MHDAC: MassHunter Data Access Component

MIDAC: MassHunter IM-MS Data Access Component

Agilent Technologies, Inc.

B.08.00 (build 8132.0)

16-June-2016

Document updated 22-July-2016

Table of Contents

[Overview 2](#_Toc456949946)

[Data File Contents 3](#_Toc456949947)

[Original MassHunter Format 3](#_Toc456949948)

[IM-MS MassHunter Format 4](#_Toc456949949)

[The MHDAC\_MIDAC Distribution 6](#_Toc456949950)

[Contents 6](#_Toc456949951)

[Building with MHDAC and MIDAC 6](#_Toc456949952)

[Example Code 7](#_Toc456949953)

[Which Files to Include 7](#_Toc456949954)

[Using MIDAC 8](#_Toc456949955)

[Opening a Data File 8](#_Toc456949956)

[Reading Chromatograms 9](#_Toc456949957)

[Reading Mass Spectra 9](#_Toc456949958)

[MIDAC Status 9](#_Toc456949959)

[MIDAC Interfaces 10](#_Toc456949960)

[MidacFileAccess 10](#_Toc456949961)

[IMidacImsReader 11](#_Toc456949962)

[File Management 11](#_Toc456949963)

[File Metadata Access 11](#_Toc456949964)

[Frame Metadata Access 13](#_Toc456949965)

[MidacSpecFormat 14](#_Toc456949966)

[Profile Total Frame Spectra 15](#_Toc456949967)

[Peak Total Frame Spectra 16](#_Toc456949968)

[Frame Spectra 17](#_Toc456949969)

[Chromatograms 19](#_Toc456949970)

[IMidacMsFiltersSpec 20](#_Toc456949971)

[IMidacSpecData, 22](#_Toc456949972)

[IMidacSpecDataMs 23](#_Toc456949973)

[IMidacFileInfo 25](#_Toc456949974)

[IMidacFrameInfo 26](#_Toc456949975)

[IMidacUnitConverter 28](#_Toc456949976)

[IMidacMsDetailsSpec 30](#_Toc456949977)

# Overview

This release of the data access components includes two different primary components that are used for reading content from different types of Agilent Mass Spectrometry data files. The two share a number of lower-level components, but each exposes a different interface designed for accessing a particular type of data file.

For current purposes, Agilent MassHunter data files fall into two categories: IM-MS (Ion Mobility – Mass Spectrometry) data files, and “everything else” (or simply “MS” data files). MS data files can be produced by the full range of mass spectrometric instruments (quadrupole, triple-quadrupole, TOF, Q-TOF). IM-MS files are produced only by the 6560A Ion Mobility Q-TOF instrument when it is operating in the “ion mobility mode”.

IM-MS data includes an extra dimension of ion separation based on differences in the mobility of ions through a low-pressure background gas under the influence of a low, constant electric field. Because of this extra dimension, the format in which the data is stored (and the parameters required to access that data) have some significant differences from the standard MassHunter MS data files. The essentials needed to understand the MS and IM-MS data formats are outlined in [Data File Contents](#_Data_File_Contents).

**MHDAC** is an acronym for MassHunter Data Access Component. This is a software component that provides licensed users with direct programmatic access to the content of MassHunter data files. Its interfaces were developed prior to the advent of the 6560A, and thus it primarily supports reading data in the original MS data format.

It is provided in both 32-bit and 64-bit versions, and supports both COM and .NET (4.0) interfaces. The version included in this distribution offers very limited support for files in the IM-MS format. It can open the data files, and access the file’s Total Frame Spectra (TFS, see [IM-MS MassHunter Format](#_IM-MS_MassHunter_Format)). However, it is “blind” to the drift time dimension and cannot extract data over specified drift time ranges. We do not currently plan on extending it to provide full support for the new format.

**MIDAC** (an acronym for MassHunter IM-MS Data Access Component) is a new software component that is tailored to reading the content of IM-MS data files. MIDAC currently supports only .NET (4.0) interfaces. It is intended primarily for 64-bit systems (though it can also be used on 32-bit systems). The initial version supports reading *only* IM-MS format files, though support for original format files may be added in the future.

MIDAC is not yet complete, and has only that functionality that has been requested by our early development partners. See [MIDAC Status](#_MIDAC_Status) for a list of what functionality is currently implemented (and not implemented) in MIDAC.

This distribution includes

* Both MHDAC and MIDAC components, both covered by the same license,
* A number of lower-level components (DLL’s) that the two share,
* Some example data files (in both MS and IM-MS formats) and,
* Three simple example projects that illustrate the basic use of the components (a COM MHDAC example, a .NET MHDAC example, and a .NET MIDAC example).

The contents of this distribution are described in more detail in [The MHDAC\_MIDAC Distribution](#_The_MHDAC_MIDAC_Distribution), along with some tips on how to build the examples and how to build the appropriate component(s) into your application

The interfaces of the older MHDAC component are described in a separate document, [MassHunter Data Access Component.doc](MassHunter%20Data%20Access%20Component.doc). The remainder of this document concentrates on MIDAC. See [Using MIDAC](#_Using_MIDAC) for quick tips on how to open a data file and on which interfaces are supplied for various tasks. See [MIDAC Interfaces](#_MIDAC_Interfaces) for details on all interfaces

## Data File Contents

### Original MassHunter Format

The following shows schematically the organization of the portion of an original format MassHunter data file that holds the mass spectra. The data in a single file corresponds to the set of mass spectra acquired at different acquisition times. (In chromatographic experiments, “acquisition time” is a synonym for “retention time”). For historical reasons, the mass spectra are called “**MS scans**”, even when (as in TOF) nothing is actually “scanned” in making the measurement.



Each mass spectrum (“scan”) may be written to the data file as “**profile spectra**” or “**peak spectra**” (or both). A profile spectrum has multiple abundance measurements made over the width of a single ion peak. A peak spectrum is derived by centroiding or otherwise peak-detecting a profile spectrum and retaining just a single m/z and abundance value for each ion peak. Peak spectra take less disc space and are faster to access, but contain less complete information that may sometimes be needed. The user decides when setting up the data acquisition method which format(s) to store.



Profile Format Peak Format

While each scan may be stored twice (once in each format), the metadata describing that scan is written only once. This contains information like the ion polarity and (for MS/MS product ion spectra) the collision energy the m/z of the precursor ion.

