Matrix Correction COM Server v. 13.0.6

User Guide and Reference

Probe Software

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

Contents

Matrix COM Server User’s Guide

Contents

Introduction To Matrix COM Server

Description

Matrix is an Active X executable server that may be utilized for calculation of matrix corrections using polynomial alpha-factor (3 coefficient). It can also query the element setup database for primary standard, MAN standard and interference standard intensities for x-ray quantification purposes 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.

Installation

Run the Matrix.msi installer application in the Matrix distribution to install the Matrix COM Server to your computer. Once installed you can start programming to the Matrix Server API if desired using Excel or another OLE compliant container.

To complete the installation of the Matrix active-X server you *must* run the TestMatrix.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.

The installation folder is C:\Program Files\Probe Software\Matrix for XP or Vista 32 bit or Win7/8/10 32 bit and C:\Program Files (x86)\Probe Software\Matrix for Win7/8/10 64 bit.

The files installed are documented here:

Matrix.pdf	PDF document with Matrix interface reference
TestMatrix.bas	Visual Basic (VB5) test program source file

TestMatrix2.bas	Visual Basic (VB5) test program source file
TestMatrix.exe	Visual Basic (VB5) test program executable
TestMatrix.frm	Visual Basic (VB5) test program source file
TestMatrix.frx	Visual Basic (VB5) test program source file
TestMatrix.vbp	Visual Basic (VB5) test program source file
TestMatrix.vbw	Visual Basic (VB5) test program source file

Note that the Matrix Server application (Matrix.exe, Matrix.hlp and Matrix.ini) are always installed to the Windows\System32 (for 32 bit OS) or C:\Windows\SysWOW64 (for 64 bit OS) folder so that the Matrix Correction Interface is available to all applications. However, the Matrix Correction 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\Probe Software\Probe for EPMA or C:\Program Files (x86)\Probe Software\Probe for EPMA\) then you must edit the Matrix.ini file (in the C:\Windows\System32(or 32 bit OS) or the C:\Windows\SysWOW64(for 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 MATRIX.INI file the first line (ProgramPath keyword) is ignored and the second line is actually used by the Matrix 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

To update the Matrix server, simply run the Matrix.msi installer.

Excel Troubleshooting

When accessing the Matrix Correction Interface from Excel you may receive the error message “Excel does not recognize the Matrix 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 Matrix Correction Interface by creating an explicit error handler. This is easily done using the "On Error GoTo ErrHandler: " statement.

Declare the Matrix Correction Object

To begin, use the statement:

```
Dim tMatrix as ClassMatrix
```

To declare your Matrix Correction object. To load the Matrix Correction object use the statement:

```
Set tMatrix as ClassMatrix
```

To unload your Matrix Correction object use the statement:

```
Set tMatrix = Nothing
```

That is all. All matrix interface initialization and de-initialization is automatically handled by the Matrix Correction 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 (tMatrix), then all properties and methods begin with the qualifier "tMatrix."

Constants

```
Global Const MAXCHAN% = 72          ' maximum elements per run
Global Const MAXSTD% = 64           ' maximum standards per run
Global Const MAXRAY% = 7            ' maximum xray symbols (ka,kb,la,lb,ma,mb," ") including
blank for non-analyzed
Global Const MAXELM% = 100          ' maximum elements
Global Const MAXINTF% = 5           ' maximum interferences per element
Global Const MAXMAN% = 16           ' maximum MAN assignments per element
Global Const MAXSPEC% = 9           ' maximum spectrometers (fixed + tunable) per run
Global Const MAXCRYS% = 6           ' maximum crystals per spectrometer
Global Const MAXCOEFF% = 9          ‘ maximum number of polynomial fit coefficients
Global Const MAXSETUP% = 256        ‘ maximum number of element setups

Global Const MAXZAF% = 10           ‘ maximum number of matrix correction options
Global Const MAXMACTYPE% = 6        ‘ maximum number of MAC options
```

Matrix Correction Methods

List of Methods

Matrix Methods

Public Sub MatrixGetMatrix(mLastElm As Long, mLastChan As Long, mTakeoffs() As Double, mKilovolts() As Double, mElements() As Long, mXrays() As Long, mStandards() As Long, mCations() As Double, mOxygens() As Double, mAtomicWts() As Double, mStdPercents() As Double, mBetaFactors() As Double, mAlphaFactors1() As Double, mAlphaFactors2() As Double, mAlphaFactors3() As Double)

Public Sub MatrixGetZAFStrings(zNumberOf as Long, zStrings() as String)

Public Sub MatrixGetMACStrings(mNumberOf as Long, mStrings() as String)

Public Sub MatrixSetZAFMode(zNumber as Long)

Public Sub MatrixSetMACMode(mNumber as Long)

Get Intensity from Database Methods

Public Sub MatrixGetStandards(sTakeoff As Double, sKilovolt As Double, sElement As Long, sXray As Long, sMotor As Long, sCrystal As Long, sNumberOf As Long, sNumbers() As Long, sPercents() As Double, sIntensities() As Double, sBetas() as Double, sDateTimes() As Double, sNames() As String)

Public Sub MatrixGetBackgrounds(bTakeoff As Double, bKilovolt As Double, bElement As Long, bXray As Long, bMotor As Long, bCrystal As Long, bNumberOf As Long, bNumbers() As Long, bIntensities() As Double, bZbars() As Double, bContinuumCorrections() As Double, bDateTimes() As Double, bNames() As String)

