

XRawfile OCX Documentation

Xcalibur Raw File OCX Documentation (XRawfile.ocx)

XRawfile OCX Documentation

1. Enumerated Types

Several functions expect or return a parameter of type 'long' that defines, typically, a type parameter.

Sample Type:

Sample type is returned in the function call GetSeqRowSampleType(...). The returned value has the following meaning.

<u>Value</u>	<u>Sample Type</u>
0	Unknown
1	Blank
2	QC
3	Standard Clear (None)
4	Standard Update (None)
5	Standard Bracket (Open)
6	Standard Bracket Start (multiple brackets)
7	Standard Bracket End (multiple brackets)

Controller Type:

Controller type determines the type of data being accessed. This type is set or returned in the calls to Get or Set controller information. This value has the following meaning.

<u>Value</u>	<u>Controller Type</u>
-1	No device
0	MS
1	Analog
2	A/D card
3	PDA
4	UV

Cutoff Type:

Cutoff type is specified in calls to GetMassListXYZ(...). The purpose of this cutoff type is to determine how the cutoff value is interpreted. This value has the following meaning.

<u>Value</u>	<u>Cutoff Type</u>
0	None (all values returned)
1	Absolute (in intensity units)
2	Relative (to base peak)

Chromatogram Type:

Chromatogram type is specified in the function call GetChroData(...). The value of this field depends on the current controller and whether or not this is the first chromatogram type parameter or the second chromatogram type parameter to this function. This value has the following meaning.

For MS devices (chromatogram trace type values are in parentheses):

Chro Type 1	Chro Operator	Chro Type 2
Mass Range (0)	+ or -	Mass Range (0)
TIC (1)	-	Mass Range (0)
TIC (1)	-	Base Peak (1)
Base Peak (2)	+ or -	Mass Range (0)

XRawfile OCX Documentation

For PDA devices (chromatogram trace type values are in parentheses):

Chro Type 1	Chro Operator	Chro Type 2
Wavelength Range (0)	+ or -	Wavelength Range (0)
Total Scan (1)	-	Wavelength Range (0)
Total Scan (1)	-	Spectrum Maximum (1)
Spectrum Maximum (2)	+ or -	Wavelength Range (0)

For UV devices (chromatogram trace type values are in parentheses):

Chro Type 1	Chro Operator	Chro Type 2
Channel A (0)	+ or -	Channel B (0)
Channel A (0)	+ or -	Channel C (1)
Channel A (0)	+ or -	Channel D (2)
Channel B (1)	+ or -	Channel A (0)
Channel B (1)	+ or -	Channel C (1)
Channel B (1)	+ or -	Channel D (2)
Channel C (2)	+ or -	Channel A (0)
Channel C (2)	+ or -	Channel B (1)
Channel C (2)	+ or -	Channel D (2)
Channel D (3)	+ or -	Channel A (0)
Channel D (3)	+ or -	Channel B (1)
Channel D (3)	+ or -	Channel C (2)

For Analog devices (chromatogram trace type values are in parentheses):

Chro Type 1	Chro Operator	Chro Type 2
Analog 1 (0)	+ or -	Analog 2 (0)
Analog 1 (0)	+ or -	Analog 3 (1)
Analog 1 (0)	+ or -	Analog 4 (2)
Analog 2 (1)	+ or -	Analog 1 (0)
Analog 2 (1)	+ or -	Analog 3 (1)
Analog 2 (1)	+ or -	Analog 4 (2)
Analog 3 (2)	+ or -	Analog 1 (0)
Analog 3 (2)	+ or -	Analog 2 (1)
Analog 3 (2)	+ or -	Analog 4 (2)
Analog 4 (3)	+ or -	Analog 1 (0)
Analog 4 (3)	+ or -	Analog 2 (1)
Analog 4 (3)	+ or -	Analog 3 (2)

XRawfile OCX Documentation

For A/D card devices (chromatogram trace type values are in parentheses):

Chro Type 1	Chro Operator	Chro Type 2
A/D Card Ch. 1 (0)	+ or -	A/D Card Ch. 2 (0)
A/D Card Ch. 1 (0)	+ or -	A/D Card Ch. 3 (1)
A/D Card Ch. 1 (0)	+ or -	A/D Card Ch. 4 (2)
A/D Card Ch. 2 (1)	+ or -	A/D Card Ch. 1 (0)
A/D Card Ch. 2 (1)	+ or -	A/D Card Ch. 3 (1)
A/D Card Ch. 2 (1)	+ or -	A/D Card Ch. 4 (2)
A/D Card Ch. 3 (2)	+ or -	A/D Card Ch. 1 (0)
A/D Card Ch. 3 (2)	+ or -	A/D Card Ch. 2 (1)
A/D Card Ch. 3 (2)	+ or -	A/D Card Ch. 4 (2)
A/D Card Ch. 4 (3)	+ or -	A/D Card Ch. 1 (0)
A/D Card Ch. 4 (3)	+ or -	A/D Card Ch. 2 (1)
A/D Card Ch. 4 (3)	+ or -	A/D Card Ch. 3 (2)

Chromatogram Operator:

The chromatogram operator type is specified in the function call GetChroData(...). This value has the following meaning.

<u>Value</u>	<u>Chro Operator</u>
0	None (single chro only)
1	Minus (subtract chro 2 from chro 1)
2	Plus (add chro 1 and chro 2)

Smoothing Type:

The smoothing type is specified in the function call GetChroData(...). This value has the following meaning.

<u>Value</u>	<u>Smoothing Type</u>
0	None (no smoothing)
1	Boxcar
2	Gaussian

XRawfile OCX Documentation

2. Error Codes

// Error codes returned to caller
cINO_DATA_PRESENT

= 2;

This code does not typically indicate an error. This code may be returned if optional data is not contained in the current raw file.

cISUCCESS

= 1;

This code indicates that the function call to the OCX was processed without error.

cIFAILED

= 0;

This code indicates that a general error has occurred. This code may be returned whenever an error of indeterminate origin occurs.

cIRAW_FILE_INVALID

= -1;

This code will be returned if no valid raw file is currently open.

cCURRENT_CONTROLLER_INVALID

= -2;

This code will be returned if no current controller has been specified.

cIOPERATION_NOT_SUPPORTED_ON_CURRENT_CONTROLLER = -3;

This code will be returned if the requested action is inappropriate for the currently defined controller. Some functions only apply to specific controllers. This code may also be returned if a parameter is passed in a call that is not supported by the current controller. For example, scan filters may only be passed in calls when the current controller is of mass spectrometer type (MS_DEVICE).

cIPARAMETER_INVALID

= -4;

This code will be returned if an invalid parameter is passed in a function call to the OCX. This can occur if a parameter is out of range or initialized incorrectly.

cIFILTER_FORMAT_INCORRECT

= -5;

This code will be returned if an incorrectly formatted scan filter is passed in a function call. See the topic **scan filters – format, definition** for scan filter format specifications.

cIMASS_RANGE_FORMAT_INCORRECT

= -6;

This code will be returned if an incorrectly formatted mass range is passed in a function call. Mass ranges should be have the same format as entered in Xcalibur applications.

XRawfile OCX Documentation

3. Function Reference

XRawfile OCX Documentation

Open

long **Open**(LPCTSTR szFileName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

szFileName A NULL terminated string containing the fully qualified path name of the raw file to open.

Remarks

Opens a raw file for reading only. This function must be called before attempting to read any data from the raw file.

Example

```
// example for Open
TCHAR* szPathName[] = _T("c:\\xcalibur\\examples\\data\\steroids15.raw");
long nRet = XRawfileCtrl.Open( szPathName );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error opening file"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

Close

long Close();

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

This function has no parameters.

Remarks

Closes a raw file and frees the associated memory.

Example

```
// example for Close
long nRet = XRawfileCtrl.Close();
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error closing file"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

GetFileName

long **GetFileName**(BSTR FAR* pbstrFileName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrFileName A valid pointer to a BSTR variable. This variable must exist and be initialized to NULL.

Remarks

Returns the fully qualified path name of an open raw file.

Example

```
// example for GetFileName
BSTR bstrFileName = NULL;
long nRet = XRawfileCtrl.GetFileName( &bstrFileName );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting file name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString( bstrFileName );
```

XRawfile OCX Documentation

GetCreatorID

long GetCreatorID(BSTR FAR* pbstrCreatorID);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrCreatorID A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the creator ID. The creator ID is the logon name of the user when the raw file was acquired.

Example

```
// example for GetCreatorID
BSTR bstrCreatorID = NULL;
long nRet = XRawfileCtrl.GetCreatorID ( &bstrCreatorID );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting creator ID"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrCreatorID);
```

XRawfile OCX Documentation

GetVersionNumber

long **GetVersionNumber**(long FAR* pnVersion);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnVersion A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the file format version number.

Example

```
// example for GetVersionNumber
long nVersionNumber;
long nRet = XRawfileCtrl.GetVersionNumber ( &nVersionNumber );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting version number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetCreationDate

long GetCreationDate(**DATE FAR*** pCreationDate);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pCreationDate A valid pointer to a DATE variable. This variable must exist.

Remarks

Returns the file creation date in DATE format.

Example

```
// example for GetCreationDate
DATE CreationDate;
long nRet = XRawfileCtrl.GetCreationDate ( &CreationDate );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting creation date"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

IsError

long IsError(BOOL FAR* pblsError);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pblsError A valid pointer to a variable of type BOOL. This variable must exist.

Remarks

Returns the error state flag of the raw file. A return value of TRUE indicates that an error has occurred. Information about the error can be obtained by calling the **GetErrorCode** or **GetErrorMessage** functions.

Example

```
// example for IsError
BOOL bError;
long nRet = XRawfileCtrl.IsError ( &bError );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting error flag"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

IsNewFile

long IsNewFile(BOOL FAR* pblsNewFile);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pblsNewFile A valid pointer to a variable of type BOOL. This variable must exist.

Remarks

Returns the creation state flag of the raw file. A return value of TRUE indicates that the file has not previously been saved.

Example

```
// example for IsNewFile
BOOL bNewFile;
long nRet = XRawfileCtrl.IsNewFile ( &bNewFile );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting new file flag"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetErrorCode

long **GetErrorCode**(long FAR* pnErrorCode);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnErrorCode A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the error code of the raw file. A return value of 0 indicates that there is no error.

Example

```
// example for GetErrorCode
long nErrorCode;
long nRet = XRawfileCtrl.GetErrorCode ( &nErrorCode );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting error code"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetErrorMessage

long GetErrorMessage(BSTR FAR* pbstrErrorMessage);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrErrorMessage A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns error information for the raw file as a descriptive string. If there is no error, the returned string will be empty.

Example

```
// example for GetErrorMessage
BSTR bstrErrorMessage = NULL;
long nRet = XRawfileCtrl.GetErrorMessage ( &bstrErrorMessage );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting error message"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrErrorMessage);
```


XRawfile OCX Documentation

GetWarningMessage

long **GetWarningMessage**(BSTR FAR* pbstrWarningMessage);

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrWarningMessage A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns warning information for the raw file as a descriptive string. If there is no warning, the returned string will be empty.

Example

```
// example for GetWarningMessage
BSTR bstrWarningMessage = NULL;
long nRet = XRawfileCtrl.GetWarningMessage ( &bstrWarningMessage );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting warning message"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrWarningMessage);
```

XRawfile OCX Documentation

GetSeqRowNumber

long GetSeqRowNumber(long FAR* pnSeqRowNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnSeqRowNumber A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the sequence row number for this sample in an acquired sequence. The numbering starts at 1.

Example

```
// example for GetSeqRowNumber
long nRow;
long nRet = XRawfileCtrl.GetSeqRowNumber ( &nRow );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowSampleType

long **GetSeqRowSampleType**(long FAR* pnSampleType);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnSampleType A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the sequence row sample type for this sample. See **Sample Type** in the **Enumerated Types** section for the possible sample type values.