The data acquisition method defines one or more **time segments**, which may be used to bound acquisition time ranges during which data acquisition parameters are changed.

So, briefly:

* Each file has a one or more time segment records, each of which includes the acquisition time range of the time segment and how many scans were acquired during that time segment.
* For each scan, there is an MS scan metadata record that stores the retention time and other information about that scan (including the location of the actual spectral data and the ID of the time segment it belongs to).
* For each scan, the spectrum data is recorded in one or both of two files reserved for that purpose (one for each format).

### IM-MS MassHunter Format

In the case of IM-MS the situation is different because of the added separation dimension (drift time). The drift time range of a complete experiment is on the order of 40-60 milliseconds, so that a complete drift separation can be done in the time that (without IM) one might spend accumulating data for a single MS scan.



Data is still acquired as a function of acquisition (retention) time. But at each recorded acquisition time, we do not have a single *scan,* but rather a single ***frame***.

A “frame” includes

* N x Frame (Mass) Spectra  
  A set of mass spectra, one for each of the N discrete ion mobility drift times recorded during the experiment. These spectra are called “frame spectra” or sometimes “Frame MS” to be clear these are not ion mobility drift spectra from the frame.   
  There are typically about 500 spectra for a drift time range of 60 milliseconds, though both the drift time range and the drift time spacing of adjacent frame spectra can vary depending on the data acquisition parameters.  
  Frame spectra are always stored in profile format only.
* 1 x Total Frame Spectrum (TFS)  
  The TFS is an additional mass spectrum that represents the sum of all frame spectra in that frame. This is, in effect, the spectrum that the TOF would have measured had the ion mobility drift separation not been done.   
  The TFS is stored in centroid and/or profile format (controlled by the acquisition method set up by the user).

So, briefly:

* Each file has a one or more time segment records, each of which includes the acquisition time range of the time segment. Unlike the original format, this records how many *frames* were acquired in the time segment rather than how many *scans*.
* For each acquired frame, there is an IM-MS Frame metadata record. This stores the acquisition time (RT) of that frame as well as other information that applies to the frame as a whole (e.g., the ID of the time segment the frame belongs to and fragmentation information if post-drift ion fragmentation is being done in that frame).
* For each mass spectral “scan”, there is (as in the original format) an MS scan metadata record. However, this does not include an acquisition time, but rather a flag that indicates if the scan represents a TFS or a Frame MS, and in the latter case the drift time of the scan. Like the original format, this record also contains information on where the actual spectral data is located in the peak and/or profile spectrum files.

## The MHDAC\_MIDAC Distribution

### Contents

This distribution is supplied as a zip file that can be unzipped to get the MHDAC\_MIDAC\_Package folder that has the following subfolders:

* Documents  
  The location of this and related documents
* ExampleData  
  Sample data files useful for testing the components and the code you write to use them. Those whose names begin “ImsSynth…” are synthetic IM-MS format data files; the remainder have the the original MS format. Note that ImsSynth\_Chrom.d has been prepared to support conversions between drift times and collision cross section (CCS) values, as described in [File Metadata Access](#_File_Metadata_Access) (HasSingleFieldCcsInformation, CcsFromDriftTime, etc).
* TestApps  
  Three projects that illustrate the use of the components.
  + DataReaderCPlusClient – illustrates COM / C++ use of MHDAC
  + DataReaderCSharpClient – illustrates .NET / C# use of MHDAC
  + MidacAppExample – illustrates .NET / C# use of MIDAC
* MHDAC\_MIDAC\_64bit/bin  
  This folder contains a 64-bit versions of all of the files needed by MHDAC or MIDAC, plus a few extra files used by the examples in TestApps. Depending on whether your application needs to read MS or IM-MS files (or both), you will want to package different subsets of these files with that application. Which files are required for various purposes is documented in [Which Files to Include](#_Which_Files_to)
* MHDAC\_32bit/bin  
  This folder contains a subset MHDAC\_MIDAC DLLs that comprise the 32-bit MHDAC component. Note that most of the DLLS here are built as “AnyCPU” and are not actually specific to 32- or 64-bit systems. Only BaseTof.dll (which is a C++ library that can’t be built AnyCPU) is actually different and specifically compiled as a 32-bit library (or 64-bit in the MHDAC\_MIDAC folder).

### Building with MHDAC and MIDAC

The MHDAC and MIDAC “components” are actually small sets of DLLs. For instance, MIDAC includes not only MIDAC.dll, but also several others (BaseError, BaseCommon, BaseTof, …) as documented in [Which Files to Include](#_Which_Files_to). Similarly, the MHDAC component includes MassSpecDataReader.dll and several other libraries.

Code that uses MIDAC will need to have a reference to at least MIDAC.dll, and will need to include all of these DLLs in the same location as its executable(s). Code that uses MHDAC will need a reference to MassSpecDataReader.dll, and include the other relevant DLLs.

All of the types (enumerations, classes and interfaces) that make up the public MIDAC interfaces are located in the namespace Agilent.MassSpectrometry.MIDAC; this should be the only namespace you need. (MHDAC uses other namespaces; see [MassHunter Data Access Component.doc](MassHunter%20Data%20Access%20Component.doc)).

### Example Code

The easiest way to build and run the examples is to copy the entire MHDAC\_MIDAC\_Package folder into a folder mounted as drive letter X. (The source code and projects in the examples assume binaries and data files are located at X:\MHDAC\_MIDAC\_Package\...) You can use some other location, but you’ll have to modify paths for DLL references and data files to make it work elsewhere.

***If you intend to use the COM interfaces*** to access MHDAC, then the dlls must be registered. The bin folders include batch files (e.g., RegisterMassHunterDataAccess.bat or RegisterMassHunterDataAccess64.bat) you can use to do that.

The MIDAC example (in the TestApps/MidacAppExample folder) builds MidacApp.exe. This is a small application that we have used for profiling the execution time of various MIDAC calls, but it may also be useful to you as an example of those calls, including drift time 🡨 🡪 CCS conversions.

The MHDAC examples (in the other TestApps folders) each make a console application that, when run, opens several data files in series and extracts data from each. The data is simply printed in the console window.

