ADINSTRUMENTS
making science easier

Simple Data File Reader/Writer

SDK Documentation (C-Interface)

Document Revision: 0.04

SDK version: 1.0

05 June 2012

Table of Contents

1 Revision History

Revision	Author	Date	Description
0.01	REG	10.05.2012	Creation
0.02	REG	15.05.2012	Incorporating feedback
0.03	PAS	05.06.2012	SDK Beta 1 revision
0.04	CDA	27.07.2012	SDK 1.0 Release

2 Purpose

The Simple Data File SDK allows third-parties to write software to load and save data in Simple Data (.adidat) files. The purpose of this document is to describe how to use this SDK.

Prior to the development of this SDK there were two ways of accessing LabChart data from third-party software.

1. Export data using a Script run from LabChart.
2. Export data as text.

With this SDK there is now a third way. The ADInstruments Simple Data File SDK exposes a set of interfaces and low-level function calls to retrieve and store data in Simple Data (.adidat) files. The SDK approach is the most difficult to learn but allows the most flexibility and performance.

A separate SDK is available which provides advanced options for creating and accessing the data in Simple Data files. This SDK exposes COM interfaces which are covered by a separate document.

3 Introduction

ADInstruments Simple Data Files (.adidat) contain:

- Raw sampled channel data,
- Channel and record information (including rates and record start time),
- Comment markers.

ADInstruments Simple Data Files do not contain LabChart settings information. This means:

- No channel calculations.
- No display settings such as graticule color.

The file format used will be compatible with all current and future versions of LabChart from v6.1.1 onwards and will give consistent and flexible access to LabChart data. ADI Simple Data Format files (*.adidat) are able to be exported from LabChart v7.3 onwards.

Using the interfaces provided with this SDK, data and other information may be read from an existing Simple Data Format file or appended to it. New files may also be created. Refer to the Appendix of this document for details of the C-style interface and example usage.

The Simple Data Format SDK includes sample projects that detail how to use the C-style interface to read and write to files. If writing your own project, your project must load the ADI Simple Data Format dll in order to access this interface. Refer to the sample projects for examples of how to configure your project to achieve this.

Once configured, your project will be able to open an *.adidat file and access its data. LabChart Data and comments may be read from the file and new data and comments may be added. Refer to the Appendix of this document for examples of these techniques, and to the code in the SDK sample projects.

Note that neither LabChart software nor a PowerLab are required to use this SDK.

The source code in the SDK remains the property of ADInstruments. While ADInstruments make every effort to maintain compatibility, the interface functions and specifiers may change from time to time to support newer versions of LabChart and enhancements to the SDK.

4 Trademarks

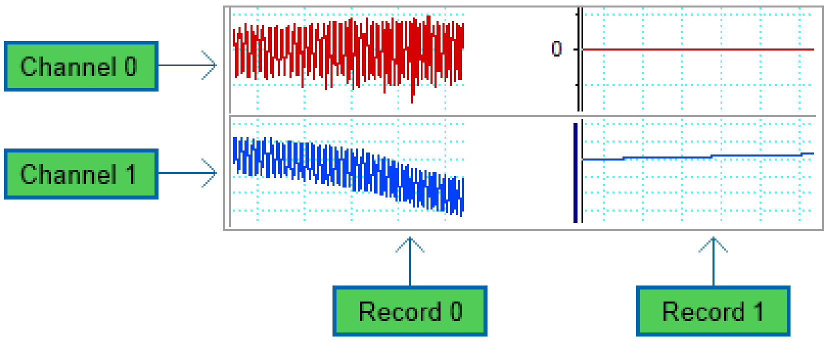
PowerLab® and LabChart® are registered trademarks of ADInstruments Pty Ltd.

5 Definitions

The following are common terms used in this document and the C-style interface in relation to ADInstruments software and data:

- *.adidat: Simple Data Format file that contains 'raw' LabChart data and no settings. The format can read by LabChart versions 6.1.1 and above and will be supported in all future versions of LabChart.
- Channel: A channel corresponds to a single input device e.g. one input on a PowerLab. A channel can contain multiple records (refer to the figure for 'Records" below). Channel numbers start from 0.
- Record: Channels are organized into segments which represent the times at which data recording was started and stopped. These segments are referred to as 'records' and correspond to the 'blocks' mentioned in the LabChart Help. Because starting and stopping recording affects all channels simultaneously, the record boundaries span across all channels. Record numbers start from 0.