Example

```
// example for GetSeqRowSampleType
long nType;
long nRet = XRawfileCtrl.GetSeqRowSampleType ( &nType );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row sample type"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowDataPath

long GetSeqRowDataPath(BSTR FAR* pbstrDataPath);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrDataPath A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the path of the directory where this raw file was acquired.

Example

```
// example for GetSeqRowDataPath
BSTR bstrPath = NULL;
long nRet = XRawfileCtrl.GetSeqRowDataPath ( &bstrPath );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting data path"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrPath);
```

XRawfile OCX Documentation

GetSeqRowRawFileName

long GetSeqRowRawFileName(BSTR FAR* pbstrRawFileName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrRawFileName A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the file name of the raw file when the raw file was acquired. This value is typically used in conjunction with **GetSeqRowDataPath** to obtain the fully qualified path name of the raw file when it was acquired.

Example

```
// example for GetSeqRowRawFileName
BSTR bstrFile = NULL;
long nRet = XRawfileCtrl.GetSeqRowRawFileName ( &bstrFile );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting raw file name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrFile);
```

XRawfile OCX Documentation

GetSeqRowSampleName

long GetSeqRowSampleName(BSTR FAR* pbstrSampleName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrSampleName A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the sample name value from the sequence row of the raw file.

Example

```
// example for GetSeqRowSampleName
BSTR bstrSampleName = NULL;
long nRet = XRawfileCtrl.GetSeqRowSampleName ( &bstrSampleName );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrSampleName);
```

XRawfile OCX Documentation

GetSeqRowSampleID

long GetSeqRowSampleID(BSTR FAR* pbstrSampleID);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrSampleID A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the sample ID value from the sequence row of the raw file.

Example

```
// example for GetSeqRowSampleID
BSTR bstrSampleID = NULL;
long nRet = XRawfileCtrl.GetSeqRowSampleID ( &bstrSampleID );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample ID"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrSampleID);
```

XRawfile OCX Documentation

GetSeqRowComment

long GetSeqRowComment(BSTR FAR* pbstrComment);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrComment A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the comment field from the sequence row of the raw file.

Example

```
// example for GetSeqComment
BSTR bstrComment = NULL;
long nRet = XRawfileCtrl.GetSeqRowComment ( &bstrComment );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting comment from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrComment);
```


XRawfile OCX Documentation

GetSeqRowLevelName

long GetSeqRowLevelName(BSTR FAR* pbstrLevelName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrLevelName A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the level name from the sequence row of the raw file. This field will be empty except for standard and QC sample types which may contain a value if a processing method was specified in the sequence at the time of acquisition.

Example

```
// example for GetSeqRowLevelName
BSTR bstrLevel = NULL;
long nRet = XRawfileCtrl.GetSeqRowLevelName ( &bstrLevel );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting level name from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrLevel);
```

XRawfile OCX Documentation

GetSeqRowUserText

long GetSeqRowUserText(long nIndex, BSTR FAR* pbstrUserText);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nIndex The index value of the user text field to return.

pbstrUserText A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns a user text field from the sequence row of the raw file. There are five user text fields in the sequence row that are indexed 0 through 4.

Example

```
// example for GetSeqRowUserText
BSTR bstrUserText = NULL;

// get user text 3
long nRet = XRawfileCtrl.GetSeqRowUserText ( 2, &bstrUserText );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting user text 3 from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrUserText);
```

XRawfile OCX Documentation

GetSeqRowInstrumentMethod

long GetSeqRowInstrumentMethod(BSTR FAR* pbstrInstrumentMethod);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstrumentMethod A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the fully qualified path name of the instrument method used to acquire the raw file. This field will be empty if the raw file was created by file format conversion or acquired from a tuning program.

Example

```
// example for GetSeqRowInstrumentMethod
BSTR bstrMethod = NULL;
long nRet = XRawfileCtrl.GetSeqRowInstrumentMethod ( &bstrMethod );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument method from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrMethod);
```

XRawfile OCX Documentation

GetSeqRowProcessingMethod

long GetSeqRowProcessingMethod(BSTR FAR* pbstrProcessingMethod);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrProcessingMethod A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the fully qualified path name of the processing method specified in the sequence used to acquire the raw file. This field will be empty if no processing method was specified at the time of acquisition.

Example

```
// example for GetSeqRowProcessingMethod
BSTR bstrMethod = NULL;
long nRet = XRawfileCtrl.GetSeqRowProcessingMethod ( &bstrMethod );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting processing method from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrMethod);
```

XRawfile OCX Documentation

GetSetRowCalibrationFile

long GetSetRowCalibrationFile(BSTR FAR* pbstrCalibrationFile);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrCalibrationFile A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the fully qualified path name of the calibration file specified in the sequence used to acquire the raw file. This field will be empty if no calibration file was specified at the time of acquisition.

Example

```
// example for GetSeqRowCalibrationFile
BSTR bstrCalFile = NULL;
long nRet = XRawfileCtrl.GetSeqRowCalibrationFile ( &bstrCalFile );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting calibration file from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrCalFile);
```

XRawfile OCX Documentation

GetSeqRowVial

long **GetSeqRowVial**(BSTR FAR* pbstrVial);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrVial A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the vial or well number of the sample when it was acquired. This value should be ignored if the raw file was not acquired using an autosampler.

Example

```
// example for GetSeqRowVial
BSTR bstrVial = NULL;
long nRet = XRawfileCtrl.GetSeqRowVial ( &bstrVial );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting vial number from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrVial);
```

XRawfile OCX Documentation

GetSeqRowInjectionVolume

long GetSeqRowInjectionVolume(double FAR* pdInjVol);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdInjVol A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the autosampler injection volume from the sequence row for this sample.

Example

```
// example for GetSeqRowInjectionVolume
double dInjVol;
long nRet = XRawfileCtrl.GetSeqRowInjectionVolume ( &dInjVol );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row injection volume"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowSampleWeight

long GetSeqRowSampleWeight(double FAR* pdSampleWt);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdSampleWt A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the sample weight from the sequence row for this sample.

Example

```
// example for GetSeqRowSampleWeight
double dWt;
long nRet = XRawfileCtrl.GetSeqRowSampleWeight ( &dWt );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row sample weight"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

GetSeqRowSampleVolume

long GetSeqRowSampleVolume(double FAR* pdSampleVolume);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdSampleVolume A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the sample volume from the sequence row for this sample.

Example

```
// example for GetSeqRowSampleVolume
double dVol;
long nRet = XRawfileCtrl.GetSeqRowSampleVolume ( &dVol );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row sample volume"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowISTDAmount

long GetSeqRowISTDAmount(double FAR* pdISTDAmount);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdISTDAmount A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the bulk ISTD correction amount from the sequence row for this sample.

Example

```
// example for GetSeqRowISTDAmount
double dISTDAmt;
long nRet = XRawfileCtrl.GetSeqRowISTDAmount ( &dISTDAmt );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row ISTD amount"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowDilutionFactor

long GetSeqRowDilutionFactor(double FAR* pdDilutionFactor);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdDilutionFactor A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the bulk dilution factor (volume correction) from the sequence row for this sample.

Example

```
// example for GetSeqRowDilutionFactor
double dDilFactor;
long nRet = XRawfileCtrl.GetSeqRowDilutionFactor ( &dDilFactor );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting seq row dilution factor"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSeqRowUserLabel

long GetSeqRowUserLabel(long nIndex, BSTR FAR* pbstrUserLabel);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nIndex The index value of the user text field to return.

pbstrUserLabel A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns a user label field from the sequence row of the raw file. There are five user label fields in the sequence row that are indexed 0 through 4. The user label fields correspond one-to-one with the user text fields.

Example

```
// example for GetSeqRowUserLabel
BSTR bstrUserLabel = NULL;

// get user label 3
long nRet = XRawfileCtrl.GetSeqRowUserLabel ( 2, &bstrUserLabel );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting user label 3 from seq row"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrUserLabel);
```

XRawfile OCX Documentation

InAcquisition

long InAcquisition(BOOL FAR* pblnAcquisition);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pblnAcquisition A valid pointer to a variable of type BOOL. This variable must exist.

Remarks

Returns the acquisition state flag of the raw file. A return value of TRUE indicates that the raw file is being acquired or that all open handles to the file during acquisition have not been closed.

Example

```
// example for InAcquisition
BOOL bInAcqu;
long nRet = XRawfileCtrl.InAcquisition ( &bInAcqu );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting acquisition state flag"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumberOfControllers

long **GetNumberOfControllers**(long FAR* pnNumControllers);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumControllers A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the number of registered device controllers in the raw file. A device controller represents an acquisition stream such as MS data, UV data, etc. Devices that do not acquire data such as autosamplers are not registered with the raw file during acquisition.

Example

```
// example for GetNumberOfControllers
long nControllers;
long nRet = XRawfileCtrl.GetNumberOfControllers ( &nControllers );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of controllers"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumberOfControllersOfType

```
long  GetNumberOfControllersOfType(long nControllerType, long FAR*
pnNumControllersOfType);
```

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nControllerType</i>	The controller type for which the number of registered controllers of that type are requested.
<i>pnNumControllers</i>	A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the number of registered device controllers of a particular type in the raw file. See **Controller Type** in the **Enumerated Types** section for a list of the available controller types and their respective values.

Example

```
// example for GetNumberOfControllersOfType
long nControllerType = 0;      // 0 == mass spec device
long nControllers;
long nRet = XRawfileCtrl.GetNumberOfControllersOfType ( nControllerType, &nControllers );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of controllers of type MS"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetControllerType

long GetControllerType(long nIndex, long FAR* pnControllerType);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nIndex</i>	The index value of the controller for which the type is to be returned.
<i>pnControllerType</i>	A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the type of the device controller registered at the specified index position in the raw file. Index values start at 0. See **Controller Type** in the **Enumerated Types** section for a list of the available controller types and their respective values.

Example

```
// example for GetControllerType
long nControllerType;

// get first device controller type
long nRet = XRawfileCtrl.GetNumberOfControllersOfType ( 0, &nControllerType );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting type for first controller"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

SetCurrentController

long SetCurrentController(long nControllerType, long nControllerNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nControllerType The type of controller for which information will be subsequently requested.

nControllerNumber The number of the controller of the specified type.

Remarks

Sets the current controller in the raw file. This function must be called before subsequent calls to access data specific to a device controller (e.g. MS or UV data) may be made. All requests for data specific to a device controller will be forwarded to the current controller until the current controller is changed. The controller number is used to indicate which device controller to use if there are more than one registered device controller of the same type (e.g. multiple UV detectors). Controller numbers for each type are numbered starting at 1. See **Controller Type** in the **Enumerated Types** section for a list of the available controller types and their respective values.

Example

```
// example for SetCurrentController
long nControllerType = 0;        // 0 == mass spec device
long nControllerNumber = 1;     // first MS device

long nRet = XRawfileCtrl.SetCurrentController ( nControllerType, nControllerNumber );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error setting current controller"), _T("Error"), MB_OK );
    ...
}

// Calls to access the current controller data may now be made.
```

XRawfile OCX Documentation

GetCurrentController

long **GetCurrentController**(long FAR* pnControllerType, long FAR* pnControllerNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnControllerType A valid pointer to a variable of type long. This variable must exist.

pnControllerNumber A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the current controller type and number for the raw file. The controller number is used to indicate which device controller to use if there are more than one registered device controller of the same type (e.g. multiple UV detectors). Controller numbers for each type are numbered starting at 1. See **Controller Type** in the **Enumerated Types** section for a list of the available controller types and their respective values.

Example

```
// example for GetCurrentController
long nControllerType;
long nControllerNumber;
long nRet = XRawfileCtrl.GetCurrentController ( &nControllerType, &nControllerNumber );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting current controller"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumSpectra

long **GetNumSpectra**(long FAR* pnNumberOfSpectra);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumberOfSpectra A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the number of spectra acquired by the current controller. For non-scanning devices like UV detectors, the number of readings per channel is returned.

Example

```
// example for GetNumSpectra
long nSpectra;
long nRet = XRawfileCtrl.GetNumSpectra (&nSpectra );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of spectra"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumStatusLog

long GetNumStatusLog(long FAR* pnNumberOfStatusLogEntries);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumberOfStatusLogEntries A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the number of status log entries recorded for the current controller.

Example

```
// example for GetNumStatusLog
long nLogs;
long nRet = XRawfileCtrl.GetNumStatusLog (&nLogs );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of status log entries"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumErrorLog

long GetNumErrorLog(long FAR* pnNumberOfErrorLogEntries);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumberOfErrorLogEntries A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the number of error log entries recorded for the current controller.

Example

```
// example for GetNumErrorLog
long nLogs;
long nRet = XRawfileCtrl.GetNumErrorLog (&nLogs );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of error log entries"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetNumTuneData

long **GetNumTuneData**(long FAR* pnNumTuneData);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumTuneData A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the number of tune data entries recorded for the current controller. Tune Data is only supported by MS controllers. Typically, if there is more than one tune data entry, each tune data entry corresponds to a particular acquisition segment.

Example

```
// example for GetNumTuneData
long nTuneData;
long nRet = XRawfileCtrl.GetNumTuneData (&nTuneData );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of tune data entries"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetMassResolution

long GetMassResolution(double FAR* pdMassResolution);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdMassResolution A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the mass resolution value recorded for the current controller. The value is returned as one half of the mass resolution. For example, a unit resolution controller would return a value of 0.5. This value is only relevant to scanning controllers such as MS.

Example

```
// example for GetMassResolution
double dHalfMassRes;
long nRet = XRawfileCtrl.GetMassResolution (&dHalfMassRes);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass resolution"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetExpectedRunTime

long GetExpectedRunTime(double FAR* pdExpectedRunTime);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdExpectedRunTime A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the expected acquisition run time for the current controller. The actual acquisition may be longer or shorter than this value. This value is intended to allow displays to show the expected run time on chromatograms. To obtain an accurate run time value during or after acquisition, use the **GetEndTime** function.

