyright 1995-2006 by John J. Donovan
Option Explicit

Dim ierror As Integer

Sub ReproduceStart()
' Example code for PFW Remote Automation Interface
' Motor reproducibility test

On Error GoTo ReproduceStartError

Dim motor As Integer, done As Integer
Dim tunablespecs As Integer, stagemotors As Integer

Dim alldone As Integer
Dim excelrow As Integer, excelcol As Integer

Dim startpositions() As Single, currentpositions() As
Single
Dim hilimits() As Single, lolimits() As Single
Dim replicate As Integer

Const MAXREPLICATES% = 50

' Create the Excel object
Dim xlsheet As Excel.Worksheet
Set xlsheet = Reproduce.Sheet1

' Create the Remote Automation Interface object
Dim remote As Realtime
Set remote = New Realtime

excelrow% = 5
excelcol% = 0

' Get number of spectrometers and stage motors from
remote
tunablespecs% = remote.RemoteNumberofTunableSpecs
stagemotors% = remote.RemoteNumberofStageMotors

' Dimension position array
ReDim startpositions(1 To tunablespecs% + stagemotors%)
As Single
ReDim currentpositions(1 To tunablespecs% +
stagemotors%) As Single
ReDim hilimits(1 To tunablespecs% + stagemotors%) As
Single
ReDim lolimits(1 To tunablespecs% + stagemotors%) As
Single
```

```
' Get limits (minus small amount)
For motor% = 1 To tunablespecs% + stagemotors%
hilimits!(motor%) = remote.RemoteMotHiLimits!(motor%)
lolimits!(motor%) = remote.RemoteMotLoLimits!(motor%)
hilimits!(motor%) = hilimits!(motor%) -
Abs(hilimits!(motor%) / 1000#)
lolimits!(motor%) = lolimits!(motor%) +
Abs(lolimits!(motor%) / 1000#)
Next motor%

' Write column labels to spreadsheet
For motor% = 1 To tunablespecs% + stagemotors%
xlsheet.Cells(excelrow%, excelcol% + (motor%)) = "Motor
" & Format$(motor%)
Next motor%

' Add column for elasped time
xlsheet.Cells(excelrow%, excelcol% + motor%) = "Time"
excelrow% = excelrow% + 1

' Get start positions
For motor% = 1 To tunablespecs% + stagemotors%
startpositions!(motor%) =
remote.RemoteGetMotorPosition(motor%)
Next motor%

For replicate% = 1 To MAXREPLICATES%

' Move motors to high limits
For motor% = 1 To tunablespecs% + stagemotors%
remote.RemoteMoveMotor motor%, hilimits!(motor%) -
(hilimits!(motor%) * 0.01)
Next motor%

' Wait for motors
alldone% = False
Do Until alldone
alldone% = True
For motor% = 1 To tunablespecs% + stagemotors%
done% = remote.RemoteGetMotorStatus(motor%)
If Not done% Then alldone% = False
DoEvents
If ierror Then GoTo ReproduceStartCancel
Next motor%
Loop

' Move motors to low limits
For motor% = 1 To tunablespecs% + stagemotors%
remote.RemoteMoveMotor motor%, lolimits!(motor%) +
(lolimits!(motor%) * 0.01)
Next motor%

' Wait for motors
alldone% = False
Do Until alldone
alldone% = True
```

```
For motor% = 1 To tunablespecs% + stagemotors%
done% = remote.RemoteGetMotorStatus(motor%)
If Not done% Then alldone% = False
DoEvents
If ierror Then GoTo ReproduceStartCancel
Next motor%
Loop

' Move motors to start positions
For motor% = 1 To tunablespecs% + stagemotors%
remote.RemoteMoveMotor motor%, startpositions!(motor%)
Next motor%

' Wait for motors
alldone% = False
Do Until alldone
alldone% = True
For motor% = 1 To tunablespecs% + stagemotors%
done% = remote.RemoteGetMotorStatus(motor%)
If Not done% Then alldone% = False
DoEvents
If ierror Then GoTo ReproduceStartCancel
Next motor%
Loop

' Get current positions
For motor% = 1 To tunablespecs% + stagemotors%
currentpositions!(motor%) =
remote.RemoteGetMotorPosition(motor%)
Next motor%

' Save current positions to spreadsheet
For motor% = 1 To tunablespecs% + stagemotors%
xlsheet.Cells(excelrow%, excelcol% + motor%) =
Format$(currentpositions!(motor%))
Next motor%