  Example: illustrates channels and records in a data file:



- LabChart stores recorded data as a sparse two-dimensional array indexed by record and channel. This means that some records may potentially not have any data in one or more of its channels.
- Sample: A discrete value taken from an analog signal during recording and stored as part of the LabChart Data.
- Sample Rate: The regular time interval at which samples (discrete values) are taken from the analog signal during recording. Sample Rate is measured in samples/second.

- Ticks and the Tick Rate: The sample rate of the fastest channel in a data block is called the "Tick Rate". The tick rate is the same for all channels in a block.

  Example: Three channels sampling at different rates for 20 seconds:

	Sampling Rate (samples/second)
Channel 1	4,000
Channel 2	10,000
Channel 3	40,000

  Ticks = Highest Sampling Rate (Tick Rate) x Sampling Duration

Tick Rate	x	Sampling Duration	=	Ticks
40,000		20		800,000

- Comments: Text items positioned at user-specified time points in the data. Comments are identified by their position within the data and are stored as a character string.
- Trigger: An event that caused data recording to start. This could be through either the start/stop button or an automated trigger in LabChart. The trigger is time-zero in the default 'Time From Start Of Block' display mode in LabChart. The time-zero does not correspond to the start of the record if either pre-trigger or post-trigger sampling was enabled, or if data in the record has been deleted.
- LabChart Data: Collective term for the samples stored in a file. These samples represent an analog signal.
- Data Types: C-Interface accesses the data samples as arrays of float values. Channel and record identifiers are long-int (32-bit). Comments are stored as wide character (wchar_t) unicode strings. Ticks, tick-rates, samples and sample-rates are all long-int. Refer to the sample projects for further details of the data-types used in the ADI Simple Data Format SDK.
- LabChart: ADInstruments software for recording, saving, viewing and performing calculations on data recorded using the PowerLab data acquisition system. Hereinafter referred to as 'LabChart'
- PowerLab: ADInstruments hardware data acquisition device for capturing analog signal data. This is used in conjunction with LabChart software. Hereinafter referred to as 'PowerLab'.

6 Application Programming Interface

This section describes all available functions in the SDK C interface.

6.1 Common Functions

6.1.1 ADI_OpenFile

Opens an existing .adidat file that allows both read and write operations, returning a pointer to it. If the function succeeds, *fileH* will be a handle to the file, otherwise 0.

Parameters:

1.	*Path*	absolute, delimited path to the file, e.g. "C:\\MyData\\TestFile.adidat"
2.	*fileH*	pointer to an ADI_FileHandle object [outparam]
3.	*mode*	ADIFileOpenMode option for controlling the how the file is to be opened

Returns *kResultSuccess* if the operation succeeded.

6.1.2 ADI_CloseFile

Closes the open file and releases memory. If operation is successful, 'fileH' param will be be 0.

Parameters:

1.	*fileH*	pointer to an ADI_FileHandle object [in/outparam]

Returns *kResultSuccess* if the operation succeeded.

6.1.3 ADI_CreateFile

Creates a new *.adidat file for write operations at the specified location and returns a pointer to it. If operation is successful, 'file' param will be be a handle to the file, else 0.

Parameters:

1.	*path*	absolute, delimited path to the file, e.g. "C:\\MyData\\NewFile.adidat"
2.	*fileH*	pointer to an ADI_FileHandle object [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.1.4 ADI_GetErrorMessage

Retrieves a text description of a result code returned from an API call.

Parameters:

1. code	an ADIResultCode value
2. message	null-terminated text for the error message [outparam]
3. maxChars	the size used for the buffer. Must not exceed this when copying
4. text into the buffer
5. textLen	receives the number of characters needed to hold the full comment text, even if parameter text is NULL (optional, may be NULL) [outparam]

Returns kResultBufferTooSmall if maxChars was too small to receive the full title text. Otherwise returns kResultSuccess.