Example

```
// example for GetExpectedRunTime
double dExpRunTime;
long nRet = XRawfileCtrl.GetExpectedRunTime (&dExpRunTime);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting expected run time"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

GetNumTrailerExtra

long **GetNumTrailerExtra**(long FAR* pnNumberOfTrailerExtraEntries);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumberOfTrailerExtraEntries A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the trailer extra entries recorded for the current controller. Trailer extra entries are only supported for MS device controllers and is used to store instrument specific information for each scan if used.

Example

```
// example for GetNumTrailerExtra
long nTrailerExtra;
long nRet = XRawfileCtrl.GetNumTrailerExtra (&nTrailerExtra);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of trailer extra entries"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetLowMass

long **GetLowMass**(double FAR* pdLowMass);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdLowMass A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the lowest mass or wavelength recorded for the current controller. This value is only relevant to scanning devices such as MS or PDA.

Example

```
// example for GetLowMass
double dLowMass;
long nRet = XRawfileCtrl.GetLowMass (&dLowMass);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting low mass"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetHighMass

long GetHighMass(double FAR* pdHighMass);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdHighMass A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the highest mass or wavelength recorded for the current controller. This value is only relevant to scanning devices such as MS or PDA.

Example

```
// example for GetHighMass
double dHighMass;
long nRet = XRawfileCtrl.GetHighMass (&dHighMass);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting high mass"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetStartTime

long **GetStartTime**(double FAR* pdStartTime);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdStartTime A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the start time of the first scan or reading for the current controller. This value is typically close to zero unless the device method contains a start delay.

Example

```
// example for GetStartTime
double dStartTime;
long nRet = XRawfileCtrl.GetStartTime (&dStartTime);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting start time"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetEndTime

long GetEndTime(double FAR* pdEndTime);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdEndTime A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the start time of the last scan or reading for the current controller.

Example

```
// example for GetEndTime
double dEndTime;
long nRet = XRawfileCtrl.GetEndTime (&dEndTime);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting end time"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetMaxIntegratedIntensity

long GetMaxIntegratedIntensity(double FAR* pdMaxIntegIntensity);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdMaxIntegIntensity A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the highest integrated intensity of all the scans for the current controller. This value is only relevant to MS device controllers.

Example

```
// example for GetMaxIntegratedIntensity
double dMaxIntegInt;
long nRet = XRawfileCtrl.GetMaxIntegratedIntensity (&dMaxIntegInt);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting max integrated intensity"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetMaxIntensity

long GetMaxIntensity(long FAR* pnMaxIntensity);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdMaxIntensity A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the highest base peak of all the scans for the current controller. This value is only relevant to MS device controllers.

Example

```
// example for GetMaxIntensity
double dMaxInt;
long nRet = XRawfileCtrl.GetMaxIntensity (&dMaxInt);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting max intensity"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetFirstSpectrumNumber

long GetFirstSpectrumNumber(long FAR* pnFirstSpectrum);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnFirstSpectrum A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the first scan or reading number for the current controller. This value will always be one if data has been acquired.

Example

```
// example for GetFirstSpectrumNumber
long nFirstScan;
long nRet = XRawfileCtrl.GetFirstSpectrumNumber (&nFirstScan);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting first scan number"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

GetLastSpectrumNumber

long GetLastSpectrumNumber(long FAR* pnLastSpectrum);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnLastSpectrum A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the last scan or reading number for the current controller.

Example

```
// example for GetLastSpectrumNumber
long nLastScan;
long nRet = XRawfileCtrl.GetLastSpectrumNumber (&nLastScan);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting last scan number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetInstrumentID

long **GetInstrumentID**(long FAR* pnInstrumentID);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnInstrumentID A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the instrument ID number for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetInstrumentID
long nInstID;
long nRet = XRawfileCtrl.GetInstrumentID (&nInstID);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting inst ID number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetInletID

long **GetInletID**(long FAR* pnInletID);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnInletID A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the inlet ID number for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetInletID
long nInletID;
long nRet = XRawfileCtrl.GetInletID (&nInletID);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting inlet ID number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetErrorFlag

long **GetErrorFlag**(long FAR* pnErrorFlag);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnErrorFlag A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the error flag value for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetErrorFlag
long nErrorFlag;
long nRet = XRawfileCtrl.GetErrorFlag (&nErrorFlag);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting error flag value"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSampleVolume

long GetSampleVolume(double FAR* pdSampleVolume);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdSampleVolume A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the sample volume value for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetSampleVolume
double dSampleVolume;
long nRet = XRawfileCtrl.GetSampleVolume (&dSampleVolume);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample volume value"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetSampleWeight

long GetSampleWeight(double FAR* pdSampleWeight);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdSampleWeight A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the sample weight value for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetSampleWeight
double dSampleWeight;
long nRet = XRawfileCtrl.GetSampleWeight (&dSampleWeight);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample weight value"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetVialNumber

long **GetVialNumber**(long FAR* pnVialNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnVialNumber A valid pointer to a variable of type long. This variable must exist.

Remarks

Gets the vial number for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetVialNumber
long nVialNum;
long nRet = XRawfileCtrl.GetVialNumber (&nVialNum);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting vial number"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetInjectionVolume

long GetInjectionVolume(double FAR* pdInjectionVolume);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pdInjectionVolume A valid pointer to a variable of type double. This variable must exist.

Remarks

Gets the injection volume for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetInjectionVolume
double dInjVol;
long nRet = XRawfileCtrl.GetInjectionVolume (&dInjVol);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting injection volume"), _T("Error"), MB_OK );
    ...
}
```


XRawfile OCX Documentation

GetFlags

long **GetFlags**(BSTR FAR* pbstrFlags);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrFlags A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the acquisition flags field for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetFlags
BSTR bstrFlags = NULL;
long nRet = XRawfileCtrl.GetFlags ( &bstrFlags );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting flags"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrFlags);
```

XRawfile OCX Documentation

GetAcquisitionFileName

long **GetAcquisitionFileName**(BSTR FAR* pbstrFileName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrFileName A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the acquisition file name for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetAcquisitionFileName
BSTR bstrAcquFile = NULL;
long nRet = XRawfileCtrl.GetAcquFile ( &bstrAcquFile );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting acquisition file name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrAcquFile);
```

XRawfile OCX Documentation

GetInstrumentDescription

long GetInstrumentDescription(BSTR FAR* pbstrInstrumentDescription);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstrumentDescription A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the instrument description field for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetInstrumentDescription
BSTR bstrInstDesc = NULL;
long nRet = XRawfileCtrl.GetInstrumentDescription ( &bstrInstDesc );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument description"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstDesc);
```

XRawfile OCX Documentation

GetAcquisitionDate

long GetAcquisitionDate(BSTR FAR* pbstrAcquisitionDate);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrAcquisitionDate A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the acquisition date for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetAcquisitionDate
BSTR bstrAcquDate = NULL;
long nRet = XRawfileCtrl.GetAcquisitionDate ( &bstrAcquDate );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting acquisition date"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrAcquDate);
```

XRawfile OCX Documentation

GetOperator

long GetOperator(BSTR FAR* pbstrOperator);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrOperator A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the operator name for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetOperator
BSTR bstrOperator = NULL;
long nRet = XRawfileCtrl.GetOperator ( &bstrOperator );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting operator name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrOperator);
```

XRawfile OCX Documentation

GetComment1

long GetComment1(BSTR FAR* pbstrComment1);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrComment1 A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the first comment for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetComment1
BSTR bstrComment1 = NULL;
long nRet = XRawfileCtrl.GetComment1 ( &bstrComment1 );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting comment 1"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrComment1);
```

XRawfile OCX Documentation

GetComment2

long GetComment2(BSTR FAR* pbstrComment2);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrComment2 A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the first comment for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetComment2
BSTR bstrComment2 = NULL;
long nRet = XRawfileCtrl.GetComment2 ( &bstrComment2 );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting comment 2"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrComment2);
```

XRawfile OCX Documentation

GetSampleAmountUnits

long **GetSampleAmountUnits**(BSTR FAR* pbstrSampleAmountUnits);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrSampleAmountUnits A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the sample amount units for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetSampleAmountUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetSampleAmountUnits ( &bstrUnits );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample amount units"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrUnits);
```


XRawfile OCX Documentation

GetInjectionAmountUnits

long GetInjectionAmountUnits(BSTR FAR* pbstrInjectionAmountUnits);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInjectionAmountUnits A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the injection amount units for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetInjectionAmountUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetInjectionAmountUnits ( &bstrUnits );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting injection amount units"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrUnits);
```

XRawfile OCX Documentation

GetSampleVolumeUnits

long GetSampleVolumeUnits(BSTR FAR* pbstrSampleVolumeUnits);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrSampleVolumeUnits A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the sample volume units for the current controller. This value is typically only set for raw files converted from other files formats.

Example

```
// example for GetSampleVolumeUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetSampleVolumeUnits ( &bstrUnits );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting sample volume units"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrUnits);
```

XRawfile OCX Documentation

GetInstName

long GetInstName(BSTR FAR* pbstrInstName);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstName A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the instrument name, if available, for the current controller.

Example

```
// example for GetInstName
BSTR bstrInstName = NULL;
long nRet = XRawfileCtrl.GetInstName ( &bstrInstName );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument name"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstName);
```

XRawfile OCX Documentation

GetInstModel

long GetInstModel(BSTR FAR* pbstrInstModel);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstModel A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the instrument model, if available, for the current controller.

Example

```
// example for GetInstModel
BSTR bstrInstModel = NULL;
long nRet = XRawfileCtrl.GetInstModel ( &bstrInstModel );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument model"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstModel);
```

XRawfile OCX Documentation

GetInstSerialNumber

long GetInstSerialNumber(BSTR FAR* pbstrInstSerialNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstSerialNumber A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the serial number, if available, for the current controller.

Example

```
// example for GetInstSerialNumber
BSTR bstrInstSerialNum = NULL;
long nRet = XRawfileCtrl.GetInstSerialNumber ( &bstrInstSerialNum );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument serial number"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstSerialNum);
```

XRawfile OCX Documentation

GetInstSoftwareVersion

long GetInstSoftwareVersion(BSTR FAR* pbstrInstSoftwareVersion);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstSoftwareVersion A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the current controller software revision information, if available.

Example

```
// example for GetInstSoftwareVersion
BSTR bstrInstSoftRev = NULL;
long nRet = XRawfileCtrl.GetInstSoftwareVersion ( &bstrInstSoftRev );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument software version"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstSoftRev);
```

XRawfile OCX Documentation

GetInstHardwareVersion

long **GetInstHardwareVersion**(BSTR FAR* pbstrInstHardwareVersion);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstHardwareVersion A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the current controller hardware or firmware revision information, if available.

Example

```
// example for GetInstHardwareVersion
BSTR bstrInstHardRev = NULL;
long nRet = XRawfileCtrl.GetInstHardwareVersion ( &bstrInstHardRev );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting instrument hardware version"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstHardRev);
```

XRawfile OCX Documentation

GetInstFlags

long GetInstFlags(BSTR FAR* pbstrInstFlags);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pbstrInstFlags A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the experiment flags, if available, for the current controller. The returned string may contain one or more fields denoting information about the type of experiment performed.

Currently defined experiment fields are:

TIM	- total ion map
NLM	- neutral loss map
PIM	- parent ion map
DDZMap	- data dependent ZoomScan map

Example

```
// example for GetInstFlags
BSTR bstrInstFlags = NULL;
long nRet = XRawfileCtrl.GetInstFlags ( &bstrInstFlags );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting experiment flags"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrInstFlags);
```


XRawfile OCX Documentation

GetInstNumChannelLabels

long GetInstNumChannelLabels(long FAR* pnInstNumChannelLabels);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnInstNumChannelLabels A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the number of channel labels specified for the current controller. This field is only relevant to channel devices such as UV detectors, A/D cards, and Analog inputs. Typically, the number of channel labels, if labels are available, is the same as the number of configured channels for the current controller.