Public Sub MatrixGetInterferences(iTakeoff As Double, iKilovolt As Double, iElement As Long, iXray As Long, iMotor As Long, iCrystal As Long, iInterfElement As Long, iNumberOf As Long, iNumbers() As Long, iPercents() As Double, iIntensities() As Double, iBetas() as Double, iDateTimes() As Double, iNames() As String)

Crystal Methods

Public Sub MatrixGetCrystals(cMotor As Long, cNumberOf As Long, cNames() As String)

Public Sub MatrixGetCrystalNumber(cMotor As Long, cCrystal as String, cNumberOf As Long)

Public Sub MatrixGetCrystalFlipPosition(cMotor As Long, cPosition As Double)

Public Sub MatrixGetCurrentCrystal(cMotor As Long, cCrystal as String)

Misc Methods

Public Sub MatrixSetDebugMode(tdebugmode As Integer)

Public Sub MatrixGetElements(nElements As Long, nAtomicNumbers() As Long, nAtomicWts() As Double, nAtomicSymlos() As String, nAtomicSymups() As String)

Public Sub MatrixGetLinearFit(mOrder As Long, mNumberOf As Long, mXdata() As Double, mYdata() As Double, mCoef() As Double)

Public Sub MatrixCalculatePosition(pMethod As Long, pMode As Long, pMotor As Long, pCrystal As Long, pElement As Long, pXray As Long, pOrder As Long, pPosition as Double

Public Sub MatrixGetSetups(sTakeoff As Double, sKilovolt As Double, sElement As Long, sNumberOf As Long, sElements() As String, sXrays() As String, sMotors() As Long, sCrystals() As String, sOnpeaks() As Double, sHiPeaks() As Double, sLoPeaks() As Double, sBaselines() As Double, sWindows() As Double, sGains() As Double, sBiases() As Double, sInteDiffs() As Long, sDeadtimes() As Double, sDatetimes() As Double, sNames() As String)

Public Sub MatrixGetSetups2(sTakeoff As Double, sKilovolt As Double, sMotor As Long, sCrystal as Long, sNumberOf As Long, sElements() As String, sXrays() As String, sMotors() As Long, sCrystals() As String, sOnpeaks() As Double, sHiPeaks() As Double, sLoPeaks() As Double, sBaselines() As Double, sWindows() As Double, sGains() As Double, sBiases() As Double, sInteDiffs() As Long, sDeadtimes() As Double, sDatetimes() As Double, sNames() As String)

Detailed Description of Methods

Matrix Methods

Public Sub MatrixGetMatrix

(mLastElm As Long, mLastChan As Long, mTakeoffs() As Double, mKilovolts() As Double, mElements() As Long, mXrays() As Long, mStandards() As Long, mCations() As Double, mOxygens() As Double, mAtomicWts() As Double, mStdPercents() As Double, mBetaFactors()

As Double, mAlphaFactors1() As Double, mAlphaFactors2() As Double, mAlphaFactors3() As Double)

This method will return various elemental parameters and the three coefficient alpha factors for the specified conditions and elements.

Passed arrays:
mLastElm = number of emitter elements (with non-blank x-ray line)
mLastChan = number of absorbing elements (with blank x-ray line)
mTakeoffs#() = array of takeoff angles (1 to mLastElm&)
mKilovolts#() = array of kilovolts (1 to mLastElm&)
mElements() = element atomic numbers (1 to mLastChan&)
mXrays() = xray lines (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb, 7=absorber only) (1 to mLastChan&)
mStandards() = primary standard (number) assignments for standard beta factors (1 to mLastElm&)

Returned arrays:
mCations#() = number of stoichiometric cations (1 to mLastChan&)
mOxygens#() = number of stoichiometric oxygens (1 to mLastChan&)
mAtomicWts#() = atomic weights (1 to mLastChan&)
mStdPercents#() = primary standard weight percents (1 to LastElm&)
mBetaFactors#() = primary standard beta factors (1 to LastElm&)
mAlphaFactors1#() = intercept coefficients alpha factors emitter x absorber array (1 to mLastElm&, 1 to mLastChan&)
mAlphaFactors2#() = slope coefficients alpha factors emitter x absorber array (1 to mLastElm&, 1 to mLastChan&)
mAlphaFactors3#() = curvature coefficients alpha factors emitter x absorber array (1 to mLastElm&, 1 to mLastChan&)

Usage:
```
Const mLastElm& = 3       ' number of emitter elements (for example)
Const mLastChan& = 4      ' number of absorber elements (must be equal
to or larger than mLastElm)

Dim mTakeoffs(1 To mLastElm&) As Double, mKilovolts(1 To mLastElm&) As
Double
Dim mElements(1 To mLastChan&) As Long, mXrays(1 To mLastChan&) As Long
Dim mCations(1 To mLastChan&) As Double, mOxygens(1 To mLastChan&) As
Double, mAtomicWts(1 To mLastChan&) As Double
Dim mStandards(1 To mLastElm&) As Long
Dim mStdBetas(1 To mLastElm&) As Double, mStdPercents(1 To mLastElm&)
As Double

Dim mAlphaFactors1(1 To mLastElm&, 1 To mLastChan&) As Double
Dim mAlphaFactors2(1 To mLastElm&, 1 To mLastChan&) As Double
Dim mAlphaFactors3(1 To mLastElm&, 1 To mLastChan&) As Double

' Get matrix factors from active-X component
tMatrix.MatrixGetMatrix mLastElm&, mLastChan&, mTakeoffs#(),
mKilovolts#(), mElements&(), mXrays&(), mStandards&(), mCations#(),
mOxygens#(), mAtomicWts#(), mStdPercents(), mStdBetas#(),
mAlphaFactors1#(), mAlphaFactors2#(), mAlphaFactors3#()
```