' Add column for elasped time
xlsheet.Cells(excelrow%, excelcol% + motor%) = Now

excelrow% = excelrow% + 1
Next replicate%

' Release the object variable
Set xlsheet = Nothing
Exit Sub

' Errors
ReproduceStartError:
MsgBox Error$, vbOKOnly + vbCritical, "ReproduceStart"
Exit Sub

ReproduceStartCancel:
MsgBox "Automation canceled", vbOKOnly + vbExclamation,
"ReproduceStart"
remote.RemoteStopAllMotors
Set xlsheet = Nothing
```

```
Exit Sub

End Sub

Sub ReproduceStop()
' Stop the automation

On Error GoTo ReproduceStopError

ierror = True

Exit Sub

' Errors
ReproduceStopError:
MsgBox Error$, vbOKOnly + vbCritical, "ReproduceStop"
Exit Sub

End Sub
```

Excel- Sample Code (Deadtime Measurements)

This code example shows how one might write a procedure to automatically acquire a dataset for use in calibrating the deadtime constants in the microprobe. The constants in this example are currently declared for use with Paul Carpenter's Deadtime.xls Excel spreadsheet.

```
' (c) Copyright 1995-2000 by John J. Donovan
Option Explicit

Dim ierror As Integer

Sub DeadtimeStart()
' Example code for PFE Remote Automation Interface
' Deadtime calibration acquisition


ierror = False
On Error GoTo DeadtimeStartError


Dim current As Integer, replicate As Integer
Dim scaler As Integer
Dim tunablespecs As Integer


Dim faradaycurrent As Single, absorbedcurrent As Single
Dim startcurrent As Single, stopcurrent As Single,
stepcurrent As Single
```

```
Dim beamaverages As Single

Dim takeoff As Single, kilovolts As Single
Dim beamcurrent As Single, beamsize As Single

Dim counttime As Single
Dim maxcounts As Long

Dim counts() As Single
Dim bdone() As Boolean

Dim alldone As Boolean
Dim excelrow As Integer, excelcol As Integer

' Create the Excel object
Dim xlSheet As Excel.Worksheet
Set xlSheet = Deadtime.Sheet1

' Create the Remote Automation Interface object
Dim Remote As Realtime
Set Remote = New Realtime

Const MAXCURRENTS% = 10
Const MAXREPLICATES% = 5
Const XLSTARTROW% = 5
Const XLSTARTCOL% = 0

' Initialize counting time
counttime = 60#
maxcounts = 1000000000
startcurrent = 10
stopcurrent = 200
stepcurrent = (stopcurrent - startcurrent) /
MAXCURRENTS%

takeoff = 40#
kilovolts = 15#
beamcurrent = 20#
```

```
beamsize = 0#

excelrow = XLSTARTROW%
excelcol = XLSTARTCOL%

beamaverages = 5#    ' number of beam current
measurements to average

' Get number of spectrometers from remote
tunablespecs = Remote.RemoteNumberofTunableSpecs

' Write faraday column label
xlSheet.Cells(XLSTARTROW, excelcol + 1) = "Faraday"

' Write column labels to spreadsheet
For scaler = 1 To tunablespecs
xlSheet.Cells(XLSTARTROW%, excelcol + scaler + 1) =
"Scaler " & Format$(scaler)
Next scaler

' Write count time label
xlSheet.Cells(XLSTARTROW%, excelcol + tunablespecs + 2)
= "Count Time"

' Write faraday column label
xlSheet.Cells(XLSTARTROW%, excelcol + tunablespecs + 3)
= "Absorbed"

' Clear the rows
For excelrow = 1 To MAXCURRENTS% * MAXREPLICATES%
For excelcol = 1 To tunablespecs + 3
xlSheet.Cells(XLSTARTROW% + excelrow, XLSTARTCOL% +
excelcol) = ""
Next excelcol
Next excelrow

excelrow = XLSTARTROW%
excelcol = XLSTARTCOL%
excelrow = excelrow + 1   ' leave for column labels
```

```
' Dimension array for counts and status flags
ReDim counts(1 To tunablespecs) As Single
ReDim bdone(1 To tunablespecs) As Boolean

' Loop on different beam currents (from start to stop in
stepcurrent intervals)
For current = 1 To MAXCURRENTS%

' Calculate beam current
beamcurrent = startcurrent + stepcurrent * (current - 1)

' Set conditions
Remote.RemoteSetConditions takeoff, kilovolts,
beamcurrent, beamsize, 0, ""

