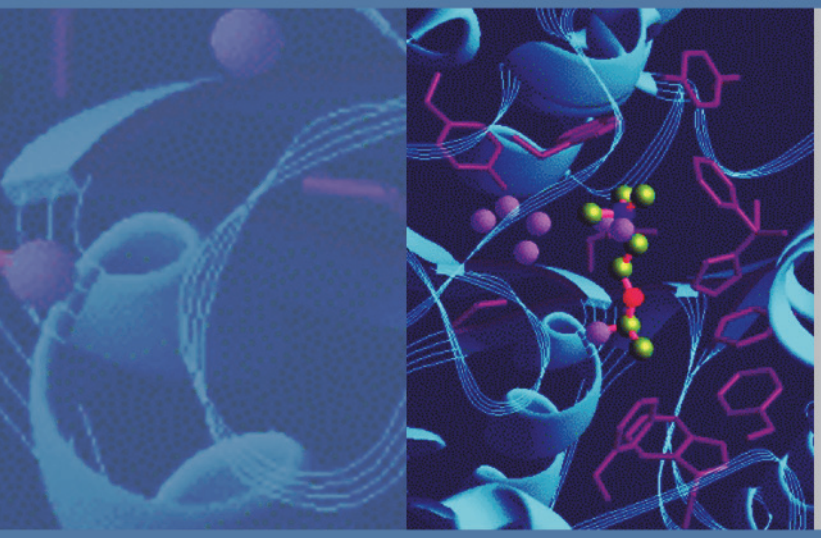

MSFileReader

Version 2.2

Reference Guide

Revision A August 2011

Part of Thermo Fisher Scientific

Thermo SCIENTIFIC

Release history: Revision A August 2011

Software Version 2.2

For Research Use Only. Not for use in diagnostic procedures.

Contents

Library Definitions

MSFileReader is designed to support read access to Xcalibur .raw files in a simple-to-use standard COM object without requiring installation of the Xcalibur data system. It is designed to be used with C++-based applications, but since it is a COM object, you can use any language that supports the COM interface.

Most of the functions described in this document are simple data access functions, but the manual also includes a number of higher-level functions that extract and manipulate the data in commonly used ways.

The description of each function includes the following information:

- The function calling sequence
- Return values
- Definitions of both input and output parameters
- An example using the function

The enumeration types described in this chapter standardize the handling of the following components: the variables set by the instrument (or instruments) during acquisition, and algorithms in the post-acquisition processing of data. Using these variables, you can determine the instrument type (mass spectrometer, analog signal, and so on), and you can select the number of the instrument (with a multiple instrument configuration) used to generate data to the raw file. The described enumerators also handle access to selected channels from multiple-channel data.

All of the described methods provide for systematic error checking and handling. This chapter also provides descriptions related to the error codes returned as a result of trapped errors.

Contents

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 GetSeqRowSampleType(…) function. The returned value has the following meaning.

Value	Sample type
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.

Value	Controller type
-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.

Value	Cutoff type
0	None (all values returned)
1	Absolute (in intensity units)
2	Relative (to base peak)

Chromatogram Type

Chromatogram type is specified in the GetChroData(…) function. 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)

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)

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 GetChroData(…) function. This value has the following meaning.

Value	Chro operator
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 GetChroData(…) function. This value has the following meaning.

Value	Smoothing type
0	None (no smoothing)
1	Boxcar
2	Gaussian

Error Codes

Table 1. Error Codes (Sheet 1 of 5)

List of error codes	
clNO_DATA_PRESENT	= 0x80002101;
Does not typically indicate an error. This code may be returned if optional data is not contained in the current raw file.	
clSUCCESS	= 0;
Indicates that the function call to the dll was processed without error.	
clFAILED	= 0x80004005;
Indicates that a general error has occurred. This code may be returned whenever an error of indeterminate origin occurs.	
clCOL_INDEX_OUT_OF_RANGE	= 0x80002206;
Returns if the column index is out of the range.	
clUSER_INDEX_OUT_OF_RANGE	= 0x80002205;
Returns if the user index is out of the range.	
clSEQ_ROW_INVALID	= 0x80002204;
Returns if the sequence row is invalid.	
clSEQ_FILE_READONLY	= 0x80002203;
Returns if the sequence file is read only.	
clNEW_FILE_READONLY	= 0x80002202;
Returns if the file to be created is read only.	
clSEQ_FILE_INVALID	= 0x80002201;
Returns if the sequence file is invalid.	

Table 1. Error Codes (Sheet 2 of 5)

List of error codes	
clINCOMPLETE_PARAMETER_SET	= 0x80002117;
Returns if the parameter set is wrong.	
clFILE_CRC_CHECK_FAILED	= 0x80002116;
Returns if the CRC check fails.	
clFILE_ALREADY_EXISTS	= 0x80002115;
Returns if creating a file that already exists.	
clFILE_NOT_FOUND	= 0x80002114;
Returns if the file specified is not found.	
clFILE_INVALID	= 0x80002113;
Returns if no valid raw file is currently open.	
clMETHOD_SCAN_EVENTS_NOT_INITIALIZED	= 0x80002112;
Returns if the scan event in the method file is not initialized.	
clTUNE_DATA_HEADER_NOT_INITIALIZED	= 0x80002111;
Returns if the tune data header is not initialized.	
clSTATUS_LOG_HEADER_NOT_INITIALIZED	= 0x80002110;
Returns if the status log header is not initialized.	
clTRAILER_HEADER_NOT_INITIALIZED	= 0x8000210f;
Returns if the trailer header is not initialized.	
clVIRUV_CREATION_FAILED	= 0x8000210e;
Returns if the UV file creation fails.	

Table 1. Error Codes (Sheet 3 of 5)

List of error codes	
clVIRUV_INVALID	= 0x8000210d;
Returns if the UV data is invalid.	
clVIRMS_CREATION_FAILED	= 0x8000210c;
Returns if the MS data creation fails.	
clVIRMS_INVALID	= 0x8000210b;
Returns if the MS data is invalid.	
clRAW_FILE_SAVE_FAILED	= 0x8000210a;
Returns if the raw file saving fails.	
clRAW_FILE_CREATION_FAILED	= 0x80002109;
Returns if the raw file creation fails.	
clMASS_RANGE_FORMAT_INCORRECT	= 0x80002108;
Returns if an incorrectly formatted mass range is passed in a function call. Mass ranges should have the same format as entered in Xcalibur applications.	
clFILTER_FORMAT_INCORRECT	= 0x80002107;
Returns if an incorrectly formatted scan filter is passed in a function call. See the topic **scan filters – format, definition** in Xcalibur Help for scan filter format specifications.	
clOPERATION_NOT_SUPPORTED_ON_CURRENT_CONTROLLER	= 0x80002105;
Returns if the requested action is inappropriate for the currently defined controller. Some functions only apply to specific controllers. This code might 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).	

Table 1. Error Codes (Sheet 4 of 5)

List of error codes	
clCURRENT_CONTROLLER_INVALID	= 0x80002104;
Returns if no current controller has been specified.	
clRAW_FILE_INVALID	= 0x80002103;
Returns if the raw file is invalid.	
clNO_DATA_PRESENT	= 0x80002101;
Returns if there is no data in the specified file.	
clRAW_FILE_OPEN_FAIL	= 0x80002100;
Returns if the raw file cannot be opened.	
clINSTMETHOD_NOT_EMBEDDED	= 0x80002099;
Returns if there is no emebedded instrument method in the raw file.	
clRAW_FILE_TOOOLDVER	= 0x80002098;
Returns if the raw file version is too old that is not supported by Xcalibur.	
clINSTMETHOD_CREATE_FAIL	= 0x80002097;
Returns if the instrument method file creation fails.	
const long clMPE_SCANFILTER	= 0x80002117;
Empty scan filters.	
const long clMPE_EMPTY_SCANHEADER	= 0x80002118;
Scan header is empty.	
const long clMPE_FILTER	= 0x80002119;
The filter format is not supported.	
const long clRAW_FILE_OPEN_FAIL	= 0x80002100;

Table 1. Error Codes (Sheet 5 of 5)

List of error codes
Failed to open raw file.
const long clMPE_RESOLUTIONERROR = 0x80002101;
The resolution value is wrong with the raw file.
const long clINSTMETHOD_NOT_EMBEDDED = 0x80002099;
In the raw file, the instrument method is not embedded.
const long clRAW_FILE_TOOOLDVER = 0x80002098;
The version of the raw file is too old or is not supported.
const long clINSTMETHOD_CREATE_FAIL = 0x80002097;
Failed to create instrument method extracted from the raw file.
const long clMPE_INVALID_SCANHEADER = 0x8000211a;
Returns when the scan header is invalid.
const long clMPE_INVALID_SCANTRAIL = 0x8000211b;
Returns when the scan trail data is invalid.
const long clMPE_INVALID_RAWDATA = 0x8000211c;
Returns when the raw data is invalid.
const long clMPE_INVALID_LABELDATA = 0x8000211d;
Returns when the label data is invalid.

Function Reference

This chapter provides extensive descriptions on the function or methods available to programmers using MSFileReader.dll. The examples contained are written using the C programming language but should also serve experienced programmers of other common programming languages well.

Open

long Open(LPCTSTR szFileName)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error opening file"), _T("Error"), MB_OK );
        …
}
```

Close

long Close()

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error closing file"), _T("Error"), MB_OK );
        …
}
```

GetFileName

long GetFileName(BSTR FAR* pbstrFileName)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting file name"), _T("Error"), MB_OK );
        …
}
…
SysFreeString( bstrFileName );
```

GetCreatorID

long GetCreatorID(BSTR FAR* pbstrCreatorID)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting creator ID"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrCreatorID);
```

GetVersionNumber

long GetVersionNumber(long FAR* pnVersion)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting version number"), _T("Error"), MB_OK );
        …
}
```

GetCreationDate

long GetCreationDate(DATE FAR* pCreationDate)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting creation date"), _T("Error"), MB_OK );
        …
}
```

IsError

long IsError(BOOL FAR* pbIsError)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pbIsError 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. For information about the error, call the GetErrorCode or GetErrorMessage functions.

Example

```
// example for IsError
BOOL bError;
long nRet = XRawfileCtrl.IsError ( &bError );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting error flag"), _T("Error"), MB_OK );
        …
}
```

IsNewFile

long IsNewFile(BOOL FAR* pbIsNewFile)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pbIsNewFile 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting new file flag"), _T("Error"), MB_OK );
        …
}
```

GetErrorCode

long GetErrorCode(long FAR* pnErrorCode)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting error code"), _T("Error"), MB_OK );
        …
}
```

GetErrorMessage

long GetErrorMessage(BSTR FAR* pbstrErrorMessage)

Return Value

0 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 is empty.

Example

```
// example for GetErrorMessage
BSTR bstrErrorMessage = NULL;
long nRet = XRawfileCtrl.GetErrorMessage ( &bstrErrorMessage );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting error message"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrErrorMessage);
```

GetWarningMessage

long GetWarningMessage(BSTR FAR* pbstrWarningMessage)

Return Value

0 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 is empty.

Example

```
// example for GetWarningMessage
BSTR bstrWarningMessage = NULL;
long nRet = XRawfileCtrl.GetWarningMessage ( &bstrWarningMessage );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting warning message"), _T("Error"), MB_OK
);
        …
}
…
SysFreeString(bstrWarningMessage);
```

GetSeqRowNumber

long GetSeqRowNumber(long FAR* pnSeqRowNumber)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row number"), _T("Error"), MB_OK );
        …
}
```

GetSeqRowSampleType

long GetSeqRowSampleType(long FAR* pnSampleType)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row sample type"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowDataPath

long GetSeqRowDataPath(BSTR FAR* pbstrDataPath)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting data path"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrPath);
```

GetSeqRowRawFileName

long GetSeqRowRawFileName(BSTR FAR* pbstrRawFileName)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting raw file name"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrFile);
```

GetSeqRowSampleName

long GetSeqRowSampleName(BSTR FAR* pbstrSampleName)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample name"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrSampleName);
```

GetSeqRowSampleID

long GetSeqRowSampleID(BSTR FAR* pbstrSampleID)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample ID"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrSampleID);
```

GetSeqRowComment

long GetSeqRowComment(BSTR FAR* pbstrComment)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting comment from seq row"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrComment);
```

GetSeqRowLevelName

long GetSeqRowLevelName(BSTR FAR* pbstrLevelName)

Return Value

0 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 is 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting level name from seq row"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrLevel);
```

GetSeqRowUserText

long GetSeqRowUserText(long nIndex, BSTR FAR* pbstrUserText)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting user text 3 from seq row"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrUserText);
```

GetSeqRowInstrumentMethod

long GetSeqRowInstrumentMethod(BSTR FAR* pbstrInstrumentMethod)

Return Value

0 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. If the raw file is created by file format conversion or acquired from a tuning program, this field is empty.

Example

```
// example for GetSeqRowInstrumentMethod
BSTR bstrMethod = NULL;
long nRet = XRawfileCtrl.GetSeqRowInstrumentMethod ( &bstrMethod );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument method from seq row"),
_T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrMethod);
```

GetSeqRowProcessingMethod

long GetSeqRowProcessingMethod(BSTR FAR* pbstrProcessingMethod)

Return Value

0 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. If no processing method is specified at the time of acquisition, this field is empty.