6.1.5 ADI_TickToSamplePos

Converts a tick position to a sample position for a given channel.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*channel*	the channel index (starting from 0)
3.	*record*	the record index (starting from 0)
4.	*tickInRecord*	the tick position to be converted
5.	*samplePosInRecord*	Out parameter that contains the converted sample position

Returns *kResultSuccess* if the operation succeeded.

6.1.6 ADI_SamplePosToTick

Converts a sample position to a tick position for a given channel.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*channel*	the channel index (starting from 0)
3.	*record*	the record index (starting from 0)
4.	*samplePosInRecord*	the tick position to be converted
5.	*tickInRecord*	the converted sample position [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2 Read from File Functions

6.2.1 ADI_GetNumberOfRecords

Retrieves the number of records in the file.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*nRecords*	the number of records in the file [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.2 ADI_GetNumberOfChannels

Retrieves the number of channels in the file.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*nChannels*	the number of channels in the file [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.3 ADI_GetNumTicksInRecord

Retrieves the number of ticks in the specified record.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*record*	the record index (starting from 0)
3.	*nTicks*	the number of ticks in the record [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.4 ADI_GetRecordTickPeriod

Retrieves the tick period for the specified record and channel.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*channel*	the channel in the record (starting from 0)
3.	*record*	the record index (starting from 0)
4.	*secsPerTick*	the tick period for the record [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.5 ADI_GetNumSamplesInRecord

Retrieves the number of samples in the specified record.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*channel*	the channel in the record (starting from 0)
3.	*record*	the record index (starting from 0)
4.	*nSamples*	the number of samples in the record [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.6 ADI_GetRecordSamplePeriod

Retrieves the sample period for the specified record.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*channel*	the channel in the record (starting from 0)
3.	*record*	the record index (starting from 0)
4.	*secsPerSample*	the sample period for the record [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.7 ADI_GetRecordTime

Retrieves time information about the specified record. The trigger time is the time origin of the record and may differ from the start time if there is a pre or post trigger delay, as specified by the trigMinusRecStart parameter.

Parameters:

1. *fileH* ADI_FileHandle for the open file
2. *record* the record index (starting from 0)
3. *triggerTime* out parameter: a time_t that receives the date and time of the trigger position for the new record. Measured as number of seconds from 1 Jan 1970
4. *fracSecs* receives the fractional seconds part of the record trigger time ('triggerTime' parameter)
5. *trigMinusRecStart* trigger-time-minus-record-start-ticks. Receives the difference between the time of trigger tick and the first tick in the record. This positive for pre-trigger delay and negative for post-trigger delay.

Returns *kResultSuccess* if the operation succeeded.

6.2.8 ADI_CreateCommentsAccessor

Creates a comments accessor handle for the specified record.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*record*	the record index (starting from 0)
3.	*commentsH*	handle to the new comments accessor for the record [outparam]

Returns *kResultSuccess* if the operation succeeded.

6.2.9 ADI_CloseCommentsAccessor

Closes the comments accessor, releasing the memory it used. Sets the ADI_CommentsHandle to 0.

Parameters:

1.	ADI_*CommentsHandle*	handle to a comments accessor.

Returns *kResultSuccess* if the operation succeeded.

6.2.10 ADI_GetCommentInfo

Retrieves information from the comment currently referenced by the comments accessor.

Parameters:

1.	ADI_CommentsHandle	handle to a comments accessor
2.	*tickPos*	out parameter that receives the tick position of the comment in the record
3.	*commentNum*	out parameter that receives the number of the comment
4.	*channel*	out parameter that receives the channel of the comment (-1 for all channel comments)
5.	*text*	out parameter for a buffer to receive null terminated text for the comment (optional, may be NULL)
6.	*maxChars*	the size of the text buffer in wchar_t s. The text will be truncated to fit in this size
7.	*textLen*	out parameter that receives the number of characters needed to hold the full comment text, even if parameter text is NULL (optional, may be NULL)

Returns *kResultSuccess* if the operation succeeded. Otherwise returns kResultBufferTooSmall if maxChars was too small to receive the full comment text. Returns kResultNoData if this accessor has reached the end of the comments in the record.

6.2.11 ADI_NextComment

Advances the comments accessor to the next comment in the record.