Example

```
// example for GetInstNumChannelLabels
long nLabels
long nRet = XRawfileCtrl. GetInstNumChannelLabels ( &nLabels );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting number of channel labels"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetInstChannelLabel

long GetInstChannelLabel(long nChannelLabelNumber, BSTR FAR* pbstrInstChannelLabel);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nChannelLabelNumber</i>	The index value of the channel number field to return.
<i>pbstrFlags</i>	A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the channel label, if available, at the specified index for the current controller. This field is only relevant to channel devices such as UV detectors, A/D cards, and Analog inputs. Channel labels indices are numbered starting at 0.

Example

```
// example for GetInstChannelLabel
long nNumber = 2;      // the channel 3 label
BSTR bstrLabel = NULL;
long nRet = XRawfileCtrl.GetInstChannelLabel ( &bstrLabel );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting channel label 3"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrLabel);
```

XRawfile OCX Documentation

GetFilters

long GetFilters(VARIANT FAR* pvarFilterArray, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pvarFilterArray A valid pointer to a variable of type VARIANT. This variable must exist and be initialized to VT_EMPTY.

pnArraySize A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the list of unique scan filters for the raw file. This function is only supported for MS device controllers. If the function succeeds, *pvarFilterArray* will point to an array of BSTR fields each containing a unique scan filter and *pnArraySize* will contain the number of scan filters in the *pvarFilterArray*.

Example

```
// example for GetFilters
VARIANT varFilters;
VariantInit(&varFilters);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetFilters ( & varFilters, &nArraySize );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting array of scan filters"), _T("Error"), MB_OK );
    return
}

if( !nArraySize || varFilters.vt != (VT_ARRAY | VT_BSTR) )
{
    ::MessageBox( NULL, _T("No valid filters returned"), _T("Error"), MB_OK );
    return;
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psa = varFilters.parray;
varFilters.parray = NULL;

BSTR* pbstrFilters = NULL;
if( FAILED(SafeArrayAccessData( psa, (void**)(&pbstrFilters) ) ) )
{
    SafeArrayUnaccessData( psa );
    SafeArrayDestroy( psa );
    ::MessageBox( NULL, _T("Failed to access scan filter array"), _T("Error"), MB_OK );
    return;
}

// display filters one at a time
TCHAR szTitle[16];
for( long i=0; i<nArraySize; i++ )
```

XRawfile OCX Documentation

```
{
    _stprintf( szTitle, _T("Scan Filter %d"), i );
    ::MessageBox( NULL, pbstrFilters[i], szTitle, MB_OK );
}

// Delete the SafeArray
SafeArrayUnaccessData( psa );
SafeArrayDestroy( psa );
```

XRawfile OCX Documentation

ScanNumFromRT

long ScanNumFromRT(double dRT, long FAR* pnScanNumber);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

dRT The run time or retention time, in minutes, for which the closest scan number is to be returned.

pnScanNumber A valid pointer to a variable of type long. This variable must exist.

Remarks

Returns the closest matching scan number that corresponds to *dRT* for the current controller. For non-scanning devices, such as UV, the closest reading number is returned. The value of *dRT* must be within the acquisition run time for the current controller. The acquisition run time for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

Example

```
// example for ScanNumFromRT
double dRT = 3.45;      // get scan number at 3.45 minutes
long nScanNum;
long nRet = XRawfileCtrl.ScanNumFromRT ( dRT, &nScanNum );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting scan number at RT of 3.45 min."), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

RTFromScanNum

long RTFromScanNum(long nScanNumber, double FAR* pdRT);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nScanNumber The scan number for which the closest run time or retention time is to be returned.

pdRT A valid pointer to a variable of type double. This variable must exist.

Remarks

Returns the closest matching run time or retention time that corresponds to *nScanNumber* for the current controller. For non-scanning devices, such as UV, the *nScanNumber* is taken to be the reading number. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

Example

```
// example for RTFromScanNum
long nScanNum = 12; // get the RT of the twelfth scan
double dRT;
long nRet = XRawfileCtrl.RTFromScanNum ( nScanNumber, &dRT );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting RT for scan number 12"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetFilterForScanNum

long GetFilterForScanNum(long nScanNumber, BSTR FAR* pbstrFilter);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which the corresponding scan filter is to be returned.
<i>pbstrFilter</i>	A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the closest matching run time that corresponds to *nScanNumber* for the current controller. This function is only support for MS device controllers. The value of *nScanNumber* must be within the range of scans for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

Example

```
// example for GetFilterForScanNum
long nScanNum = 12; // get the scan filter for the twelfth scan
BSTR bstrFilter = NULL;
long nRet = XRawfileCtrl.GetFilterForScanNum ( nScanNumber, &bstrFilter );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting scan filter for scan number 12"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrFilter);
```

XRawfile OCX Documentation

GetFilterForScanRT

long GetFilterForScanRT(double dRT, BSTR FAR* pbstrFtiler);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

dRT The run time for which the corresponding scan filter is to be returned.

pbstrFilter A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the scan filter for the closest matching scan that corresponds to *dRT* for the current controller. This function is only support for MS device controllers. The value of *dRT* must be within the acquisition run time for the current controller. The acquisition run time for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

Example

```
// example for GetFilterForScanRT
double dRT = 3.45;        // get scan filter at 3.45 minutes
BSTR bstrFilter = NULL;
long nRet = XRawfileCtrl.GetFilterForScanNum ( dRT, &bstrFilter );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting scan filter for RT of 3.45 min."), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrFilter);
```


XRawfile OCX Documentation

GetMassListFromScanNum

long GetMassListFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number for which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan corresponding to *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan to *pnScanNumber* that matches the scan filter will be returned. The requested scan number must be valid for the current controller. Valid scan number limits may be obtained by calling **GetFirstSpectrumNumber** and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

XRawfile OCX Documentation

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

// example for GetMassListFromScanNum

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

long nScanNumber = 12;          // read the contents of scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetMassListFromScanNum ( &nScanNumber,
                                                  NULL,                // no filter
                                                  0,                  // no cutoff
                                                  0,                  // no cutoff
                                                  0,                  // all peaks returned
                                                  FALSE,              // do not centroid
                                                  &varMassList,         // mass list data
                                                  &varPeakFlags,      // peak flags data
                                                  &nArraySize );      // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for scan 12."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)(&pDataPeaks) );

    for( long j=0; j<nArraySize; j++ )
    {
```

XRawfile OCX Documentation

```
        double dMass = pDataPeaks[jj].dMass;
        double dIntensity = pDataPeaks[jj].dIntensity;

        // Do something with mass intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetMassListFromRT

long GetMassListFromRT(double FAR* pdRT, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a double precision variable containing the retention time, in minutes, for which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the closest scan to *pdRT* will be returned. If a scan filter is provided, the closest matching scan to *pdRT* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pdRT* will contain the actual retention time of the returned scan. Valid retention time limits may be obtained by calling **GetStartTime** and **GetEndTime**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

XRawfile OCX Documentation

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

// example for GetMassListFromRT

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

double dRT = 3.8;          // read the contents of the scan at RT = 3.8 minutes
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetMassListFromRT ( &dRT,
                                             NULL,           // no filter
                                             0,              // no cutoff
                                             0,              // no cutoff
                                             0,              // all peaks returned
                                             FALSE,          // do not centroid
                                             &varMassList,     // mass list data
                                             &varPeakFlags,   // peak flags data
                                             &nArraySize );    // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for scan 12."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)(&pDataPeaks) );

    for( long j=0; j<nArraySize; j++ )
    {
        double dMass = pDataPeaks[j].dMass;
        double dIntensity = pDataPeaks[j].dIntensity;
    }
}
```

XRawfile OCX Documentation

```
        // Do something with mass intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetNextMassListFromScanNum

long GetNextMassListFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number after which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan after *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan after *pnScanNumber* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* will contain the actual scan number of the returned scan. Valid scan number limits may be obtained by calling

GetFirstSpectrumNumber and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

XRawfile OCX Documentation

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

// example for GetNextMassListFromScanNum

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

long nScanNumber = 12;           // read the contents of the scan after scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetNextMassListFromScanNum (    &nScanNumber,
                                                         NULL,           // no filter
                                                         0,             // no cutoff
                                                         0,             // no cutoff
                                                         0,             // all peaks returned
                                                         FALSE,         // do not centroid
                                                         &varMassList,  // mass list data
                                                         &varPeakFlags, // peak flags data
                                                         &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for next scan after 12."), _T("Error"),
    MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)(&pDataPeaks) );

    for( long j=0; j<nArraySize; j++ )
```


XRawfile OCX Documentation

```
{
    double dMass = pDataPeaks[jj].dMass;
    double dIntensity = pDataPeaks[jj].dIntensity;

    // Do something with mass intensity values
    ...
}

// Release the data handle
SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetPrevMassListFromScanNum

long GetPrevMassListFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number before which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan before *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan before *pnScanNumber* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* will contain the actual scan number of the returned scan. Valid scan number limits may be obtained by calling

GetFirstSpectrumNumber and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

XRawfile OCX Documentation

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

// example for GetPrevMassListFromScanNum

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

long nScanNumber = 12;           // read the contents of the scan before scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetPrevMassListFromScanNum (    &nScanNumber,
                                                         NULL,           // no filter
                                                         0,             // no cutoff
                                                         0,             // no cutoff
                                                         0,             // all peaks returned
                                                         FALSE,         // do not centroid
                                                         &varMassList,  // mass list data
                                                         &varPeakFlags, // peak flags data
                                                         &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for next scan after 12."), _T("Error"),
    MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)(&pDataPeaks) );

    for( long j=0; j<nArraySize; j++ )
```

XRawfile OCX Documentation

```
{
    double dMass = pDataPeaks[jj].dMass;
    double dIntensity = pDataPeaks[jj].dIntensity;

    // Do something with mass intensity values
    ...
}

// Release the data handle
SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetMassListRangeFromScanNum

long GetMassListRangeFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, LPCTSTR csMassRange1 , long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number for which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>csMassRange1</i>	A string containing the mass range.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan corresponding to *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan to *pnScanNumber* that matches the scan filter will be returned. The requested scan number must be valid for the current controller. Valid scan number limits may be obtained by calling **GetFirstSpectrumNumber** and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

XRawfile OCX Documentation

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

To get a range of masses between two points that will be returned in the mass list, set the string of *szMassRange1* to a valid range.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

```
// example for GetMassListRangeFromScanNum
```

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

long nScanNumber = 12;          // read the contents of scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
TCHAR* szMassRange1[] = _T("450.00-640.00");
long nRet = XRawfileCtrl.GetMassListFromScanNum ( &nScanNumber,
    NULL,                                     // no filter
    0,                                       // no cutoff
    0,                                       // no cutoff
    0,                                       // all peaks returned
    FALSE,                                   // do not centroid
    &varMassList,                           // mass list data
    &varPeakFlags,                         // peak flags data
    szMassRange1,                          // mass range
    &nArraySize );                          // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for scan 12."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;
```

XRawfile OCX Documentation

```
DataPeak* pDataPeaks = NULL;
SafeArrayAccessData( psa, (void**)&pDataPeaks );

for( long j=0; j<nArraySize; j++ )
{
    double dMass = pDataPeaks[j].dMass;
    double dIntensity = pDataPeaks[j].dIntensity;

    // Do something with mass intensity values
    ...
}

// Release the data handle
SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetMassListRangeFromRT

long GetMassListRangeFromRT(double FAR* pdRT, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, LPCTSTR szMassRange1, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a double precision variable containing the retention time, in minutes, for which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>szMassRange1</i>	A string containing the mass range.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the closest scan to *pdRT* will be returned. If a scan filter is provided, the closest matching scan to *pdRT* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pdRT* will contain the actual retention time of the returned scan. Valid retention time limits may be obtained by calling **GetStartTime** and **GetEndTime**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

XRawfile OCX Documentation

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

To get a range of masses between two points that will be returned in the mass list, set the string of *szMassRange1* to a valid range.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

// example for GetMassListRangeFromRT

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

double dRT = 3.8;          // read the contents of the scan at RT = 3.8 minutes
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
TCHAR* szMassRange1[] = _T("450.00-640.00");
long nArraySize = 0;
long nRet = XRawfileCtrl.GetMassListRangeFromRT (    &dRT,
                                                    NULL,           // no filter
                                                    0,             // no cutoff
                                                    0,             // no cutoff
                                                    0,             // all peaks returned
                                                    FALSE,        // do not centroid
                                                    &varMassList,  // mass list data
                                                    &varPeakFlags, // peak flags data
                                                    czMassRange1, // mass range
                                                    &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for scan 12."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;
```

XRawfile OCX Documentation

```
DataPeak* pDataPeaks = NULL;
SafeArrayAccessData( psa, (void**)&pDataPeaks );

for( long j=0; j<nArraySize; j++ )
{
    double dMass = pDataPeaks[j].dMass;
    double dIntensity = pDataPeaks[j].dIntensity;

    // Do something with mass intensity values
    ...
}

// Release the data handle
SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if(varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetNextMassListRangeFromScanNum

long GetNextMassListFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, LPCTSTR szMassRange1, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number after which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>szMassRange1</i>	A string containing the mass range.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan after *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan after *pnScanNumber* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* will contain the actual scan number of the returned scan. Valid scan number limits may be obtained by calling **GetFirstSpectrumNumber** and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

XRawfile OCX Documentation

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

To get a range of masses between two points that will be returned in the mass list, set the string of *szMassRange1* to a valid range.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

```
// example for GetNextMassListFromScanNum
```

```
typedef struct _datapeak
{
    double dMass;
    double dIntensity;
} DataPeak;

long nScanNumber = 12;           // read the contents of the scan after scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
TCHAR* szMassRange1[] = _T("450.00-640.00");
long nArraySize = 0;
long nRet = XRawfileCtrl.GetNextMassListFromScanNum (    &nScanNumber,
                                                         NULL,           // no filter
                                                         0,             // no cutoff
                                                         0,             // no cutoff
                                                         0,             // all peaks returned
                                                         FALSE,        // do not centroid
                                                         &varMassList, // mass list data
                                                         &varPeakFlags, // peak flags data
                                                         szMassRange1, // mass range
                                                         &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for next scan after 12."), _T("Error"),
    MB_OK );
    ...
}
```

XRawfile OCX Documentation

```
if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)( &pDataPeaks ) );

    for( long j=0; j<nArraySize; j++ )
    {
        double dMass = pDataPeaks[j].dMass;
        double dIntensity = pDataPeaks[j].dIntensity;

        // Do something with mass intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if( varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetPrecursorInfoFromScanNum

```
long GetPrecursorInfoFromScanNum(  
    long nScanNumber,  
    LPVARIANT pvarPrecursorInfos,  
    LPLONG pnArraySize)
```

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which the corresponding precursor info is to be returned.
<i>pvarPrecursorInfos</i>	A valid pointer to a VARIANT variable to receive the precursor info.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of precursor info packets returned in the precursor info array.

