User's Manual

WDF File Access Library

YOKOGAWA
Yokogawa Test & Measurement Corporation

IM B8074XP-01EN
5th Edition

Foreword

This user's manual contains useful information about the precautions, functions, and API specifications of the WDF File Access Library.
To ensure correct use, please read this manual thoroughly during operation.
After reading the manual, keep it in a convenient location for quick reference whenever a question arises during operation.
For information about the handling precautions, functions, and operating procedures of the measurement instrument, and the handling and operating procedures of Windows, see the manuals that accompany the particular instrument you are using.

Notes

- The contents of this manual are subject to change without prior notice as a result of continuing improvements to the instrument's performance and functions.
- Every effort has been made in the preparation of this manual to ensure the accuracy of its contents. However, should you have any questions or find any errors, please contact your nearest YOKOGAWA dealer.

Trademarks

- Microsoft, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- For purposes of this manual, the TM and ® symbols do not accompany their respective trademark names or registered trademark names.
- Other company and product names are trademarks or registered trademarks of their respective companies.

Revisions

- 1st Edition: October 2010
- 2nd Edition: January 2013
- 3rd Edition: February 2014
- 4th Edition: October 2017
- 5th Edition: December 2018

5th Edition: December 2018 (YMI)

Contents

1.1 Software Overview

Overview

This software offers API (Application Program Interface) for acquiring data from waveform data files (WDF) saved on the measurement instrument.

Functions

This software enables the following functions. Please refer to Chapter4.3 (API Detailed Specification) for details.

- Getting Measuring Date/Time
- Getting Measuring Trace(Channel) Information
- Getting Measuring Points
- Getting Time Data/Measuring Data
- Getting Units of Time Axis/Measuring Axis
- Getting Resolutions of Time Axis/Measuring Axis

Software Structure

This software is composed of the following software packages.

- WDF File Access Library User's Manual(this manual)
- API Related files

File Name	Content
****.dll	WDF File Access Library Body
****.lib	DLL Import Libraryl
WDFAPIPublic.h	Header file of Prototype Declaration
WDFAPIWrapper.cpp	Sample Code by Explicit DLL Linking

System Condition

PC

PC capable of running Windows 7(32bit/64bit), Windows 8(32bit/64bit), and Windows 10(32bit/64bit) with at least a Core2Duo 2GHz processor and at least 2GB of memory.

Application Developing Environment

VisualStudio 2008 SP1 or later

2.1 Notes on Using the Software

Disclaimers

By downloading and installing this software, the customer agrees to all of the following disclaimers.

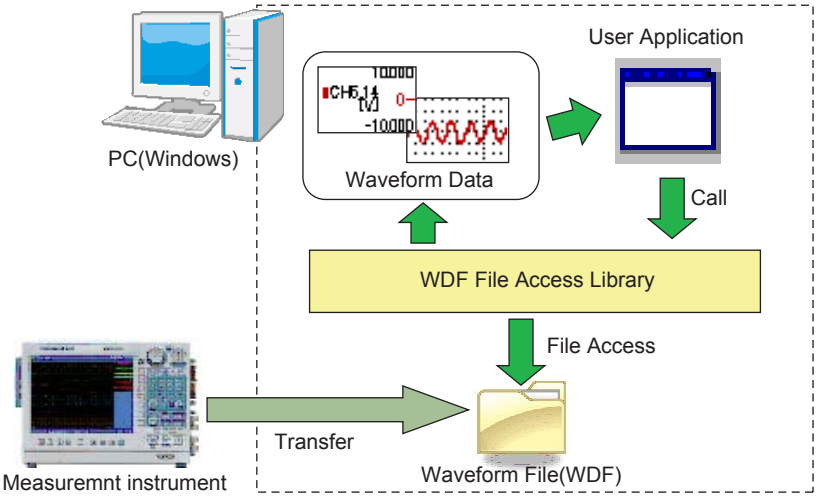
- Yokogawa bears no liability for any problems occurring as a result of downloading or installing this software.
- Yokogawa bears no responsibility for any damage caused directly or indirectly as a result of using this software.
- This software is provided free of charge, however no unlimited warranty against software defects exists, nor is any claim made that the product is free of all defects whatsoever. Also, Yokogawa is not always able to repair defects (“bugs”) in, or respond to questions or inquiries about this software.
- Yokogawa reserves all rights to this software, including but not limited to all property rights, ownership rights, and intellectual property rights.

Precautions Concerning the Use of the Software

- This software is the special library for waveform data files saved in WDF format. Please do not use for waveform data files saved on other measuring instruments.
- Can not load waveform data files saved in WDF format of newer version than the software.
- VisualStudio 2008 SP1 is required to use this software. Please install VisualStudio 2008 SP1 Redistributable Package, when VisualStudio 2008 SP1 is not installed in PC.

3.1 WDF File Access API Specification Overview

API is offered as the DLL (Dynamic Link Library) and DLL enables application software to use API by linking DLL with it.
As the diagram below indicates, API offers some functions for application development, such as getting measuring conditions, getting waveform data, about recording waveform data files(WDF) accessible over PC storage environment.



3.2 API Specification Overview

The following indicates API specification overview.

Creating/Discarding Handle

The following indicates API for creating/discarding handle.

API Name	Function	Page
WdfOpenFileEx	Create handle for WDF file access	4-5
WdfOpenFile	Create handle for WDF file access	4-6
WdfCloseFile	Discard handle for WDF file access	4-7

Getting Measuring Trace Information

The following indicates API for getting measuring trace (channel) information.

API Name	Function	Page
WdfGetTraceNumber	Get total waveform trace number	4-7
WdfGetTraceName	Get waveform trace name	4-8
WdfGetLogicBitName	Get logic bit name of waveform trace	4-8
WdfGetTraceBlockNumber	Get history total block number of waveform trace	4-9

Getting Waveform Accompanying Information

The following indicates API for getting accompanying information of waveform data.