Public Sub MatrixGetZAFStrings

(zNumberOf As Long, zStrings() As String)

Returns the available matrix correction options

Passed parameter
None

Returned parameter and array
zNumberOf& = number of matrix corrections
zStrings$() = matrix correction options

Usage:

```
Dim zNumberOf as long

Dim zStrings(1 to MAXZAF&) as String

tMatrix.MatrixGetZAFStrings zNumberOf&, zStrings$()
```

Public Sub MatrixGetMACStrings

(mNumberOf As Long, mStrings() As String)

Returns the available MAC options

Passed parameter
None

Returned parameter and array
mNumberOf& = number of MAC options
mStrings$() = MAC options

Usage:

```
Dim mNumberOf as long

Dim mStrings(1 to MAXMACTYPE&) as String

tMatrix.MatrixGetMACStrings mNumberOf&, mStrings$()
```

Public Sub MatrixSetZAFMode

(zMode as Long)

Sets the ZAF matrix correction option mode. Be sure to call MatrixGetMatrix again after calling this procedure to obtain the recalculated alpha-factors

Passed parameter
zMode& = ZAF matrix correction option specified by user (1 to 10)

Usage:

```
Dim zMode as long

tMatrix.MatrixSetZAFMode zMode&
```

Public Sub MatrixSetAlphaMode

(cMode as Long, p1Mode as Long, p2Mode as Boolean, p2Value as Double)

Sets the alpha factor correction mode. Be sure to call MatrixGetMatrix again after calling this procedure to obtain the recalculated alpha-factors

Passed parameter
cMode& = alpha factor correction mode specified by user (1, 2 or 3)
p1Mode& = Penepma binary mode (1 = do not used Penepma binaries, 2 = use Penepma binaries)
p2Mode = Penepma binary limit mode (false for do not use, true for use)
p2Value = Penepma limit value (50 to 99, default = 90)

Usage:

```
Dim cMode as long, p1Mode as Long, p2Mode as Boolean, p2Value as Double

tMatrix.MatrixSetAlphaMode cMode&, p1Mode&, p2Mode, P2Value#
```

Public Sub MatrixSetMACMode

(mMode as Long)

Sets the MAC option mode. Be sure to call MatrixGetMatrix again after calling this procedure to obtain the recalculated alpha-factors

Passed parameter
mMode& = matrix correction option specified by user

Usage:

```
Dim mMode as long

tMatrix.MatrixSetMACMode mMode&
```

Get Intensity from Database Methods

Public MatrixGetStandards

(sTakeoff As Double, sKilovolt As Double, sElement As Long, sXray As Long, sMotor As Long, sCrystal As Long, sNumberOf As Long, sNumbers() As Long, sPercents() As Double, sIntensities() As Double, sBetas() as Double, sDateTimes() As Double, sNames() As String)

Get standard intensities for the indicated element, x-ray, spectrometer, etc (from SETUP.MDB). The returned intensities are in cps/nA units and are already corrected for deadtime, beam drift and background.

Passed parameters and arrays:
sTakeoff# = takeoff angle
sKilovolt# = kilovolt
sElement& = element atomic number, use values 1 to MAXELM&
sXray& = element x-ray line, use values 1 to MAXRAY& (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb, 7=absorber only)
sMotor& = spectrometer number, use values 1 to MAXSPEC&
sCrystal& = crystal number on that spectrometer, use values 1 to MAXCRYS&

Returned parameters and arrays
sNumberOf& = number of standards returned
sNumbers&() = array of returned standard numbers
sPercents#() = array of returned standard weight percents
sIntensities#() = array of returned standard intensities
sBetas#() = array of returned standard beta factors
sDateTimes#() = array of standard dates and times (of acquisition)
sNames$() = array of returned standard names (string)

Usage:

```
Const MAXSTD& = 40

Dim n As Long
Dim sTakeoff As Double, sKilovolt As Double
Dim sElement As Long, sXray As Long
Dim sMotor As Long, sCrystal As Long

Dim sNumberOf As Long
Dim sNumbers(1 To MAXSTD&) As Long
Dim sPercents(1 To MAXSTD&) As Double
Dim sIntensities(1 To MAXSTD&) As Double
Dim sBetas(1 To MAXSTD&) as Double
Dim sDateTimes(1 To MAXSTD&) As Double
Dim sNames(1 To MAXSTD&) As String

sTakeoff# = Val(FormMAIN.TextTakeOff.Text)
sKilovolt# = Val(FormMAIN.TextKilovolts.Text)

sElement& = Val(FormMAIN.TextStdElement.Text)
sXray& = Val(FormMAIN.TextStdXray.Text)
sMotor& = Val(FormMAIN.TextStdMotor.Text)
sCrystal& = Val(FormMAIN.TextStdCrystal.Text)

' Get standard intensities from active-X component
tMatrix.MatrixGetStandards sTakeoff#, sKilovolt#, sElement&, sXray&,
sMotor&, sCrystal&, sNumberOf&, sNumbers&(), sPercents#(),
sIntensities#(), sBetas#(), sDateTimes#(), sNames$()
```

Public MatrixGetBackgrounds

(bTakeoff As Double, bKilovolt As Double, bElement As Long, bXray As Long, bMotor As Long, bCrystal As Long, bNumberOf As Long, bNumbers() As Long, bIntensities() As Double, bZbars() As Double, bContinuumCorrections() As Double, bDateTimes() As Double, bNames() As String)

Get the MAN background intensities (from SETUP2.MDB). The returned intensities are in cps/nA units and are already corrected for deadtime and beam drift (not background corrected!).