For replicate = 1 To MAXREPLICATES%

' Get absorbed current
absorbedcurrent = Remote.RemoteGetAbsorbed(beamaverages)

' Unblank beam
Remote.RemoteFaraday Int(2)

' Start counters
For scaler = 1 To tunablespecs
Remote.RemoteStartCounts scaler, counttime, maxcounts
bdone(scaler) = False
Next scaler

' Wait for counters (counts returned in cps)
alldone = False
Do Until alldone
alldone = True

For scaler = 1 To tunablespecs
If Not bdone(scaler) Then
bdone(scaler) = Remote.RemoteGetCountStatus(scaler)
```

```
If Not bdone(scaler) Then alldone = False
DoEvents
If ierror Then GoTo DeadtimeStartCancel
End If
Next scaler
Loop

' Get counts
For scaler = 1 To tunablespecs
counts(scaler) = Remote.RemoteGetCountCount(scaler)
Next scaler

' Blank beam
Remote.RemoteFaraday Int(1)

' Get faraday current
faradaycurrent! = Remote.RemoteGetFaraday(beamaverages)

' Save faraday to spreadsheet
xlSheet.Cells(excelrow, excelcol + 1) =
Format$(faradaycurrent)

' Save counts to spreadsheet (un-normalize)
For scaler = 1 To tunablespecs
xlSheet.Cells(excelrow, excelcol + scaler + 1) =
Format$(counts(scaler) * counttime)
Next scaler

' Save count time to spreadsheet
xlSheet.Cells(excelrow, excelcol + tunablespecs + 2) =
Format$(counttime)

' Save absorbed to spreadsheet
xlSheet.Cells(excelrow, excelcol + tunablespecs + 3) =
Format$(absorbedcurrent)

' Loop to next replicate or beam current
excelrow = excelrow + 1
Next replicate
```

```
Next current

' Release the object variable
Set xlSheet = Nothing
Exit Sub

' Errors
DeadtimeStartError:
MsgBox Error$, vbOKOnly + vbCritical, "DeadtimeStart"
Exit Sub

DeadtimeStartCancel:
MsgBox "Automation canceled", vbOKOnly + vbExclamation,
"DeadtimeStart"
Remote.RemoteFaraday Int(1)
Exit Sub

End Sub

Sub DeadtimeStop()
' Stop the automation

On Error GoTo DeadtimeStopError

ierror = True

Exit Sub

' Errors
DeadtimeStopError:
MsgBox Error$, vbOKOnly + vbCritical, "DeadtimeStop"
Exit Sub

End Sub
```

Visual Basis Code Examples

Visual Basic- Coding Suggestions

To create a Visual Basic project to access the Remote Automation Interface, first create a new project based on the Standard EXE template.

Then click on Project | References and scroll down to the Remote Automation Interface object and check it. Click OK, and return to the project to create user forms and code modules.

For an example see the supplied Visual Basic project: TestRemote.vbp.

To declare, load and unload the remote Automation Interface in your code use the following code as an example:

Declare Remote Object Example

In this code fragment, the Remote Automation object is declared at the module level so that all procedures can access it.

```
' (c) Copyright 1995-2016 by John J. Donovan
Option Explicit

Dim ierror As Integer

Dim Remote As Realtime   ' Remote automation Interface
object (shared by all modules)
```

Load Remote Object Example

This code fragment shows how to load the Remote Automation object for use in your code.

```
Sub TestRemoteLoadRemoteObject()
' Load the Remote Automation object (call from Form Load
event)

ierror = False
On Error GoTo TestRemoteLoadRemoteObjectError

Set Remote = New Realtime

Exit Sub

' Errors
```

```
TestRemoteLoadRemoteObjectError:
MsgBox Error$, vbOKOnly + vbCritical,
"TestRemoteLoadRemoteObject"
ierror = True
Exit Sub

End Sub
```

Unload Remote Object Example

This code fragment shows how to unload the Remote Automation object after you are finished.

```
Sub TestRemoteUnLoadRemoteObject()
' UnLoad the Remote Automation object (call from Form
Unload event)

ierror = False
On Error GoTo TestRemoteUnLoadRemoteObjectError

If Not Remote Is Nothing Then Set Remote = Nothing

Exit Sub

' Errors
TestRemoteUnLoadRemoteObjectError:
MsgBox Error$, vbOKOnly + vbCritical,
"TestRemoteUnLoadRemoteObject"
ierror = True
Exit Sub

End Sub
```

Cancel Automation

If you have the following procedure and "ierror" is a global variable referenced in your automation routine, then calling this will cancel your automation.