Parameters:

1.	ADI_*CommentsHandle*	handle to a comments accessor

Returns *kResultNoData* if this accessor has reached the end of the comments in the record. Otherwise returns *kResultSuccess*.

6.2.12 ADI_GetSamples

Retrieves a block of sample data from the file into a buffer. Samples are in physical prefixed units.

Parameters:

1. *fileH* ADI_FileHandle for the open file
2. *channel* the channel containing the desired data (starting from 0)
3. *record* the record containing the start position of the desired data (starting from 0)
4. *startPos* the start position as an offset from the start of the record (0)
5. *dataType* specifies the type of data (ticks or samples) to retrieve
6. *nLength* number of elements (ticks/samples) to retrieve
7. *data* out parameter: pointer to a float array of 'nLength' in size e.g. float* data=(float*)malloc(sizeof(float*kDataSize));
8. *returned* out parameter that will contain the number of samples actually returned by the function (may be fewer than the amount requested)

Returns *kResultSuccess* if the operation succeeded.

6.2.13 ADI_GetUnitsName

Retrieves the prefixed units of a channel, as a string.

Parameters:

1. *fileH* ADI_FileHandle for the open file
2. *channel* the unit's channel (starting from 0)
3. *record* the unit's record (starting from 0)
4. *units* out parameter: a buffer to receive null terminated text for the units name (optional, may be NULL)
5. *maxChars* the size of the text buffer in wchar_t s. The text will be truncated to fit in this size
6. *textLen* out parameter that receives the number of characters needed to hold the full comment text, even if parameter text is NULL (optional, may be NULL)

Returns *kResultBufferTooSmall* if the passed-in *maxChars* was too small to receive the full comment text. Otherwise, returns *kResultSuccess*.

6.2.14 ADI_GetChannelName

Retrieves the name of a channel, as a string.

Parameters:

1. *fileH* ADI_FileHandle for the open file
2. *channel* the channel index (starting from 0)
3. *name* out parameter that will contain null-terminated text for the name
4. *maxChars* the size used for the buffer. Must not exceed this when copying text into the *buffer*
5. *textLen* out parameter: receives the number of characters needed to hold the full comment text, even if parameter text is NULL (optional, may be NULL)

Returns *kResultBufferTooSmall* if the passed-in *maxChars* was too small to receive the full title text. Otherwise, returns *kResultSuccess*.

6.3 Write to File Functions

6.3.1 ADI_SetChannelName

Sets the name of the specified channel.

Parameters:

1.	*file*	ADI_FileHandle for the open file
2.	*channel*	the channel to set the name (starting from 0)
3.	*name*	null-terminated text string with the channel name

Returns kResultSuccess if the operation succeeded.

6.3.2 ADI_CreateWriter

Creates a new writer session for writing new data and returns a handle to that open writer for use in other related functions.

Parameters:

1.	*fileH*	ADI_FileHandle for the open file
2.	*writerH*	out parameter that will contain the ADI_WriterHandle for the new writing session

Returns *kResultSuccess* if the operation succeeded.

6.3.3 ADI_SetChannelInfo

Sets the channel information for the specified channel in a new record to be written by the writer session.

Parameters:

1.	*writerH*	ADI_WriterHandle for the writing session
2.	*channel*	the channel to receive the new info (starting from 0)
3.	*enabled*	boolean value set true (1) if data is to be added to this channel in the new record
4.	*secondsPerSample*	new sample period for this channel in the new record
5.	*unitsName*	null-terminated text string units for the channel record
6.	*limits*	Optional pointer to a struct holding maximum and minimum values between which valid data values in the channel fall. If NULL, the limits are +ve and -ve infinity.

Returns *kResultSuccess* if the operation succeeded.

6.3.4 ADI_StartRecord

Starts the writer recording a new record, setting the trigger time. The trigger time is the time origin of the record and may differ from the start time if there is a pre or post trigger delay, as specified by the trigMinusRecStart parameter.

Parameters:

1.	*writerH*	ADI_WriterHandle for the writing session
2.	*triggerTime*	time_t specifying the date and time of the trigger position for the new record. Measured as number of seconds from 1 Jan 1970
3.	*fracSecs*	Provides fraction-resolution to the start position of the record ('triggerTime' parameter)
4.	*trigMinusRecStart*	trigger-time-minus-record-start-ticks. Specifies the difference between the time of trigger tick and the first tick in the record. This +ve for pre-trigger delay and -ve for post-trigger delay.