Example

```
// example for GetSeqRowProcessingMethod
BSTR bstrMethod = NULL;
long nRet = XRawfileCtrl.GetSeqRowProcessingMethod ( &bstrMethod );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting processing method from seq row"),
_T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrMethod);
```

GetSetRowCalibrationFile

long GetSetRowCalibrationFile(BSTR FAR* pbstrCalibrationFile)

Return Value

0 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. If no calibration file is specified at the time of acquisition, this field is empty.

Example

```
// example for GetSeqRowCalibrationFile
BSTR bstrCalFile = NULL;
long nRet = XRawfileCtrl.GetSeqRowCalibrationFile ( &bstrCalFile );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting calibration file from seq row"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrCalFile);
```

GetSeqRowVial

long GetSeqRowVial(BSTR FAR* pbstrVial)

Return Value

0 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. If the raw file is not acquired using an autosampler, this value should be ignored.

Example

```
// example for GetSeqRowVial
BSTR bstrVial = NULL;
long nRet = XRawfileCtrl.GetSeqRowVial ( &bstrVial );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting vial number from seq row"), _T("Error"),
MB_OK
);
        …
}
…
SysFreeString(bstrVial);
```

GetSeqRowInjectionVolume

long GetSeqRowInjectionVolume(double FAR* pdInjVol)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row injection volume"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowSampleWeight

long GetSeqRowSampleWeight(double FAR* pdSampleWt)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row sample weight"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowSampleVolume

long GetSeqRowSampleVolume(double FAR* pdSampleVolume)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row sample volume"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowISTDAmount

long GetSeqRowISTDAmount(double FAR* pdISTDAmount)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row ISTD amount"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowDilutionFactor

long GetSeqRowDilutionFactor(double FAR* pdDilutionFactor)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting seq row dilution factor"), _T("Error"),
MB_OK );
        …
}
```

GetSeqRowUserLabel

long GetSeqRowUserLabel(long nIndex, BSTR FAR* pbstrUserLabel)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting user label 3 from seq row"), _T("Error"),
MB_OK
);
        …
}
…
SysFreeString(bstrUserLabel);
```

InAcquisition

long InAcquisition(BOOL FAR* pbInAcquisition)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pbInAcquisition 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting acquisition state flag"), _T("Error"),
MB_OK );
        …
}
```

GetNumberOfControllers

long GetNumberOfControllers(long FAR* pnNumControllers)

Return Value

0 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, and so on. 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of controllers"), _T("Error"),
MB_OK );
        …
}
```

GetNumberOfControllersOfType

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nControllerType	The controller type that are requested for the number of registered controllers of that type.
pnNumControllers	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 != 0 )
{
          ::MessageBox( NULL, _T("Error getting number of controllers of type MS"),
_T("Error"), MB_OK );
          …
}
```

GetControllerType

long GetControllerType(long nIndex, long FAR* pnControllerType)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nIndex	The index value of the controller type that is returned.
pnControllerType	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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting type for first controller"), _T("Error"),
MB_OK );
        …
}
```

SetCurrentController

long SetCurrentController(long nControllerType, long nControllerNumber)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nControllerType The type of controller for which information is 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 (for example, MS or UV data) may be made. All requests for data specific to a device controller are 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 is more than one registered device controller of the same type (for example, 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 nContorllerNumber = 1;        // first MS device

long nRet = XRawfileCtrl.SetCurrentController ( nControllerType, nControllerNumber );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error setting current controller"), _T("Error"), MB_OK );
        …
}

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

GetCurrentController

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

Return Value

0 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 is more than one registered device controller of the same type (for example, 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 nContorllerNumber;
long nRet = XRawfileCtrl.GetCurrentController ( &nControllerType, &nControllerNumber );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting current controller"), _T("Error"), MB_OK );
        …
}
```

GetNumSpectra

long GetNumSpectra(long FAR* pnNumberOfSpectra)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of spectra"), _T("Error"), MB_OK
);
        …
}
```

GetNumStatusLog

long GetNumStatusLog(long FAR* pnNumberOfStatusLogEntries)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of status log entries"), _T("Error"),
MB_OK );
        …
}
```

GetNumErrorLog

long GetNumErrorLog(long FAR* pnNumberOfErrorLogEntries)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of error log entries"), _T("Error"),
MB_OK );
        …
}
```

GetNumTuneData

long GetNumTuneData(long FAR* pnNumTuneData)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of tune data entries"), _T("Error"),
MB_OK );
        …
}
```

GetMassResolution

long GetMassResolution(double FAR* pdMassResolution)

Return Value

0 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 returns 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting mass resolution"), _T("Error"), MB_OK );
        …
}
```

GetExpectedRunTime

long GetExpectedRunTime(double FAR* pdExpectedRunTime)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting expected run time"), _T("Error"), MB_OK
);
        …
}
```

GetNumTrailerExtra

long GetNumTrailerExtra(long FAR* pnNumberOfTrailerExtraEntries)

Return Value

0 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 are used to store instrument specific information for each scan if used.

Example

```
// example for GetNumTrailerExtra
long nTrailerExtra;
long nRet = XRawfileCtrl.GetNumTrailerExtra (&nTrailerExtra);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of trailer extra entries"),
_T("Error"), MB_OK );
        …
}
```

GetLowMass

long GetLowMass(double FAR* pdLowMass)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting low mass"), _T("Error"), MB_OK );
        …
}
```

GetHighMass

long GetHighMass(double FAR* pdHighMass)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting high mass"), _T("Error"), MB_OK );
        …
}
```

GetStartTime

long GetStartTime(double FAR* pdStartTime)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting start time"), _T("Error"), MB_OK );
        …
}
```

GetEndTime

long GetEndTime(double FAR* pdEndTime)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting end time"), _T("Error"), MB_OK );
        …
}
```

GetMaxIntegratedIntensity

long GetMaxIntegratedIntensity(double FAR* pdMaxIntegIntensity)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting max integrated intensity"), _T("Error"),
MB_OK );
        …
}
```

GetMaxIntensity

long GetMaxIntensity(long FAR* pnMaxIntensity)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting max intensity"), _T("Error"), MB_OK );
        …
}
```

GetFirstSpectrumNumber

long GetFirstSpectrumNumber(long FAR* pnFirstSpectrum)

Return Value

0 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. If data has been acquired, this value is always one.

Example

```
// example for GetFirstSpectrumNumber
long nFirstScan;
long nRet = XRawfileCtrl.GetFirstSpectrumNumber (&nFirstScan);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting first scan number"), _T("Error"), MB_OK );
        …
}
```

GetLastSpectrumNumber

long GetLastSpectrumNumber(long FAR* pnLastSpectrum)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting last scan number"), _T("Error"), MB_OK );
        …
}
```

GetInstrumentID

long GetInstrumentID(long FAR* pnInstrumentID)

Return Value

0 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 file formats.

Example

```
// example for GetInstrumentID
long nInstID;
long nRet = XRawfileCtrl.GetInstrumentID (&nInstID);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting inst ID number"), _T("Error"), MB_OK );
        …
}
```

GetInletID

long GetInletID(long FAR* pnInletID)

Return Value

0 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 file formats.

Example

```
// example for GetInletID
long nInletID;
long nRet = XRawfileCtrl.GetInletID (&nInletID);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting inlet ID number"), _T("Error"), MB_OK );
        …
}
```

GetErrorFlag

long GetErrorFlag(long FAR* pnErrorFlag)

Return Value

0 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 file formats.

Example

```
// example for GetErrorFlag
long nErrorFlag;
long nRet = XRawfileCtrl.GetErrorFlag (&nErrorFlag);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting error flag value"), _T("Error"), MB_OK );
        …
}
```

GetSampleVolume

long GetSampleVolume(double FAR* pdSampleVolume)

Return Value

0 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 file formats.

Example

```
// example for GetSampleVolume
double dSampleVolume;
long nRet = XRawfileCtrl.GetSampleVolume (&dSampleVolume);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample volume value"), _T("Error"),
MB_OK );
        …
}
```

GetSampleWeight

long GetSampleWeight(double FAR* pdSampleWeight)

Return Value

0 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 file formats.

Example

```
// example for GetSampleWeight
double dSampleWeight;
long nRet = XRawfileCtrl.GetSampleWeight (&dSampleWeight);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample weight value"), _T("Error"),
MB_OK );
        …
}
```

GetVialNumber

long GetVialNumber(long FAR* pnVialNumber)

Return Value

0 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 file formats.

Example

```
// example for GetVialNumber
long nVialNum;
long nRet = XRawfileCtrl.GetVialNumber (&nVialNum);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting vial number"), _T("Error"), MB_OK );
        …
}
```

GetInjectionVolume

long GetInjectionVolume(double FAR* pdInjectionVolume)

Return Value

0 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 file formats.

Example

```
// example for GetInjectionVolume
double dInjVol;
long nRet = XRawfileCtrl.GetInjectionVolume (&dInjVol);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting injection volume"), _T("Error"), MB_OK );
        …
}
```

GetFlags

long GetFlags(BSTR FAR* pbstrFlags)

Return Value

0 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 file formats.

Example

```
// example for GetFlags
BSTR bstrFlags = NULL;
long nRet = XRawfileCtrl.GetFlags ( &bstrFlags );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting flags"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrFlags);
```

GetAcquisitionFileName

long GetAcquisitionFileName(BSTR FAR* pbstrFileName)

Return Value

0 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 file formats.

Example

```
// example for GetAcquisitionFileName
BSTR bstrAcquFile = NULL;
long nRet = XRawfileCtrl.GetAcquFile ( &bstrAcquFile );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting acquisition file name"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrAcquFile);
```

GetInstrumentDescription

long GetInstrumentDescription(BSTR FAR* pbstrInstrumentDescription)

Return Value

0 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 file formats.

Example

```
// example for GetInstrumentDescription
BSTR bstrInstDesc = NULL;
long nRet = XRawfileCtrl.GetInstrumentDescription ( &bstrInstDesc );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument description"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrInstDesc);
```

GetAcquisitionDate

long GetAcquisitionDate(BSTR FAR* pbstrAcquisitionDate)

Return Value

0 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 file formats.

Example

```
// example for GetAcquisitionDate
BSTR bstrAcquDate = NULL;
long nRet = XRawfileCtrl.GetAcquisitionDate ( &bstrAcquDate );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting acquisition date"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrAcquDate);
```

GetOperator

long GetOperator(BSTR FAR* pbstrOperator)

Return Value

0 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 file formats.

Example

```
// example for GetOperator
BSTR bstrOperator = NULL;
long nRet = XRawfileCtrl.GetOperator ( &bstrOperator );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting operator name"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrOperator);
```

GetComment1

long GetComment1(BSTR FAR* pbstrComment1)

Return Value

0 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 file formats.

Example

```
// example for GetComment1
BSTR bstrComment1 = NULL;
long nRet = XRawfileCtrl.GetComment1 ( &bstrComment1 );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting comment 1"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrComment1);
```

GetComment2

long GetComment2(BSTR FAR* pbstrComment2)

Return Value

0 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 file formats.

Example

```
// example for GetComment2
BSTR bstrComment2 = NULL;
long nRet = XRawfileCtrl.GetComment2 ( &bstrComment2 );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting comment 2"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrComment2);
```

GetSampleAmountUnits

long GetSampleAmountUnits(BSTR FAR* pbstrSampleAmountUnits)

Return Value

0 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 file formats.

Example

```
// example for GetSampleAmountUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetSampleAmountUnits ( &bstrUnits );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample amount units"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrUnits);
```

GetInjectionAmountUnits

long GetInjectionAmountUnits(BSTR FAR* pbstrInjectionAmountUnits)

Return Value

0 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 file formats.

Example

```
// example for GetInjectionAmountUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetInjectionAmountUnits ( &bstrUnits );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting injection amount units"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrUnits);
```

GetSampleVolumeUnits

long GetSampleVolumeUnits(BSTR FAR* pbstrSampleVolumeUnits)

Return Value

0 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 file formats.