API Name	Function	Page
WdfGetVDataType	Get data type of history block	4-9
WdfGetVOffset	Get vertical axis offset of history block	4-10
WdfGetVResolution	Get vertical axis resolution of history block	4-11
WdfIsEnabledVIllegalData	Determine whether hidden code is valid.	4-11
WdfGetVIllegalData	Get invalid value of history block	4-12
WdfGetVUnit	Get vertical axis unit of history block	4-12
WdfGetHOffset	Get horizontal axis start data offset of history block	4-13
WdfGetHResolution	Get horizontal axis resolution of history block	4-13
WdfGetHUnit	Get horizontal axis unit of history block	4-14
WdfGetDate	Get measuring date of history block	4-14
WdfGetTime	Get measuring time of history block	4-15

Getting Waveform

The following indicates getting waveform data.

API Name	Function	Page
WdfGetBlockSize	Get data block size of history block	4-15
WdfGetBlockSize64	Get data block size of history block	4-16
WdfGetScaleWave	Get waveform raw data	4-16
WdfGetScaleWave64	Get waveform raw data	4-20

DLL Types and Supported APIs

The supported APIs vary depending on the DLL type.

API Name	DLL Type			
	DL850E.dll[1]	DL350.dll[2]	XVWDF.dll[3]	DLM3000.dll[4]
WdfOpenFileEx	Y	Y	Y	N
WdfOpenFile	N	N	N	Y
WdfCloseFile	Y	Y	Y	Y
WdfGetTraceNumber	Y	Y	Y	Y
WdfGetTraceName	Y	Y	Y	Y
WdfGetLogicBitName	Y	Y	Y	Y
WdfGetTraceBlockNumber	Y	Y	Y	Y
WdfGetVDataType	Y	Y	Y	Y
WdfGetVOffset	Y	Y	Y	Y
WdfGetVResolution	Y	Y	Y	Y
WdfIsEnabledVIllegalData	Y	Y	N	N
WdfGetVIllegalData	Y	Y	Y	Y
WdfGetVUnit	Y	Y	Y	Y
WdfGetHOffset	Y	Y	Y	Y
WdfGetHResolution	Y	Y	Y	Y
WdfGetHUnit	Y	Y	Y	Y
WdfGetDate	Y	Y	Y	Y
WdfGetTime	Y	Y	Y	Y
WdfGetBlockSize	N	N	N	Y
WdfGetBlockSize64	Y	Y	Y	N
WdfGetScaleWave	N	N	N	Y
WdfGetScaleWave64	Y	Y	Y	N

Y: Supported, N: Not supported

1 Applies to waveform data saved with the DL850E series.
2 Applies to waveform data saved with the DL350.
3 Applies to waveform data saved with Yokogawa's Xviewer software.
4 Applies to waveform data saved with the DLM3000 series.

3.3 Flow of API Usage

Each API is used by file handle. Initially, create handle with WDF file specified by full-path file name, and access WDF file by passing the handle as the argument of API.
The following indicates flow of API usage and code description examples.

1. Creation of API handle

Include API header file (WDFAPIPublic.h) in your program.

Get the specified WDF file handle by using the WdfOpenFileEx API.

```
Ex.
#include "WDFAPIPublic.h"
• • •
WDFHandle handle;
WdfOpenFileEx(&handle, "C:\\Hoge\\0001.WDF", …);
```

2. Getting waveform information from WDF file

Utilize each API by passing the handle as the first argument.

```
Ex.
WdfGetTraceNumber(handle, &traceNumber);
WdfGetTraceName(handle, traceNumber - 1, traceName);
WdfGetTraceBlockNumber(handle, traceNumber - 1, &blockNumber);
WdfGetTime(handle, traceNumber - 1, blockNumber -1, time);
  • • •
```

3. Discarding API handle

Discard the created file handle by using the WdfCloseFile API.

```
Ex. WdfCloseFile(&handle);
```

3.4 Definition of API Term

The following indicates the term used in API.

Trace Number

The channel (sub channel) is equivalent to zero-origin trace number. For example, if CH1, CH3_1, CH3_2, Ch5 are specified as the target channels for saving, the relation between channel and trace number is as indicated bellow.

Channel (Sub Channel)	Trace Number
CH1	0
CH3_1	1
CH3_2	2
CH5	3

You can get the number of channels (sub channels) in a WDF file by using the WdfGetTraceNumber API.
The trace number of above example is 4, and valid trace number is 0-3(=4-1).
The relation between trace number and display channel is obtained by getting waveform channel label (WdfGetTraceName API).

History Block Number

In case of selecting “History ALL” waveform save, historical acquired data are also saved.
History waveform is specified by record number, and history block number of API is equivalent to record number.
Record number #0000 means the latest waveform data, but history block number 0 means the earliest.
The following indicates the example of four history waveforms.

History Waveform	Record Number	History Block Number
Latest	# 0000	3
↑	#-0001	2
↓	#-0002	1
Earliest	#-0003	0

You can get the total number of history waveforms in a WDF file by using the WdfGetTraceBlockNumber API. The total number of history waveform of above example is 4, and valid history block number is 0-3(=4-1).

In case of selecting “History One” waveform save, the number of history waveform is 1, and history block number is constantly 0.

Waveform Data Block

History block of each trace has waveform data (waveform data block). Waveform data block is obtained by using the WdfGetScaleWave API or WdfGetScaleWave64 API. And waveform block size (waveform points) is obtained by using the WdfGetBlockSize API or WdfGetBlockSize64 API.

Data type of waveform data block depends on mounted modules or measurement settings. The data type of each trace is obtained by using the WdfGetVDataType API.