Returns *kResultSuccess* if the operation succeeded.

6.3.5 ADI_AddChannelSamples

Writes new data samples into the specified channel record.

Parameters:

1. *writerH* ADI_WriterHandle for the writing session
2. *channel* the channel to receive the new data (starting from 0)
3. *data* an array of new float data e.g. float* data = (float*)malloc(sizeof(float * numSamples));
4. *dataType* specifies the type of data (ticks or samples) being added
5. *nSamples* number of samples in the array
6. *newTicksAdded* number of ticks by which the record increased as a result of adding these samples.This will usually be 0 for channels other than the first to which samples are added.In the multi-rate case, this can be greater than the number of samples added if the channel has a sample rate lower than the tick rate.

Returns *kResultSuccess* if the operation succeeded.

6.3.6 ADI_FinishRecord

Ends the current record being written by the writer.

Parameters:

1. *writerH* - ADI_WriterHandle for the writer session

Returns *kResultSuccess* if the operation succeeded.

6.3.7 ADI_CommitFile

Ensures all changes to the file made via the writer session are written to the file.

Parameters:

1. *writerH* - ADI_WriterHandle for the writer session

Returns *kResultSuccess* if the operation succeeded.

6.3.8 ADI_CloseWriter

Terminates the writer session and releases resources used by the session. Sets the ADI_WriterHandle to 0. It also internally calls ADI_CommitFile() to ensure all changes to the file made via the writer session are written to the file.

Parameters:

1. *writerH* ADI_WriterHandle for the writer session

Returns *kResultSuccess* if the operation succeeded.

6.3.9 ADI_AddComment

Adds a new comment at the specified position in the existing data.

Parameters:

1. *file* ADI_FileHandle for the open file
2. *channel* the channel to add the comment into, or -1 for all-channel comment
3. *record* the record containing the desired position of the new comment (starting from 0)
4. *tickPos* the tick position of the comment as an offset from the start of the record (0)
5. *text* the null-terminated text string for the new comment

6. *commentNum* out parameter that will contain the number of the newly added comment (optional, may be NULL)

Returns *kResultSuccess* if the operation succeeded.

6.3.10 ADI_DeleteComment

Deletes a comment from the specified position in the existing data.

Parameters:

1.	*file*	ADI_FileHandle for the open file
2.	*commentNum*	the number of the comment to delete

Returns *kResultSuccess* if the operation succeeded.

6.4 Error Handling

Most functions return an error code in the form of an ADIResultCode enum.

From this error code, a text description of the error can be retrieved by calling ADI_GetErrorMessage() as shown in the following example:

```
ADI_FileHandle fileH(0);
ADIResultCode result = ADI_OpenFile(path, &fileH, kOpenFileForReadOnly);

// If we failed to open the file, print the error message returned from
// ADI_OpenFile().
if(result != kResultSuccess)
   {
   ADI_GetErrorMessage(result,errorMsg,kMaxErrorMsgLen,0);
   wprintf_s(L"\n%s\n", errorMsg);
   return 1;
   }
```

8 Appendix

The following is an excerpt of the declarations and functions for the C-Interface (comments have been removed here for brevity):

8.1 General

```
ResultCode ADI_OpenFile(const TCHAR* path, ADI_FileHandle* fileH, FileOpenMode mode);
ResultCode ADI_CreateFile(const TCHAR* path, ADI_FileHandle* fileH);
const TCHAR* ADI_GetErrorMessage(ResultCode code);
ResultCode ADI_TickToSamplePos(ADI_FileHandle fileH, long channel, long record,
                               long tick, long* samplePos);
```