Example

```
// example for GetSampleVolumeUnits
BSTR bstrUnits = NULL;
long nRet = XRawfileCtrl.GetSampleVolumeUnits ( &bstrUnits );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting sample volume units"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrUnits);
```

GetInstName

long GetInstName(BSTR FAR* pbstrInstName)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument name"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrInstName);
```

GetInstModel

long GetInstModel(BSTR FAR* pbstrInstModel)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument model"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrInstModel);
```

GetInstSerialNumber

long GetInstSerialNumber(BSTR FAR* pbstrInstSerialNumber)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument serial number"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrInstSerialNum);
```

GetInstSoftwareVersion

long GetInstSoftwareVersion(BSTR FAR* pbstrInstSoftwareVersion)

Return Value

0 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 revision information for the current controller software, if available.

Example

```
// example for GetInstSoftwareVersion
BSTR bstrInstSoftRev = NULL;
long nRet = XRawfileCtrl.GetInstSoftwareVersion ( &bstrInstSoftRev );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument software version"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString(bstrInstSoftRev);
```

GetInstHardwareVersion

long GetInstHardwareVersion(BSTR FAR* pbstrInstHardwareVersion)

Return Value

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

Example

```
// example for GetInstHardwareVersion
BSTR bstrInstHardRev = NULL;
long nRet = XRawfileCtrl.GetInstHardwareVersion ( &bstrInstHardRev );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting instrument hardware version"),
_T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrInstHardRev);
```

GetInstFlags

long GetInstFlags(BSTR FAR* pbstrInstFlags)

Return Value

0 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.

These are the currently defined experiment fields:

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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting experiment flags"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrInstFlags);
```

GetInstNumChannelLabels

long GetInstNumChannelLabels(long FAR* pnInstNumChannelLabels)

Return Value

0 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting number of channel labels"), _T("Error"),
MB_OK
);
        …
}
```

GetInstChannelLabel

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nChannelLabelNumber	The index value of the channel number field to return.
pbstrFlags	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 label 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting channel label 3"), _T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrLabel);
```

GetFilters

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

Return Value

0 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* points to an array of BSTR fields, each containing a unique scan filter, and *pnArraySize* contains 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 != 0 )
{
        ::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++ )
{
            _stprintf( szTitle, _T("Scan Filter %d"), i );
            ::MessageBox( NULL, pbstrFilters[i], szTitle, MB_OK );
}

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

ScanNumFromRT

long ScanNumFromRT(double dRT, long FAR* pnScanNumber)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

dRT	The run time or retention time, in minutes, that is returned for the closest scan number.
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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting scan number at RT of 3.45 min."),
_T("Error"), MB_OK );
        …
}
```

RTFromScanNum

long RTFromScanNum(long nScanNumber, double FAR* pdRT)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the closest run time or retention time.

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 the reading number. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for RTFromScanNum
long nScanNum = 12;    // get the RT of the twelfth scan
double dRT;
long nRet = XRawfileCtrl.RTFromScanNum ( nScanNumber, &dRT );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting RT for scan number 12"), _T("Error"),
MB_OK );
        …
}
```

GetFilterForScanNum

long GetFilterForScanNum(long nScanNumber, BSTR FAR* pbstrFilter)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding scan filter.
pbstrFilter	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 supported for MS device controllers. The value of *nScanNumber* must be within the range of scans for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting scan filter for scan number 12"),
_T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrFilter);
```

GetFilterForScanRT

long GetFilterForScanRT(double dRT, BSTR FAR* pbstrFtiler)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

dRT	The run time that is returned for the corresponding scan filter.
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 supported 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting scan filter for RT of 3.45 min."),
_T("Error"), MB_OK );
        …
}
…
SysFreeString(bstrFilter);
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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 scan corresponding to *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan to *pnScanNumber* that matches the scan filter is 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a double precision variable containing the retention time, in minutes, that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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 closest scan to *pdRT* is returned. If a scan filter is provided, the closest matching scan to *pdRT* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pdRT* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number after which the corresponding mass list data is returned.
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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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 scan after *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan after *pnScanNumber* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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++ )
        {
                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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number before which the corresponding mass list data is returned.
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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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 scan before *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan before *pnScanNumber* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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++ )
        {
                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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
csMassRange1	A string containing the mass range.
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 scan corresponding to *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan to *pnScanNumber* that matches the scan filter is 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a double precision variable containing the retention time, in minutes, that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
szMassRange1	A string containing the mass range.
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 closest scan to *pdRT* is returned. If a scan filter is provided, the closest matching scan to *pdRT* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pdRT* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
}
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number after which the corresponding mass list data is returned.
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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
szMassRange1	A string containing the mass range.
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 scan after *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan after *pnScanNumber* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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++ )
        {
                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 );
}
```

GetPrecursorInfoFromScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding precursor information.
pvarPrecursorInfos	A valid pointer to a VARIANT variable to receive the precursor information.
pnArraySize	A valid pointer to a long variable to receive the number of precursor information packets returned in the precursor information array.

Remarks

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

You 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 get access to the scan data of the parent scan in the form of a XSpectrumRead object.

Further refine the charge state and the monoisotopic mass 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);

        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."));
     }
  }
```

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

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number before which the corresponding mass list data is returned.
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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
szMassRange1	A string containing the mass range.
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 scan before *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan before *pnScanNumber* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pnScanNumber* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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);
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 != 0 )
{
        ::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++ )
        {
                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 );
}
```

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

0 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 that is returned for the corresponding averaged mass list data.
pnLastAvgScanNumber	A valid pointer to a long variable containing the last scan number of the scan number range that is returned for the corresponding averaged mass list data.
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, are returned. Likewise, all the scans between *pnFirstBkg1ScanNumber* and *pnLastBkg1ScanNumber* and *pnFirstBkg2ScanNumber* and *pnLastBkg2ScanNumber*, inclusive, are 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 are utilized in obtaining the background subtracted mass list. The specified scan numbers must be valid for the current controller. If no background subtraction is performed, the background scan numbers should be set to zero. On return, the scan number variables 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 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 Xcalibur 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 are returned in the mass list, set *nMaxNumberOfPeaks* to a value greater than zero. To have all data peaks returned, set *nMaxNumberOfPeaks* to zero.

The mass list contents are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

Example

```
// example for GetAverageMassList

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

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,
                                               &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 != 0 )
{
        ::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 );
}
```

GetAveragedMassSpectrum

long GetAveragedMassSpectrum(long FAR* pnScanNumbers,
long nScansToAverage,
BOOL bCentroidResult,
VARIANT FAR* pvarMassList,
VARIANT FAR* pvarPeakFlags,
long FAR* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumbers	A valid pointer to an array of scan numbers that this routine will average.
nScansToAverage	The number of scans that are averaged.
bCentroidResult	A flag indicating if the mass spectral data is centroided before it is returned by this function.
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.

GetAveragedMassSpectrum returns the average spectrum for the list of scans that are supplied to the function in *pnScanNumbers*. If no scans are provided in *pnScanNumbers*, or if *nScansToAverage* is zero, then the function returns an error code.

If the *bCentroidData* value is true, profile data is centroided before it is returned by this routine.

The mass list contents are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

Example

```
// example for GetAveragedMassSpectrum

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

long nScans[3];
long nScans[0] = 12;
long nScans[1] = 18;
long nScans[2] = 25;
long nScansToAverage =3;
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet;

nRet = XRawfileCtrl.GetAveragedMassSpectrum ( nScans,
                                                nScansToAverage, // the number of
scans
                                                bCentroidData, // centroid the data
                                                &varMassList,  // mass list data
                                                &varPeakFlags, // peak flags data
                                                &nArraySize ); // size of mass list array

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting average mass spectrum 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 );
}
```

GetSummedMassSpectrum

long GetSummedMassSpectrum(long FAR* pnScanNumbers,
long nScansToSum,
BOOL bCentroidResult,
VARIANT FAR* pvarMassList,
VARIANT FAR* pvarPeakFlags,
long FAR* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumbers	A valid pointer to an array of scan numbers that this routine will sum.
nScansToSum	The number of scans that is summed.
bCentroidResult	A flag indicating if the mass spectral data is centroided before it is returned by this function.
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.

GetSummedMassSpectrum returns the summed spectrum for the list of scans that are supplied to the function in *pnScanNumbers*. If no scans are provided in *pnScanNumbers*, or if *nScansToSum* is zero, then the function returns an error code.

If the *bCentroidResult* value is true, then profile data is centroided before it is returned by this routine.

The mass list contents are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

Example

```
// example for GetSummedMassSpectrum

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

long nScans[3];
long nScans[0] = 12;
long nScans[1] = 18;
long nScans[2] = 25;
long nScansToSum =3;
VARIANT varMassList;
VariantInit(&varMassList);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
long nRet;

nRet = XRawfileCtrl.GetSummedMassSpectrum ( nScans,
                                            nScansToSum, // the number of scans
                                            bCentroidResult, // centroid the data
                                            &varMassList,  // mass list data
                                            &varPeakFlags, // peak flags data
                                            &nArraySize ); // size of mass list array

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting summed mass spectrum 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 );
}
```

GetLabelData

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

Return Value

0 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 containing the scan number that is returned for the corresponding label data.

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. The flags are saturated, fragmented, merged, exception, reference, and modified.

Example

```
// example for GetLabelData

long            nRet;
long            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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting label data."), _T("Error"), MB_OK );
        …
}

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.
            ...
}
```

GetAveragedLabelData

long GetAveragedLabelData(long *pnScanNumbers,
long nScansToAverage,
VARIANT *pvarMassList,
VARIANT *pvarPeakFlags,
long *pnArraySize);

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumbers	Input parameter: the scans that you want to average.
nScansToAverage	The number of scans in *pnScanNumbers* that are averaged.
pvarMassList	A valid pointer to a VARIANT variable to receive the mass list data. This should be a two-dimensional array *n* * 6 of doubles, where the second dimension is an array of doubles consisting of `mass`, `intensity`, `resolution`, `basel ne`, `noise`, and `charge` from the raw scan label peaks. *N* is sized to *pnArraySize*.
pvarPeakFlags	A valid pointer to a VARIANT variable to receive the peak flags. This should be a two-dimensional array *n* * 6 of unsigned character, where the second dimension is an array of bytes anded out of bits from each raw scan label peak flag member consisting of `LABEL_SATURATED_MASK`, `LABEL_FRAGMENTED_MASK`, `LABEL_MERGED_MASK`, `LABEL_EXCEPTION_MASK`, `LABEL_REFERENCE_MASK`, and `LABEL_MODIFIED_MASK`.
pnArraySize	A valid pointer to a long variable to receive the number of data peaks returned in the mass list array.

Remarks

This method enables you to read the averaged FT-PROFILE labels for the list of scans represented by the *pnScanNumbers*. If no scans are provided in *pnScanNumbers,* or if *nScansToAverage* is zero, the function returns an error code.

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

The mass list contents are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order, for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3.

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

Example

```
// example for GetAveragedLabelData

long nScans[3];
long nScans[0] = 12;
long nScans[1] = 18;
long nScans[2] = 25;
long nScansToAverage =3;
long            nRet;
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.GetAverageLabelData(nScans, nScansToAverage, pvarLabels,
pvarFlags,
                                                &nScanNumber);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting label data."), _T("Error"), MB_OK );
        …
}

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.
        ...
}
```

GetNoiseData

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

Return Value

0 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 containing the scan number that is returned for the corresponding noise packets.

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;
long        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 != 0 )
{
        ::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.
        …
}
```

IsProfileScanForScanNum

long IsProfileScanForScanNum(long nScanNumber, long pbIsProfileScan)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the profile data type information.

pbIsProfileScan 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for IsProfileScanForScanNum
long nScanNum = 12;    // Is the twelfth scan a profile scan
long bIsProfileScan;
long nRet = XRawfileCtrl. IsProfileScanForScanNum (nScanNum, & bIsProfileScan);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting profile flag for scan number 12"),
_T("Error"), MB_OK );
        …
}
```

IsCentroidScanForScanNum

long IsCentroidScanForScanNum(long nScanNumber, long pbIsCentroidScan)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the profile data type information.
pbIsCentroidScan	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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for IsCentroidScanForScanNum
long nScanNum = 12;    // Is the twelfth scan a centroid scan
long bIsCentroidScan;
long nRet = XRawfileCtrl. IsCentroidScanForScanNum (nScanNum, & bIsCentroidScan);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T(“Error getting centroid flag for scan number 12”),
_T(“Error”), MB_OK );
        …
}
```

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, long pbUniformTime, double FAR* pdFrequency)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the scan header information.
pnNumPackets	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.
pdStartTime	A valid pointer to a variable of type double to receive the retention time of the specified scan. This variable must exist.
pdLowMass	A valid pointer to a variable of type double to receive the low mass value of the specified scan. This variable must exist.
pdHighMass	A valid pointer to a variable of type double to receive the high mass value of the specified scan. This variable must exist.
pdTIC	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.
pdBasePeakMass	A valid pointer to a variable of type double to receive the base peak mass of the specified scan. This variable must exist.
pdBasePeakIntensity	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.
pnNumChannels	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.

pbUniformTime	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.
pdFrequency	A valid pointer to a variable of type double to receive the sampling frequency for the current controller if *pbUniformTime* 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

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 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;
long bUniformTime = FALSE;
double dFrequency = 0.0;
long nRet = XRawfileCtrl. GetScanHeaderInfoForScanNum (     nScanNum,
                                                            &nPackets,
                                                            &dStartTime,
                                                            &dLowMass,
                                                            &dHighMass,
                                                            &dTIC,
                                                            &dBasePeakMass,
                                                            &dBasePeakIntensity,
                                                            &nChannels,
                                                            &bUniformTime,
                                                            &dFrequency );
```

```
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting scan header info"), _T("Error"), MB_OK );
        …
}
```

GetStatusLogForScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for status log information.
pdStatusLogRT	A valid pointer to a variable of type double to receive the retention time when the status log entry was recorded. This variable must exist.
pvarLabels	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.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* and *pvarValues* 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

On return, *pdStatusLogRT* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but is 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 are of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains the number of entries in the *pvarLabels* and *pvarValues* arrays.