Passed parameters and arrays
bTakeoff# = takeoff angle
bKilovolt# = kilovolt
bElement = element atomic number, use values 1 to MAXELM&
bXray = element x-ray line, use values 1 to MAXRAY& (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb, 7=absorber only)
bMotor = spectrometer number, use values 1 to MAXSPEC&
bCrystal = crystal number on that spectrometer, use values 1 to MAXCRYS&

Returned parameter and arrays
bNumberOf = number of MAN backgrounds returned
bNumbers() = array of returned MAN standard numbers
bIntensities#() = array of returned MAN standard intensities (y plot axis)
bZbars#() = array of returned MAN standard zbars (x plot axis)
bContinuumCorrections#() = array of returned continuum intensity corrections
bDateTimes() = MAN standard dates and times (of acquisition)
bNames() = array of returned MAN standard names

Usage:

```
Const MAXMAN& = 16

Dim bTakeoff As Double, bKilovolt As Double
Dim bElement As Long, bXray As Long
Dim bMotor As Long, bCrystal As Long

Dim bNumberOf As Long
Dim bNumbers(1 To MAXMAN&) As Long
Dim bIntensities(1 To MAXMAN&) As Double
Dim bZbars(1 To MAXMAN&) As Double
Dim bContinuumCorrections(1 To MAXMAN&) As Double
Dim bDateTimes(1 To MAXMAN&) As Double
Dim bNames(1 To MAXMAN&) As String

' Get background intensities from active-X component
tMatrix.MatrixGetBackgrounds bTakeoff#, bKilovolt#, bElement&, bXray&,
bMotor&, bCrystal&, bNumberOf&, bNumbers&(), bIntensities#(),
bZbars#(), bContinuumCorrections#(), bDateTimes#(), bNames$()
```

Public Sub MatrixGetInterferences

(iTakeoff As Double, iKilovolt As Double, iElement As Long, iXray As Long, iMotor As Long, iCrystal As Long, iInterfElement As Long, iNumberOf As Long, iNumbers() As Long, iPercents() As Double, iIntensities() As Double, iBetas() as Double, iDateTimes() As Double, iNames() As String)

Get the interference intensities (from SETUP3.MDB). The returned intensities are in cps/nA units and are already corrected for deadtime, beam drift and background.

Passed parameters and arrays
iTakeoff# = takeoff angle
iKilovolt# = kilovolt
iElement = element atomic number, use values 1 to MAXELM&
iXray = element x-ray line, use values 1 to MAXRAY& (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb, 7=absorber only)
iMotor = spectrometer number, use values 1 to MAXSPEC&
iCrystal = crystal number on that spectrometer, use values 1 to MAXCRYS&
iInterfElement = interfering element atomic number which is present in the standard, use values 1 to MAXELM%

Returned parameter and arrays
iNumberOf = number of interferences standards returned
iNumbers() = array of returned interference standard numbers
iPercents#() = array of returned standard weight percents (of the interfering element)
iIntensities#() = array of returned interference standard intensities
iBetas#() = array of returned interference standard beta factors
iDateTimes() = interference standard dates and times (of acquisition)
iNames() = array of returned interference standard names

Usage:

```
Const MAXSTD& = 40

Dim iTakeoff As Double, iKilovolt As Double
Dim iElement As Long, iXray As Long
Dim iMotor As Long, iCrystal As Long
Dim iInterfElement As Long

Dim iNumberOf As Long
Dim iNumbers(1 To MAXSTD&) As Long
Dim iPercents(1 To MAXSTD&) As Double
Dim iIntensities(1 To MAXSTD&) As Double
Dim iBetas(1 To MAXSTD&) as Double
Dim iDateTimes(1 To MAXSTD&) As Double
Dim iNames(1 To MAXSTD&) As String

' Get standard intensities from active-X component
tMatrix.MatrixGetInterferences iTakeoff#, iKilovolt#, iElement&,
iXray&, iMotor&, iCrystal&, iInterfElement&, iNumberOf&, iNumbers&(),
iPercents#(), iIntensities#(), iBetas#(), iDateTimes#(), iNames$()
```

Crystal Methods

Public Sub MatrixGetCrystals

(cMotor As Long, cNumberOf As Long, cNames() As String)

Returns the available crystals for a specific spectrometer

Passed parameter
cMotor& = spectrometer number, use values 1 to MAXSPEC&

Returned parameter and array
cNumberOf& = number of crystals on the specified spectrometer
cNames$() = crystal names of each crystal

Usage:

```
Dim cMotor as Long
Dim cNumberOf as long

Dim cNames(1 to MAXCRYS&) as String

tMatrix.MatrixGetCrystals cMotor&, cNumberOf&, cNames$()
```

Public Sub MatrixGetCrystalNumber

(cMotor As Long, cCrystal as String, cNumber As Long)

Returns the driver level crystal number for the specified spectro and crystal string. Note spectrometer must be in proper position before flipping crystal to avoid damage to crystal (see procedure MatrixGetCrystalFlipPosition).