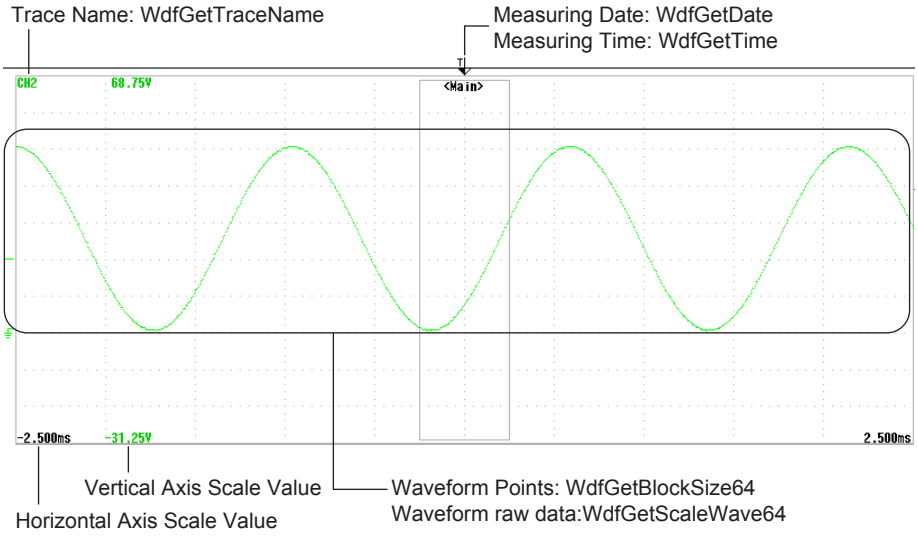
How to convert waveform data block to graphical display data differs from data type to data type.
Further information about waveform data block and data conversion flow, please refer to detailed specification of WdfGetScaleWave API or WdfGetScaleWave64 API.

3.5 Display Screens and Supported APIs

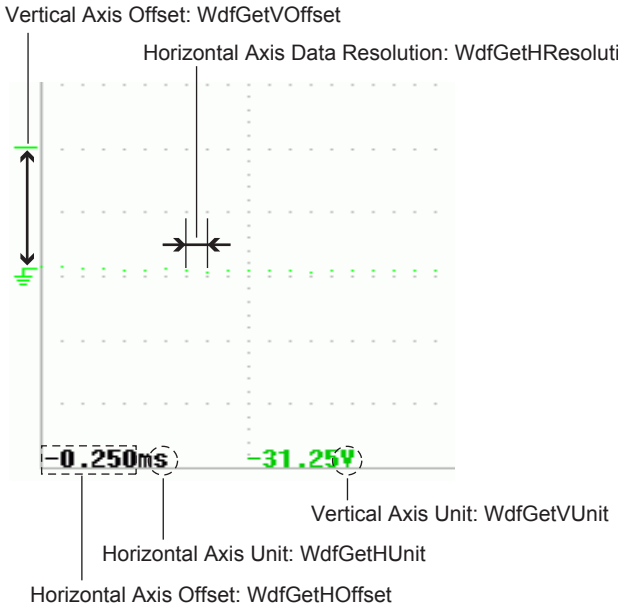
The waveform display and the supported APIs are indicated using the DL850E series display screen as an example.

Main Window

The following chart indicates DL850 series waveform main display measured by typical voltage module and corresponding API.

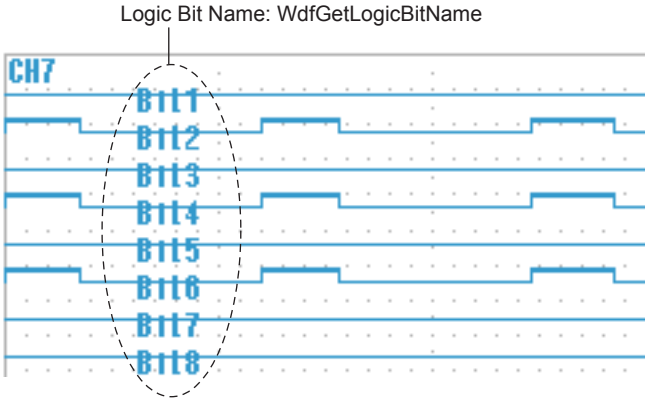


The following chart indicates DL850 series waveform zoom display around leftmost vertical axis and corresponding API.



Logic Module

The following chart indicates DL850 series waveform display measured by logic module and corresponding API.



4.1 Definition of Data Structure

The following indicates Definition of API data structure.

WDFResult

Function: API Error Type

Format:
```
typedef INT WDFResult;
```

Detail: Type of API return.
When the return value of the execution result is WDF_OK, 0 is returned. For other return values, a value other than 0 is returned.

WDFResultKind

Function: API Error Detailed Information

Format:
```
typedef enum{
        WDF_OK,
        WDF_ERR,
        WDF_ERR_Error,
        WDF_ERR_InvalidHandle,
        WDF_ERR_InvalidParameter,
        WDF_ERR_Open,
        WDF_ERR_Alloc,
        WDF_ERR_FileAccess,
        WDF_ERR_NotDualCap,
        WDF_ERR_UnknownVersion,
        WDF_ERR_UnknownFormat,
        WDF_ERR_ParamBoundary,
        WDF_ERR_DataValue,
        WDF_ERR_CantReadDataType,
}WDFResultKind;
```

Detail: Detailed information of WDFResult.
Refer to API detailed specification about each information.

WDFHandle

Function: API Handle Type

Format:
```
typedef HANDLE WDFHandle;
```

Detail: File handle used in API.

WDFTrace

Function: Waveform Trace Number Type

Format:
```
typedef UINT WDFTrace;
```

Detail: Corresponds to channel (sub channel) number.

WDFHistoryBlock

Function: History Block Number Type of Waveform Trace

Format:
```
typedef UINT WDFHistoryBlock;
```

Detail: Corresponds to history record number.

WDFDataParameter

Function: Parameter Type of Waveform Data

Format:
```
typedef double WDFDataParameter;
```

Detail: General numeric value (floating point) type meaning parameter values of waveform data.

WDFDataBlock

Function: Waveform Data Points Type

Format:
```
typedef INT WDFDataBlock;
```

Detail: Data size (points) of waveform data block.

WDFDataBlock64

Function: Waveform Data Points Type

Format:
```
typedef INT64 WDFDataBlock;
```

Detail: Data size (points) of waveform data block.

WDFBitIndex

Function: Logic Bit Index Type

Format:
```
typedef INT WDFBitIndex;
```

Detail: Bit location of logic data.

WDFOpenMode

Function: Mode Type for WdfOpenFileEx

Format:
```
typedef UINT32 WDFOpenMode;
```

Detail: Specify as the argument (Mode) of WdfOpenFileEx.
Specified value is defined in WDFOpenModeKind.