Example

```
// example for GetStatusLogForScanNum
long nScan = 12;                    // use twelth scan
double dStatusLogRT = 0.0;
```

```
VARIANT varLabels;
VariantInit(&varLabels);
VARIANT varValues;
VariantInit(&varValues);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetStatusLogForScanNum (    nScan,
                                                      &dStatusLogRT,
                                                      &varLabels,
                                                      &varValues,
                                                      &nArraySize);
if( nRet != 0 )
{
        ::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 );
```

GetStatusLogForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the retention time that is returned for the closest status log entry.
pvarLabels	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.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* and *pvarValues* 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* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time specified but is 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 are of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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 );
```

GetStatusLogLabelsForScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for status log information.
pdStatusLogRT	A valid pointer to a variable of type double to receive the retention time when the status log entry was recorded. This variable must exist.
pvarLabels	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

On return, *pdStatusLogRT* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but is the closest status log entry to the scan time.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable is of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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;

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 );
```

GetStatusLogLabelsForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the retention time that is returned for the closest status log entry.
pvarLabels	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* 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* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time specified but is the closest status log entry to the specified time.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable is of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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;

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 );
```

GetStatusLogValueForScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for status log information.
szLabel	A string containing the label that is returned for the status log parameter value.
pdStatusLogRT	A valid pointer to a variable of type double to receive the retention time when the status log entry was recorded. This variable must exist.
pvarValue	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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

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

On return, *pdStatusLogRT* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time corresponding to the specified scan number but is the closest status log entry to the scan time.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable is of the paramter type 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 != 0 )
{
        ::MessageBox( NULL, _T(“Error getting status log information”), _T(“Error”),
MB_OK );
        …
}

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

GetStatusLogValueForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the retention time that is returned for the closest status log entry.
szLabel	A string containing the label that is returned for the status log parameter value.
pvarValue	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* contains the retention time when the status log entry was recorded. This time may not be the same as the retention time specified but is the closest status log entry to the specified time.

The variable *pvarValue* must be initialized to VARIANT type VT_EMPTY. On return, this variable is of the paramter type 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting status log information"), _T("Error"),
MB_OK );
```

```
            …
}

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

GetTrailerExtraForScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for trailer extra information.
pvarLabels	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.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* and *pvarValues* 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

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

Example

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

```
                                                        &nArraySize);
if( nRet != 0 )
{
        ::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;

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 );
```

GetTrailerExtraForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the retention time that is returned for the trailer extra entry.
pvarLabels	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.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* and *pvarValues* arrays. This variable must exist.

Remarks

Returns the recorded trailer extra entry labels and values for the current controller. This funciton 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* contains the retention time when the trailer extra entry was recorded. This time may not be the same as the retention time specified but is 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 are of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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;

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 );
```

GetTrailerExtraLabelsForScanNum

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for trailer extra information.
pvarLabels	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* 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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The variable *pvarLabels* must be initialized to VARIANT type VT_EMPTY. On return, this variable is of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
```

```
{
        ::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 );
        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 );
```

GetTrailerExtraLabelsForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the scan retention time that is returned for the trailer extra labels.
pvarLabels	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* 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* contains the retention time when the trailer extra entry was recorded. This time may not be the same as the retention time specified but is 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 is of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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 );
        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 );
```

GetTrailerExtraValueForScanNum

long GetTrailerExtraValueForScanNum(long nScanNumber, LPCTSTR szLabel, VARIANT FAR* pvarValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for trailer extra information.
szLabel	A string containing the label that is returned for the trailer extra parameter value.
pvarValue	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 scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

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

The *pvarValue* variable must be initialized to VARIANT type VT_EMPTY. On return, this variable is 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"),
MB_OK );
        …
}
```

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

GetTrailerExtraValueForRT

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

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a variable of type double containing the retention time that is returned for the closest trailer extra entry.
szLabel	A string containing the label that is returned for the trailer extra parameter value.
pvarValue	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* contains the retention time when the trailer extra entry was recorded. This time may not be the same as the retention time specified but is 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 is 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 != 0 )
```

```
{
        ::MessageBox( NULL, _T("Error getting trailer extra information"), _T("Error"),
MB_OK );
        …
}

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

GetErrorLogItem

long GetErrorLogItem(long nItemNumber, double FAR* pdRT, BSTR FAR* pbstrErrorMessage)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nItemNumber	The error log item number that is returned for information.
pdRT	A valid pointer to a variable of type double to receive the retention time when the error occurred. This variable must exist.
pbstrErrorMessage	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 when 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting error log information"), _T("Error"),
MB_OK );
        …
}
…
SysFreeString( bstrMessage );
```

GetTuneData

long GetTuneData(long nSegmentNumber, VARIANT FAR* pvarLabels, VARIANT FAR* pvarValues, long FAR* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nSegmentNumber	The acquisition segment that is returned for tune information.
pdRT	A valid pointer to a variable of type double to receive the retention time when the error occurred. This variable must exist.
pvarLabels	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.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* and *pvarValues* 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 are of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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;

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 );
```

GetTuneDataValue

long GetTuneDataValue(long nSegmentNumber, LPCTSTR szLabel, VARIANT FAR* pvarValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nSegmentNumber	The acquisition segment that is returned for tune information.
szLabel	A string containing the label that is returned for the tune parameter value.
pvarValue	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 is 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 != 0 )
{
        ::MessageBox( NULL, _T("Error getting tune record information"), _T("Error"),
MB_OK );
        …
}
```

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

GetTuneDataLabels

long GetTuneDataLabels(long nSegmentNumber, VARIANT FAR* pvarLabels, long FAR* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nSegmentNumber	The acquisition segment that is returned for tune information.
pvarLabels	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarLabels* 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 is of type VT_ARRAY | VT_BSTR. On return, *pnArraySize* contains 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 != 0 )
{
        ::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 );
}

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

        // do something with label
        …
}

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

GetNumInstMethods

long GetNumInstMethods(long FAR* pnNumInstMethods)

Return Value

0 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 with a method that was created in Instrument Setup (for example, autosampler, LC, MS, PDA) has its instrument method contained in the raw file.

Example

```
// example for GetNumInstMethods
long nMethods;
long nRet = XRawfileCtrl.GetNumInstMethods ( &nMethods );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error retrieving number of inst methods"), _T("Error"),
MB_OK );
        …
}
```

GetInstMethod

long GetInstMethod(long nInstMethodItem, BSTR FAR* pbstrInstMethod)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nInstMethodItem	A long variable containing the index value of the instrument method to be returned.
pbstrFilter	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 are 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 != 0 )
{
          ::MessageBox( NULL, _T("Error getting fifth instrument method."), _T("Error"),
MB_OK );
          …
}
…
SysFreeString(bstrMethod);
```

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

0 if successful; otherwise, see Error Codes.

Parameters

nChroType1	A long variable containing the first chromatogram trace type of interest.
nChroOperator	A long variable containing the chromatogram trace operator.
nChroType2	A long variable containing the second chromatogram trace type of interest.
szFilter	A string containing the formatted scan filter.
szMassRanges1	A string containing the formatted mass ranges for the first chromatogram trace type.
szMassRanges2	A string containing the formatted mass ranges for the second chromatogram trace type.
dDelay	A double precision variable containing the chromatogram delay in minutes.
pdStartTime	A pointer to a double precision variable containing the start time of the chromatogram time range to return.
pdEndTime	A pointer to a double precision variable containing the end time of the chromatogram time range to return.
nSmoothingType	A long variable containing the type of chromatogram smoothing to be performed.
nSmoothingValue	A long variable containing the chromatogram smoothing value.
pvarChroData	A valid pointer to a VARIANT variable to receive the chromatogram data.
pvarPeakFlags	A valid pointer to a VARIANT variable to receive the peak flag data.

pnArraySize 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*.

The chromatogram trace types and operator values of *nChroType1*, *nChroOperator*, and *nChroType2* depend 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 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 Xcalibur 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 Xcalibur 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. Or, if the entire chromatogram is returned, *pdStartTime* and *pdEndTime* may be set to zero. On return, *pdStartTime* and *pdEndTime* 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 are 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* is set to type VT_ARRAY | VT_R8. The format of the chromatogram list returned is an array of double precision values in time intensity pairs in ascending time order (for example, time 1, intensity 1, time 2, intensity 2, time 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains 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

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 != 0 )
{
        ::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 );
}

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

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

GetMassListRangeFromScanNum

**HRESULT GetMassListRangeFromScanNum(long* pnScanNumber, BSTR bstrFilter,
long nIntensityCutoffType,
long nIntensityCutoffValue,
long nMaxNumberOfPeaks,
BOOL bCentroidResult,
double* pdCentroidPeakWidth,
VARIANT* pvarMassList,
VARIANT* pvarPeakFlags,
LPCTSTR csMassRange1,
long* pnArraySize)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
pdCentroidPeakWidth	The peak width to use when centroiding the peaks.
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.
csMassRange1	A string containing the mass range.
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 scan corresponding to *pnScanNumber* is returned. If a scan filter is provided, the closest matching scan to *pnScanNumber* that matches the scan filter is 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
```

GetMassListRangeFromRT

HRESULT GetMassListRangeFromRT(double* pdRT, BSTR bstrFilter, long nIntensityCutoffType, long nIntensityCutoffValue, long nMaxNumberOfPeaks, BOOL bCentroidResult, double* pdCentroidPeakWidth, VARIANT* pvarMassList, VARIANT* pvarPeakFlags, LPCTSTR szMassRange1, long* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a double precision variable containing the retention time, in minutes, that is returned for the corresponding 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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
pdCentroidPeakWidth	The peak width to use when centroiding the peaks.
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.
szMassRange1	A string containing the mass range.
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 closest scan to *pdRT* is returned. If a scan filter is provided, the closest matching scan to *pdRT* that matches the scan filter is returned. The requested scan must be valid for the current controller. On return, *pdRT* contains 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 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 Xcalibur 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 are returned in the mass list, set *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 are 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* is set to type VT_ARRAY | VT_R8. The format of the mass list returned is an array of double precision values in mass intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so on).

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

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

On successful return, *pnArraySize* contains 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 != 0 )
{
        ::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;

                // 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 );
}
```

GetPrecursorInfoFromScanNum

**HRESULT GetPrecursorInfoFromScanNum(long nScanNumber,
VARIANT* pvarPrecursorInfos,
LONG* pnArraySize)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding precursor information.
pvarPrecursorInfos	A valid pointer to a VARIANT variable to receive the precursor information.
pnArraySize	A valid pointer to a long variable to receive the number of precursor information packets returned in the precursor information array.

Remarks

This function is used to retrieve information about the parent scans of a data-dependent MS^n scan.

You 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 also get access to the scan data of the parent scan in the form of an XSpectrumRead object.

Further refine the charge state and the monoisotopic mass 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);

        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."));
     }
}
```

RefreshViewOfFile

long RefreshViewOfFile()

Return Value

0 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 reopening 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 != 0 )
{
        ::MessageBox( NULL, _T("Error file refreshing view of file"), _T("Error"), MB_OK );
        …
}
```

ExtractInstMethodFromRaw

HRESULT ExtractInstMethodFromRaw(BSTR szInstMethodFileName)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

szInstMethodFileName	The path and file name of the instrument method. An example is C:\Xcalibur\Methods\MyMethod.meth.

Remarks

This function enables you to save the embedded instrument method in the raw file in a separated method (.meth)) file. It overwrites any pre-existing method file in the same path with the same name.

Example

```
// example for ExtratInstMethodFrom Raw

BSTR bstrInstMethodFileName = _T("c:\\xcalibur\\example\\InstMethod.meth");

long nRet = XRawfileCtrl.ExtractInstMethodFromRaw(bstrInstMethodFileName );
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error extract instrument method"), _T("Error"), MB_OK
);
        …
}
…
SysFreeString( bstrFileName );
```

GetActivationTypeForScanNum

long GetActivationTypeForScanNum(long nScanNumber,
long nMSOrder,
long FAR *pnActivationType)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the activation type information.
nMSOrder	The MS^n order for the scan.
pnActivationType	A valid pointer to a variable of type long. This variable must exist.

Remarks

This function returns the activation type for the scan specified by *nScanNumber* and the transition specified by *nMSorder* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The value returned in the *pnActivationType* variable is one of the following:

CID	0
MPD	1
ECD	2
PQD	3
ETD	4
HCD	5
Any activation type	6
SA	7
PTR	8
NETD	9
NPTR	10

Example

```
// example for GetActivationTypeForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nMSOrder = 2;     // The MS2 transition
long nType;
```

```
long nRet = XRawfileCtrl. GetActivationTypeForScanNum(nScanNum, nMSOrder,
&nType);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T(“Error getting the activation type for scan number 12”),
_T(“Error”), MB_OK );
        …
}
```

GetMassAnalyzerTypeForScanNum

**long GetMassAnalyzerForScanNum(long nScanNumber,
long FAR *pnMassAnalyzerType)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the mass analyzer type information.
pnMassAnalyzerType	A valid pointer to a variable of type long. This variable must exist.

Remarks

This function returns the mass analyzer type for the scan specified by *nScanNumber* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The value returned in the *pnMassAnalyzerType* variable is one of the following:

ITMS	0
TQMS	1
SQMS	2
TOFMS	3
FTMS	4
Sector	5

Example

```
// example for GetMassAnalyzerTypeForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nType;

long nRet = XRawfileCtrl. GetMassAnalyzerTypeForScanNum(nScanNum, & nType);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the mass analyzer type for scan number
                        12"),
                        _T("Error"), MB_OK );
        …
}
```

GetDetectorTypeForScanNum

**long GetDetectorTypeForScanNum(long nScanNumber,
long FAR *pnDetectorType)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the detector type information.

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

Remarks

This function returns the detector type for the scan specified by *nScanNumber* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The value returned in the *pnDetectorType* variable is one of the following:

CID	0
PQD	1
ETD	2
HCD	3

Example