### Which Files to Include

This table summarizes which of the files (in the MHDAC\_MIDAC\_64bit folder) you will need for an application, depending on whether that application is intended to use MHDAC or MIDAC or both. If your application uses only MHDAC, you also have the option of using the .NET or COM interfaces.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| If you use ----> | MHDAC COM | MHDAC .NET | MIDAC | MHDAC.NET and MIDAC |
| MIDAC.dll |  |  | x | x |
| BaseCommon.dll | x | x | x | x |
| BaseDataAccess.dll | x | x | x | x |
| BaseError.dll | x | x | x | x |
| BaseTof.dll | x | x | x | x |
| BaseDataAccess.dll.config | x | x |  | x |
| MassSpecDataReader.dll | x | x |  | x |
| BaseCommon.tlb | x |  |  |  |
| BaseDataAccess.tlb | x |  |  |  |
| MassSpecDataReader.tlb | x |  |  |  |
| RegisterMassHunterDataAccess[64].bat | x |  |  |  |
| UnregisterMassHunterDataAccess[64].bat | x |  |  |  |

Additional files (included in MHDAC\_MIDAC\_64bit/bin) that are used in the example applications are listed below. These should *not* be packaged as part of your application; they are included to make debugging and intellisense work better for you when “playing” with the examples.

|  |  |  |
| --- | --- | --- |
| Example Application ----> | MhdacApp | MidacApp |
| BaseCommon.pdb | x | x |
| BaseDataAccess.pdb | x | x |
| BaseError.pdb | x | x |
| BaseTof.pdb | x | x |
| MassSpecDataReader.pdb | x | x |
| AgtFileSelectionDialog.dll |  | x |
| AgtFileSelectionDialog.pdb |  | x |
| agtsampleinforw.dll |  | x |
| BaseCommon.XML |  | x |
| BaseDataAccess.XML |  | x |
| Interop.shell32.dll |  | x |
| MIDAC.pdb |  | x |
| MIDAC.xml |  | x |

# Using MIDAC

## Opening a Data File

All access to MassHunter data files with MIDAC begins with the static class [MidacFileAccess](#_MidacFileAccess). This has a function you can use to determine whether a data file is an IM-MS MassHunter data file and functions that can open a data file and return an interface to the reader class appropriate for that data file.

MIDAC has one reader class whose main interface, [IMidacImsReader](#_IMidacImsReader), provides the functions needed to access IM-MS data files, and another whose interface, IMidacMsReader, (not yet implemented) will be appropriate for original format files.

You cannot directly instantiate either reader. Rather, use one of the MidacFileAccess functions to obtain the reader (and open the file). Note that the FileHasImsData() function will tell you, without opening a file, whether or not that file is an IM-MS data file.

Current Recommendation

Because IMidacMsReader is not yet implemented, you should first check whether the file is an IM-MS format file. If so, you can get the IMidacImsReader for that file. If not, your only current recourse is to use the original MHDAC to open the file.

## Reading Chromatograms

Not yet implemented.

## Reading Mass Spectra

When reading mass spectra, the first decision is what kind of spectra are needed – frame spectra or total frame spectra. The answer will be frame spectra if you need data from a limited drift time range; otherwise the total frame spectra are probably appropriate.

There are different functions for reading each type of spectrum because

* Only for Frame Spectra is a drift time value or range relevant, and
* Only for Total Frame Spectra is there a choice of reading peak or profile format data.

In both cases, the question of “which frames’ data should be read?” is relevant, especially if the file contains data acquired in different ways. As an example, consider a file acquired in the “All Ions” data mode. This will have frames representing low fragmentation data interleaved with frames in which all ions leaving the drift tube are nonselectively fragmented. There are several approaches you can take, depending on what you want to do with the data.

* Read the spectrum for each frame in the file sequentially from 1 to the number of frames in the file. Examine the metadata attached to the returned spectrum ([IMidacSpecData](#_IMidacSpecData,)) and decide if it is a spectrum you’re interested in, and proceed . . .
* Create a set of MS spectrum filters (see [IMidacSpecFiltersMs](#_IMidacSpecFiltersMs)) that specify the conditions you want the spectra to meet. Note that these filters can include an acquisition (RT) range as well as ion polarity, fragmentation class, etc.   
  Call IMidacImsReader.FilteredFrameNumbers to get the collection of frame numbers whose spectra meet those conditions, and then read the spectra just from those frame numbers.
* (For summing or averaging the qualified frames) Call the IMidacImsReader function that produces the type of spectrum you want (TFS or frame) and has an IMidacSpecFiltersMs parameter.

## MIDAC Status

#### Implemented

* Function to test for whether a file is IM-MS or the original format
* A reader class for IM-MS files, including
  + Access functions for whole-file, per-frame and sum-of-frames metadata
  + Access functions for *profile* Total Frame Spectra
  + Access functions for drift resolved Frame Spectra (always profile)

#### Not Yet Implemented

* A reader class for original (not IM-MS) format files
* IM-MS access functions for *peak* format Total Frame Spectra
* IM-MS access functions for Chromatograms
* Access to non-MS data (e.g., from GC or LC detectors and devices) contained within the IM-MS data file.

# MIDAC Interfaces

## MidacFileAccess

This is a static class, not an interface. Functions include

public static bool FileHasImsData(string filePath)

Checks (without opening the file) whether a specified file is a MassHunter IM-MS data

file.   
"filePath" - the full path name of the file, e.g., C:\My\Path\stuff.d

public static IMidacImsReader ImsDataReader(string dataFilePath)

Returns a reader capable of accessing data from a MassHunter IM-MS data file, already opened to that file.

public static IMidacMsReader MsDataReader(string dataFilePath)

(not yet implemented; returns null)  
Returns a reader capable of accessing data from a MassHunter traditional MS data file, already opened to that file.

public static IMidacMsFiltersSpecs DefaultMsSpectrumFilters

Instantiate and return spectrum data filters with default values (no filtering).

public static IMidacMsFiltersChrom DefaultMsChromFilters

Instantiate and return chromatogram data filters with default values (unfiltered TIC).

## IMidacImsReader

This interface provides access to data in IM-MS format files. Its functions (below) are categorized according to the type of information they return.