```
' (c) Copyright 1995-2016 by John J. Donovan
Option Explicit

Global ierror As Integer

Dim Remote As Realtime   ' Remote automation Interface
object (shared by all modules)



Sub TestRemoteCancel()

ierror = False
On Error GoTo TestRemoteCancelError

ierror = True
```

```
Exit Sub

' Errors
TestRemoteCancelError:
MsgBox Error$, vbOKOnly + vbCritical, "TestRemoteCancel"
ierror = True
Exit Sub

End Sub
```

Visual Basic- Sample Code (Move Motors)

This Visual Basic code example, moves the spectrometer and stage motors between the high and low spectrometer and stage limits and back to the current positions.

```
Sub TestRemoteMotorTest()
' Performs a motor test using the remote motor interface

ierror = False
On Error GoTo TestRemoteMotorTestError

Dim maxspec As Integer, maxstage As Integer
Dim motor As Integer, done As Integer, alldone As
Integer
Dim pos As Single, xpos As Single, ypos As Single, zpos
As Single, wpos As Single

' Get number of motors
maxspec% = Remote.RemoteNumberofTunableSpecs
maxstage% = Remote.RemoteNumberofStageMotors

' Save current positions
ReDim motorpositions(1 To maxspec% + maxstage%) As
Single
For motor% = 1 To maxspec% + maxstage%
motorpositions!(motor%) =
Remote.RemoteGetMotorPosition(motor%)
Next motor%

' Get low limits and high limits
ReDim lolimits(1 To maxspec% + maxstage%) As Single
ReDim hilimits(1 To maxspec% + maxstage%) As Single
For motor% = 1 To maxspec% + maxstage%
hilimits!(motor%) = Remote.RemoteMotHiLimits!(motor%)
lolimits!(motor%) = Remote.RemoteMotLoLimits!(motor%)
Next motor%

' Move all motors to high limit
For motor% = 1 To maxspec%
pos! = hilimits!(motor%) - (hilimits!(motor%) -
lolimits!(motor%)) / 1000#
Remote.RemoteMoveMotor motor%, pos!
Next motor%
```

```
For motor% = maxspec% + 1 To maxspec% + maxstage%
If motor% = maxspec% + 1 Then xpos! = hilimits!(motor%)
- (hilimits!(motor%) - lolimits!(motor%)) / 1000#
If motor% = maxspec% + 2 Then ypos! = hilimits!(motor%)
- (hilimits!(motor%) - lolimits!(motor%)) / 1000#
If motor% = maxspec% + 3 Then zpos! = hilimits!(motor%)
- (hilimits!(motor%) - lolimits!(motor%)) / 1000#
Next motor%

If Remote.RemoteMoveAllStageMotorsHardwarePresent Then
Remote.RemoteMoveStageMotors Int(1), xpos!, ypos!,
zpos!, wpos!
Else
Remote.RemoteMoveMotor maxspec% + 1, xpos!
Remote.RemoteMoveMotor maxspec% + 2, ypos!
Remote.RemoteMoveMotor maxspec% + 3, zpos!
End If

' Wait for motion complete
alldone% = False
Do Until alldone%
alldone% = True
For motor% = 1 To maxspec% + maxstage%
done% = Remote.RemoteGetMotorStatus(motor%)
DoEvents
If ierror Then GoTo TestRemoteMotorTestCancel
If Not done% Then alldone% = False
Next motor%
Loop

' Move all motors to low limit
For motor% = 1 To maxspec%
pos! = lolimits!(motor%) + (hilimits!(motor%) -
lolimits!(motor%)) / 1000#
Remote.RemoteMoveMotor motor%, pos!
Next motor%

For motor% = maxspec% + 1 To maxspec% + maxstage%
If motor% = maxspec% + 1 Then xpos! = lolimits!(motor%)
+ (hilimits!(motor%) - lolimits!(motor%)) / 1000#
If motor% = maxspec% + 2 Then ypos! = lolimits!(motor%)
+ (hilimits!(motor%) - lolimits!(motor%)) / 1000#
If motor% = maxspec% + 3 Then zpos! = lolimits!(motor%)
+ (hilimits!(motor%) - lolimits!(motor%)) / 1000#
Next motor%