```
// example for GetDetectorTypeForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nType;

long nRet = XRawfileCtrl. GetDetectorTypeForScanNum(nScanNum, & nType);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the detector type for scan number 12"),
                        _T("Error"), MB_OK );
        …
}
```

GetScanTypeForScanNum

**long GetScanTypeForScanNum(long nScanNumber,
long FAR *pnScanType)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the scan type information.

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

Remarks

This function returns the scan type for the scan specified by *nScanNumber* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The value returned in the *pnScanType* variable is one of the following:

ScanTypeFull	0
ScanTypeSIM	1
ScanTypeZoom	2
ScanTypeSRM	3

Example

```
// example for GetScanTypeForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nType;

long nRet = XRawfileCtrl. GetScanTypeForScanNum(nScanNum, & nType);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the scan type for scan number 12"),
                        _T("Error"), MB_OK );
        …
}
```

GetMSOrderForScanNum

**long GetMSOrderForScanNum(long nScanNumber,
long FAR *pnMassOrder)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the scan type information.

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

Remarks

This function returns the MS order for the scan specified by *nScanNumber* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

The value returned in the *pnScanType* variable is one of the following:

Neutral gain	-3
Neutral loss	-2
Parent scan	-1
Any scan order	0
MS	1
MS2	2
MS3	3
MS4	4
MS5	5
MS6	6
MS7	7
MS8	8
MS9	9
MS10	10

Example

```
// example for GetMSOrderForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nOrder;

long nRet = XRawfileCtrl. GetMSOrderForScanNum(nScanNum, & nOrder);
```

```
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the MS order for scan number 12"),
                        _T("Error"), MB_OK );
        …
}
```

GetPrecursorMassForScanNum

long GetPrecursorMassForScanNum(long nScanNumber,
long nMSOrder,
double FAR *pdPrecursorMass)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the scan type information.
nMSOrder	The MS^n order for the scan.
pdPrecursorMass	A valid pointer to a variable of type double. This variable must exist.

Remarks

This function returns the precursor mass for the scan specified by *nScanNumber* and the transition specified by *nMSorder* from the scan event structure in the RAW file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. The range of scans or readings for the current controller may be obtained by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for GetPrecursorMassForScanNum

long nScanNum = 12;    // Is the twelfth scan from the file
long nMSOrder = 2;
double dMass;

long nRet = XRawfileCtrl. GetPrecursorMassForScanNum(nScanNum, nMSOrder, &
dMass);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the precursor mass for scan number 12"),
                        _T("Error"), MB_OK );
        …
}
```

Version

**long Version(long *pnMajorVersion, long *pnMinorVersion, long *pnSubMinorVersion,
long *pnBuildNumber)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnMajorVersion	The major version number for the DLL. This variable must exist.
pnMinorVersion	The minor version number for the DLL. This variable must exist.
pnsubMinorVersion	The sub-minor version number for the dll. This variable must exist.
pnsubBuildNumber	The build number for the dll. This variable must exist.

Remarks

This function returns the version number for the DLL.

Example

```
// example for Version

long nMajorVersion, nMinorVersion, nSubMinorVersion, nBuildNumber;

long nRet = XRawfileCtrl.Version(&nMajorVersion, &nMinorVersion, &nSubMinorVersion,
                                 &nBuildNumber);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error reading the version number"),
                        _T("Error"), MB_OK );
        …
}
```

IsThereMSData

long IsThereMSData(BOOL FAR* pbMSData)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

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

Remarks

This function checks to see if there is MS data in the raw file. A return value of TRUE means that the raw file contains MS data. You must open the raw file before performing this check.

Example

```
// example for IsThereMSData
BOOL bIsThereMSData;

long nRet = XRawfileCtrl.IsThereMSData ( &bIsThereMSData );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while checking for MS data"), _T("Error"),
MB_OK );
        …
}
```

HasExpMethod

long HasExpMethod(BOOL FAR* pbHasMethod)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

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

Remarks

This function checks to see if the raw file contains an experimental method. A return value of TRUE indicates that the raw file contains the method. You must open the raw file before performing this check.

Example

```
// example for HasExpMethod
BOOL bHasMethod

long nRet = XRawfileCtrl.HasExpMethod ( &bHasMethod );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while check for experimental method"),
_T("Error"),
                        MB_OK );
        …
}
```

GetFilterMassPrecision

long GetFilterMassPrecision(long* pnFilterMassPrecision)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

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

Remarks

This function gets the mass precision for the filter associated with an MS scan.

Example

```
// example for GetFilterMassPrecision
long nMassPrecison = 0;

long nRet = XRawfileCtrl. GetFilterMassPrecision ( & nMassPrecison);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while getting the filter mass precision"),
_T("Error"),
                        MB_OK );
        …
}
```

GetStatusLogForPos

long GetStatusLogForPos(long nPos, VARIANT *pvarRT, VARIANT *pvarValues, long *pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nPos	The position that the status log information is to be returned for.
pvarRT	A valid pointer to a variable of type VARIANT to receive the retention time when the status log entry was recorded. This variable must exist and be initialized to VT_EMPTY.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type LONG to receive the number of records returned in the *pvarRT* and *pvarValues* arrays. This variable must exist.

Remarks

This function returns the recorded status log entry labels and values for the current controller.

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

Example

```
// example for GetStatusLogForPos

VARIANT varRT;
VariantInit(&varRT);

VARIANT varValues;
VariantInit(&varValues);

long nPosition = 0;
long nArraySize = 0;

long nRet = XRawfileCtrl. GetStatusLogForPos ( nPosition, &varRT, & varValues,
&nArraySize);

if( nRet != 0 )
```

```
{
        ::MessageBox( NULL, _T("Error while getting the status log information"),
                                _T("Error"),  MB_OK );
        …
}
```

GetStatusLogPlottableIndex

long GetStatusLogPlottableIndex(VARIANT *pvarIndex, VARIANT *pvarValues, long *pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pvarIndex	A valid pointer to a variable of type VARIANT to receive the retention time when the status log entry was recorded. This variable must exist and be initialized to VT_EMPTY.
pvarValues	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.
pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarIndex* and *pvarValues* arrays. This variable must exist.

Remarks

This function returns the recorded status log entry labels and values for the current controller.

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

Example

```
// example for GetStatusLogPlottableIndex

VARIANT varIndex;
VariantInit(&varIndex);

VARIANT varValues;
VariantInit(&varValues);

long nArraySize = 0;

long nRet = XRawfileCtrl. GetStatusLogPlottableIndex (&varIndex, & varValues,
&nArraySize);
```

```
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while getting the status log information"),
                        _T("Error"),  MB_OK );
        …
}
```

GetInstMethodNames

long GetInstMethodNames(long *pnSize, VARIANT *pvarNames)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnArraySize	A valid pointer to a variable of type long to receive the number of records returned in the *pvarNames* array. This variable must exist.
pvarNames	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.

Remarks

This function returns the recorded names of the instrument methods for the current controller.

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

Example

```
// example for GetInstMethodNames

VARIANT varNames;
VariantInit(&varNames);
long nArraySize = 0;

long nRet = XRawfileCtrl. GetInstMethodNames ( &nArraySize, &varNames);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while getting the instrument method names"),
                        _T("Error"),  MB_OK );
        …
}

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

BSTR* pbstrNames = NULL;

if( FAILED(SafeArrayAccessData( psa, (void**)(&pbstrNames) ) ) )
{
```

```
        SafeArrayUnaccessData( psa );
        SafeArrayDestroy( psa );
        ::MessageBox( NULL, _T("Failed to access scan filter array"), _T("Error"), MB_OK );
        return;
}

// display names one at a time
TCHAR szTitle[24];

for( long i=0; i<nArraySize; i++ )
{
        _stprintf( szTitle, _T("Method name %d"), i );
        ::MessageBox( NULL, pbstrNames[i], szTitle, MB_OK );
}

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

SetMassTolerance

long SetMassTolerance(BOOL bUserDefined, double dMassTolerance, long nUnits)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

bUserDefined	A flag indicating whether the mass tolerance is user-defined (TRUE) or based on the values in the raw file (FALSE).
dMassTolerance	The mass tolerance value.
nUnits	The type of tolerance value (amu, mmu, ppm).

Remarks

This function sets the mass tolerance that will be used with the raw file.

Example

```
// example for SetMassTolerance
double dMassTolerance = 0.5;
long nUnits = 0;
BOOL bUserDefined = TRUE;

long nRet = XRawfileCtrl. SetMassTolerance ( bUserDefine, dMassTolerance, nUnits );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error while setting the mass tolerance"), _T("Error"),
                        MB_OK );
        …
}
```

GetChros

long GetChros(long nChros, double *pdStartTime, double *pdEndTime, VARIANT *pvarChroParamsArray, VARIANT *pvarSizeArray, VARIANT *pvarChroDataArray, VARIANT *pvarPeakFlagsArray)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nChros	A long variable containing the number of chromatograms to get from the raw file.
pdStartTime	A pointer to a double-precision variable containing the start time of the chromatogram time range to return.
pdEndTime	A pointer to a double-precision variable containing the end time of the chromatogram time range to return.
pvarChroParamsArray	A valid pointer to a VARIANT variable to parameters that will be used to generate each chromatogram.
pvarSizeArray	A valid pointer to a VARIANT variable to the size of each chromatogram.
pvarChroDataArray	A valid pointer to a VARIANT variable to receive the chromatogram data.
pvarPeakFlagsArray	A valid pointer to a VARIANT variable to receive the peak flag data.

Remarks

This function returns the requested chromatogram data as an array of double-precision time-intensity pairs in *pvarChroDataArray*. The number of time-intensity pairs is returned in *pvarSizeArray*.

You can use the start and end times, *pdStartTime* and *pdEndTime*, 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 you can obtain by calling *GetStartTime* and *GetEndTime*, respectively. Or, if the entire chromatogram is to be returned, you can set *pdStartTime* and *pdEndTime* to zero. On return, *pdStartTime* and *pdEndTime* contain the actual time range of the returned chromatographic data.

The parameters that are used to generate the chromatograms are contained in *pvarChroParamsArray*. They include (in this order) the trace type 1 (int), trace operator (int), trace type 2 (int), filter string (bstr) or a VARIANT array of compound names, mass range 1 (bstr), mass range 2 (bstr), delay (double), start RT (double), end RT (double), start scan number (int), end scan number (int), smoothing type (int), and number of smoothing points (int). The description of the GetChroData function contains additional information on these values.

The filter string or VARIANT array of compound names allows the caller to either use a specific filter to derive the chromatograms or allow the library to select the chromatograms by using compound names. Use the array of compound names in the same manner as you would in GetChroByCompoundName().

The chromatogram list contents are returned in a *SafeArray* attached to the *pvarChroDataArray* VARIANT variable. When passed in, the *pvarChroData* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarChroDataArray* is set to type VT_ARRAY|VT_R8. The format of the chromatogram list returned is an array of double-precision values in time-intensity pairs in ascending time order (for example, time 1, intensity 1, time 2, intensity 2, time 3, intensity 3, and so forth).

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

On successful return, *pvarSizeArray* contains the number of time-intensity pairs stored in *pvarChroDataArray.* When passed in, the *pvarSizeArray* variable must exist and be initialized to VARIANT type VT_EMPTY. If the function returns successfully, *pvarSizeArray* is set to type VT_ARRAY|VT_I4.

GetSegmentedMassListFromRT

**long GetSegmentMassListFromRT(double *pdRT, BSTR bstrFilter,
long nIntensityCutoffType,
long nIntensityCutoffValue,
long nMaxNumberOfPeaks, BOOL bCentroidResult,
double *pdCentroidPeakWidth,
VARIANT * pvarMassList, VARIANT * pvarPeakFlags,
long *pnArraySize, VARIANT *pvarSegments,
long *pnNumSegments,
VARIANT *pvarLowHighMassRange)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pdRT	A valid pointer to a double-precision variable containing the retention time, in minutes, that the corresponding mass list data is to be returned for.
bstrFilter	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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
pvarSegments	A valid pointer to a VARIANT variable to receive the segment data.
pnNumSegments	A valid pointer to a long variable to receive the number of segments returned in the segments array.
pvarMassRange	A valid pointer to a VARIANT variable to receive the mass range data.

Remarks

This function applies only to scanning devices such as MS.

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

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

To reduce the number of low-intensity data peaks returned, you can apply an intensity cutoff, *nIntensityCutoffType*. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted on the basis of 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 are 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 are 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* is set to type VT_ARRAY|VT_R8. The format of the mass list returned is an array of double-precision values in mass-intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so forth).

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

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

The *varSegments* array contains information about the segments, and the *varMassRange* array contains the mass range for each segment. The *nSegments* variable contains the number of segments.

Example

```
// example for GetSegmentedMassListFromRT

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

VARIANT varMassList;
VariantInit(&varMassList);
```

```
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);

VARIANT varSegments;
VariantInit(&varSegments);

VARIANT varMassRange;
VariantInit(&varMassRange);

double dRT = 3.8;// read the contents of the scan at RT = 3.8 minutes
long nArraySize = 0;
long nSegments = 0;

long nRet = XRawfileCtrl.GetSegmentedMassListFromRT ( &dRT,
   NULL, 0, 0, 0, FALSE,
   &varMassList,
   &varPeakFlags,
   &nArraySize,
            &varSegments,
   &nSegments,
   &varMassRange);

if( nRet != 0 )
{
          ::MessageBox( NULL, _T("Error getting mass list data for scan at 3.8 minutes."),
         _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 );
}

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

        // Delete the SafeArray
        SafeArrayDestroy( psa );
}

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

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

GetSegmentedMassListFromScanNum

**long GetSegmentMassListFromScanNum(long *pnScanNumber, BSTR bstrFilter,
long nIntensityCutoffType,
long nIntensityCutoffValue,
long nMaxNumberOfPeaks, BOOL bCentroidResult,
double *pdCentroidPeakWidth,
VARIANT * pvarMassList, VARIANT * pvarPeakFlags,
long *pnArraySize, VARIANT *pvarSegments,
long *pnNumSegments,
VARIANT *pvarMassRange)**

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number that the corresponding mass list data is to be returned for.
bstrFilter	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.
bCentroidResult	Boolean flag indicating that returned mass list contents should be centroided.
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.
pvarSegments	A valid pointer to a VARIANT variable to receive the segment data.
pnNumSegments	A valid pointer to a long variable to receive the number of segments returned in the segments array.
pvarMassRange	A valid pointer to a VARIANT variable to receive the mass range data.