Passed parameter
cMotor& = spectrometer number, use values 1 to MAXSPEC&
cCrystal$ = crystal name of selected crystal

Returned parameter
cNumber& = number of crystal for crystal flip driver level call

Usage:

```
Dim cMotor as Long
Dim cCrystal as String
Dim cNumber as long

tMatrix.MatrixGetCrystalNumber cMotor&, cCrystal$, cNumber&

Special note: the actual crystal number specified in the driver level
call depends on the interface type. However, some interfaces do not use
an integer value, for example the SESAME and SX50/51 interfaces use the
first 4 characaters of the actual crystal name.

Also the AMB interface uses a bit mask written to the PMC DCX I/O
channels:

       09  J3   Channel 11 (output)  Crystal flip circuit strobe
       (low=enabled)
       08  J3   Channel 12 (output)  Crystal flip I/O bit 1 (high=active)
       07  J3   Channel 13 (output)  Crystal flip I/O bit 2 (high=active)
       06  J3   Channel 14 (output)  Crystal flip I/O bit 3 (high=active)
       05  J3   Channel 15 (output)  Crystal flip I/O bit 4 (high=active)
       03  J3   Channel 16 (output)  Crystal flip I/O bit 5 (high=active)

The procedure to flip a crystal using the PMC DCX board is to:

1.  Set the PMC DCX board I/O channel 11 high (FALSE) to disable
    crystal flipping
2.  Write the returned bit mask to channels 12 to 16 (bit set = TRUE)
3.  Now strobe channel 11 (0.05 second delay)
4.  Wait two seconds for the crystal to flip
```

Public Sub MatrixGetCrystalFlipPosition

(cMotor As Long, cPosition As Double)

Returns the crystal flip (change) spectrometer position for the specified spectro. Use this position to move spectrometer to before changing crystal. Note that a crystal flip position of zero means that the instrument does not have automated crystal flip capability.

Passed parameter
cMotor& = spectrometer number, use values 1 to MAXSPEC&

Returned parameter
cPosition# = spectrometer position for safe crystal flip operation

Usage:

```
Dim cMotor as Long
Dim cPosition as Double

tMatrix.MatrixGetCrystalFlipPosition cMotor&, cPosition#
```

Public Sub MatrixGetCurrentCrystal

(cMotor As Long, cCrystal as String)

Returns the current crystal for the specified spectro. If this value is the same as the user selected crystal, then no crystal flip is necessary.

Passed parameter
cMotor& = spectrometer number, use values 1 to MAXSPEC&

Returned parameter
CString$ = current spectrometer crystal string

Usage:

```
Dim cMotor as Long
Dim cCrystal as String


tMatrix.MatrixGetCurrentCrystal cMotor&, cCrystal$
```

Misc Methods

Public Sub MatrixSetDebugMode

(tdebugmode As Boolean)

Set the debugmode for testing purposes

Usage:

```
tMatrix.MatrixSetDebugMode (True)
```

Public Sub MatrixGetLinearFit

(mOrder As Long, mNumberOf As Long, mXdata() As Double, mYdata() As Double, mCoef() As Double)

Return linear least square fit coefficients of specified degree. The polynomial is of the form: mCoeff(1) + mCoeff(2)*x + ... + mCoeff(mOrder)*x**(mOrder-1)

Passed parameters and arrays
mOrder = degree of fit, 0 = constant, 1 = linear (two points), 2 = parabolic (3 points), etc.
mNumberOf = the number of pairs of data points to fit
mXdata(), mYdata() = x and y data

Returned array
mCoef() = polynomial fit coefficients (1 to MAXCOEFF%)

Usage (example shown for Si Ka at 15 keV):

```
Dim mOrder as Long, mNumberOf as long

mNumber& = 4
ReDim mXdata(1 to mNumberOf&) as Double
ReDim mYdata(1 to mNumberOf&) as Double

Dim mCoef(1 to MAXCOEFF&) as Double

MXdata#(1) = 10.42668                ‘ MgO Zbar
MYdata#(1) = .9582438 * 1.568938     ‘ MgO MAN intensity and MANbeta

MXdata#(1) = 16.39201                ‘ TiO2 Zbar
MYdata#(1) = 1.712424 * 1.21558      ‘ TiO2 MAN intensity and MANbeta

MXdata#(1) = 21.16582                ‘ MnO Zbar
MYdata#(1) = .9582438 * 1.326036     ‘ MnO MAN intensity and MANbeta

MXdata#(1) = 22.94331                ‘ CoO Zbar
MYdata#(1) = 2.295885 * 1.439189     ‘ CoO MAN intensity and MANbeta

tMatrix.MatrixGetLinearFit mOrder&, mNumberOf&, mXdata#(), mYdata#(),
mCoef#()
```

Public Sub MatrixCalculatePosition

(pMethod As Long, pMode As Long, pMotor As Long, pCrystal As Long, pElement As Long, pXray As Long, pOrder As Long, pPosition as Double)

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

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

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

Usage:
```
Dim pMethod as Long, pMode as Long, pMotor as Long
Dim pCrystal as Long, pElement as Long, pXray as Long
Dim pOrder as Long
Dim pPosition as Double

pMethod& = 1      ‘ calculate theoretical position
pMode& = 0        ‘ calculate on-peak position
pMotor& = 2       ‘ spectrometer 2
pCrystal& = 2     ‘ crystal 2
pElement& = 22    ‘ Ti
pXray& = 1        ‘ Ka
```

```
pOrder% = 1       ‘ first order diffraction

Call MatrixCalculatePosition(pMethod&, pMode&, pMotor&,
pCrystal&, pElement&, pXray&, pOrder&, pPosition#)
```

Public MatrixGetSetups

(sTakeoff As Double, sKilovolt As Double, sElement As Long, sNumberOf As Long, sElements() As String, sXrays() As String, sMotors() As Long, sCrystals() As String, sOnpeaks() As Double, sHiPeaks() As Double, sLoPeaks() As Double, sBaselines() As Double, sWindows() As Double, sGains() As Double, sBiases() As Double, sInteDiffs() As Long, sDeadtimes() As Double, sDatetimes() As Double, sNames() As String)