If Remote.RemoteMoveAllStageMotorsHardwarePresent Then
Remote.RemoteMoveStageMotors Int(1), xpos!, ypos!,
zpos!, wpos!
Else
Remote.RemoteMoveMotor maxspec% + 1, xpos!
Remote.RemoteMoveMotor maxspec% + 2, ypos!
Remote.RemoteMoveMotor maxspec% + 3, zpos!
End If

' Wait for motion complete
alldone% = False
```

```
Do Until alldone%
alldone% = True
For motor% = 1 To maxspec% + maxstage%
done% = Remote.RemoteGetMotorStatus(motor%)
DoEvents
If ierror Then GoTo TestRemoteMotorTestCancel
If Not done% Then alldone% = False
Next motor%
Loop

' Move to previous position
For motor% = 1 To maxspec%
pos! = motorpositions!(motor%)
Remote.RemoteMoveMotor motor%, pos!
Next motor%

For motor% = maxspec% + 1 To maxspec% + maxstage%
If motor% = maxspec% + 1 Then xpos! =
motorpositions!(motor%)
If motor% = maxspec% + 2 Then ypos! =
motorpositions!(motor%)
If motor% = maxspec% + 3 Then zpos! =
motorpositions!(motor%)
Next motor%

If Remote.RemoteMoveAllStageMotorsHardwarePresent Then
Remote.RemoteMoveStageMotors Int(1), xpos!, ypos!,
zpos!, wpos!
Else
Remote.RemoteMoveMotor maxspec% + 1, xpos!
Remote.RemoteMoveMotor maxspec% + 2, ypos!
Remote.RemoteMoveMotor maxspec% + 3, zpos!
End If

' Wait for motion complete
alldone% = False
Do Until alldone%
alldone% = True
For motor% = 1 To maxspec% + maxstage%
done% = Remote.RemoteGetMotorStatus(motor%)
DoEvents
If ierror Then GoTo TestRemoteMotorTestCancel
If Not done% Then alldone% = False
Next motor%
Loop

Exit Sub

' Errors
TestRemoteMotorTestError:
MsgBox Error$, vbOKOnly + vbCritical,
"TestRemoteMotorTest"
ierror = True
Exit Sub

TestRemoteMotorTestCancel:
```

```
MsgBox "Test canceled", vbOKOnly + vbExclamation,
"TestRemoteMotorTest"
Remote.RemoteStopAllMotors
ierror = True
Exit Sub

End Sub
```

Visual Basic- Sample Code (Count X-rays)

This Visual Basic code example cycles the scalers and records the counts to an ASCII data file.

```
Sub TestRemoteCountTest()
' Performs a counter test using the remote counter
interface

ierror = False
On Error GoTo TestRemoteCountTestError

Dim maxtunable As Integer
Dim scaler As Integer, alldone As Integer, done As
Integer
Dim counts() As Single

Dim counttime As Single, maxcounts As Long
Dim outputfile As String, astring As String

' Define output file
outputfile$ = App.Path & "\TESTREMOTE.DAT"

' Get number of scalers
maxtunable% = Remote.RemoteNumberofTunableSpecs

' Dimension array for counts
ReDim counts(1 To maxtunable%) As Single

' Unblank beam
Remote.RemoteFaraday Int(2)

' Start counters
counttime! = 10#
maxcounts& = 100000000
For scaler% = 1 To maxtunable%
Remote.RemoteStartCounts scaler%, counttime!, maxcounts&
Next scaler%

' Wait for counts
alldone% = False
Do Until alldone%
alldone% = True
For scaler% = 1 To maxtunable%
done% = Remote.RemoteGetCountStatus(scaler%)
```

```
DoEvents
If ierror Then GoTo TestRemoteCountTestCancel
If Not done% Then alldone% = False
Next scaler%
Loop

' Blank beam
Remote.RemoteFaraday Int(1)

' Get counts
astring$ = ""
For scaler% = 1 To maxtunable%
counts!(scaler%) = Remote.RemoteGetCountCount(scaler%)
astring$ = astring$ & Format$(counts!(scaler%)) & vbTab
Next scaler%

' Write data to file
Open outputfile$ For Append As #1
Print #1, astring$
Close #1

Exit Sub

' Errors
TestRemoteCountTestError:
MsgBox Error$, vbOKOnly + vbCritical,
"TestRemoteCountTest"
ierror = True
Exit Sub

TestRemoteCountTestCancel:
MsgBox "Test canceled", vbOKOnly + vbExclamation,
"TestRemoteCountTest"
Remote.RemoteStopAllCounters
ierror = True
Exit Sub

End Sub
```

Index