WDFOpenModeKind

Function: Value Definition of WDFOpenMode Type

Format:
```
typedef enum{
        wdfOpenModeNormal,
        wdfOpenModeDualCapture,
  }WDFOpenModeKind;
```

WDFVDataType

Function: Waveform Data Block Type

Format:
```
typedef enum{
        wdfDataTypeUINT16
        wdfDataTypeSINT16,
        wdfDataTypeLOGIC16,
        wdfDataTypeUINT32
        wdfDataTypeSINT32,
  }WDFVDataType;
```

Detail: Numeric data type of waveform data block.
Data type of waveform trace is obtained by using the WdfGetVDataType API.

WDFAccessParam

Function: Acquired Waveform Data Parameter Type

Format:
```
typedef struct{
        UINT            version;
        WDFTrace        trace;
        WDFHistoryBlock block;
        WDFDataBlock    start;
        WDFDataBlock    count;
        INT             ppRate;
        INT             waveType;
        INT             dataType;
        WDFDataBlock    cntOut;
        void*           dst;
        INT             box;
        INT             compMode;
        INT             rsv1;
        INT             rsv2;
        INT             rsv3;
        INT             rsv4;
}WDFAccessParam;
```

Detail: Parameter data type of acquired waveform data.
This data type is specified as the argument of WdfGetScaleWave API.

WDFAccessParam64

Function: Acquired Waveform Data Parameter Type

Format:
```
typedef struct{
        UINT            version;
        WDFTrace        trace;
        WDFHistoryBlock block;
        WDFDataBlock64  start;
        WDFDataBlock64  count;
        INT64           ppRate;
        INT             waveType;
        INT             dataType;
        WDFDataBlock64  cntOut;
        void*           dst;
        INT             box;
        INT             compMode;
        INT             rsv1;
        INT             rsv2;
        INT             rsv3;
        INT             rsv4;
}WDFAccessParam64;
```

Detail: Parameter data type of acquired waveform data.
This data type is specified as the argument of WdfGetScaleWave64 API.

WDFBool

Function: Boolean Type

Format:
```
typedef BOOL WDFBool;
```

Detail: Boolean Value. TRUE or FALSE.

4.2 Definition of Constant Value

Buffer Size of Strings

Function: Buffer Size for Character Strings

Format:

```
enum{
  MAX_WDF_TRACE_NAME    = 16 + 1,
  MAX_WDF_BIT_NAME      = 16 + 1,
  MAX_WDF_V_UNIT        = 16 + 1,
  MAX_WDF_H_UNIT        = 16 + 1,
  MAX_WDF_DATE          = 32 + 1,
  MAX_WDF_TIME          = 32 + 1,
};
```

Detail: Definition of buffer size passed as the argument of API.
Refer to API sample code about usage.

4.3 API Detailed Specification

The following indicates API detailed specification.

WdfOpenFileEx

Function: Create file handle for accessing WDF file.

Format:
```
WDFResult WdfOpenFileEx(WDFHandle* handle, const char* fileName,
WDFOpenMode mode);
```

Argument:

[OUT] handle	Created API file handle
[IN] filename	WDF File Name (Full path designation)
[IN] mode	Open Mode

Return Value:

WDF_OK	Success
WDF_ERR_Open	File Open Error (unknown file name)
WDF_ERR_Alloc	Memory Allocation Error (insufficient memory)
WDF_ERR_FileAccess	File Access Error (storage failure)
WDF_ERR_UnknownFormat	Illegal WDF Format (file defect, file of other than ScopeCorder)
WDF_ERR_DataValue	Illegal WDF Format (file defect)
WDF_ERR_CantReadDataType	Other error

Detail: API file handle is created by specifying argument “filename” and created handle is assigned to argument “handle”. Argument “mode” is constant value (wdfOpenModeNormal).
Each API is used with acquired API file handle to access WDF file.
If a proper file is specified, the API returns WDF_OK(0); otherwise, the API returns a value other than WDF_OK(0).

Caution: The handle value when the return value is not WDF_OK is invalid and cannot be used. At the end of the application, be sure to use the WdfCloseFile API to discard any handles that have been created.
In case of opening WDF file saved by HD Recording with file divide or more than 2G bytes, head file (ex. HOGE_000.WDF) should be specified. Other file is not covered under warranty.

Example:
```
WDFHandle wdfHandle;
const char* wdfFilePath = “C:\\HOGE\\0001.WDF”;
WDFResult result = WdfOpenFileEx(&wdfHandle, wdfFilePath,
wdfOpenModeNormal);
if(result != 0) {
printf('Error: WdfOpenFileEx()\r\n”);
return;
}

• • •
WdfCloseFile(&wdfHandle);
```

WdfOpenFile

Function: Create file handle for accessing WDF file.

Format:
```
WDFResult WdfOpenFile(WDFHandle* handle, const char* fileName);
```

Argument:

[OUT] handle	Created API file handle
[IN] filename	WDF File Name (Full path designation)

Return Value:

WDF_OK	Success
WDF_ERR_Open	File Open Error (unknown file name)
WDF_ERR_Alloc	Memory Allocation Error (insufficient memory)
WDF_ERR_FileAccess	File Access Error (storage failure)
WDF_ERR_UnknownFormat	Illegal WDF Format (file defect, file of other than ScopeCorder)
WDF_ERR_DataValue	Illegal WDF Format (file defect)
WDF_ERR_CantReadDataType	Other error

Detail: API file handle is created by specifying argument “filename” and created handle is assigned to argument “handle”.
Each API is used with acquired API file handle to access WDF file.
If a proper file is specified, the API returns WDF_OK(0); otherwise, the API returns a value other than WDF_OK(0).

Caution: The handle value when the return value is not WDF_OK is invalid and cannot be used.
At the end of the application, be sure to use the WdfCloseFile API to discard any handles that have been created.

Example:
```
WDFHandle wdfHandle;
const char* wdfFilePath = “C:\\HOGE\\0001.WDF”;
WDFResult result = WdfOpenFile(&wdfHandle, wdfFilePath);
if(result != 0) {
printf('Error: WdfOpenFile()\r\n”);
return;
}

• • •
WdfCloseFile(&wdfHandle);
```

WdfCloseFile

Function: Close WDF File.