### File Management

You can’t directly instantiate a file reader (use MidacFileAccess for that purpose), but given an active instance, you *can* close the currently-open file and open a new one.

void Close();

Terminate access to the current IMS data file and clear all contents of the reader.

void OpenFile(string dataFilePath);

Open the specified data file after first closing any currently-open file.

### File Metadata Access

These IMidacImsReader functions provide access to information about the file as a whole.

These are all relatively quick, as they don’t require accessing the spectrum records.

IMidacFileInfo FileInfo { get; }

Get overall file metadata

IMidacUnitConverter FileUnitConverter { get; }

Get a unit converter that bases its conversions on values averaged over the entire data file. This can be used to convert acquisition times between units of (minutes, seconds, frame number); to convert drift times between units of (milliseconds, drift bin); and to convert mass values between units of (m/z, flight time in nanoseconds).

int[] FilteredFrameNumbers(IMidacMsFiltersSpec filters);

Get a collection of all frame numbers for which the mass spectra meet specified filter criteria. If “filters” is null or if filters.AppliedFilters is Unspecified, returns a list of all frame numbers in the file. Note: Use MidacFileAccess to instantiate new filter instances.

int[] FilteredFrameNumbers(IMidacMsFiltersChrom filters);

Get a collection of all frame numbers for which the data meet specified filter criteria. If “filters” is null or if filters.AppliedFilters is Unspecified, returns a list of all frame numbers in the file. Note: Use MidacFileAccess to instantiate new filter instances.

double[] FilteredAcqTimes(IMidacMsFiltersSpec filters);

Similar to FilteredFrameNumbers, except it returns an array of acquisition (retention) times with values in minutes.

bool HasPeakTfsSpectra { get; }

Returns true if this file has any Total Frame Spectra stored in peak format.

bool HasProfileTfsSpectra { get; }

Returns true if this file has any Total Frame Spectra stored in profile format.

bool HasSingleFieldCcsInformation { get; }

Returns true if this file records the data required to convert between Collision Cross Section (CCS) and drift time values.

double CcsFromDriftTime(double mz, int z, double dtMilliseconds);

Convert the specified drift time to CCS for an ion with the specified m/z and charge. Returns NaN if required information is missing.

double DriftTimeFromCcs(double mz, int z, double ccs);

Convert the specified CCS to drift time (in milliseconds) for an ion with the specified m/z and charge. Returns NaN if required information is missing.

### Frame Metadata Access

These IMidacImsReader functions return information that applies to all of the data in a specific frame. The second function in each set returns information about the frame that would result if you were to add all of the frames whose frame numbers are specified (though that frame addition is not actually performed).

These are relatively quick because they don’t read spectrum records.

Notes:

* frame number values are 1, 2, … N, where N is the number of frames in the file.
* drift bin index values are 0, 1, … (N-1), where N is the number of drift bins in the frame

IMidacFrameInfo FrameInfo(int frameNumber);

IMidacFrameInfo FrameInfo(int[] frameNumbers);

Get frame-specific metadata for a specific frame (or summed frame). Note that the abundance range in the returned IMidacFrameInfo will be empty because this function doesn’t access spectrum records.

short[] NonEmptyDriftBins(int frameNumber);

short[] NonEmptyDriftBins(int[] frameNumbers);

Get a list of the drift bin indices for which a specific frame (or summed frame) has mass spectra with non-zero abundance. For summed frames, this includes all drift bins that have non-zero abundance in any of the frames contributing to the sum.

IMidacUnitConverter FrameUnitConverter(int frameNumber);

IMidacUnitConverter FrameUnitConverter(int[] frameNumbers);

Get a unit converter that bases its conversions on values from a specific frame (or array of frames). This differs from the FileUnitConverter only if per-frame internal reference mass correction has been done for this file. In that case, this gets the mass calibration for the specified frame (or average for the array of frames), while FileUnitConverter uses a whole-file average calibration.

### MidacSpecFormat

Functions that return a profile mass spectrum (see sections [Profile Total Frame Spectra](#_Profile_Total_Frame) and [Frame Spectra](#_Frame_Spectra), below) take a MidacSpecFormat parameter. This is an enumeration whose value controls the format in which the spectrum is returned from these functions. The spectrum is always a profile spectrum; the difference lies in how and whether zero-abundance values are included.

Spectra are stored in the data file in a compressed format without zero-abundance values. The performance of reading those spectra is considerably better if the “missing” points don’t have to be restored – but then your code must be prepared to deal with spectra that have “gaps”.

The allowed values of MidacSpecFormat are

Profile A full spectrum containing all points, even those with zero abundance.  
This takes the most memory and most time.

ZeroTrimmed A spectrum that does not include any zero-abundance points  
Takes the least memory and time, but directly plotting this spectrum with lines connecting successive values will produce strange results.

ZeroBounded A spectrum without zero-abundance points, except that a single zero-abundance point *is* included at each end of a region containing non-zero points.   
Nearly as efficient as ZeroTrimmed, with the advantage that plotting the spectrum produces better results.

For example, consider a region of a spectrum in which successive Y-values are as follows (there is always a corresponding array of X values):

10 5 1 0 0 0 0 0 1 5 10 5 1 0 10 12

When returned in the various MidacSpecFormats, the result Y arrays will be as follows (where “.” indicates a missing point; X arrays will naturally also be missing those same points):

Profile 10 5 1 0 0 0 0 0 1 5 10 5 1 0 10 12

ZeroTrimmed 10 5 1 . . . . . 1 5 10 5 1 . 10 12

ZeroBounded 10 5 1 0 . . . 0 1 5 10 5 1 0 10 12

### Profile Total Frame Spectra

These IMidacImsReader functions read the TFS for individual frames, or compute the sum or average of the TFS for multiple frames. Because a TFS represents the sum of all frame spectra over the entire drift time range of the frame, these functions to not have any drift time parameter.

You can only get a profile TFS if that format is stored in the file. Unlike MHDAC, there is no “desired format” parameter like “PeakElseProfile” or “ProfileElsePeak”. Instead, there are IMidacImsReader functions (HasPeakTfsSpectra and HasProfileTfsSpectra) that make it easy to find out what format(s) are present in the file, and then you can ask for what you want.

See [MidacSpecFormat](#_MidacSpecFormat) for the interpretation of the specFormat parameter.