Remarks

This function is used to retrieve information about the parent scans of a data dependent MSn scan.

You will retrieve the scan number of the parent scan, the isolation mass used, the charge state and the monoisotopic mass as determined by the instrument firmware. You will also get access to the scan data of the parent scan in the form of a XSpectrumRead object.

For the charge state and the monoisotopic mass it is tried to further refine these values from the actual parent scan data.

Example

```
struct PrecursorInfo  
{  
    double dIsolationMass;  
    double dMonoIsoMass;  
    long nChargeState;  
    long nScanNumber;  
};  
  
void CTestOCXDlg::OnOpenParentScansOcx()  
{  
    try  
    {  
        VARIANT vPrecursorInfos;  
        VariantInit(&vPrecursorInfos);  
        long nPrecursorInfos = 0;  
  
        // Get the precursor scan information  
        m_Rawfile.GetPrecursorInfoFromScanNum(m_nScanNumber,  
                                              &vPrecursorInfos,  
                                              &nPrecursorInfos);  
  
        // Access the safearray buffer  
        BYTE* pData;  
        SafeArrayAccessData(vPrecursorInfos.parray, (void**)&pData);
```

XRawfile OCX Documentation

```
for (int i=0; i < nPrecursorInfos; ++i)
{
    // Copy the scan information from the safearray buffer
    PrecursorInfo info;
    memcpy(&info,
           pData + i * sizeof(MS_PrecursorInfo),
           sizeof(PrecursorInfo));

    // Process the paraent scan information ...
}

SafeArrayUnaccessData(vPrecursorInfos.parray);
}
catch (...)
{
    AfxMessageBox(_T("There was a problem while getting the parent scan
                    information."));
}
}
```

XRawfile OCX Documentation

GetPrevMassListRangeFromScanNum

long GetPrevMassListFromScanNum(long FAR* pnScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, LPCTSTR szMassRange1, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pnScanNumber</i>	A valid pointer to a long variable containing the scan number before which the corresponding mass list data is to be returned.
<i>szFilter</i>	A string containing the optional scan filter.
<i>nIntensityCutoffType</i>	The type of intensity cutoff to apply.
<i>nIntensityCutoffValue</i>	The intensity cutoff value.
<i>nMaxNumberOfPeaks</i>	The maximum number of data peaks to return in the mass list.
<i>bCentroidResult</i>	Boolean flag indicating that returned mass list contents should be centroided.
<i>pvarMassList</i>	A valid pointer to a VARIANT variable to receive the mass list data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>szMassRange1</i>	A string containing the mass range.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scan before *pnScanNumber* will be returned. If a scan filter is provided, the closest matching scan before *pnScanNumber* that matches the scan filter will be returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* will contain the actual scan number of the returned scan. Valid scan number limits may be obtained by calling **GetFirstSpectrumNumber** and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

XRawfile OCX Documentation

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

To have profile scans centroided, set *bCentroidResult* to TRUE. This parameter is ignored for centroid scans.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

To get a range of masses between two points that will be returned in the mass list, set the string of *szMassRange1* to a valid range.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

```
// example for GetPrevMassListFromScanNum
```

```
typedef struct _datapoint
{
    double dMass;
    double dIntensity;
} DataPoint;

long nScanNumber = 12;           // read the contents of the scan before scan 12
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
TCHAR* szMassRange1[] = _T("450.00-640.00");
long nArraySize = 0;
long nRet = XRawfileCtrl.GetPrevMassListFromScanNum (    &nScanNumber,
                                                         NULL,           // no filter
                                                         0,             // no cutoff
                                                         0,             // no cutoff
                                                         0,             // all peaks returned
                                                         FALSE,         // do not centroid
                                                         &varMassList,  // mass list data
                                                         &varPeakFlags, // peak flags data
                                                         szMassRange1, // mass range
                                                         &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting mass list data for next scan after 12."), _T("Error"),
    MB_OK );
    ...
}
```

XRawfile OCX Documentation

```
if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)&pDataPeaks );

    for( long j=0; j<nArraySize; j++ )
    {
        double dMass = pDataPeaks[j].dMass;
        double dIntensity = pDataPeaks[j].dIntensity;

        // Do something with mass intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if( varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

GetAverageMassList

long GetAverageMassList(long FAR* pnFirstAvgScanNumber, long FAR* pnLastAvgScanNumber, long FAR* pnFirstBkg1ScanNumber, long FAR* pnLastBkg1ScanNumber, long FAR* pnFirstBkg2ScanNumber, long FAR* pnLastBkg2ScanNumber, LPCTSTR szFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, VARIANT FAR* pvarMassList, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnFirstAvgScanNumber A valid pointer to a long variable containing the first scan number of the scan number range for which the corresponding averaged mass list data is to be returned.

pnLastAvgScanNumber A valid pointer to a long variable containing the last scan number of the scan number range for which the corresponding averaged mass list data is to be returned.

pnFirstBkg1ScanNumber A valid pointer to a long variable containing the first scan number of the first scan number range to be subtracted from the averaged mass list data.

pnLastBkg1ScanNumber A valid pointer to a long variable containing the last scan number of the first scan number range to be subtracted from the averaged mass list data.

pnFirstBkg2ScanNumber A valid pointer to a long variable containing the first scan number of the second scan number range to be subtracted from the averaged mass list data.

pnLastBkg2ScanNumber A valid pointer to a long variable containing the last scan number of the second scan number range to be subtracted from the averaged mass list data.

szFilter A string containing the optional scan filter.

nIntensityCutoffType The type of intensity cutoff to apply.

nIntensityCutoffValue The intensity cutoff value.

nMaxNumberOfPeaks The maximum number of data peaks to return in the mass list.

pvarMassList A valid pointer to a VARIANT variable to receive the mass list data.

pvarPeakFlags A valid pointer to a VARIANT variable to receive the peak flag data.

pnArraySize A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This function is only applicable to scanning devices such as MS and PDA.

If no scan filter is supplied, the scans between *pnFirstAvgScanNumber* and *pnLastAvgScanNumber*, inclusive, will be returned. Likewise, all the scans between *pnFirstBkg1ScanNumber* and

XRawfile OCX Documentation

pnLastBkg1ScanNumber and *pnFirstBkg2ScanNumber* and *pnLastBkg2ScanNumber*, inclusive, will be averaged and subtracted from the *pnFirstAvgScanNumber* to *pnLastAvgScanNumber* averaged scans. If a scan filter is provided, the scans in the preceding scan number ranges that match the scan filter will be utilized in obtaining the background subtracted mass list. The specified scan numbers must be valid for the current controller. If no background subtraction is to be performed, the background scan numbers should be set to zero. On return, the scan number variables will contain the actual first and last scan numbers, respectively, for the scans used. Valid scan number limits may be obtained by calling **GetFirstSpectrumNumber** and **GetLastSpectrumNumber**.

If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

To reduce the number of low intensity data peaks returned, an intensity cutoff, *nIntensityCutoffType*, may be applied. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted based on the value of *nIntensityCutoffType*. See **Cutoff Type** in the **Enumerated Types** section for the possible cutoff type values.

To limit the total number of data peaks that will be returned in the mass list, set the value of *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

The mass list contents will be returned in a SafeArray attached to the *pvarMassList* VARIANT variable. When passed in, the *pvarMassList* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarMassList* will be set to type VT_ARRAY | VT_R8. The format of the mass list returned will be an array of double precision values in mass intensity pairs in ascending mass order (e.g. mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each mass intensity pair.

On successful return, *pnArraySize* will contain the number of mass intensity pairs stored in the *pvarMassList* array.

Example

```
// example for GetAverageMassList
```

```
typedef struct _datapoint
{
    double dMass;
    double dIntensity;
} DataPoint;

long nFirstAvgScanNumber = 12;           // average scans 12 through 18
long nLastAvgScanNumber = 18;
long nFirstBkg1ScanNumber = 5;          // subtract scans 5 through 8
long nLastBkg1ScanNumber = 8;
long nFirstBkg2ScanNumber = 0;          // do not use second background scan number range
long nLastBkg2ScanNumber = 0;
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet = XRawfileCtrl.GetAverageMassList ( &nFirstAvgScanNumber,
```

XRawfile OCX Documentation

```

        &nLastAvgScanNumber,
        &nFirstBkg1ScanNumber,
        &nLastBkg1ScanNumber,
        &nFirstBkg2ScanNumber,
        &nLastBkg2ScanNumber,
        NULL,           // no filter
        0,              // no cutoff
        0,              // no cutoff
        0,              // all peaks returned
        &varMassList,   // mass list data
        &varPeakFlags, // peak flags data
        &nArraySize ); // size of mass list array

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting average mass list data."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varMassList.parray;

    DataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)&pDataPeaks );

    for( long j=0; j<nArraySize; j++ )
    {
        double dMass = pDataPeaks[j].dMass;
        double dIntensity = pDataPeaks[j].dIntensity;

        // Do something with mass intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if( varMassList.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varMassList.parray;
    varMassList.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

if( varPeakFlags.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varPeakFlags.parray;
    varPeakFlags.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}

```

XRawfile OCX Documentation

GetLabelData

```
long    GetLabelData( VARIANT FAR* pvarLabels,
                     VARIANT FAR* pvarFlags,
                     long FAR* pnScanNumber );
```

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pvarLabels	A valid pointer to a VARIANT variable to receive the label data.
pvarFlags	A valid pointer to a VARIANT variable to receive the flags.
pnScanNumber	A valid pointer to a long variable which contains the scan number for which the corresponding label data are to be returned.

Remarks

This method enables you to read the FT-PROFILE labels of a scan represented by the scanNumber.

pvarFlags can be NULL if you are not interested in receiving the flags.

The label data contains values of mass (double), intensity (double), resolution (float), baseline (float), noise (float) and charge (int).

The flags are returned as unsigned char values. There are the flags saturated, fragmented, merged, exception, reference and modified.

Example

// example for GetLabelData

```
long        nRet, nScanNumber = 1; // get the label data of the first scan.
int         dim, inx, charge;
double      *pdval;
unsigned char *pcval;
SAFEARRAY   *parray, *parray2;
_variant_t  vSpecData, vFlags;
VARIANT     varLabels, *pvarLabels;
VARIANT     varFlags, *pvarFlags;

double      dMass, dInt ;
unsigned char cMerged, cFragmented, cReference, cException, cModified, cSaturated;
TCHAR       flags[7];
float       fRes, fBase, fNoise;

pvarLabels = &varLabels;
pvarFlags  = &varFlags;
nRet = XRawfileCtrl.GetLabelData(pvarLabels, pvarFlags, &nScanNumber);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting label data."), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

```
vSpecData    = pvarLabels;
parray       = vSpecData.parray;
dim          = parray->rgsabound[0].cElements;
pdval        = (double *) parray->pvData;

if(pvarFlags)
{
    vFlags = pvarFlags;
    parray2 = vFlags.parray;
    pcval   = (unsigned char *) parray2->pvData;
}

for (inx = 0; inx < dim; inx++)
{
    dMass      = (double)    pdval[((inx)*6)+0] ;
    dInt       = (double)    pdval[((inx)*6)+1] ;
    fRes       = (float)     pdval[((inx)*6)+2] ;
    fBase      = (float)     pdval[((inx)*6)+3] ;
    fNoise     = (float)     pdval[((inx)*6)+4] ;
    charge     = (int)       pdval[((inx)*6)+5] ;
    if(pVarFlags)
    {
        cSaturated   = (unsigned char) pcval[((inx)*6)+0] ;
        cFragmented  = (unsigned char) pcval[((inx)*6)+1] ;
        cMerged      = (unsigned char) pcval[((inx)*6)+2] ;
        cException    = (unsigned char) pcval[((inx)*6)+3] ;
        cReference    = (unsigned char) pcval[((inx)*6)+4] ;
        cModified     = (unsigned char) pcval[((inx)*6)+5] ;

        // write the flags into a String
        flags[0] = _T("\0");
        if(cSaturated)
            _tcscat(flags, _T("S"));
        if(cFragmented)
            _tcscat(flags, _T("F"));
        if(cMerged)
            _tcscat(flags, _T("M"));
        if(cException)
            _tcscat(flags, _T("E"));
        if(cReference)
            _tcscat(flags, _T("R"));
        if(cModified)
            _tcscat(flags, _T("O"));
    }
    // Do something with the data.
    ...
}
```

XRawfile OCX Documentation

GetNoiseData

```
long    GetNoiseData( VARIANT FAR* pvarNoisePacket,
                    long FAR* pnScanNumber );
```

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pvarNoisePacket	A valid pointer to a VARIANT variable to receive the noise packets.
pnScanNumber	A valid pointer to a long variable which contains the scan number for which the corresponding noise packets are to be returned.