Format:
```
WDFResult WdfCloseFile(WDFHandle* handle);
```

Argument:

[IN] handle	API File Handle

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid Handle

Detail: Discard the handles that have been created by using the WdfOpenFileEx API or WdfOpenFile API.

Caution: Discarded handle is not available. Access API with discarded handle is not covered under warranty.

Example: Refer to WdfOpenFileEx API, WdfOpenFile API.

WdfGetTraceNumber

Function: Get total waveform trace number.

Format:
```
WDFResult WdfGetTraceNumber(WDFHandle handle, WDFTrace* traceNumber);
```

Argument:

[IN] handle	API File Handle
[OUT] traceNumber	Total Waveform Trace Number

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get total number of waveform trace. Specified trace number is zero-origin, and maximum value is “Total Waveform Trace Number – 1”.

Caution: If an illegal trace number (e.g., a value more than “traceNumber”) is specified, the API returns WDF_ERR_Error.

Example:
```
WDFTrace traceNumber;
WdfGetTraceNumber(wdfHandle, &traceNumber);
```

WdfGetTraceName

Function: Get waveform trace name.

Format:
```
WDFResult WdfGetTraceName(WDFHandle handle, WDFTrace trace, char*
traceName);;
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[OUT] traceName	Waveform Trace Name

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get the name of specified waveform trace. Trace name corresponds to channel label name of measurement instrument.

Example:
```
// Show All Trace Name
WDFTrace traceNumber;
WdfGetTraceNumber(wdfHandle, &traceNumber);
for(WDFTrace trace = 0; trace < traceNumber; trace++){
    char traceName[MAX_WDF_TRACE_NAME];
    WdfGetTraceName(wdfHandle, trace, traceName);
    printf('Trace:%d Name:%s\r\n”, trace, traceName);
}
```

WdfGetLogicBitName

Function: Get logic bit name of waveform trace.

Format:
```
WDFResult WdfGetLogicBitName(WDFHandle handle, WDFTrace trace,
WDFBitIndex index, char* bitName);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] index	Logic Bit Location
[OUT] bitName	Logic Bit Name

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get the logic bit name of specified waveform trace. Bit name corresponds to channel logic bit name of measurement instrument logic settings.

Caution: This API is valid for only waveform trace containing logic data (Waveform Data Type: other than . wdfDataTypeLOGIC16). Other case is not covered under warranty.

Example:
```
// Show All Trace Bit Name
for(int i=0;i<8;i++){
    char bitName[MAX_WDF_BIT_NAME];
    WdfGetLogicBitName(wdfHandle, trace , i, bitName);
    printf('Bit:%02d Name:%s”, i, bitName);
}
```

WdfGetTraceBlockNumber

Function: Get total history block number of waveform trace.

Format:
```
WDFResult WdfGetTraceBlockNumber(WDFHandle handle, WDFTrace trace,
                                 WDFHistoryBlock* blockNumber);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[OUT] blockNumber	Total History Block Number

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get total history block number of waveform trace. Specified history block number is zero-origin, and maximum value is “Total History Block Number – 1”.
Total history block number corresponds to history record number of measurement instrument.

Caution: If an illegal history block number (e.g., a value more than “blockNumber”) is specified, the API returns WDF_ERR_Error.

Example: Refer to WdfGetVDataType API.

WdfGetVDataType

Function: Get data type of waveform history block.

Format:
```
WDFResult WdfGetVDataType(WDFHandle handle, WDFTrace trace,
                          WDFHistoryBlock block, WDFVDataType* vDataType);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] vDataType	Data Type
wdfDataTypeUINT16	Unsigned 16bit (data size: 2byte)
wdfDataTypeSINT16	Signed 16bit (data size: 2byte)
wdfDataTypeLOGIC16	Logic 16bit (data size: 2byte)
wdfDataTypeUINT32	Unsigned 32bit (data size: 4byte)
wdfDataTypeSINT32	Signed 32bit (data size: 4byte)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get data type of waveform history block data. In case of getting waveform raw data (WdfGetScaleWave, WdfGetScaleWave64), acquired data should be converted base on data type.

Example:
```
//Show All Block Data Type
WDFTrace traceNumber;
WdfGetTraceNumber(wdfHandle, &traceNumber);
for(WDFTrace trace =0; trace<traceNumber; trace++){
    WDFHistoryBlock blockNumber;
    WdfGetTraceBlockNumber(wdfHandle, trace, &blockNumber);

    for(WDFHistoryBlock block=0; block<blockNumber; block++){
          WDFVDataType vDataType;
          WdfGetVDataType(wdfHandle, trace, block, &vDataType);
          printf('Trace:%d Block:%d DataType:%d \r\n", trace, block,
          vDataType);
    }
}
```

WdfGetVOffset

Function: Get vertical axis offset of waveform history block.

Format:
```
WDFResult WdfGetVOffset(WDFHandle handle, WDFTrace trace,
                        WDFHistoryBlock block, WDFDataParameter* vOffset);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] vOffset	Vertical Axis Offset

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get vertical axis offset of waveform history block. In case of getting waveform raw data(WdfGetScaleWave, WdfGetScaleWave64), acquired data should be converted base on vertical axis offset.

Example:
```
WDFDataParameter vOffset;
WdfGetVOffset(wdfHandle, trace, block, &vOffset);
```

WdfGetVResolution

Function: Get vertical axis resolution of waveform history block.

Format:
```
WDFResult WdfGetVResolution(WDFHandle handle, WDFTrace trace,
                            WDFHistoryBlock block, WDFDataParameter*
                            vResolution);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] vResolution	Vertical Axis Resolution

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get vertical axis resolution of waveform history block. In case of getting waveform raw data(WdfGetScaleWave, WdfGetScaleWave64), acquired data should be converted base on vertical axis resolution.

Example:
```
WDFDataParameter vResolution;
WdfGetVResolution(wdfHandle, trace, block, &vResolution);
```

WdfIsEnabledVIllegalData

Function: Judge whether hidden code is valid.