8.2 Read From File

```
ResultCode ADI_GetNumberOfRecords(ADI_FileHandle fileH, long* nRecords);
ResultCode ADI_GetNumberOfChannels(ADI_FileHandle fileH, long* nChannels);
ResultCode ADI_GetNumTicksInRecord(ADI_FileHandle fileH, long record, long* nTicks);
ResultCode ADI_GetRecordTickRate(ADI_FileHandle fileH, long channel, long record,
                                 double* secsPerTick);
ResultCode ADI_GetNumSamplesInRecord(ADI_FileHandle fileH, long channel, long record,
                                     long* nSamples);
ResultCode ADI_GetRecordSampleRate(ADI_FileHandle fileH, long channel, long record,
                                   double* secsPerSample);
ResultCode ADI_GetNumberOfComments(ADI_FileHandle fileH, long* nComments);
ResultCode ADI_GetCommentText(ADI_FileHandle fileH, long commentNum, TCHAR* comment,
                              long maxChars);
ResultCode ADI_GetCommentChannel(ADI_FileHandle fileH, long commentNum, long* channel);

ResultCode ADI_GetCommentPosition(ADI_FileHandle fileH, long commentNum, long* record,
                                  long* tickPos);
ResultCode ADI_GetSamples(ADI_FileHandle fileH, long channel, long record, long startPos,
                          DataType dataType, long nLength,  float* data, long* returned);
ResultCode ADI_GetUnits(ADI_FileHandle fileH, long channel, long record, TCHAR* units,
                        long maxChars);
ResultCode ADI_GetChannelTitle(ADI_FileHandle fileH, long channel, TCHAR* title,
                               long maxChars);
```

8.3 Write To File

```
ResultCode ADI_AppendNewRecord(ADI_FileHandle fileH, time_t startTime, double fracSecs,
                               long trigMinusRecStart, ADI_RecordHandle* recordH);
ResultCode ADI_SetChannelInfo(ADI_RecordHandle recordH, long channel, const TCHAR* units,
                              long samplingRate);
ResultCode ADI_SetChannelTitle(ADI_FileHandle fileH, long channel, const TCHAR* title);
ResultCode ADI_AddChannelSamples(ADI_RecordHandle recordH, long channel,
                                 DataType dataType, float* data, long nSamples);
ResultCode ADI_FinishRecord(ADI_RecordHandle* recordH);
ResultCode ADI_SetNumberOfChannels(ADI_FileHandle fileH, long nChannels);
ResultCode ADI_AddComment(ADI_FileHandle fileH, long channel, long record, long tickPos,
                          const TCHAR* text, long* commentNum);
ResultCode ADI_DeleteComment(ADI_FileHandle fileH, long commentNum);
ResultCode ADI_CloseFile(ADI_FileHandle* fileH);
```

8.4 Enumerations, Types and Constants

```
   enum ADIResultCode
      {
      kResultSuccess = 0,
      kResultErrorFlagBit = 0x80000000,
      kResultFail         = kResultErrorFlagBit,
      kResultFileNotFound = kResultErrorFlagBit | 1,
      kResultFileOpenFail = kResultErrorFlagBit | 2,
      kResultFileIOError  = kResultErrorFlagBit | 3,
      kResultInvalidFileHandle  = kResultErrorFlagBit | 4,
      kResultInvalidPosition    = kResultErrorFlagBit | 5,
      kResultInvalidCommentNum  = kResultErrorFlagBit | 6,
      kResultInvalidArg         = kResultErrorFlagBit | 7,
      kResultNoData             = kResultErrorFlagBit | 8,
      kResultBufferTooSmall     = kResultErrorFlagBit | 9
      };

   // File open modes
   enum ADIFileOpenMode
      {
      kOpenFileForReadOnly = 0,
      kOpenFileForReadAndWrite,
      };

   // Data Types : ADICDataFlags
   enum ADICDataFlags
      {
      kADICDataAtSampleRate = 0,
      kADICDataAtTickRate = 0x80000000,
      };

   typedef struct ADIDataLimits
      {
      float mMaxLimit;
      float mMinLimit;
      } ADIDataLimits;

   struct ADI_FileHandle__ { int unused; }; typedef struct ADI_FileHandle__
*ADI_FileHandle;
   struct ADI_WriterHandle__ { int unused; }; typedef struct ADI_WriterHandle__
*ADI_WriterHandle;
   struct ADI_CommentsHandle__ { int unused; }; typedef struct
ADI_CommentsHandle__ *ADI_CommentsHandle;
```

8.5 Example Usage of the Interface

For simple examples of how to read and write to ADInstruments Simple Data files see the examples folder in the accompanying SDK.