Returns element setups (from SETUP.MDB) based on takoff, keV and element number

Passed parameters
sTakeoff# = takeoff angle
sKilovolt# = kilovolts
sElement = element atomic number to search for, use values 1 to MAXELM%

Returned parameter and arrays
sNumberOf = number of elements setups returned
sElements = array of element symbols
sXrays = array of element x-ray symbols (ka,kb,la,lb,ma,mb) (emitters only)
sMotors = array of spectrometer numbers, (values 1 to MAXSPEC%)
sCrystals = array of crystal names on that spectrometer
sOnPeaks = array of on peak positions on that spectrometer
sHiPeaks = array of hi peak positions on that spectrometer
sLoPeaks = array of lo peak positions on that spectrometer
sBaselines = array of PHA baselines on that spectrometer
sWindows = array of PHA windows on that spectrometer
sGains = array of PHA gains on that spectrometer
sBiases = array of PHA biases on that spectrometer
sInteDiffs = array of PHA InteDiff modes on that spectrometer
sDeadtimes = array of PHA deadtimes on that spectrometer
sDateTimes() = array of sample dates and times (of acquisition)
sNames() = array of returned sample names

Public MatrixGetSetups2

(sTakeoff As Double, sKilovolt As Double, sMotor As Long, sCrystal as Long, sNumberOf As Long, sElements() As String, sXrays() As String, sMotors() As Long, sCrystals() As String, sOnpeaks() As Double, sHiPeaks() As Double, sLoPeaks() As Double, sBaselines() As Double, sWindows() As Double, sGains() As Double, sBiases() As Double, sInteDiffs() As Long, sDeadtimes() As Double, sDatetimes() As Double, sNames() As String)

Returns element setups (from SETUP.MDB) based on takoff, keV and spectrometer number (also crystal number if non-zero)

Passed parameters
sTakeoff# = takeoff angle
sKilovolt# = kilovolts
sMotor = spectrometer number to search for, use values 1 to MAXSPEC%
sCrystal = spectrometer crystal number to search for, use values 0 to MAXCRYS% (0 = ignore)

Returned parameter and arrays
sNumberOf = number of elements setups returned
sElements = array of element symbols
sXrays = array of element x-ray symbols (ka,kb,la,lb,ma,mb) (emitters only)
sMotors = array of spectrometer numbers, (values 1 to MAXSPEC%)
sCrystals = array of crystal names on that spectrometer
sOnPeaks = array of on peak positions on that spectrometer

sHiPeaks = array of hi peak positions on that spectrometer
sLoPeaks = array of lo peak positions on that spectrometer
sBaselines = array of PHA baselines on that spectrometer
sWindows = array of PHA windows on that spectrometer
sGains = array of PHA gains on that spectrometer
sBiases = array of PHA biases on that spectrometer
sInteDiffs = array of PHA InteDiff modes on that spectrometer
sDeadtimes = array of PHA deadtimes on that spectrometer
sDateTimes() = array of sample dates and times (of acquisition)
sNames() = array of returned sample names

Visual Basis Code Examples

Visual Basic- Coding Suggestions

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

Then click on Project | References and scroll down to the Matrix Correction 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: TestMatrix.vbp.

Note the following constants:

```
Global Const MAXSETUP& = 256   ' maximum number of element setups to return
Global Const MAXSTD& = 64      ' up to 64 standards per emitting element
Global Const MAXMAN& = 16      ' up to 16 MAN standards per emitting element
Global Const MAXINTF& = 5      ' up to 5 interfering elements per emitting element
Global Const MAXCOEF% = 3      ' maximum MAN fir coefficients
Global Const MAXCRYS& = 6      ' maximum crystals per spectrometer
```

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

Declare Matrix Object Example

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

```
' (c) Copyright 1995-2010 by John J. Donovan
Option Explicit

Dim tMatrix As ClassMatrix   ' Matrix Interface object (shared by all
modules)
```

Load Matrix Object Example

This code fragment shows how to load the Matrix Correction object for use in your code.

```
Sub TestMatrixLoadRemoteObject()
' Load the Matrix interface object (call from Form Load event)

ierror = False
```

```
On Error GoTo TestMatrixLoadRemoteObjectError

' Load matrix active-x object
Set tMatrix = New ClassMatrix

' Set debug mode for testing
tMatrix.MatrixSetDebugMode (True)
If ierror Then Exit Sub

Exit Sub

' Errors
TestMatrixLoadRemoteObjectError:
Set tMatrix = Nothing
MsgBox Error$, vbOKOnly + vbCritical, "TestMatrixLoadRemoteObject"
ierror = True
Unload FormMAIN
Exit Sub

End Sub
```

Unload Matrix Object Example

This code fragment shows how to unload the Matrix Correction object after you are finished.

```
Sub TestMatrixUnLoadRemoteObject()
' UnLoad the Matrix interface object (call from Form Unload event)

ierror = False
On Error GoTo TestMatrixUnLoadRemoteObjectError

' Set debug mode for exiting
If tMatrix Is Nothing Then Exit Sub
tMatrix.MatrixSetDebugMode (False)
If ierror Then Exit Sub

If Not tMatrix Is Nothing Then Set tMatrix = Nothing
DoEvents

Exit Sub

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

End Sub
```

Visual Basic- Sample Code (Get Matrix Factors)