Format:
```
WDFResult WdfIsEnabledVIllegalData(WDFHandle handle, WDFTrace trace,
WDFBool * enable);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[OUT] enable	Judgement (TRUE or FALSE)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Check whether the value obtained by using the WdfGetVIllegalData API can be used as a hidden code.
If this function returns FALSE, the target waveform doesn't have a hidden code.
Therefore, the value obtained by using the WdfGetVIllegalData API cannot be used as a hidden code.

Example:
```
WDFBool vEnable;
WdfIsEnabledVIllegalData(wdfHandle, trace, &vEnable);
```

WdfGetVIllegalData

Function: Get invalid value of waveform history block.

Format:
```
WDFResult WdfGetVIllegalData(WDFHandle handle, WDFTrace trace,
                             WDFHistoryBlock block, WDFDataParameter*
                             vIllegalData);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] vIllegalData	Invalid(Hidden) Value

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get invalid (hidden) value of waveform history block. If the hidden code of the retrieved value is valid, the raw data retrieved using the WdfGetScaleWave or WdfGetScaleWave64 API is handled as hidden data.

Caution: With some waveform traces, the hidden code is invalid.
The WdfIsEnabledVIllegalData API can be used to check the validity of the hidden code.
If the WdfIsEnabledVIllegalData API returns invalid, do not handle the trace as hidden data.

Example:
```
WDFDataParameter vIllegalData;
WdfGetVIllegalData(wdfHandle, trace, block, &vIllegalData);
```

WdfGetVUnit

Function: Get vertical axis unit of waveform history block.

Format:
```
WDFResult WdfGetVUnit(WDFHandle handle, WDFTrace trace,
                      WDFHistoryBlock block, char* vUnit);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] vUnit	Vertical Axis Unit

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get vertical axis unit string(ex. “V”, “C”).

Example:
```
char vUnit[MAX_WDF_V_UNIT];
WdfGetVUnit(wdfHandle, trace, block, vUnit);
```

WdfGetHOffset

Function: Get horizontal axis offset of waveform history block.

Format:
```
WDFResult WdfGetHOffset(WDFHandle handle, WDFTrace trace,
                        WDFHistoryBlock block, WDFDataParameter* hOffset);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] hOffset	Horizontal Axis Data Start Offset

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get horizontal axis data start offset of waveform history block. In case of getting waveform raw data(WdfGetScaleWave, WdfGetScaleWave64), acquired data should be converted base on data start offset.
The absolute time of the data start point can be calculated from the measurement time, which can be obtained by using the WdfGetTime API.

Example:
```
WDFDataParameter hOffset;
WdfGetHOffset(wdfHandle, trace, block, &hOffset);
```

WdfGetHResolution

Function: Get horizontal axis resolution of waveform history block.

Format:
```
WDFResult WdfGetHResolution(WDFHandle handle, WDFTrace trace,
                            WDFHistoryBlock block, WDFDataParameter*
                            hResolution);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] hResolution	Horizontal Axis Resolution

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get horizontal axis resolution of waveform history block. In case of getting waveform raw data(WdfGetScaleWave, WdfGetScaleWave64), acquired data should be converted base on horizontal axis resolution.
Absolute time of specified data block can be calculated from horizontal axis data start offset and horizontal axis resolution.

Example:
```
WDFDataParameter hResolution;
WdfGetHResolution(wdfHandle, trace, block, &hResolution);
```

WdfGetHUnit

Function: Get horizontal axis unit of waveform history block.

Format:
```
WDFResult WdfGetHUnit(WDFHandle handle, WDFTrace trace,
                      WDFHistoryBlock block, char* hUnit);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] hUnit	Horizontal axis unit

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get horizontal axis unit string(ex. “s”, “Hz”).

Example:
```
char hUnit[MAX_WDF_H_UNIT];
WdfGetHUnit(wdfHandle, trace, block, hUnit);
```

WdfGetDate

Function: Get measuring date of waveform history block.

Format:
```
WDFResult WdfGetDate(WDFHandle handle, WDFTrace trace,
                     WDFHistoryBlock block, char* date);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] date	Measuring Date

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get measuring date of waveform history block. String format is “YYYY/MM/DD” (ex. 2010/09/14).

Example:
```
char date[MAX_WDF_DATE];
WdfGetDate(wdfHandle, trace, block, date);
```

WdfGetTime

Function: Get measuring time of waveform history block.

Format:
```
WDFResult WdfGetTime(WDFHandle handle, WDFTrace trace,
                     WDFHistoryBlock block, char* time);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] time	Measuring Time

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get measuring time of waveform history block. String format is “HH:MM:SS.uSecond” (ex. 16:50:43.012345).

Example:
```
char time[MAX_WDF_TIME];
WdfGetDate(wdfHandle, trace, block, time);
```

WdfGetBlockSize

Function: Get data block size of waveform history block.