Remarks

This method enables you to read the FT-PROFILE noise packets of a scan represented by the scanNumber.

The noise packets contain values of mass (double), noise (float) and baseline (float).

Example

// example for GetNoiseData

```
long        nRet, nScanNumber = 1; // get the noise packets of the first scan.
int         dim, inx;
double      *pdval;
SAFEARRAY   *parray;
_variant_t  vSpecData;
VARIANT     varNoisePackets, *pvarNoisePackets;
double      dMass;
float       fBase, fNoise;

pvarNoisePackets = &varNoisePackets;
nRet = XRawfileCtrl.GetNoiseData(pvarNoisePackets, &nScanNumber);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting noise packets."), _T("Error"), MB_OK );
    ...
}
vSpecData      = pvarNoisePackets;
parray         = vSpecData.parray;
dim            = parray->rgsabound[0].cElements;
pdval          = (double *) parray->pvData;

for (inx = 0; inx < dim; inx++)
{
    dMass = (double)    pdval[((inx)*3)+0] ;
    fNoise = (float)     pdval[((inx)*3)+1] ;
    fBase  = (float)     pdval[((inx)*3)+2] ;

    // Do something with the data.
    ...
}
```


XRawfile OCX Documentation

IsProfileScanForScanNum

long IsProfileScanForScanNum(long nScanNumber, BOOL FAR* pblsProfileScan);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nScanNumber The scan number for which the profile data type information is to be returned.

pblsProfileScan A valid pointer to a variable of type BOOL. This variable must exist.

Remarks

Returns TRUE if the scan specified by *nScanNumber* is a profile scan, FALSE if the scan is a centroid scan. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

Example

```
// example for IsProfileScanForScanNum
long nScanNum = 12; // Is the twelfth scan a profile scan
BOOL blsProfileScan;
long nRet = XRawfileCtrl. IsProfileScanForScanNum (nScanNum, & blsProfileScan);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting profile flag for scan number 12"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

IsCentroidScanForScanNum

long IsCentroidScanForScanNum(long nScanNumber, BOOL FAR* pblsCentroidScan);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

nScanNumber The scan number for which the profile data type information is to be returned.

pblsCentroidScan A valid pointer to a variable of type BOOL. This variable must exist.

Remarks

Returns TRUE if the scan specified by *nScanNumber* is a centroid scan, FALSE if the scan is a profile scan. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

Example

```
// example for IsCentroidScanForScanNum
long nScanNum = 12; // Is the twelfth scan a centroid scan
BOOL blsCentroidScan;
long nRet = XRawfileCtrl. IsCentroidScanForScanNum (nScanNum, & blsCentroidScan);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting centroid flag for scan number 12"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetScanHeaderInfoForScanNum

long GetScanHeaderInfoForScanNum(long nScanNumber, long FAR* pnNumPackets, double FAR* pdStartTime, double FAR* pdLowMass, double FAR* pdHighMass, double FAR* pdTIC, double FAR* pdBasePeakMass, double FAR* pdBasePeakIntensity, long FAR* pnNumChannels, BOOL FAR* pbUniformTime, double FAR* pdFrequency);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which the scan header information is to be returned.
<i>pnNumPackets</i>	A valid pointer to a variable of type long to receive the number of mass intensity value pairs in the specified scan. This variable must exist.
<i>pdStartTime</i>	A valid pointer to a variable of type double to receive the retention time of the specified scan. This variable must exist.
<i>pdLowMass</i>	A valid pointer to a variable of type double to receive the low mass value of the specified scan. This variable must exist.
<i>pdHighMass</i>	A valid pointer to a variable of type double to receive the high mass value of the specified scan. This variable must exist.
<i>pdTIC</i>	A valid pointer to a variable of type double to receive the integrated total ion current value for the specified scan. This variable must exist.
<i>pdBasePeakMass</i>	A valid pointer to a variable of type double to receive the base peak mass of the specified scan. This variable must exist.
<i>pdBasePeakIntensity</i>	A valid pointer to a variable of type double to receive the intensity of the base peak mass for the specified scan. This variable must exist.
<i>pnNumChannels</i>	A valid pointer to a variable of type long to receive the number of channels acquired at the specified scan number index. This variable must exist.
<i>pbUniformTime</i>	A valid pointer to a variable of type BOOL to receive the flag indicating whether or not the sampling time increment for the current controller is uniform. This variable must exist.
<i>pdFrequency</i>	A valid pointer to a variable of type double to receive the sampling frequency for the current controller if <i>pbUniformTime</i> is TRUE. This variable must exist.

Remarks

For a given scan number, this function returns information from the scan header for the current controller. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

The validity of these parameters depends on the current controller. For example, *pdLowMass*, *pdHighMass*, *pdTIC*, *pdBasePeakMass*, and *pdBasePeakIntensity* are only likely to be set on return for

XRawfile OCX Documentation

MS or PDA controllers. *PnNumChannels* is only likely to be set on return for Analog, UV, and A/D Card controllers. *PdUniformTime*, and *pdFrequency* are only likely to be set on return for UV, and A/D Card controllers and may be valid for Analog controllers. In cases where the value is not set, a value of zero is returned.

Example

```
// example for GetScanHeaderInfoForScanNum
long nScanNum = 12; // get info for the twelfth scan
long nPackets = 0;
double dStartTime = 0.0;
double dLowMass = 0.0;
double dHighMass = 0.0;
double dTIC = 0.0;
double dBasePeakMass = 0.0;
double dBasePeakIntensity = 0.0;
long nChannels = 0;
BOOL bUniformTime = FALSE;
double dFrequency = 0.0;
long nRet = XRawfileCtrl. GetScanHeaderInfoForScanNum (    nScanNum,
                                                         &nPackets,
                                                         &dStartTime,
                                                         &dLowMass,
                                                         &dHighMass,
                                                         &dTIC,
                                                         &dBasePeakMass,
                                                         &dBasePeakIntensity,
                                                         &nChannels,
                                                         &bUniformTime,
                                                         &dFrequency );

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting scan header info"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetStatusLogForScanNum

long GetStatusLogForScanNum(long nScanNumber, double* pdStatusLogRT, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which status log information is to be returned.
<i>pdStatusLogRT</i>	A valid pointer to a variable of type double to receive the retention time at which the status log entry was recorded. This variable must exist.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pvarValues</i>	A valid pointer to a variable of type VARIANT to receive the array of text string values for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> and <i>pvarValues</i> arrays. This variable must exist.

Remarks

Returns the recorded status log entry labels and values for the current controller. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

On return, *pdStatusLogRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but will be the closest status log entry to the scan time.

The variables *pvarLabels* and *pvarValues* must be initialized to VARIANT type VT_EMPTY. On return, these variables will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetStatusLogForScanNum
long nScan = 12;           // use twelfth scan
double dStatusLogRT = 0.0;
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetStatusLogForScanNum ( nScan,
                                                    &dStatusLogRT,
                                                    &varLabels,
                                                    &varValues,
```

XRawfile OCX Documentation

```
                                &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

SAFEARRAY FAR* psaValues = varValues.parray;
varValues.parray = NULL;

BSTR* pbstrLabels = NULL;
BSTR* pbstrValues = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

if( FAILED(SafeArrayAccessData( psaValues, (void**>(&pbstrValues) ) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    SafeArrayUnaccessData( psaValues );
    SafeArrayDestroy( psaValues );
    ::MessageBox( NULL, _T("Failed to access values array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];
    sData = pbstrValues[i];

    // do something with label and value
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
SafeArrayUnaccessData( psaValues );
SafeArrayDestroy( psaValues );
```

XRawfile OCX Documentation

GetStatusLogForRT

long GetStatusLogForRT(double FAR* pdRT, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the retention time for which the closest status log entry is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pvarValues</i>	A valid pointer to a variable of type VARIANT to receive the array of text string values for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> and <i>pvarValues</i> arrays. This variable must exist.

Remarks

Returns the recorded status log entry labels and values for the current controller. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

On return, *pdRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time specified but will be the closest status log entry to the specified time.

The variables *pvarLabels* and *pvarValues* must be initialized to VARIANT type VT_EMPTY. On return, these variables will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetStatusLogForRT
double dRT = 3.8;      // 3.8 minutes
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetStatusLogForRT ( &dRT,
                                              &varLabels,
                                              &varValues,
                                              &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

```
// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

SAFEARRAY FAR* psaValues = varValues.parray;
varValues.parray = NULL;

BSTR* pbstrLabels = NULL;
BSTR* pbstrValues = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

if( FAILED(SafeArrayAccessData( psaValues, (void**>(&pbstrValues) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    SafeArrayUnaccessData( psaValues );
    SafeArrayDestroy( psaValues );
    ::MessageBox( NULL, _T("Failed to access values array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];
    sData = pbstrValues[i];

    // do something with label and value
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
SafeArrayUnaccessData( psaValues );
SafeArrayDestroy( psaValues );
```


XRawfile OCX Documentation

GetStatusLogLabelsForScanNum

long GetStatusLogLabelsForScanNum(long nScanNumber, double* pdStatusLogRT, VARIANT FAR* pvarLabels, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which status log information is to be returned.
<i>pdStatusLogRT</i>	A valid pointer to a variable of type double to receive the retention time at which the status log entry was recorded. This variable must exist.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> arrays. This variable must exist.

Remarks

Returns the recorded status log entry labels for the current controller. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

On return, *pdStatusLogRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but will be the closest status log entry to the scan time.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* array.

Example

```
// example for GetStatusLogLabelsForScanNum
long nScan = 1;           // first scan status log record
double dStatusLogRT = 0.0;
VARIANT varLabels;
VariantInit(&varLabels);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetStatusLogLabelsForScanNum (    nScan,
                                                            &dStatusLogRT,
                                                            &varLabels,
                                                            &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
```

XRawfile OCX Documentation

```
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

BSTR* pbstrLabels = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];

    // do something with label
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
```

XRawfile OCX Documentation

GetStatusLogLabelsForRT

long GetStatusLogLabelsForRT(double FAR* pdRT, VARIANT FAR* pvarLabels, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the retention time for which the closest status log entry is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested status log information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> arrays. This variable must exist.

Remarks

Returns the recorded status log entry labels for the current controller. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

On return, *pdRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time specified but will be the closest status log entry to the specified time.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* array.

Example

```
// example for GetStatusLogLabelsForRT
double dStatusLogRT = 3.8;    // 3.8 minutes
VARIANT varLabels;
VariantInit(&varLabels);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetStatusLogLabelsForRT (    &dStatusLogRT,
                                                    &varLabels,
                                                    &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

BSTR* pbstrLabels = NULL;
```

XRawfile OCX Documentation

```
if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];

    // do something with label
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
```

XRawfile OCX Documentation

GetStatusLogValueForScanNum

long GetStatusLogValueForScanNum(long nScanNumber, LPCTSTR szLabel, double* pdStatusLogRT, VARIANT FAR* pvarValue);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which status log information is to be returned.
<i>szLabel</i>	A string containing the label for which the status log parameter value is to be returned.
<i>pdStatusLogRT</i>	A valid pointer to a variable of type double to receive the retention time at which the status log entry was recorded. This variable must exist.
<i>pvarValue</i>	A valid pointer to a variable of type VARIANT to receive the status log parameter value. This variable must exist and be initialized to VT_EMPTY.