Remarks

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

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

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

To reduce the number of low-intensity data peaks returned, you can apply an intensity cutoff, *nIntensityCutoffType*. The available types of cutoff are None, Absolute (intensity), and Relative (relative intensity). The value of *nIntensityCutoffValue* is interpreted on the basis of 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 are 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 are 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* is set to type VT_ARRAY|VT_R8. The format of the mass list returned is an array of double-precision values in mass-intensity pairs in ascending mass order (for example, mass 1, intensity 1, mass 2, intensity 2, mass 3, intensity 3, and so forth).

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

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

The *varSegments* array contains information about the segments, and the *varMassRange* array contains the mass range for each segment. The *nSegments* variable contains the number of segments.

Example

```
// example for GetSegmentedMassListFromScanNum

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

VARIANT varMassList;
VariantInit(&varMassList);
```

```
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);

VARIANT varSegments;
VariantInit(&varSegments);

VARIANT varMassRange;
VariantInit(&varMassRange);

long nScanNumber = 12;// read the contents of scan 12
long nArraySize = 0;
long nSegments = 0;

long nRet = XRawfileCtrl.GetSegmentedMassListFromScanNum ( &nScanNumber,
  NULL, 0, 0, 0, FALSE,
 &varMassList,
&varPeakFlags,
&nArraySize,
&varSegments,
&nSegments,
&varMassRange);

if( nRet != 0 )
{
        ::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;

        // 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 );
}

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

        // Delete the SafeArray
        SafeArrayDestroy( psa );
}

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

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

GetScanEventForScanNum

long GetScanEventForScanNumberTextEx(long nScan, BSTR FAR* pbstrScanEvent)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScan The scan number that the scan event is being requested for.

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

Remarks

This function returns scan event information as a string for the specified scan number.

Example

```
// example for GetScanEventForScanNum

long nScan = 10;
BSTR bstrScanEvent = NULL;

long nRet = XRawfileCtrl. GetScanEventForScanNum ( nScan, & bstrScanEvent);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting the scan event for scan 10"), _T("Error"),
MB_OK );
        …
}
…

SysFreeString(bstrScanEvent);
```

GetSeqRowUserTextEx

long GetSeqRowUserTextEx(long nIndex, BSTR FAR* pbstrUserText)

Return Value

0 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

This function 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.GetSeqRowUserTextEx ( 2, &bstrUserText );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting user text 3 from seq row"), _T("Error"), MB_OK );
        …
}
…

SysFreeString(bstrUserText);
```

GetSeqRowBarcode

long GetSeqRowBarcode(BSTR FAR* pbstrBarcode)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

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

Remarks

This function returns the barcode used to acquire the raw file. This field is empty if the raw file was created by file format conversion or acquired from a tuning program.

Example

```
// example for GetSeqRowBarcode
BSTR bstrBarcode = NULL;

long nRet = XRawfileCtrl.GetSeqRowBarcode ( &bstrBarcode );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting barcode from seq row"), _T("Error"), MB_OK );
        …
}
…

SysFreeString(bstrBarcode);
```

GetSeqRowBarcodeStatus

long GetSeqRowBarcodeStatus(long* pnStatus)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnStatus A valid pointer to a long. This variable must exist and be initialized to 0.

Remarks

This function returns the barcode status from the raw file. This field is empty if the raw file was created by file format conversion or acquired from a tuning program.

Example

```
// example for GetSeqRowBarcodeStatus
long nStatus = 0;

long nRet = XRawfileCtrl.GetSeqRowBarcodeStatus ( &nStatus );

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting barcode status from seq row"), _T("Error"),
                        MB_OK );
        …
}
…
```

GetSegmentAndScanEventForScanNum

long GetSegmentAndScanEventForScanNum(long nScanNumber, long *pnSegment, long* pnScanEvent)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnSegment A valid pointer to a long. This variable must exist and be initialized to 0.

pnScanEvent A valid pointer to a long. This variable must exist and be initialized to 0.

Remarks

Returns the segment and scan event indexes for the specified scan.

Example

```
// example for GetSegmentAndScanEventForScanNum
long nScanNumber = 1;
long nSegment = 0;
long nScanEvent =0;

long nRet = XRawfileCtrl. GetSegmentAndScanEventForScanNum ( nScanNumber, &nSegment,
                                                             &nScanEvent );

if( nRet != 0 )
{
::MessageBox( NULL, _T(“Error getting the segment and scan event indexes for a scan”),
                    _T(“Error”),  MB_OK );
          …
}
…
```

GetMassPrecisionEstimate

long GetMassPrecisionEstimate(long nScanNumber, VARIANT *pvarMassList, long *pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pnScanNumber	A valid pointer to a long variable containing the scan number for which the corresponding mass list data is to be returned.
pvarMassList	A valid pointer to a VARIANT variable to receive the mass precision 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. It gets the mass precision information for an accurate mass spectrum (that is, acquired on an FTMS- or Orbitrap-class instrument).

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

The mass list contents are 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* is set to type VT_ARRAY|VT_R8. The format of the mass list returned is an array of double-precision values in the order of intensity, mass, accuracy in MMU, accuracy in PPM, and resolution.

Example

```
// example for GetMassPrecisionEstimate

typedef struct _datapeak
{
        double dIntensity;
        double dMass;
        double dAccuracyMMU;
        double dAccuracyPPM;
        double dResolution;
} DataPeak;

VARIANT varMassList;
```

```
VariantInit(&varMassList);

long nScanNumber = 12;// read the contents of scan 12
long nArraySize = 0;
long nSegments = 0;

long nRet = XRawfileCtrl. GetMassPrecisionEstimate ( nScanNumber,  &varMassList,
                                                     &nArraySize);

if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting mass precision 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;
double dAccuracyMMU = pDataPeaks[j].dAccuracyMMU;
double dAccuracyMMU = pDataPeaks[j].dAccuracyPPM;
double dResolution = pDataPeaks[j]. dResolution;

        // Do something with mass precision values
        …
        }

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

GetNumberOfMassRangesFromScanNum

long GetNumberOfMassRangesFromScanNum(int nScanNumber, long *pnNumMassRanges)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that the data is being requested for.
pnNumMassRanges	A valid pointer to a long variable to receive the number of mass ranges in this scan.

Remarks

This function gets the number of MassRange data items in the scan.

Example

```
// example for GetNumberOfMassRangesFromScanNum
long nMassRanges;
long nRet = XRawfileCtrl.GetNumberOfMassRangesFromScanNum(0, &nMassRanges);
if( nRet != 0 )
{
::MessageBox( NULL, _T("Error getting number of mass ranges for scan 0"), _T("Error"), MB_OK
);
…
}
```

GetNumberOfMSOrdersFromScanNum

long GetNumberOfMSOrdersFromScanNum(long nScanNumber, long *pnNumMSOrders)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
pnNumMSOrders	A valid pointer to a long variable to receive the number of mass orders in this scan.

Remarks

This function gets the number of MS reaction data items in the scan event for the scan specified by *nScanNumber* and the transition specified by *nMSOrder* from the scan event structure in the raw file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for GetNumberOfMSOrdersFromScanNum
long nMSOrders;
long nRet = XRawfileCtrl.GetNumberOfMSOrdersFromScanNum(0, &nMSOrders);
if( nRet != 0 )
{
::MessageBox( NULL, _T("Error getting number of mass orders for scan 0"), _T("Error"), MB_OK
);
…
}
```

GetNumberOfMassCalibratorsFromScanNum

long GetNumberOfMassCalibratorsFromScanNum(int nScanNumber, long *pnNumMassCalibrators)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number from which the data is being requested.
pnNumMassCalibrators	A valid pointer to a long variable to receive the number of mass calibrators in this scan.

Remarks

This function gets the number of mass calibrators (each of which is a double) in the scan.

Example

```
// example for GetNumberOfMassCalibratorsFromScanNum
long nNumMassCalibrators;
long nRet = XRawfileCtrl.GetNumberOfMassCalibratorsFromScanNum(0,
&nNumMassCalibrators);
if( nRet != 0 )
{
::MessageBox( NULL, _T("Error getting number of mass calibrators for scan 0"), _T("Error"),
MB_OK );
…
}
```

GetCycleNumberFromScanNumber

long GetCycleNumberFromScanNumber(long nScanNumber, long *pnCycleNumber)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the scan type information.

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

Remarks

This function returns the cycle number for the scan specified by *nScanNumber* from the scan index structure in the raw file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for GetCycleNumberFromScanNumber
long nScanNum = 12; // Is the twelfth scan from the start of file
long nCycleNumber
long nRet = XRawfileCtrl. GetCycleNumberFromScanNumber (nScanNum, &nCycleNumber);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting the cycle number for scan number 12"), _T("Error"),
MB_OK );
…
}
```

GetUniqueCompoundNames

long GetUniqueCompoundNames(VARIANT FAR* pvarCompoundNamesArray, long FAR* pnArraySize)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pvarCompoundNamesArray	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

This function returns the list of unique compound names for the raw file. If the function succeeds, pvarFilterArray points to an array of BSTR fields, each containing a unique compound name. *PnArraySize* contains the number of compound names in the pvarFilterArray.

Example

```
// example for GetUniqueCompoundNames
VARIANT varNames;
VariantInit(&varFilters);
long nArraySize = 0;
long nRet = XRawfileCtrl. GetUniqueCompoundNames ( &varNames, &nArraySize );
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting array of compound names "), _T("Error"), MB_OK);
  return
}
if( !nArraySize || varFilters.vt != (VT_ARRAY | VT_BSTR) )
{
  ::MessageBox( NULL, _T("No valid compound names 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 compound names "), _T("Error"), MB_OK);
  return;
}
```

```
// display names one at a time
TCHAR szTitle[16];
for( long i=0; i
{
  _tprintf( szTitle, _T("Compouund Name:  %d"), i );
  ::MessageBox( NULL, pbstrFilters[i], szTitle, MB_OK );
}


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

GetCompoundNameFromScanNum

long GetCompoundNameFromScanNum(int nScanNumber , BSTR *pbstrCompoundName)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that the compound name is being requested for.

pbstrCompoundName A valid pointer to a variable of type BSTR to receive the compound names string value for the requested scan. This variable must exist.

Remarks

This function returns a compound name as a string for the specified scan number. The value of *nScanNumber* must be within the range of scans for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

Example

```
// example for GetCompoundNameFromScanNum
BSTR bstrName = NULL;
long nRet = XRawfileCtrl. GetCompoundNameFromScanNum ( 10, &bstrName );
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting the compound  name for scan 10"), _T("Error"),
      MB_OK );
    …
}
…
SysFreeString(bstrScanEvent);
```

GetAValueFromScanNum

long GetAValueFromScanNum(long nScanNumber, long *pnAValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnAValue A valid pointer to a long variable to receive the value of the A parameter in this scan.

Remarks

This function gets the A parameter value in the scan event. The value returned is either 0, 1, or 2 for parameter A off, parameter A on, or accept any parameter A, respectively.

Example

```
// example for GetAValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetAValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting A value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetBValueFromScanNum

long GetBValueFromScanNum(long nScanNumber, long *pnBValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnBValue A valid pointer to a long variable to receive the value of the B parameter in this scan.

Remarks

This function gets the B parameter value in the scan event. The value returned will be either 0, 1, or 2 for parameter B off, parameter B on, or accept any parameter B, respectively.

Example

```
// example for GetBValueFromScanNum
long nResult;

long nRet = XRawfileCtrl.GetBValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting B value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetFValueFromScanNum

long GetFValueFromScanNum(long nScanNumber, long *pnFValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnFValue A valid pointer to a long variable to receive the value of the F parameter in this scan.

Remarks

This function gets the F parameter value in the scan event. The value returned is either 0, 1, or 2 for parameter F off, parameter F on, or accept any parameter F, respectively.