IMidacSpecDataMs ProfileTotalFrameMs(MidacSpecFormat specFormat,

int frameNumber);

Get the profile TFS from a specified frame and return it in the specified format.

IMidacSpecDataMs[] ProfileTotalFrameMs(MidacSpecFormat specFormat,

int[] frameNumbers,

bool addSpectra);

Get a combined profile TFS from an array of specified frames and return it in the specified format. The spectra are added if “addSpectra” is true, else they are averaged. The returned array of spectra will be null if the file doesn’t have profile TFS, or if none of the specified frames exist in the file. Else, it will have the desired spectrum or multiple spectra if the specified frames have spectra of types that cannot be mixed. Examples of "non-mixable" spectra include those with different ion polarities, different MS levels or spectra from different scan segments. However, data acquisition in IM-MS mode does not yet support multiple ion polarities or multiple scan segments, so mixed MSLevels is currently the only practical example of this.

Note that spectra with different fragmentation energies ARE combined.

IMidacSpecDataMs[] ProfileTotalFrameMs(MidacSpecFormat specFormat,

IMidacSpecFiltersMs filters,

bool addSpectra);

Get a combined profile TFS from the frames meeting specified filter criteria and return it in the specified format. The spectra are added if “addSpectra” is true, else they are averaged. As above, the returned array has multiple spectra only if the filters allow frames with spectra that cannot be mixed.   
Note: Use MidacFileAccess to instantiate new filter instances

Note: this same operation can be done by calling FilteredFrameNumbers (with

MsStorageType == ProfileSpectra), and passing the frame number array to the ProfileTotalFrameMs function described just above.

### Peak Total Frame Spectra

The IMidacImsReader interface currently defines two functions intended to return a peak TFS (both named PeakDetectedFrameMs, with different parameters).

Neither of these functions is implemented yet, and their signature is subject to change.

IMidacSpecData PeakDetectedTotalFrameMs(int frameNumber,

IMidacPeakFilters pkFilters,

bool convertFromProfileIfNecessary);

IMidacSpecData[] PeakDetectedTotalFrameMs(int[] frameNumbers,

bool addSpectra,

IMidacPeakFilters pkFilters,

bool convertFromProfileIfNecessary);

### Frame Spectra

IMidacImsReader functions for reading spectra from specific acquisition times (RT) and drift times. These are always profile spectra.

Note:

* Frame number values are 1, 2, … N, where N is the number of frames in the file.
* Drift bin index values are 0, 1, … (N-1), where N is the number of drift bins in the frame
* A drift range can have units of Milliseconds or DriftBinIndex (the latter is usually faster)
* See [MidacSpecFormat](#_MidacSpecFormat) for the interpretation of the specFormat parameter.

IMidacSpecDataMs FrameMs(int frameNumber,

int driftIdx,

MidacSpecFormat specFormat,

bool useAvgCal);

Read the drift-resolved mass spectrum at a specified drift bin in a specified frame and return in the specified format. If “useAvgCal” is true, then the average mass calibration for the file is used to compute m/z values from TOF flight times. Otherwise, if there is an internal reference mass calibration for the frame that calibration is used. The latter is slower because the m/z arrays need to be recomputed more often, but may be necessary if the values must be as accurate as possible.

Note: performance for accessing many spectra is best if multiple spectra from the same

frame are read in successive calls before moving on to another frame.

void FrameMs(int frameNumber,

int driftIdx,

MidacSpecFormat specFormat,

bool useAvgCal,

ref IMidacSpecDataMs specData);

As above, but rather than creating a new spectrum object, the specified “specData” is modified to hold the new results. This tends to run significantly faster because it can frequently avoid allocating new (possibly large) blocks of memory.

IMidacSpecDataMs FrameMs(int frameNumber,

IDoubleRange driftRange,

MidacSpecFormat specFormat,

bool useAvgCal,

AbundanceMeasure driftCombine);

Get a single mass spectrum that represents the sum or average of the spectra falling within a specified drift range in a specified frame and return in the specified format. If “useAvgCal” is true, then the average mass calibration for the file is used to compute m/z values from TOF flight times. Otherwise, if there is an internal reference mass calibration for the frame that calibration is used. The latter is slower because the m/z arrays need to be recomputed more often, but may be necessary if the values must be as accurate as possible.

The “driftCombine” parameter specifies how spectra at different drift times are combined; a value of Sum or Average is expected.

Note: performance for accessing many spectra is best if multiple spectra from the same frame are read before proceeding on to a different frame.

void FrameMs(int frameNumber,

IDoubleRange driftRange,

MidacSpecFormat specFormat,

bool useAvgCal,

AbundanceMeasure driftCombine,

ref IMidacSpecDataMs specData);

As above, but replace the data in the specified mass spectrum with that of the sum or average of the spectra falling within a specified drift range in a specified frame. Tends to be faster because it can frequently avoid allocating new (potentially large) blocks of memory.

IMidacSpecDataMs FrameMs(int[] frameNumbers,

IDoubleRange driftRange,

MidacSpecFormat specFormat,

bool useAvgCal,

AbundanceMeasure driftCombine,

AbundanceMeasure frameCombine);

Get a single mass spectrum that represents the sum or average of the spectra

falling within a specified drift range from specified frames and return in the specified format. This is similar to the other functions except that it combines data from multiple frames as well as multiple drift times. The “frameCombine” parameter dictates how spectra from different frames are combined (Sum or Average expected) while “driftCombine” dictates how spectra from different drift times are combined.

### Chromatograms

IMidacImsReader interfaces for chromatogram extraction are not yet designed. The following are for prototype timing tests only (though they do work!).

IMidacChromDataMs Chromatogram(ChromatogramType chromType,

bool doCycleSum,

IMidacMsFiltersChrom filters);

Barely functional (e.g., chromType must be TIC);

float[][] ChromYArrays(IDoubleRange[] mzRanges);

Return the Y-Arrays for a collection of EICs, each extracted over a specified m/z range. Intended only for rough timing comparison with MHDAC, this uses Total Frame Spectra (there's no parameter for drift time)

## IMidacMsFiltersSpec

This is a collection of values on which mass spectra may be filtered during mass spectrum or chromatogram extraction. One property (ApplicableFilters) is a [Flags] enumeration for which one bit is set for every data attribute on which filtering is to be done. Only if the corresponding bit is set does the corresponding field’s value need to be specified.