This Visual Basic code example gets the matrix alpha factors for a given condition and set of elements.

```
Sub TestMatrixGetAlphaFactors()
' Get matrix of alpha factors (1 to mLastElm&, 1 to mLastChan&)

' Passed arrays
'  mLastElm = number of emitter elements (with non-blank x-ray line)
'  mLastChan = number of absorbing elements (with blank x-ray line)
(mLastChan& must be >= mLastElm&)
'  mTakeoffs#() = array of takeoff angles (1 to mLastElm)
'  mKilovolts#() = array of kilovolts (1 to mLastElm)
'  mElements&() = element atomic numbers  (1 to mLastChan)
'  mXrays&() = xray lines (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb,
7=absorber only) (1 to mLastChan)
'  mStandards&() = primary standard (number) assignments for standard
beta factors (1 to mLastElm&)
'
' Returned arrays
'  mCations#() = number of stoichiometric cations (1 to mLastChan&)
'  mOxygens#() = number of stoichiometric oxygens (1 to mLastChan&)
'  mAtomicWts#() = atomic weights (1 to mLastChan&)
'  mStdPercents#() = primary standard weight percents (1 to mLastElm&)
'  mStdBetas#() =  primary standard beta factors (1 to mLastElm)
'  mAlphaFactors1#() = intercept coefficients alpha factors emitter x
absorber array (1 to mLastElm, 1 to mLastChan)
'  mAlphaFactors2#() = slope coefficients alpha factors emitter x
absorber array (1 to mLastElm, 1 to mLastChan)
'  mAlphaFactors3#() = curvature coefficients alpha factors emitter x
absorber array (1 to mLastElm, 1 to mLastChan)

ierror = False
On Error GoTo TestMatrixGetAlphaFactorsError

Const mLastElm& = 3       ' number of emitter elements
Const mLastChan& = 4      ' number of absorber elements (must be equal
to or larger than LastElm)

Dim chan As Long, emitter As Long, absorber As Long

Dim mTakeoffs(1 To mLastElm&) As Double, mKilovolts(1 To mLastElm&) As
Double
Dim mElements(1 To mLastChan&) As Long, mXrays(1 To mLastChan&) As Long
Dim mCations(1 To mLastChan&) As Double, mOxygens(1 To mLastChan&) As
Double, mAtomicWts(1 To mLastChan&) As Double
Dim mStandards(1 To mLastElm&) As Long
Dim mStdBetas(1 To mLastElm&) As Double, mStdPercents(1 To mLastElm&)
As Double

Dim mAlphaFactors1(1 To mLastElm&, 1 To mLastChan&) As Double
Dim mAlphaFactors2(1 To mLastElm&, 1 To mLastChan&) As Double
Dim mAlphaFactors3(1 To mLastElm&, 1 To mLastChan&) As Double

' Load emitters
For chan& = 1 To mLastElm&
mTakeoffs#(chan&) = Val(FormMAIN.TextTakeOff.Text)
mKilovolts#(chan&) = Val(FormMAIN.TextKilovolts.Text)
mElements&(chan&) = Val(FormMAIN.TextElements(chan& - 1).Text)
mXrays&(chan&) = Val(FormMAIN.TextXrays(chan& - 1).Text)
mStandards&(chan&) = Val(FormMAIN.TextStandards(chan& - 1).Text)
Next chan&

' Load absorbers
For chan& = mLastElm& + 1 To mLastChan&
mElements&(chan&) = Val(FormMAIN.TextElements(chan& - 1).Text)
mXrays&(chan&) = Val(FormMAIN.TextXrays(chan& - 1).Text)      ' must be
MAXRAY& for absorber
Next chan&

' Get matrix factors from active-X component
Screen.MousePointer = vbHourglass
tMatrix.MatrixGetMatrix mLastElm&, mLastChan&, mTakeoffs#(),
mKilovolts#(), mElements&(), mXrays&(), mStandards&(), mCations#(),
mOxygens#(), mAtomicWts#(), mStdPercents(), mStdBetas#(),
mAlphaFactors1#(), mAlphaFactors2#(), mAlphaFactors3#()
Screen.MousePointer = vbDefault

' Display results
msg$ = ""
FormMAIN.TextResults.Text = msg$
For chan& = 1 To mLastElm&
msg$ = msg$ & Str$(CSng(mTakeoffs#(chan&))) & ", " &
Str$(CSng(mKilovolts#(chan&))) & "," & Str$(mElements&(chan&)) & ", " &
Str$(mXrays&(chan&)) & ", " & Str$(mStandards&(chan&)) & ", " &
```

```
Str$(CSng(mStdPercents#(chan&))) & ", " & Str$(CSng(mStdBetas#(chan&)))
& vbCrLf
Next chan&

For chan& = mLastElm& + 1 To mLastChan&
msg$ = msg$ & Str$(mElements&(chan&)) & ", " & Str$(mXrays&(chan&)) &
vbCrLf
Next chan&

' Display alpha-factors
msg$ = msg$ & vbCrLf
For emitter& = 1 To mLastElm&
For absorber& = 1 To mLastChan&
msg$ = msg$ & Str$(mElements&(emitter&)) & " " &
Str$(mXrays&(emitter&)) & " by " & Str$(mElements&(absorber&)) & ", " &
Str$(CSng(mAlphaFactors1#(emitter&, absorber&))) & ", " &
Str$(CSng(mAlphaFactors2#(emitter&, absorber&))) & ", " &
Str$(CSng(mAlphaFactors3#(emitter&, absorber&))) & vbCrLf
Next absorber&
Next emitter&

FormMAIN.TextResults.Text = msg$
Exit Sub

' Errors
TestMatrixGetAlphaFactorsError:
Screen.MousePointer = vbDefault
MsgBox Error$, vbOKOnly + vbCritical, "TestMatrixGetAlphaFactors"
ierror = True
Exit Sub

End Sub
```

Visual Basic- Sample Code (Get Standards)