Format:
```
WDFResult WdfGetBlockSize(WDFHandle handle, WDFTrace trace,
                          WDFHistoryBlock block, WDFDataBlock*
                          blockSize);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] blockSize	Data Block Size(Waveform Points)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get data block size(waveform points) of waveform history block. Waveform data position is zero-origin, and maximum value is “Data Block Size-1”.
Acquired data block size is utilized for specifying data points(WDFAccessParam::start, count) of waveform raw data.

Example:
```
WDFDataBlock blockSize;
WdfGetBlockSize(wdfHandle, trace, block, blockSize);
```

WdfGetBlockSize64

Function: Get data block size of waveform history block.

Format:
```
WDFResult WdfGetBlockSize64(WDFHandle handle, WDFTrace trace,
                            WDFHistoryBlock block, WDFDataBlock64*
                            blockSize);
```

Argument:

[IN] handle	API File Handle
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[OUT] blockSize	Data Block Size(Waveform Points)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error (Illegal argument)

Detail: Get data block size(waveform points) of waveform history block. Waveform data position is zero-origin, and maximum value is “Data Block Size-1”.
Acquired data block size is utilized for specifying data points(WDFAccessParam64::start, count) of waveform raw data.

Example:
```
WDFDataBlock64 blockSize;
WdfGetBlockSize64(wdfHandle, trace, block, blockSize);
```

WdfGetScaleWave

Function: Get waveform raw data.

Format:
```
WDFResult WdfGetScaleWave(WDFHandle handle, WDFAccessParam* param);
```

Argument:

[IN] handle	API File Handle
[IN] param	Acquiring Parameter
[IN] version	Constant Value(WDF_DEFAULT_ACSPRM_VERSION)
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[IN] start	Data Block Start Position
[IN] count	Required Data Points
[IN] ppRate	Constant Value(WDF_DEFAULT_ACSPRM_PPRATE)
[IN] waveType	Constant Value(WDF_DEFAULT_ACSPRM_WAVETYPE)
[IN] dataType	Constant Value(WDF_DEFAULT_ACSPRM_DATATYPE)
[OUT] cntOut	Acquired Data Points
[IN] dst	Data Output Buffer
[OUT] box	Not in Use
[IN] compMode	Constant Value(WDF_DEFAULT_ACSPRM_COMPMODE)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error
WDF_ERR_ParamBoundary	Illegal acquiring parameter

Detail: Get waveform raw data specified by parameter (param). If the data block start position (start) is set to ”0” and the required data points (count) is set to the block size acquired by using the WdfGetBlockSize API, all waveform data are obtained.
To allocate enough memory in case of long record length of measuring waveform, call this API repeatedly with specifying data block start position (start).
Acquired data points is assigned to argument (cntOut), and valid waveform data is obtained by accessing data output buffer (dst) based on acquired data points.

Acquired waveform data is stored in data output buffer (dst) as little endian order based on data block type (WDFVDataType) and data size of data block type.
Acquired data (vertical axis) needs to be converted to physical value.
The following indicates conversion equation.
Logic16bit(wdfDataTypeLOGIC16): Only low 8 bits are valid. No conversion is necessary.
Signed16bit(wdfDataTypeSINT16): Convert by the following equation.
Unsigned16bit(wdfDataTypeUINT16): Convert by the following equation.
Signed32bit(wdfDataTypeSINT32) : Converted by the following equation.
Unsigned32bit(wdfDataTypeUINT32): Converted by the following equation.

(Acquired Raw Data * Vertical Axis Resolution) + Vertical Axis Offset

Conversion equation of horizontal axis data is as follows.

(Data Block Position * Horizontal Axis Resolution) + Horizontal Axis Offset

Caution: Argument “version”, “ppRate”, “waveType”, “dataType”, and “compMode” should be assigned defined value (WDF_DEFAULT_ACSPRM_*). Other case is not covered under warranty.
As the buffer “dst”, enough memory allocation is needed. At least, buffer size should be more than (data size of waveform data type) * (required data points).
For the data size of the waveform data type, it is recommended that 4 bytes be assumed per data value.

Example:
```
// Get waveform raw-data.
typedef signed short INT16;
typedef unsigned short UINT16;
typedef signed long INT32;
typedef unsigned long UINT32;

WDFAccessParam param;
param.version  = WDF_DEFAULT_ACSPRM_VERSION;  // fixed
param.trace    = TargetTrace; // given
param.block    = TargetBlcok; // given
param.start    = StartCount;  // given
param.count    = ReadCount;   // given
param.ppRate   = WDF_DEFAULT_ACSPRM_PPRATE;   // fixed
param.waveType = WDF_DEFAULT_ACSPRM_WAVETYPE; // fixed
param.dataType = WDF_DEFAULT_ACSPRM_DATATYPE; // fixed
param.compMode = WDF_DEFAULT_ACSPRM_COMPMODE; // fixed
param.dst = malloc((size_t)param.count * 4); // MaxDataSize:4byte
if(!param.dst){
   return; // Error, Not Enough Memory
}
WDFResult result = WdfGetScaleWave(wdfHandle, &param);
```

```
if(result != 0){
    free(param.dst);
    return; // Error
}

// Get Horizontal waveform Infomations.
WDFDataParameter hOffset;
WdfGetHOffset(wdfHandle, param.trace, param.block, &hOffset);
WDFDataParameter hResolution;
WdfGetHResolution(wdfHandle, param.trace, param.block, &hResolution);

// Get Vertical waveform Infomations.
WDFDataParameter vResolution;
WdfGetVResolution(wdfHandle, param.trace, param.block, &vResolution);
WDFDataParameter vOffset;
WdfGetVOffset(wdfHandle, param.trace, param.block, &vOffset);
WDFDataParameter vIllegalData;
WdfGetVIllegalData(wdfHandle, param.trace, param.block, &vIllegalData);
WDFDataParameter vEnabledIllegalData;
WdfIsEnabledVIllegalData(wdfHandle, trace, &vEnabledIllegalData);

// Convert raw data and save text.
FILE fp = fopen('C:\\Output.csv", "w");
WDFVDataType vDataType;
WdfGetVDataType(wdfHandle, param.trace, param.block, &vDataType);
switch(vDataType){
   case wdfDataTypeSINT16:
      for(int i=0;i<param.cntOut/* read count */;i++){
         double outDataX = (param.start + i) * hResolution + hOffset;
         INT16 rawData = ((INT16*)param.dst)[i];
         if(vEnableIllegalData == True && rawData == vIllegalData){
            fprintf(fp, "%f, nan\r\n", outDataX); // Hidden Data
         }
         else{
            double outDataY = rawData * vResolution + vOffset;
            fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
         }
      }
      break;

   case wdfDataTypeLOGIC16:
      for(int i=0;i<param.cntOut/* read count */;i++){
         UINT16 outDataY = ((UINT16*)param.dst)[i];
         double outDataX = (param.start + i) * hResolution + hOffset;
            fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
   }
   break;
```