This design supports filter attributes that are not applicable to IM-MS data files, but only to some original format files. Conversely, some attributes will only be found in IM-MS files.

ApplicableFilters ApplicableFilters { get; set; }

The [Flags] enumeration that specifies which of the other properties in this interface should actually be used to filter data. Any property whose corresponding bit in this enumeration is not set is not used as a filter.

IDoubleRange CompensationFieldRange { get; set; }

(FAIMS only) Compensation field value

IDoubleRange DispersionFieldRange { get; set; }

(FAIMS only) Compensation field value

IDoubleRange[] DriftTimeRanges { get; set; }

(IM-MS only) The drift time range(s) over which data should be extracted. A zero-width range implies using the single drift time value closest to the specified range. A null or empty range results in an exception if the corresponding bit is set in ApplicableFilters.

FragmentationClass FragmentationClass { get; set; }

(IM-MS only) The fragmentation class (LowEnergy, HighEnergy or both) to which a spectrum

must belong if its FragmentationOpMode includes HiLowFrag

IDoubleRange FragmentationEnergyRange { get; set; }

The range into which a spectrum's fragmentation (collision) energy must fall for it to be included. Ignored for unfragmented spectra (i.e., MS(1) spectra having FragmentationOpMode == None). A null or empty range results in an exception if the corresponding bit is set in ApplicableFilters.

FragmentationOpMode FragmentationOpMode { get; set; }

(IM-MS only) A [Flags] enumeration specifying fragmentation mode(s) that make a spectrum

eligible for inclusion. (None, NonSelective, Selective, HiLowFrag, ...)

IDoubleRange FragmentorVoltageRange { get; set; }

(Original format only) For mass spectrometers with fragmentor voltages, the range of fragmentor voltages into which a spectrum's value must fall for it to be included. A null or empty range results in an exception if the corresponding bit is set in ApplicableFilters.

Ionization IonizationMode { get; set; }

Type of ionization (ESI, APCI, EI, ...)

Polarity IonPolarity { get; set; }

Ion polarities of MS scans to be extracted. A value of "Mixed" may result in returning two spectra if both MS and MS/MS spectra meet other filter criteria. Note that current IM-MS data files always have data of a single polarity, though non-IM data files may have mixed polarity data.

MsLevel MsLevel { get; set; }

The MS level(s) of spectra to be extracted. A value of "Mixed" may result in returning two spectra if both MS and MS/MS spectra meet other filter criteria.

MsScanType MsScanType { get; set; }

The type(s) of MS scans to be extracted. For [Q]TOF and IM-MS data, only Scan and ProductIonScan make any sense; for QQQ data many more types are used. A value of "All" may result in returning two spectra if both MS and MS/MS spectra meet other filter criteria.

MsStorageMode MsStorageMode { get; set; }

Mass spectrum storage mode. For IM-MS data, this is ignored except Total Frame Spectra, which are the only spectra that can have PeakDetected spectra. A value of "Mixed" or "Unspecified" results in an exception if the corresponding bit is set in ApplicableFilters.

IDoubleRange[] MzOfInterestRanges { get; set; }

The range into which an MS/MS spectrum's m/z of interest must fall for it to be included (this is the quadrupole’s isolation m/z for product ion scans). Ignored for MS(1) spectra. Zero-width ranges require an "exact" match (within a floating point epsilon). At least one range must be specified if the corresponding bit is set in ApplicableFilters

new IMidacSpecFiltersMs Clone();

Make a deep copy of this object

## IMidacSpecData,

IMidacSpecData is the base interface from which IMidacSpecDataMs is derived (along with other interfaces in the future, such as for UV-Vis spectra). These are roughly analogous to the IBdaSpecData interface used in MHDAC.

IDoubleRange[] AcquiredTimeRanges { get; }

Get the collection of time ranges over which spectra were summed or averaged to create this object. If SpectrumCount == 1, this will be a single, zero-width range.

IMidacDeviceInfo DeviceInfo { get; }

Get information about the device that produced the data in this object

bool IsAverage {get;}

True if this spectrum is the average of two or more individual spectra. If false but SpectrumCount > 1, then this is the sum of individual spectra rather than an average.

int MaxProfilePoints {get;}

For profile spectra, the number of points the spectrum would have if all zero-abundance points were included. Zero for peak spectra.

int NonZeroPoints {get;}

The number of points with non-zero abundance in this spectrum (applies to both profile and peak spectra).

int SpectrumCount {get;}

The number of individual measured spectra that are included in this one (if an average or sum). (Fulfills the same role as IBDASpecData.TotalScanCount)

MidacSpecFormat SpectrumFormat { get; }

What is the format in which the (x,y) data are stored within this object?

MidacSpecType SpectrumType { get; }

What kind of spectrum does this data represent?

double[] XArray { get; }

Get the array of x values

double XSamplingPeriod { get; }

If the spectrum has (or is derived from) equally-spaced data points, the spacing of points. This is in units of nanometers for UV-Vis spectra; m/z for Q or QQQ mass spectra,

nanoseconds for [Q]TOF mass spectra, and milliseconds for drift spectra.

MidacUnits XUnit { get; }

Get the units of the x values (fulfills the function of IBdaSpecData.IsDataInMassUnit).

float[] YArray { get; }

Get the array of y values

## IMidacSpecDataMs

This is an extension of IMidacSpecData that provides access to mass spectrum specific information.

double AbundanceLimit { get; }

For [Q]TOF spectra, the abundance value that is the largest value that could be seen in this spectrum (the "fully saturated" value).

IDoubleRange CompensationFieldRange { get; }

For FAIMS instruments (there are none yet), the range of compensation field values spanned by the spectra contributing to this average or summed spectrum. Null for non-FAIMS instruments

IDoubleRange DispersionFieldRange { get; }

For FAIMS instruments (there are none yet), the range of dispersion field values spanned by the spectra contributing to this average or summed spectrum. Null for non-FAIMS instruments

IDoubleRange[] DriftTimeRanges { get; }

For IM-MS instruments, the range(s) of ion drift times spanned by the spectra contributing to this average or summed spectrum. Null for other instruments or for IM-MS instruments operating in Q-TOF (non-mobility) mode.

FragmentationClass FragmentationClass { get; }

For High/Low fragmentation experiments, which class this spectrum belongs. Unspecified for other experiment types.