This Visual Basic code example gets the standard intensities from the SETUP.MDB element setup database.

```
Sub TestMatrixGetStandards()
' Get standard intensities for the indicated element, x-ray,
spectrometer, etc

' Passed arrays
'  sTakeoff# = takeoff angle
'  sKilovolt# = kilovolt
'  sElement& = element atomic number (1 to MAXELM&)
'  sXray& = element x-ray line (1=Ka, 2=Kb, 3=La, 4=Lb, 5=Ma, 6=Mb,
7=absorber only) (1 to MAXRAY&)
'  sMotor& = spectrometer number (1 to MAXSPEC&)
'  sCrystal& = crystal number on that spectrometer (1 to MAXCRYS&)
'
' Returned parameters and arrays
'  sNumberOf& = number of standards returned
'  sNumbers&() = array of returned standard numbers
'  sPercents#() = array of returned standard weight percents
'  sIntensities#() = array of returned standard intensities
‘  sBetas#() = array of returned standard beta factors
'  sDateTimes#() = standard dates and times (of acquisition)
'  sNames$() = array of returned standard names (string)

ierror = False
On Error GoTo TestMatrixGetStandardsError

Const MAXSTD& = 64

Dim n As Long
Dim sTakeoff As Double, sKilovolt As Double
Dim sElement As Long, sXray As Long
Dim sMotor As Long, sCrystal As Long

Dim sNumberOf As Long
Dim sNumbers(1 To MAXSTD&) As Long
Dim sPercents(1 To MAXSTD&) As Double
Dim sIntensities(1 To MAXSTD&) As Double
Dim sBetas(1 To MAXSTD&) as Double
Dim sDateTimes(1 To MAXSTD&) As Double
Dim sNames(1 To MAXSTD&) As String

sTakeoff# = Val(FormMAIN.TextTakeOff.Text)
sKilovolt# = Val(FormMAIN.TextKilovolts.Text)

sElement& = Val(FormMAIN.TextStdElement.Text)
sXray& = Val(FormMAIN.TextStdXray.Text)
sMotor& = Val(FormMAIN.TextStdMotor.Text)
sCrystal& = Val(FormMAIN.TextStdCrystal.Text)

' Get standard intensities from active-X component
Screen.MousePointer = vbHourglass
tMatrix.MatrixGetStandards sTakeoff#, sKilovolt#, sElement&, sXray&,
sMotor&, sCrystal&, sNumberOf&, sNumbers&(), sPercents#(),
sIntensities#(), sBetas#(), sDateTimes#(), sNames$()
Screen.MousePointer = vbDefault

' Display returned results
msg$ = ""
FormMAIN.TextResults.Text = msg$
msg$ = "Standardizations for " & Str$(CSng(sTakeoff#)) & ", " &
Str$(CSng(sKilovolt#)) & "," & Str$(sElement&) & ", " & Str$(sXray&) &
", spectro " & Str$(sMotor&) & ", crystal " & sCrystal& & vbCrLf
If sNumberOf& > 0 Then
For n& = 1 To sNumberOf&
msg$ = msg$ & Str$(sNumbers&(n&)) & ", " & Str$(CSng(sPercents#(n&))) &
", " & Str$(CSng(sIntensities#(n&))) & ", " & Str$(CSng(sBetas#(n&))) &
", " & sNames$(n&) & vbCrLf
Next n&

Else
msg$ = msg$ & "No standardizations found in SETUP.MDB"
End If

FormMAIN.TextResults.Text = msg$

' Load listbox with results
FormMAIN.ListStandards.Clear
For n& = 1 To sNumberOf&
msg$ = Format$(sNumbers&(n&)) & ", " & Format$(CSng(sPercents#(n&))) &
", " & Format$(CSng(sIntensities#(n&))) & ", "
FormMAIN.ListStandards.AddItem msg$
Next n&
FormMAIN.ListStandards.ListIndex = sNumberOf& - 1

Exit Sub

' Errors
TestMatrixGetStandardsError:
Screen.MousePointer = vbDefault
MsgBox Error$, vbOKOnly + vbCritical, "TestMatrixGetStandards"
ierror = True
Exit Sub

End Sub
```

Glossary of Terms

ActiveX EXE

Components provide reusable code in the form of objects. An application that uses a component’s code, by creating objects and calling their properties and methods, is referred to as a *client*.

Components can run either in-process or out-of-process with respect to the clients that use their objects. An out-of-process component, or ActiveX EXE, runs in its own address space. The client is usually an application running in another process.

The fact that an out-of-process component runs in its own process means that a client can tell it to do something, and then go about its business while the component does the work.

Error Handler

If you don't use an On Error statement, any run-time error that occurs is fatal; that is, an error message is displayed and execution stops.

An "enabled" error handler is one that is turned on by an On Error statement; an "active" error handler is an enabled handler that is in the process of handling an error. If an error occurs while an error handler is active (between the occurrence of the error and a Resume, Exit Sub, Exit Function, or Exit Property statement), the current procedure's error handler can't handle the error. Control returns to the calling procedure. If the calling procedure has an enabled error handler, it is activated to handle the error. If the calling procedure's error handler is also active, control passes back through previous calling procedures until an enabled, but inactive, error handler is found. If no inactive, enabled error handler is found, the error is fatal at the point at which it actually occurred. Each time the error handler passes control back to a calling procedure, that procedure becomes the current procedure. Once an error is handled by an error handler in any procedure, execution resumes in the current procedure at the point designated by the Resume statement.

Note : An error-handling routine is not a Sub procedure or Function procedure. It is a section of code marked by a line label or line numbe

Index