Example

```
// example for GetFValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetFValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting F value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetKValueFromScanNum

long GetKValueFromScanNum(long nScanNumber, long *pnKValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnKValue A valid pointer to a long variable to receive the value of the K parameter in this scan.

Remarks

This function gets the K parameter value in the scan event. The value returned is either 0, 1, or 2 for parameter K off, parameter K on, or accept any parameter K, respectively.

Example

```
// example for GetKValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetKValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting K value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetRValueFromScanNum

long GetRValueFromScanNum(long nScanNumber, long *pnRValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnRValue A valid pointer to a long variable to receive the value of the R parameter in this scan.

Remarks

This function gets the R parameter value in the scan event. The value returned is either 0, 1, or 2 for parameter R off, parameter R on, or accept any parameter R, respectively.

Example

```
// example for GetRValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetRValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting R value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetVValueFromScanNum

long GetVValueFromScanNum(long nScanNumber, long *pnVValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnVValue A valid pointer to a long variable to receive the value of the V parameter in this scan.

Remarks

This function gets the R parameter value in the scan event. The value returned is either 0, 1, or 2 for parameter R off, parameter R on, or accept any parameter R, respectively.

Example

```
// example for GetVValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetVValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting V value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetMSXMultiplexValueFromScanNum

long GetMSXMultiplexValueFromScanNum(long nScanNumber, long *pnMSXValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber The scan number that is returned for the corresponding precursor information.

pnMSXValue A valid pointer to a long variable to receive the value of the msx-multiplex parameter in this scan.

Remarks

This function gets the msx-multiplex parameter value in the scan event. The value returned is either 0, 1, or 2 for multiplex off, multiplex on, or accept any multiplex, respectively.

Example

```
// example for GetMSXMultiplexValueFromScanNum
long nResult;
long nRet = XRawfileCtrl.GetMSXMultiplexValueFromScanNum(0, &nResult);
if( nRet != 0 )
{
  ::MessageBox( NULL, _T("Error getting multiplex value for scan 0"), _T("Error"), MB_OK );
  …
}
```

GetMassCalibrationValueFromScanNum

long GetMassCalibrationValueFromScanNum(long nScanNumber, long nMassCalibrationIndex, double *pdMassCalibrationValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding precursor information.
nMassCalibrationIndex	The index of the mass calibration wanted.
pdMassCalibrationValue	A valid pointer to a double data item where you can place the data. This must not be null.

Remarks

This function retrieves information about one of the mass calibration data values of a scan. You can find the count of mass calibrations for the scan by calling GetNumberOfMassCalibratorsFromScanNum().

Example

```
// example for GetMassCalibrationValueFromScanNum
double dMassCalibratorValue;
long nRet = XRawfileCtrl.GetMassCalibrationValueFromScanNum(0, 0, &dMassCalibratorValue);
if( nRet != 0 )
{
    ::MessageBox( NULL, _T("Error getting the mass calibrator value for scan 0, calibrator 0"),
        _T("Error"), MB_OK );
…
}
```

GetFullMSOrderPrecursorDataFromScanNum

long GetFullMSOrderPrecursorDataFromScanNum(long nScanNumber, long nMSOrder, LPVARIANT pvarFullMSOrderPrecursorInfo)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding precursor information.
nMSOrder	The MS^n order for the scan. This should be a value between MS_AcceptAnyMSorder and MS_ms100 (see values for the MS_MSOrder enumeration) but not more than the value returned from GetNumberOfMSOrdersFromScanNum().
pvarFullMSOrderPrecursorInfo	A valid pointer to a VARIANT array that places the full mass order precursor information into a safe array.

Remarks

This function retrieves information about the reaction data of a data-dependent MS^n for the scan specified by *nScanNumber* and the transition specified by *nMSOrder* from the scan event structure in the raw file.

- Reaction data refers to precursor mass, isolation width, collision energy, whether the collision energy is valid, whether the precursor mass is valid, the first precursor mass, the last precursor mass, and the isolation width offset.
- Specify the data-dependent MS^n through the *nMSOrder* input. You can find the count of MS orders by calling GetNumberOfMSOrdersFromScanNum.
- Specify the scan through the *nScanNumber* input. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber.

When you input the scan number and mass order that you want data from, GetFullMSOrderPrecursorDataFromScanNum returns a full mass order precursor information structure. You receive access to this data in the form of a FullMSOrderPrecursorInfo object. The return is in an array since the VARIANT does not allow you to store a structure.

```
struct FullMSOrderPrecursorInfo
{
   double dPrecursorMass;
   double dIsolationWidth;
   double dCollisionEnergy;
```

```
    UINT uiCollisionEnergyValid;    // set to 1 to use in scan filtering. High-order bits hold the
                                    // activation type enum bits 0xffe and the flag for multiple
                                    // activation (bit 0x1000). You can implement yhese features
                                    // individually with the new access functions or as a UINT
                                    // with the new CollisionEnergyValueEx function.
    BOOL bRangeIsValid;             // If TRUE, dPrecursorMass is still the center mass, but
                                    // dFirstPrecursorMass and dLastPrecursorMass are also
                                    // valid.
    double dFirstPrecursorMass;     // If bRangeIsValid == TRUE, this value defines the start of
                                    // the precursor isolation range.
    double dLastPrecursorMass;      // If bRangeIsValid == TRUE, this value defines the end of the
                                    // precursor isolation range.
    double dIsolationWidthOffset;
};
```

Example

```
// example for GetFullMSOrderPrecursorDataFromScanNum
void CTestOCXDlg::OnOpenParentScansOcx()
{
  try
  {
    VARIANT vPrecursorInfos;
    VariantInit(&vPrecursorInfos);

    // Get the precursor scan information
   long nRet =  m_Rawfile.GetFullMSOrderPrecursorDataFromScanNum(0, 1, &vPrecursorInfos);
   if( nRet != 0 )
   {
       ::MessageBox( NULL, _T("Error getting the precursor information for scan 0, MS order 1"),
                _T("Error"), MB_OK );
       …
   }

    // Access the FullMSOrderPrecursorInfo data buffer
    BYTE* pData;
    SafeArrayAccessData(vPrecursorInfos.parray, (void**)&pData);

    // Copy the scan information from the safearray buffer
    FullMSOrderPrecursorInfo info;
    memcpy(&info, pData , sizeof(FullMSOrderPrecursorInfo));

    // Process the full mass order precursor info scan information ...

    SafeArrayUnaccessData(vPrecursorInfos.parray);
  }
  catch (...)
  {
    AfxMessageBox(_T("There was a problem while getting this scan's precursor information."));
  }
}
```

GetMassRangeFromScanNum

long GetMassRangeFromScanNum(long nScanNumber, long nMassRangeIndex, double *pdMassRangeLowValue, double *pdMassRangeHighValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number that is returned for the corresponding precursor information.
nMassRangeIndex	The index of the mass range requested.
pdMassRangeLowValue	A valid pointer to a double to place the mass range low value into. You cannot set it to NULL.
pdMassRangeHighValue	A valid pointer to a double to place the mass range high value into. You cannot set it to NULL.

Remarks

This function retrieves information about the mass range data of a scan (high and low masses). You can find the count of mass ranges for the scan by calling GetNumberOfMassRangesFromScanNum().

Example

```
// example for GetMassRangeFromScanNum
void CTestOCXDlg::OnOpenParentScansOcx()
{
 try
 {
  double dMassRangeLowValue, dMassRangeHighValue

  // Get the precursor scan information
 long nRet =  m_Rawfile.GetMassRangeFromScanNum(0, 1, &dMassRangeLowValue,
         &dMassRangeHighValue);
 if( nRet != 0 )
 {
      ::MessageBox( NULL, _T("Error getting the scan's range information  value for scan 0, range
item
                    0"),   _T("Error"), MB_OK );
       …
  }

  // Process the mass range information ...
 }
 catch (...)
 {
```

```
    AfxMessageBox(_T("There was a problem while getting this scan's range information."));
  }
}
```

GetMassTolerance

long GetMassTolerance(BOOL *bUserDefined, double *dMassTolerance, long* nUnits)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

bUserDefined	Returned flag indicating whether the mass tolerance is user-defined (TRUE) or based on the values in the raw file (FALSE).
dMassTolerance	The returned mass tolerance value.
nUnits	The returned type of tolerance value (amu = 2, mmu = 0, or ppm = 1).

Remarks

This function gets the mass tolerance that is being used with the raw file. To set these values, use the SetMassTolerance() function.

Example

```
// example for GetMassTolerance
double dMassTolerance;
long nUnits;
BOOL bUserDefined;
long nRet = XRawfileCtrl. GetMassTolerance ( &bUserDefine, &dMassTolerance, &nUnits );
if( nRet != 0 )
{
    ::MessageBox( NULL, _T("Error while getting the mass tolerance"), _T("Error"), MB_OK );
    …
}
```

GetNumberOfSourceFragmentsFromScanNum

long GetNumberOfSourceFragmentsFromScanNum(long nScanNumber, long *pnNumSourceFragments)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number from which the data being requested.
pnNumSourceFragments	A valid pointer to a long variable to receive the number of source fragments in this scan.

Remarks

This function gets the number of source fragments (or compensation voltages) in the scan.

Example

```
// example for GetNumberOfSourceFragmentsFromScanNum
long nSourceFragments
long nRet = XRawfileCtrl. GetNumberOfSourceFragmentsFromScanNum (0, &
nSourceFragments);
if( nRet != 0 )
{
::MessageBox( NULL, _T("Error getting number of Source Fragments for scan 0"), _T("Error"),
MB_OK );
…
}
```

GetSourceFragmentValueFromScanNum

long GetSourceFragmentValueFromScanNum(long nScanNumber, long nSourceFragmentIndex, double *pdSourceFragmentValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
nSourceFragmentIndex	The index of the source fragment wanted.
pdSourceFragmentValue	A valid pointer to a double data item where the data can be placed. This must not be null.

Remarks

This function retrieves information about one of the source fragment values of a scan. It is also the same value as the compensation voltage. You can find the count of source fragments for the scan by calling GetNumberOfSourceFragmentsFromScanNum ().

Example

```
// example for GetSourceFragmentValueFromScanNum
double dSourceFragmentsValue;
long nRet = XRawfileCtrl. GetSourceFragmentValueFromScanNum (0, 0,
&dSourceFragmentsValue);
if( nRet != 0 )
{
    ::MessageBox( NULL, _T("Error getting the Source Fragments value for scan 0, calibrator 0"),
        _T("Error"), MB_OK );
    …
}
```

GetNumberOfSourceFragmentationMassRangesFromScanNum

long GetNumberOfSourceFragmentationMassRangesFromScanNum(long nScanNumber, long *pnNumSourceFragmentationMassRanges)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number from which the data is being requested.
pnNumSourceFragmentationMassRanges	A valid pointer to a long variable to receive the number of source fragmentation mass ranges in this scan.

Remarks

This function gets the number of source fragmentation mass ranges in the scan.

Example

```
// example for GetNumberOfSourceFragmentationMassRangesFromScanNum
long nSourceFragmentationMassRanges
long nRet = XRawfileCtrl. GetNumberOfSourceFragmentationMassRangesFromScanNum (0, &
nSourceFragmentationMassRanges);
if( nRet != 0 )
{
   ::MessageBox( NULL, _T("Error getting number of Source Fragments for scan 0"), _T("Error"),
        MB_OK );
     …
}
```

GetSourceFragmentationMassRangeFromScanNum

long GetSourceFragmentationMassRangeFromScanNum(long nScanNumber, long nSourceFragmentIndex, double *pdSourceFragmentRangeLowValue, double *pdSourceFragmentRangeHighValue)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
nSourceFragmentIndex	The index of the source fragment mass range wanted.
pdSourceFragmentRangeLowValue	A valid pointer to a double to place the source fragment mass range low value into. You cannot set it to NULL.
pdSourceFragmentRangeHighValue	A valid pointer to a double to place the source fragment mass range high value into. You cannot set it to NULL.

Remarks

This function retrieves information about the source fragment mass range data of a scan (high and low source fragment masses). You can find the count of mass ranges for the scan by calling GetNumberOfSourceFragmentationMassRangesFromScanNum ().

Example

```
// example for GetSourceFragmentationMassRangeFromScanNum
void CTestOCXDlg::OnOpenParentScansOcx()
{
  try
  {
    double dSourceFragmentMassRangeLowValue, dSourceFragmentMassRangeHighValue

    // Get the precursor scan information
   long nRet =  m_Rawfile. GetSourceFragmentationMassRangeFromScanNum (0, 1,
          &dSourceFragmentMassRangeLowValue,  &dSourceFragmentMassRangeHighValue);

   if( nRet != 0 )
   {
        ::MessageBox( NULL, _T("Error getting the Source Fragmentation Mass Range value for
scan 0,
                  calibrator 0"),  _T("Error"), MB_OK );
        …
    }

    // Process the mass range information ...
```

```
  }
  catch (...)
  {
      AfxMessageBox(_T("There was a problem while getting this scan's fragment mass range
              Information."));
  }
}
```

GetIsolationWidthForScanNum

long GetIsolationWidthForScanNum(long nScanNumber, long nMSOrder, double FAR *pdIsolationWidth)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
nMSOrder	The MS^n order for the scan. This should be a value between MS_AcceptAnyMSorder and MS_ms100 (see values for the MS_MSOrder enumeration), but not more than the value returned from GetNumberOfMSOrdersFromScanNum().
pdIsolationWidth	A valid pointer to a variable of type double. This variable must exist.