IDoubleRange FragmentationEnergyRange { get; }

If fragmentation was done, the range of fragmentation (collision) energies applied to the spectra contributing to this average or summed spectrum. Null if no fragmentation was done. IsEmpty == true if no fragmentation was done. IsZeroWidth == true if all contributing spectra had the same value.

Note that zero is a legitimate value for fragmentation energy and that FragmentorVoltageRange is a separate property.

FragmentationOpMode FragmentationOpMode { get; }

A [Flags] enumeration that specifies whether (intentional) fragmentation was done while measuring the data included in this spectrum. If so, specifies whether that fragmentation was selective (using quadrupole m/z isolation) or not, and whether the spectrum was acquired as a part of a hi/low fragmentation experiment.

IDoubleRange FragmentorVoltageRange { get; }

For instruments with a pre-Quadrupole Fragmentor Voltage setting, the range of fragmentor voltages of all spectra contributing to the sum (or average). IsEmpty == true if there is no fragmentor. IsZeroWidth == true if all contributing spectra had the same value.

Note that zero is a legitimate (if unlikely) fragmentor voltage.

Ionization IonizationMode { get; }

Get the ionization technique that was used

Polarity IonPolarity { get; }

Get the polarity of the ions in the spectrum

MsLevel MsLevel { get; }

MS for single-stage spectra (including nonselective All-Ions spectra), MSMS for two-stage spectra (including wide-band isolation type spectra).

MsScanType MsScanType { get; }

Get the MS scan type. All-Ions spectra (non-selective fragmentation) are regarded as

MS(1) spectra of type Scan. Wideband isolation experiments would be recorded as MS/MS

ProductIon spectra with a non-zero-width m/z of interest range.

MsStorageMode MsStorageMode { get; }

Whether the data in this object represents a profile spectrum or a peak-detected (centroided) spectrum.

IDoubleRange[] MzOfInterestRanges { get; }

For MS/MS spectra, the set of unique "m/z of interest" values for the individual spectra contributing to this average or summed spectrum. Null for MS(1) spectra. For an IDoubleCwRange, the Center corresponds to the designated m/z (e.g., 235.0) while the range Min/Max correspond to the span of the quadrupole isolation window (e.g., 235.0 -0.3 amu + 1.7 amu). May also be a Min/Max range (zero width to report just a single value).

The interpretation of the m/z of interest varies depending on the MsScanType (precursor m/z for product ion scan or MRM transision, neutral loss for neutral loss scan, product m/z for precursor ion scan, etc.)

int[] ParentScanIdArray { get; }

For Auto-MS/MS or Triggered MRM spectra, the ScanId(s) of the spectra that triggered the acquisition of this spectrum. Null in other cases.

int ScanId { get; }

For single (non-summed and non-averaged) spectra, an integer identifier for the spectrum that is unique within the data file. Zero for any "combined" spectrum.

## IMidacFileInfo

File-level metadata for a MassHunter IM-MS data file

DateTime AcquisitionDate { get; }

The date/time that the file was acquired

Bool AcquisitionIsComplete { get; }

Flag that indicates whether acquisition of the data is complete. If not, data acquisition may yet write more data into it.

string AcquisitionSoftwareVersion { get; }

Version information about the software used to aquire the data file

string DriftGas { get; }

The forumula of the drift gas used during this run. (Could be “Nitrogen” instead of “N2” for older data files)

bool FileHasTrapPressure { get; }

Does the file have the trap pressure recorded?

string FilePath { get; }

The file path, e.g., “X:/Some/Folder/MyFile.d”

IMidacUnitConverter FileUnitConverter { get; }

Get a unit converter. This can accurately convert chromatographic units (minutes, seconds, frame number), and can convert mass units (m/z, flight time) and drift-spectrum units (milliseconds, drift bins) using file average values.

bool HasHiLoFragData { get; }

True if the data file has alternating high and low fragmentation frames (e.g., AllIons data)

bool HasTgtMsMsData { get; }

True if the file contains Targeted MS/MS data (in which the quadrupole is used to select the m/z of ions to be fragmented after the drift region)

string InstrumentName { get; }

The name of the instrument on which the data were acquired.

double MaxAcqTime { get; }

The acquisition (retention) time of the last frame in the data file, in minutes.

int MaxFlightTimeBin { get; }

The bin number (0-based) of the longest flight time possible for any mass spectrum in this file.

Note: MaxFlightTime (ns) = MaxFlightTimeBin \* TofBinWidth   
(TofBinWidth is available from IMidacUnitConverter or IMidacMsDetailsSpec)

int MaxNonTfsMsPerFrame { get; }

The largest number of mass spectra (each corresponding to one "drift bin") in any frame in the file. Does not count the TFS present in each frame. Does count any drift bin with an empty (all zero) mass spectrum.

int NumFrames { get; }

The number of frames in the file

Polarity Polarity { get; }

The ion polarity (or polarities, if mixed-polarity acquisition is ever supported) of data in the file.

IMidacMsDetailsSpec TfsMsDetails { get; }

A mass spectrum metadata summary for the entire file

IMidacFileInfo Clone();

Make a deep copy of this object

## IMidacFrameInfo

Frame-specific metadata applicable to an extracted frame (which may represent a single acquired frame or a sum of multiple acquired frames).

IDoubleRange AbundRange { get; }

The range of abundance values in the frame.

IDoubleRange AcqTimeRange { get; }

The range of acquisition times (e.g., retention times) represented by the frames that were combined to create the frame. In minutes.

IIntRange DriftBinRange { get; }

The range of drift bins in the frame. This is the same as DriftTimeRange except for units. Spans all acquired data, including bins with all-zero abundance points

double DriftField { get; }

The drift field during this frame (V/cm)

double DriftPressure { get; }

The drift region pressure measured when this frame was acquired.

double DriftTemperature { get; }

The drift region temperature measured when this frame was acquired.

IDoubleRange DriftTimeRange { get; }

The range of drift times in the frame. In milliseconds.

IMidacFileInfo FileInfo { get; }

A reference to overall information about the file from which this frame was extracted.

IDoubleRange FragmentationEnergyRange { get; }

The range of fragmentation energies used in this frame. Null if this FrameInfo represents an entire run rather than a specific [averaged] frame. Empty or 0:0 if no fragmentation is done.