Remarks

Returns the recorded status log parameter value for the specified status log parameter label for the current controller. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

To obtain a list of the status log parameter labels, call **GetStatusLogLabelsForScanNum**.

On return, *pdStatusLogRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but will be the closest status log entry to the scan time.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of the type of the parameter stored in the data file.

Example

```
// example for GetStatusLogValueForScanNum
long nScan= 1;           // status log record for first scan
double dRT = 0.0;
VARIANT varValue;
VariantInit(&varValue);
TCHAR szLabel;
_tcscpy(szLabel, _T("Multiplier (V):")); // call GetStatusLogLabels for correct labels
long nRet = XRawfileCtrl. GetStatusLogValueForScanNum (nScan, szLabel, &dRT, &varValue);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}

// determine type and do something with value
...
```

XRawfile OCX Documentation

GetStatusLogValueForRT

long GetStatusLogValueForRT(double FAR* pdRT, LPCTSTR szLabel, VARIANT FAR* pvarValue);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the retention time for which the closest status log entry is to be returned.
<i>szLabel</i>	A string containing the label for which the status log parameter value is to be returned.
<i>pvarValue</i>	A valid pointer to a variable of type VARIANT to receive the status log parameter value. This variable must exist and be initialized to VT_EMPTY.

Remarks

Returns the recorded status log parameter value for the specified status log parameter label for the current controller. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

To obtain a list of the status log parameter labels, call **GetStatusLogLabelsForRT**.

On return, *pdRT* will contain the retention time at which the status log entry was recorded. This time may not be the same as the retention time specified but will be the closest status log entry to the specified time.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of the type of the parameter stored in the data file.

Example

```
// example for GetStatusLogValueForRT
double dRT = 3.8 minutes;
VARIANT varValue;
VariantInit(&varValue);
TCHAR szLabel;
_tcscpy(szLabel, _T("Multiplier (V)")); // call GetStatusLogLabels for correct labels
long nRet = XRawfileCtrl. GetStatusLogValueForRT (&dRT, szLabel, &varValue);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"), MB_OK );
    ...
}

// determine type and do something with value
...
```

XRawfile OCX Documentation

GetTrailerExtraForScanNum

long GetTrailerExtraForScanNum(long nScanNumber, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which trailer extra information is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pvarValues</i>	A valid pointer to a variable of type VARIANT to receive the array of text string values for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> and <i>pvarValues</i> arrays. This variable must exist.

Remarks

Returns the recorded trailer extra entry labels and values for the current controller. This function is only valid for MS controllers. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

The variables *pvarLabels* and *pvarValues* must be initialized to VARIANT type VT_EMPTY. On return, these variables will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetTrailerExtraForScanNum
long nScan = 12;           // use twelfth scan
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTrailerExtraForScanNum ( nScan,
                                                    &varLabels,
                                                    &varValues,
                                                    &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
```

XRawfile OCX Documentation

```
varLabels.parray = NULL;

SAFEARRAY FAR* psaValues = varValues.parray;
varValues.parray = NULL;

BSTR* pbstrLabels = NULL;
BSTR* pbstrValues = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

if( FAILED(SafeArrayAccessData( psaValues, (void**>(&pbstrValues) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    SafeArrayUnaccessData( psaValues );
    SafeArrayDestroy( psaValues );
    ::MessageBox( NULL, _T("Failed to access values array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];
    sData = pbstrValues[i];

    // do something with label and value
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
SafeArrayUnaccessData( psaValues );
SafeArrayDestroy( psaValues );
```


XRawfile OCX Documentation

GetTrailerExtraForRT

long GetTrailerExtraForRT(double FAR* pdRT, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the retention time for which the trailer extra entry is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pvarValues</i>	A valid pointer to a variable of type VARIANT to receive the array of text string values for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> and <i>pvarValues</i> arrays. This variable must exist.

Remarks

Returns the recorded trailer extra entry labels and values for the current controller. This function is only valid for MS controllers. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

On return, *pdRT* will contain the retention time at which the trailer extra entry was recorded. This time may not be the same as the retention time specified but will be the scan retention time of the scan closest to the specified time.

The variables *pvarLabels* and *pvarValues* must be initialized to VARIANT type VT_EMPTY. On return, these variables will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetTrailerExtraForRT
double dRT = 3.8;           // 3.8 minutes
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTrailerExtraForRT (&dRT,
                                                &varLabels,
                                                &varValues,
                                                &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
}
```

XRawfile OCX Documentation

```
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

SAFEARRAY FAR* psaValues = varValues.parray;
varValues.parray = NULL;

BSTR* pbstrLabels = NULL;
BSTR* pbstrValues = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

if( FAILED(SafeArrayAccessData( psaValues, (void**>(&pbstrValues) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    SafeArrayUnaccessData( psaValues );
    SafeArrayDestroy( psaValues );
    ::MessageBox( NULL, _T("Failed to access values array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];
    sData = pbstrValues[i];

    // do something with label and value
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
SafeArrayUnaccessData( psaValues );
SafeArrayDestroy( psaValues );
```

XRawfile OCX Documentation

GetTrailerExtraLabelsForScanNum

long GetTrailerExtraLabelsForScanNum(long nScanNumber, VARIANT FAR* pvarLabels, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which trailer extra information is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> arrays. This variable must exist.

Remarks

Returns the recorded trailer extra entry labels for the current controller. This function is only valid for MS controllers. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* array.

Example

```
// example for GetTrailerExtraLabelsForScanNum
long nScan = 1;           // first scan trailer extra record
VARIANT varLabels;
VariantInit(&varLabels);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTrailerExtraLabelsForScanNum ( nScan,
                                                            &varLabels,
                                                            &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

BSTR* pbstrLabels = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**)(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
}
```

XRawfile OCX Documentation

```
        SafeArrayDestroy( psaLabels );
        ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
    }

    for( long i=0; i<nArraySize; i++ )
    {
        sLabel = pbstrLabels[i];

        // do something with label
        ...
    }

    // Delete the SafeArray
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
```

XRawfile OCX Documentation

GetTrailerExtraLabelsForRT

long GetTrailerExtraLabelsForRT(double FAR* pdRT, VARIANT FAR* pvarLabels, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the scan retention time for which the trailer extra labels are to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested trailer extra information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> arrays. This variable must exist.

Remarks

Returns the recorded trailer extra entry labels for the current controller. This function is only valid for MS controllers. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

On return, *pdRT* will contain the retention time at which the trailer extra entry was recorded. This time may not be the same as the retention time specified but will be the retention time of the scan closest to the specified time.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* array.

Example

```
// example for GetTrailerExtraLabelsForRT
double dRT = 3.8;           // 3.8 minutes
VARIANT varLabels;
VariantInit(&varLabels);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTrailerExtraLabelsForRT ( &dRT,
                                                       &varLabels,
                                                       &nArraySize);

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

BSTR* pbstrLabels = NULL;
```

XRawfile OCX Documentation

```
if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];

    // do something with label
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
```

XRawfile OCX Documentation

GetTrailerExtraValueForScanNum

long GetTrailerExtraValueForScanNum(long nScanNumber, LPCTSTR szLabel, VARIANT FAR* pvarValue);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nScanNumber</i>	The scan number for which trailer extra information is to be returned.
<i>szLabel</i>	A string containing the label for which the trailer extra parameter value is to be returned.
<i>pvarValue</i>	A valid pointer to a variable of type VARIANT to receive the trailer extra parameter value. This variable must exist and be initialized to VT_EMPTY.

Remarks

Returns the recorded trailer extra parameter value for the specified trailer extra parameter label for the current controller. This function is only valid for MS controllers. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scan or readings for the current controller may be obtained by calling **GetFirstScanNumber** and **GetLastScanNumber**.

To obtain a list of the status log parameter labels, call **GetTrailerExtraLabelsForScanNum**.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of the type of the parameter stored in the data file.

Example

```
// example for GetTrailerExtraValueForScanNum
long nScan= 1;           // trailer extra record for first scan
VARIANT varValue;
VariantInit(&varValue);
TCHAR szLabel;
_tcscpy(szLabel, _T("Charge State:")); // call GetTrailerExtraLabelsForScanNum for correct labels
long nRet = XRawfileCtrl. GetTrailerExtraValueForScanNum (nScan, szLabel, &varValue);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
    ...
}

// determine type and do something with value
...
```

XRawfile OCX Documentation

GetTrailerExtraValueForRT

long GetTrailerExtraValueForRT(double FAR* *pdRT*, LPCTSTR *szLabel*, VARIANT FAR* *pvarValue*);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>pdRT</i>	A valid pointer to a variable of type double containing the retention time for which the closest trailer extra entry is to be returned.
<i>szLabel</i>	A string containing the label for which the trailer extra parameter value is to be returned.
<i>pvarValue</i>	A valid pointer to a variable of type VARIANT to receive the trailer extra parameter value. This variable must exist and be initialized to VT_EMPTY.

Remarks

Returns the recorded trailer extra parameter value for the specified trailer extra parameter label for the current controller. This function is only valid for MS controllers. The value of *pdRT* must be within the retention time range for the current controller. The retention time range for the current controller may be obtained by calling **GetStartTime** and **GetEndTime**.

To obtain a list of the trailer extra parameter labels, call **GetTrailerExtraLabelsForRT**.

On return, *pdRT* will contain the retention time at which the trailer extra entry was recorded. This time may not be the same as the retention time specified but will be the retention time of the scan closest to the specified time.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of the type of the parameter stored in the data file.

Example

```
// example for GetTrailerExtraValueForRT
double dRT = 3.8 minutes;
VARIANT varValue;
VariantInit(&varValue);
TCHAR szLabel;
_tcscpy(szLabel, _T("Charge State:")); // call GetTrailerExtraLabelsForRT for correct labels
long nRet = XRawfileCtrl. GetTrailerExtraValueForRT (&dRT, szLabel, &varValue);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"), MB_OK );
    ...
}

// determine type and do something with value
...
```


XRawfile OCX Documentation

GetErrorLogItem

long GetErrorLogItem(long nItemNumber, double FAR* pdRT, BSTR FAR* pbstrErrorMessage);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nItemNumber</i>	The error log item number for which information is to be returned.
<i>pdRT</i>	A valid pointer to a variable of type double to receive the retention time at which the error occurred. This variable must exist.
<i>pbstrErrorMessage</i>	A valid pointer to a variable of type BSTR to receive the text string describing the error. This variable must exist and be initialized to NULL.

Remarks

Returns the specified error log item information and the retention time at which the error occurred. The value of *nItemNumber* must be within the range of one to the number of error log items recorded for the current controller. The number of error log items for the current controller may be obtained by calling **GetNumErrorLog**.

Example

```
// example for GetErrorLogItem
long nItem = 1;           // first error item number
double dRT = 0.0;
BSTR bstrMessage = NULL;
long nRet = XRawfileCtrl. GetErrorLogItem (nItem, &dRT, &bstrMessage);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting error log information"), _T("Error"), MB_OK );
    ...
}
...
SysFreeString( bstrMessage );
```

XRawfile OCX Documentation

GetTuneData

long GetTuneData(long nSegmentNumber, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nSegmentNumber</i>	The acquisition segment for which tune information is to be returned.
<i>pdRT</i>	A valid pointer to a variable of type double to receive the retention time at which the error occurred. This variable must exist.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested tune information. This variable must exist and be initialized to VT_EMPTY.
<i>pvarValues</i>	A valid pointer to a variable of type VARIANT to receive the array of text string values for the requested tune information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> and <i>pvarValues</i> arrays. This variable must exist.

Remarks

Returns the recorded tune parameter labels and values for the current controller. This function is only valid for MS controllers. The value of *nSegmentNumber* must be within the range of one to the number of tune data items recorded for the current controller. The number of tune data items for the current controller may be obtained by calling **GetNumTuneData**.