Remarks

This function returns the isolation width for the scan specified by *nScanNumber* and the transition specified by *nMSOrder* from the scan event structure in the raw file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber. You can acquire the number of MS orders by calling GetNumberOfMSOrdersFromScanNum.

Example

```
// example for GetIsolationWidthForScanNum
long nScanNum = 12; // Is the twelfth scan from the file
long nMSOrder = 2;
double dIsolationWidth;
long nRet = XRawfileCtrl. GetIsolationWidthForScanNum (nScanNum, nMSOrder,
&dIsolationWidth);
if( nRet != 0 )
{
   ::MessageBox( NULL, _T("Error getting the Isolation Width at index 2 for scan number 12"),
         _T("Error"), MB_OK );
   …
}
```

GetCollisionEnergyForScanNum

long GetCollisionEnergyForScanNum(long nScanNumber, long nMSOrder, double FAR *pdCollisionEnergy)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
nMSOrder	The MS^n order for the scan. This should be a value between MS_AcceptAnyMSorder and MS_ms100 (see values for the MS_MSOrder enumeration), but not more than the value returned from GetNumberOfMSOrdersFromScanNum().
pdCollisionEnergy	A valid pointer to a variable of type double. This variable must exist.

Remarks

This function returns the collision energy for the scan specified by *nScanNumber* and the transition specified by *nMSOrder* from the scan event structure in the raw file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can find the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber. You can acquire the number of MS orders by calling GetNumberOfMSOrdersFromScanNum.

Example

```
// example for GetCollisionEnergyForScanNum
long nScanNum = 12; // Is the twelfth scan from the file
long nMSOrder = 2;
double dCollisionEnergy;
long nRet = XRawfileCtrl. GetCollisionEnergyForScanNum (nScanNum, nMSOrder, &
dCollisionEnergy);
if( nRet != 0 )
{
   ::MessageBox( NULL, _T("Error getting the collision energy at index 2 for scan number 12"),
      _T("Error"), MB_OK );
   …
}
```

GetPrecursorRangeForScanNum

long GetPrecursorRangeForScanNum(long nScanNumber, long nMSOrder, double FAR *pdFirstPrecursorMass, double FAR *pdLastPrecursorMass, BOOL FAR *pbIsValid)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
nMSOrder	The MS^n order for the scan. This should be a value between MS_AcceptAnyMSorder and MS_ms100 (see values for the MS_MSOrder enumeration), but not more than the value returned from GetNumberOfMSOrdersFromScanNum().
pdFirstPrecursorMass	A valid pointer to a variable of type double. This variable must exist. If pbIsValid is returned as TRUE, this value defines the start of the precursor isolation range.
pdLastPrecursorMass	A valid pointer to a variable of type double. This variable must exist. If pbIsValid is returned as TRUE, this value defines the end of the precursor isolation range.
pbIsValid	A valid pointer to a variable of type boolean. This variable must exist. If it is returned as TRUE, the precursor mass returned in GetPrecursorMassForScanNum() is still the center mass, and pdFirstPrecursorMass and pdLastPrecursorMass are valid.

Remarks

This function returns the first and last precursor mass values of the range and whether they are valid for the scan specified by *nScanNumber* and the transition specified by *nMSOrder* from the scan event structure in the raw file. The value of *nScanNumber* must be within the range of scans or readings for the current controller. You can obtain the range of scans or readings for the current controller by calling GetFirstSpectrumNumber and GetLastSpectrumNumber. You can acquire the number of MS orders by calling GetNumberOfMSOrdersFromScanNum.

Example

```
// example for GetPrecursorRangeForScanNum
long nScanNum = 12; // Is the twelfth scan from the file
long nMSOrder = 2;
double dFirstPrecursorMass, dLastPrecursorMass;
BOOL bIsValid
long nRet = XRawfileCtrl. GetPrecursorRangeForScanNum (nScanNum, nMSOrder,
      &dFirstPrecursorMass, &dLastPrecursorMass, &bIsValid);
```

```
if( nRet != 0 )
{
   ::MessageBox( NULL, _T("Error getting the precursor mass ranges  at index 2 for scan number
         12"),_T("Error"),  MB_OK );
   …
}
else if( ! bIsValid )
{
   ::MessageBox( NULL, _T("The precursor mass ranges at index 2 for scan number 12 are
          invalid."),_T("Bad Ranges"), MB_OK );
   …
}

// Process the precursor ranges
```

GetAllMSOrderData

long GetAllMSOrderData(long nScanNumber, VARIANT FAR* pvarDoubleData, VARIANT FAR* pvarFlagsData, long FAR* pnNumberOfMSOrders)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

nScanNumber	The scan number for which the data is being requested.
pvarDoubleData	A valid pointer to a VARIANT variable to receive the label data. This is an array of 6 X *n*, where *n* is the number of MS orders returned in the *pnNumberOfMassOrders* parameter. The items in the array are all doubles: precursor mass, isolation width, collision energy, first precursor mass, last precursor mass, and isolation width offset.
pvarFlagData	A valid pointer to a VARIANT variable to receive the flags. This is an array of 2 x *n*, where *n* is the number of MS orders returned in the *pnNumberOfMassOrders* parameter. The items in the array are all 2-byte integers (short): activation type (an enumeration of 11 states (see the MS_Activations enum in the XRawfile2.idl0)) and whether the precursor range is valid (BOOLEAN).
pnNumberOfMassOrders	A valid pointer to a long variable that receives the number of MS orders (entries in the arrays provided in pvarDoubleData and pvarFlagData).

Remarks

This method enables you to obtain all of the precursor information from the scan (event).

The FT-PROFILE labels of a scan are represented by *nScanNumber*. PvarFlags can be NULL if you do not want to receive 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 character values. The flags are saturated, fragmented, merged, exception, reference, and modified.

Example

```
// example for GetAllMSOrderData
long nScanNumber = 1; // get the noise packets of the first scan.
VARIANT varDoubleData, varFlagsData;
long numberOfMSOrders;

long nRet = XRawfileCtrl.GetAllMSOrderData(nScanNumber, &varDoubleData, &varFlagsData,
        &numberOfMSOrders);
if( nRet != 0 )
{
        ::MessageBox( NULL, _T("Error getting noise packets."), _T("Error"), MB_OK );
```

```
        …
}
_variant_t vDoubleData = &varDoubleData;
_variant_t vFlagsData = &varFlagsData;
SAFEARRAY pDoubleArray = vDoubleData.parray;
SAFEARRAY pFlagsArray = vFlagsData.parray;

double pdval = (double *)vDoubleData->pvData;
short psval = (short)vFlagsData->pvData;
for (int inx = 0; inx < numberOfMSOrders; inx++)
{
        double dPrecursorMassMass = (double) pdval[((inx)*6)+0] ;
        double dIsolationWidth = (double) pdval[((inx)*6)+1] ;
        double dCollisionEnergy = (double) pdval[((inx)*6)+2] ;
        double dFirstPrecursorMass = (double) pdval[((inx)*6)+3] ;
        double dLastPrecursorMass = (double) pdval[((inx)*6)+4] ;
        double dIsolationWidthOffset = (double) pdval[((inx)*6)+5] ;

        enum MS_RActivations eActivation = (enum MS_RActivations) psval[((inx)*2)+0] ;
        BOOL bPrecursorRangeIsValid = (BOOL) psval[((inx)*2)+1] ;
        // Do something with the data.
        …
}
```

GetChroByCompoundName

long GetChroByCompoundName(long nChroType1, long nChroOperator, long nChroType2, VARIANT *pCompoundNames, 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

0 if successful; otherwise, see Error Codes.

Parameters

nChroType1	A long variable containing the first chromatogram trace type of interest.
nChroOperator	A long variable containing the chromatogram trace operator.
nChroType2	A long variable containing the second chromatogram trace type of interest.
pCompoundNames	(Input) An array of strings containing the compounds to filter the chromatogram with.
szMassRanges1	A string containing the formatted mass ranges for the first chromatogram trace type.
szMassRanges2	A string containing the formatted mass ranges for the second chromatogram trace type.
dDelay	A double-precision variable containing the chromatogram delay in minutes.
pdStartTime	A pointer to a double-precision variable containing the start time of the chromatogram time range to return.
pdEndTime	A pointer to a double-precision variable containing the end time of the chromatogram time range to return.
nSmoothingType	A long variable containing the type of chromatogram smoothing to be performed.
nSmoothingValue	A long variable containing the chromatogram smoothing value.
pvarChroData	A valid pointer to a VARIANT variable to receive the chromatogram 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 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*.

The chromatogram trace types and operator values of *nChroType1*, *nChroOperator*, and *nChroType2* depend 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 compound names are only valid for MS controllers. If you do not provide compound names, you cannot filter by compound names. Use the GetUniqueCompoundNames method to get all of the compound names available in the raw file.

The *dDelay* value contains the retention-time offset to add to the returned chromatogram times. You can set the value to 0.0 if you do not want an offset is. 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. You can leave these values empty for Base Peak or Spectrum Maximum trace types, but you must specify them for Mass Range or Wavelength Range trace types. See the "Mass1 (m/z) text box" topic in the Xcalibur Help for information on how to format mass ranges.

You can use the start and end times, *pdStartTime* and *pdEndTime*, 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 you can obtain by calling *GetStartTime* and *GetEndTime*, respectively. Or, if the entire chromatogram is returned, you can set *pdStartTime* and *pdEndTime* to zero. On return, *pdStartTime* and *pdEndTime* 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. For a list of the valid values for *nSmoothingType*, see Smoothing Type in the Enumerated Types section. The value of *nSmoothingValue* must be an odd number in the range of 3-15 if smoothing is desired. The chromatogram list contents are 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* is set to type VT_ARRAY|VT_R8. The format of the chromatogram list returned is an array of double-precision values in time-intensity pairs in ascending time order (for example, time 1, intensity 1, time 2, intensity 2, time 3, intensity 3, and so on).

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

On successful return, *pnArraySize* contains the number of time-intensity pairs stored in the *pvarChroData* array.

Example

```
// example for GetChroByCompoundName to return the MS TIC trace
typedef struct _datapeak
{
double dTime;
double dIntensity;
} ChroDataPeak;

XRawfileCtrl.SetCurrentController ( 0, 1 ); // first MS controller

VARIANT varChroData;
VariantInit(&varChroData);
VARIANT varPeakFlags;
VariantInit(&varPeakFlags);
long nArraySize = 0;
double dStartTime = 0.0;
double dEndTime = 0.0;

// Create the variant that contains the strings
CStringArray compoundNames;
compoundNames.Add("methyltestosterone");// Filter by this compound
SAFEARRAY FAR* psa;
SAFEARRAYBOUND rgsabound[1];

// Allocate safearray with room for the strings (indexed from 0).
rgsabound[0].lLbound = 0;
rgsabound[0].cElements = rCStrArray.GetSize();
psa = SafeArrayCreate(VT_BSTR, 1, rgsabound);
if (psa == NULL) {
::MessageBox( NULL, _T("Error converting compound names to VARIANT type."), _T("Error"),
MB_OK );
…
}

// Get a pointer to the the elements of the array.
BSTR* pstrItem;
if (FAILED(SafeArrayAccessData(psa, (void**) &pstrItem))) {
::MessageBox( NULL, _T("Error converting compound names to VARIANT type."), _T("Error"),
MB_OK );
…
}

//  add compounds to the array
for (int i = 0; i < *pnArraySize; i++) {
        *pstrItem++ = rCStrArray[i].AllocSysString();
}

if (FAILED(SafeArrayUnaccessData(psa))) {
::MessageBox( NULL, _T("Error converting compound names to VARIANT type."), _T("Error"),
MB_OK );
…
}
```

```
// Store new array in variant
VARIANT varArray
varArray->vt = VT_ARRAY | VT_BSTR;
varArray->parray = psa;

long nRet = XRawfileCtrl.GetChroByCompoundName( 1, // TIC trace
0,
0,
varArray,
NULL,
NULL,
0.0,
&dStartTime,
&dEndTime,
0,
0,
&varChroData,
&varPeakFlags,
&nArraySize );

if( nRet != 0 )
{
::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 );
}
if(varPeakFlags.vt != VT_EMPTY )
{
```

```
SAFEARRAY FAR* psa = varPeakFlags.parray;
varPeakFlags.parray = NULL;
// Delete the SafeArray
SafeArrayDestroy( psa );
}
```

IsQExactive

long IsQExactive(BOOL *pVal)

Return Value

0 if successful; otherwise, see Error Codes.

Parameters

pVal A valid pointer to a BOOL. This variable must exist.

Remarks

Checks the instrument name by calling *GetInstName()* and comparing the result to Q Exactive Orbitrap. If it matches, IsQExactive *pVal* is set to TRUE. Otherwise, *pVal* is set to FALSE.

Example

```
// example for IsQExactive
BOOL isRawFileFromQExactive;
long nRet = XRawfileCtrl. IsQExactive ( & isRawFileFromQExactive);
if( nRet != 0 )
{
::MessageBox( NULL, _T("Error verifying instrument"), _T("Error"), MB_OK );
…
}
…
```

Index

H

I

O

R

S

V