FragmentationOpMode FragmentationOpMode { get; }

The fragmentation operation mode used to acquire the frame (or the modes, if a summed frame using different modes)

IIntRange FrameNumRange { get; }

The range of acquisition times (e.g., retention times) represented by the frames that were combined to create the frame. In units of frame number.

IMidacUnitConverter FrameUnitConverter { get; }

Get a unit converter. This can accurately convert mass units (m/z, flight time) and drift-spectrum unit s(milliseconds, drift bins) for the related frame, and chromatographic units (minutes, seconds, frame number) via a nested FileInfo unit converter.

IDoubleRange MzRange { get; }

The range of m/z values in the frame. May be in units of MassToCharge, Nanoseconds or FlightTimeBins. Spans all acquired data, including all-zero abundance points.

Int NumTransients { get; }

The number of TOF transients summed for each of the frame spectra in this frame.

IMidacMsDetailsSpec SpectrumDetails { get; }

Additional, more detailed metadata

Double Tic { get; }

Get the sum of ion abundance over both drift time and m/z for the entire frame.

short TimeSegmentId { get; }

The ID of the time segment from which this frame or frame sum was extracted; The value is negative if this data derives from multiple time segments.

IIntRange TofBinRange { get; }

The range of TOF flight time bin values for the frame. Spans all acquired data, including all-zero abundance points.

double TrapPressure { get; }

Get the pressure in the trap preceding the drift region

IMidacFrameInfo Clone()

Get a (SHALLOW) copy of this object. Doubles and ranges are cloned, references to larger objects are not.

## IMidacUnitConverter

Access to functions that may be used to intercovert values between the units supported by the IM-MS Browser.

IDoubleRange AcqTimeRange { get; }

The range of acquisition (retention) times of all frames in the file (always in minutes)

double DriftBinWidth { get; }

The drift bin width (in milliseconds) or NaN if there is no drift calibration set.

dDouble TofBinWidth { get; }

The value of the TOF bin width (in nanoseconds), or NaN if there is no TOF calibration set.

double TofMassCalA { get; }

The "A" coefficient of the traditional TOF mass calibration equation

double TofMassCalTo { get; }

The "T0" coefficient of the traditional TOF mass calibration equation

Bool Convert(MidacUnits newUnits,   
 ref IDoubleRange doubleRange);

Convert a specified doubleRange from its current units to specified new units. Returns true if the value was changed, false if it already had the desired units. Throws an exception if asked for an unsupported conversion (e.g., m/z to minutes).  
Note: IDoubleRange “knows” its current units.

Bool Convert(MidacUnits oldUnits,   
 MidacUnits newUnits,   
 double[] doubleArray);

Convert all values in a specified double array from specified old units to new units. Returns true if the values were changed, false if oldUnits == newUnits.

Bool Convert(MidacUnits oldUnits,   
 MidacUnits newUnits,   
 ref double value);

Convert a value from specified old units to new units. Returns true if the values were changed, false if oldUnits == newUnits.

Bool Equals(IMidacUnitConverter other);

Returns true if two converters are functionally equivalent (will return the same conversion values).

Bool SupportsConversion(MidacUnits oldUnits,   
 MidacUnits newUnits);

Returns true if oldUnits and newUnits are interconvertable (e.g., minutes and seconds) and false if not (e.g., drift time and m/z)

Void TofMassCalPolynomial(out double[] coefficients,   
 out double tMin,  
 out double tMax);

Get the TOF polynomial information. *Do NOT use these directly (call Convert functions instead) – there is additional information that is required to apply them to measured data*.

## IMidacMsDetailsSpec

Additional metadata about a mass spectrum.

Although most properties support both get and set, setting the values is not useful in the context of reading file data via MIDAC.

Double AbundanceLimit { get; set; }

The abundance value that is the largest value that could be seen in this spectrum (the theoretical "full scale" value).

IDoubleRange[] AcqTimeRanges { get; set; }

The acquisition (retention) time range(s) from which this spectrum was extracted.

IDoubleRange CompensationFieldRange { get; set; }

(FAIMS only – not used for IM-MS)

IDoubleRange DispersionFieldRange { get; set; }

(FAIMS only – not used for IM-MS)

IDoubleRange[] DriftTimeRanges { get; set; }

The drift time range(s) from which this spectrum was extracted. Null if no drift time filtering was done.

FragmentationClass FragmentationClass { get; set; }

The fragmentation class (e.g., LowEnergy, HighEnergy, Mixed) of the spectrum. Unspecified if unknown or no fragmentation was done.

IDoubleRange FragmentationEnergyRange { get; set; }

The range of fragmentation (collision) energies used. Null if no fragmentation was done.

A range of 10-30 V indicates that at least one contributing spectrum had a CE of 10V, another had 30V, and there may have been others in between.

FragmentationOpMode FragmentationOpMode { get; set; }

What fragmentation operation(s) were used to collect the data in this spectrum.

IDoubleRange FragmentorVoltageRange { get; set; }

If acquired on an instrument with a fragmentor voltage parameter, the range of that voltage used in producing this spectrum.

Ionization IonizationMode { get; set; }

The ionization mode being used when these data were acquired.

Polarity IonPolarity { get; set; }

The ion polarity of this spectrum

MsLevel MsLevel { get; set; }

The MS level (MS or MS/MS) of this spectrum. Note that All-Ions data is MS (even the high fragmentation spectra)

MsScanType MsScanType { get; set; }

The MS scan type(s) contributing to this spectrum

MsStorageMode MsStorageMode { get; set; }

Whether this data is a profile or peak-detected representation of the spectrum

IDoubleRange[] MzOfInterestRanges { get; set; }

For MS/MS spectra the m/z(s) used as the precursor ion, product ion, or neutral gain or loss. A single range unless this is the combination of multiple spectra with different m/z of interest values; then one range for each unique value.

Quadrupole mass selection is represented as a CenterWidth range with the isolation m/z as the center. If the actual limits of the (possibly asymmetric) isolation window are known, they are used for the range; otherwise, the range has zero width.

Double TofBinWidth { get; set; }

The TOF flight time bin width, in nanoseconds.

IMidacMsDetailsSpec Clone();

Makes a deep copy of this object.