The variables *pvarLabels* and *pvarValues* must be initialized to VARIANT type VT_EMPTY. On return, these variables will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetTuneData
long nSegment = 1;           // first tune record
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTuneData (nSegment, &varLabels, &varValues, &nArraySize);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting tune record information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
```

XRawfile OCX Documentation

```
varLabels.parray = NULL;

SAFEARRAY FAR* psaValues = varValues.parray;
varValues.parray = NULL;

BSTR* pbstrLabels = NULL;
BSTR* pbstrValues = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**>(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}

if( FAILED(SafeArrayAccessData( psaValues, (void**>(&pbstrValues) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    SafeArrayUnaccessData( psaValues );
    SafeArrayDestroy( psaValues );
    ::MessageBox( NULL, _T("Failed to access values array"), _T("Error"), MB_OK );
}

for( long i=0; i<nArraySize; i++ )
{
    sLabel = pbstrLabels[i];
    sData = pbstrValues[i];

    // do something with label and value
    ...
}

// Delete the SafeArray
SafeArrayUnaccessData( psaLabels );
SafeArrayDestroy( psaLabels );
SafeArrayUnaccessData( psaValues );
SafeArrayDestroy( psaValues );
```

XRawfile OCX Documentation

GetTuneDataValue

long GetTuneDataValue(long nSegmentNumber, LPCTSTR szLabel, VARIANT FAR* pvarValue);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nSegmentNumber</i>	The acquisition segment for which tune information is to be returned.
<i>szLabel</i>	A string containing the label for which the tune parameter value is to be returned.
<i>pvarValue</i>	A valid pointer to a variable of type VARIANT to receive the tune parameter value. This variable must exist and be initialized to VT_EMPTY.

Remarks

Returns the recorded tune parameter value for the specified tune parameter label for the current controller. This function is only valid for MS controllers. The value of *nSegmentNumber* must be within the range of one to the number of tune data items recorded for the current controller. The number of tune data items for the current controller may be obtained by calling **GetNumTuneData**.

To obtain a list of the tune parameter labels, call **GetTuneDataLabels**.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of the type of the parameter stored in the data file.

Example

```
// example for GetTuneDataValue
long nSegment = 1;           // first tune record
VARIANT varValue;
VariantInit(&varValue);
TCHAR szLabel;
_tcscpy(szLabel, _T("Ion Time (ms):")); // call GetTuneDataLabels for correct labels
long nRet = XRawfileCtrl. GetTuneDataValue (nSegment, szLabel, &varValue);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting tune record information"), _T("Error"), MB_OK );
    ...
}

// determine type and do something with value
...
```

XRawfile OCX Documentation

GetTuneDataLabels

long GetTuneDataLabels(long nSegmentNumber, VARIANT FAR* pvarLabels, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nSegmentNumber</i>	The acquisition segment for which tune information is to be returned.
<i>pvarLabels</i>	A valid pointer to a variable of type VARIANT to receive the array of text string labels for the requested tune information. This variable must exist and be initialized to VT_EMPTY.
<i>pnArraySize</i>	A valid pointer to a variable of type long to receive the number of records returned in the <i>pvarLabels</i> array. This variable must exist.

Remarks

Returns the recorded tune parameter labels for the current controller. This function is only valid for MS controllers. The value of *nSegmentNumber* must be within the range of one to the number of tune data items recorded for the current controller. The number of tune data items for the current controller may be obtained by calling **GetNumTuneData**.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable will be of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* will contain the number of entries in the *pvarLabels* array.

Example

```
// example for GetTuneDataLabels
long nSegment = 1;           // first tune record
VARIANT varLabels;
VariantInit(&varLabels);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetTuneDataLabels (nSegment, &varLabels, &nArraySize);
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting tune record information"), _T("Error"), MB_OK );
    ...
}

// Get a pointer to the SafeArray
SAFEARRAY FAR* psaLabels = varLabels.parray;
varLabels.parray = NULL;

BSTR* pbstrLabels = NULL;

if( FAILED(SafeArrayAccessData( psaLabels, (void**)(&pbstrLabels) ) ) )
{
    SafeArrayUnaccessData( psaLabels );
    SafeArrayDestroy( psaLabels );
    ::MessageBox( NULL, _T("Failed to access labels array"), _T("Error"), MB_OK );
}
```

XRawfile OCX Documentation

```
}  
  
for( long i=0; i<nArraySize; i++ )  
{  
    sLabel = pbstrLabels[i];  
  
    // do something with label  
    ...  
}  
  
// Delete the SafeArray  
SafeArrayUnaccessData( psaLabels );  
SafeArrayDestroy( psaLabels );
```

XRawfile OCX Documentation

GetNumInstMethods

long **GetNumInstMethods**(long FAR* pnNumInstMethods);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

pnNumInstMethods A valid pointer to a long variable to receive the number of instrument methods contained in the raw file.

Remarks

Returns the number of instrument methods contained in the raw file. Each instrument used in the acquisition for which a method was created in Instrument Setup (e.g. autosampler, LC, MS, PDA) will have its instrument method contained in the raw file.

Example

```
// example for GetNumInstMethods
long nMethods;
long nRet = XRawfileCtrl.GetNumInstMethods ( &nMethods );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error retrieving number of inst methods"), _T("Error"), MB_OK );
    ...
}
```

XRawfile OCX Documentation

GetInstMethod

long GetInstMethod(long nInstMethodItem, BSTR FAR* pbstrInstMethod);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nInstMethodItem</i>	A long variable containing the index value of the instrument method to be returned.
<i>pbstrFilter</i>	A valid pointer to a BSTR. This variable must exist and be initialized to NULL.

Remarks

Returns the channel label, if available, at the specified index for the current controller. This field is only relevant to channel devices such as UV detectors, A/D cards, and Analog inputs. Channel labels indices are numbered starting at 0.

Returns the instrument method, if available, at the index specified in *nInstMethodItem*. The instrument method indices are numbered starting at 0. The number of instrument methods can be obtained by calling **GetNumInstMethods**.

Example

```
// example for GetInstMethod
long nInstMethodItem = 4;      // get the fifth instrument method
BSTR bstrMethod = NULL;
long nRet = XRawfileCtrl.GetInstMethod ( nInstMethodItem, &bstrMethod );
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting fifth instrument method."), _T("Error"), MB_OK );
    ...
}
...
SysFreeString(bstrMethod);
```


XRawfile OCX Documentation

GetChroData

long GetChroData(long nChroType1, long nChroOperator, long nChroType2, LPCTSTR szFilter, LPCTSTR szMassRanges1, LPCTSTR szMassRanges2, double dDelay, double FAR* pdStartTime, double FAR* pdEndTime, long nSmoothingType, long nSmoothingValue, VARIANT FAR* pvarChroData, VARIANT FAR* pvarPeakFlags, long FAR* pnArraySize);

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

<i>nChroType1</i>	A long variable containing the first chromatogram trace type of interest.
<i>nChroOperator</i>	A long variable containing the chromatogram trace operator.
<i>nChroType2</i>	A long variable containing the second chromatogram trace type of interest.
<i>szFilter</i>	A string containing the formatted scan filter.
<i>szMassRanges1</i>	A string containing the formatted mass ranges for the first chromatogram trace type.
<i>szMassRanges2</i>	A string containing the formatted mass ranges for the second chromatogram trace type.
<i>dDelay</i>	A double precision variable containing the chromatogram delay in minutes.
<i>pdStartTime</i>	A pointer to a double precision variable containing the start time of the chromatogram time range to return.
<i>pdEndTime</i>	A pointer to a double precision variable containing the end time of the chromatogram time range to return.
<i>nSmoothingType</i>	A long variable containing the type of chromatogram smoothing to be performed.
<i>nSmoothingValue</i>	A long variable containing the chromatogram smoothing value.
<i>pvarChroData</i>	A valid pointer to a VARIANT variable to receive the chromatogram data.
<i>pvarPeakFlags</i>	A valid pointer to a VARIANT variable to receive the peak flag data.
<i>pnArraySize</i>	A valid pointer to a long variable to receive the number of data peaks returned in the chromatogram array.
<i>pnArraySize</i>	A pointer to a long variable to receive the size of the returned chromatogram array.

Remarks

Returns the requested chromatogram data as an array of double precision time intensity pairs in *pvarChroData*. The number of time intensity pairs is returned in *pnArraySize*.

XRawfile OCX Documentation

The chromatogram trace types and operator values of *nChroType1*, *nChroOperator*, and *nChroType2* are dependent on the current controller. See **Chromatogram Type** and **Chromatogram Operator** in the **Enumerated Types** section for a list of the valid values for the different controller types.

The scan filter field is only valid for MS controllers. If no scan filter is to be provided, the value of *szFilter* may be NULL or an empty string. Scan filters must match the Xcalibur scan filter format. See the topic **scan filters format, definition** in the Xcalibur online help for information on how to construct a scan filter.

The *dDelay* value contains the retention time offset to add to the returned chromatogram times. The value may be set to 0.0 if no offset is desired. This value must be 0.0 for MS controllers. It must be greater than or equal to 0.0 for all other controller types.

The mass ranges are only valid for MS or PDA controllers. For all other controller types, these fields must be NULL or empty strings. For MS controllers, the mass ranges must be correctly formatted mass ranges and are only valid for Mass Range and Base Peak chromatogram trace types. For PDA controllers, the mass ranges must be correctly formatted wavelength ranges and are only valid for Wavelength Range and Spectrum Maximum chromatogram trace types. These values may be left empty for Base Peak or Spectrum Maximum trace types but must be specified for Mass Range or Wavelength Range trace types. See the topic **Mass1 (m/z) text box** in the Xcalibur online help for information on how to format mass ranges.

The start and end times, *pdStartTime* and *pdEndTime*, may be used to return a portion of the chromatogram. The start time and end time must be within the acquisition time range of the current controller which may be obtained by calling **GetStartTime** and **GetEndTime**, respectively. Alternatively, if the entire chromatogram is to be returned, *pdStartTime* and *pdEndTime* may be set to zero. On return, *pdStartTime* and *pdEndTime* will contain the actual time range of the returned chromatographic data.

The *nSmoothingType* variable contains the type of smoothing to perform on the returned chromatographic data. See **Smoothing Type** in the **Enumerated Types** section for a list of the valid values for *nSmoothingType*. The value of *nSmoothingValue* must be an odd number in the range of 3-15 if smoothing is desired.

The chromatogram list contents will be returned in a SafeArray attached to the *pvarChroData* VARIANT variable. When passed in, the *pvarChroData* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarChroData* will be set to type VT_ARRAY | VT_R8. The format of the chromatogram list returned will be an array of double precision values in time intensity pairs in ascending time order (e.g. time 1, intensity 1, time 2, intensity 2, time 3, intensity 3, etc.)

The *pvarPeakFlags* variable is currently not used. This variable is reserved to future use to return flag information, such as saturation, about each time intensity pair.

On successful return, *pnArraySize* will contain the number of time intensity pairs stored in the *pvarChroData* array.

Example

```
// example for GetChroData to return the MS TIC trace
```

```
typedef struct _datapeak
{
    double dTime;
    double dIntensity;
} ChroDataPeak;
```

```
XRawfileCtrl.SetCurrentController ( 0, 1 );    // first MS controller
```

XRawfile OCX Documentation

```
VARIANT varChroData;
VariantInit(&varChroData);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
double dStartTime = 0.0;
double dEndTime = 0.0;
long nRet = XRawfileCtrl.GetChroData ( 1,          // TIC trace
                                     0,
                                     0,
                                     NULL,
                                     NULL,
                                     NULL,
                                     0.0,
                                     &dStartTime,
                                     &dEndTime,
                                     0,
                                     0,
                                     &varChroData,
                                     &varPeakFlags,
                                     &nArraySize );

if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error getting chro data."), _T("Error"), MB_OK );
    ...
}

if( nArraySize )
{
    // Get a pointer to the SafeArray
    SAFEARRAY FAR* psa = varChroData.parray;

    ChroDataPeak* pDataPeaks = NULL;
    SafeArrayAccessData( psa, (void**)&pDataPeaks );

    for( long j=0; j<nArraySize; j++ )
    {
        double dTime = pDataPeaks[j].dTime;
        double dIntensity = pDataPeaks[j].dIntensity;

        // Do something with time intensity values
        ...
    }

    // Release the data handle
    SafeArrayUnaccessData( psa );
}

if(varChroData.vt != VT_EMPTY )
{
    SAFEARRAY FAR* psa = varChroData.parray;
    varChroData.parray = NULL;

    // Delete the SafeArray
    SafeArrayDestroy( psa );
}
```

XRawfile OCX Documentation

```
}  
  
if(varPeakFlags.vt != VT_EMPTY )  
{  
    SAFEARRAY FAR* psa = varPeakFlags.parray;  
    varPeakFlags.parray = NULL;  
  
    // Delete the SafeArray  
    SafeArrayDestroy( psa );  
}
```

XRawfile OCX Documentation

RefreshViewOfFile

long RefreshViewOfFile();

Return Value

1 if successful; otherwise, see **Error Codes**.

Parameters

This function has no parameters.

Remarks

Refreshes the view of a file currently being acquired. This function provides a more efficient mechanism for gaining access to new data in a raw file during acquisition without closing and re-opening the raw file. This function has no effect with files that are not being acquired.

Example

```
// example for RefreshViewOfFile
long nRet = XRawfileCtrl.RefreshViewOfFile();
if( nRet != 1 )
{
    ::MessageBox( NULL, _T("Error file refreshing view of file"), _T("Error"), MB_OK );
    ...
}
```