```
   case wdfDataTypeUINT16:
      for(int i=0;i<param.cntOut/* read count */;i++){
         double outDataX = (param.start + i) * hResolution + hOffset;
         UINT16 rawData = ((UINT16*)param.dst)[i];
         if (vEnableIllegalData == True && rawData == vIllegalData){
            fprintf(fp, "%f, nan\r\n", outDataX); // Hidden Data
         }
         else{
            double outDataY = rawData * vResolution + vOffset;
            fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
         }
      }
      break;

   case wdfDataTypeSINT32:
      for(int i=0;i<param.cntOut/* read count */;i++){
         double outDataX = (param.start + i) * hResolution + hOffset;
         INT32 rawData = ((INT32*)param.dst)[i]; // no illegaldata on
         4byte type
            double outDataY = rawData * vResolution + vOffset;
            fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
         }
      break;

   case wdfDataTypeUINT32:
      for(int i=0;i<param.cntOut/* read count */;i++){
         double outDataX = (param.start + i) * hResolution + hOffset;
         UINT32 rawData = ((UINT32*)param.dst)[i]; // no illegaldata on
         4byte type
            double outDataY = rawData * vResolution + vOffset;
            fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
         }
      break;

   default:
      break; // Unknown data type
}

fclose(fp);

free(param.dst);
```

WdfGetScaleWave64

Function: Get waveform raw data.

Format:
```
WDFResult WdfGetScaleWave64(WDFHandle handle, WDFAccessParam64* param);
```

Argument:

[IN] handle	API File Handle
[IN] param	Acquiring Parameter
[IN] version	Constant Value(WDF_DEFAULT_ACSPRM64_VERSION)
[IN] trace	Waveform Trace Number
[IN] block	History Block Number
[IN] start	Data Block Start Position
[IN] count	Required Data Points
[IN] ppRate	Constant Value(WDF_DEFAULT_ACSPRM64_PPRATE)
[IN] waveType	Constant Value(WDF_DEFAULT_ACSPRM64_WAVETYPE)
[IN] dataType	Constant Value(WDF_DEFAULT_ACSPRM64_DATATYPE)
[OUT] cntOut	Acquired Data Points
[IN] dst	Data Output Buffer
[OUT] box	Not in Use
[IN] compMode	Constant Value(WDF_DEFAULT_ACSPRM64_ COMPMODE)

Return Value:

WDF_OK	Success
WDF_ERR_InvalidHandle	Invalid handle
WDF_ERR_Error	Retrieval error
WDF_ERR_ParamBoundary	Illegal acquiring parameter

Detail: Get waveform raw data specified by parameter (param). If the data block start position (start) is set to ”0” and the required data points (count) is set to the block size acquired by using the WdfGetBlockSize64 API, all waveform data are obtained.
To allocate enough memory in case of long record length of measuring waveform, call this API repeatedly with specifying data block start position (start).
Acquired data points is assigned to argument (cntOut), and valid waveform data is obtained by accessing data output buffer (dst) based on acquired data points.

Acquired waveform data is stored in data output buffer (dst) as little endian order based on data block type (WDFVDataType) and data size of data block type.
Acquired data (vertical axis) needs to be converted to physical value.
The following indicates conversion equation.
Logic16bit(wdfDataTypeLOGIC16): Only low 8 bits are valid. No conversion is necessary.
Signed16bit(wdfDataTypeSINT16): Convert by the following equation.
Unsigned16bit(wdfDataTypeUINT16): Convert by the following equation.
Signed32bit(wdfDataTypeSINT32) : Converted by the following equation.
Unsigned32bit(wdfDataTypeUINT32): Converted by the following equation.

(Acquired Raw Data * Vertical Axis Resolution) + Vertical Axis Offset

Conversion equation of horizontal axis data is as follows.

(Data Block Position * Horizontal Axis Resolution) + Horizontal Axis Offset

Caution: Argument “version”, “ppRate”, “waveType”, “dataType”, and “compMode” should be assigned defined value (WDF_DEFAULT_ACSPRM64_*). Other case is not covered under warranty.
As the buffer “dst”, enough memory allocation is needed. At least, buffer size should be more than (data size of waveform data type) * (required data points).
For the data size of the waveform data type, it is recommended that 4 bytes be assumed per data value.

Example:

```
// Get waveform raw-data.
typedef signed short INT16;
typedef unsigned short UINT16;
typedef signed long INT32;
typedef unsigned long UINT32;

WDFAccessParam64 param;
param.version  = WDF_DEFAULT_ACSPRM64_VERSION;  // fixed
param.trace    = TargetTrace; // given
param.block    = TargetBlcok; // given
param.start    = StartCount;  // given
param.count    = ReadCount;   // given
param.ppRate   = WDF_DEFAULT_ACSPRM64_PPRATE;   // fixed
param.waveType = WDF_DEFAULT_ACSPRM64_WAVETYPE; // fixed
param.dataType = WDF_DEFAULT_ACSPRM64_DATATYPE; // fixed
param.compMode = WDF_DEFAULT_ACSPRM64_COMPMODE; // fixed
param.dst = malloc((size_t)param.count * 4); // MaxDataSize:4byte
if(!param.dst){
   return; // Error, Not Enough Memory
}
WDFResult result = WdfGetScaleWave64(wdfHandle, &param);

if(result != 0){
    free(param.dst);
    return; // Error
}

// Get Horizontal waveform Infomations.
WDFDataParameter hOffset;
WdfGetHOffset(wdfHandle, param.trace, param.block, &hOffset);
WDFDataParameter hResolution;
WdfGetHResolution(wdfHandle, param.trace, param.block, &hResolution);

// Get Vertical waveform Infomations.
WDFDataParameter vResolution;
WdfGetVResolution(wdfHandle, param.trace, param.block, &vResolution);
WDFDataParameter vOffset;
WdfGetVOffset(wdfHandle, param.trace, param.block, &vOffset);
WDFDataParameter vIllegalData;
WdfGetVIllegalData(wdfHandle, param.trace, param.block, &vIllegalData);
WDFDataParameter vEnabledIllegalData;
WdfIsEnabledVIllegalData(wdfHandle, trace, &vEnabledIllegalData);

// Convert raw data and save text.
FILE fp = fopen('C:\\Output.csv”, “w”);
WDFVDataType vDataType;
WdfGetVDataType(wdfHandle, param.trace, param.block, &vDataType);
switch(vDataType){
```

```
        case wdfDataTypeSINT16:
           for(int i=0;i<param.cntOut/* read count */;i++){
              double outDataX = (param.start + i) * hResolution + hOffset;
              INT16 rawData = ((INT16*)param.dst)[i];
              if(vEnableIllegalData == True && rawData == vIllegalData){
                 fprintf(fp, "%f, nan\r\n", outDataX); // Hidden Data
              }
              else{
                 double outDataY = rawData * vResolution + vOffset;
                 fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
              }
           }
           break;
        case wdfDataTypeLOGIC16:
           for(int i=0;i<param.cntOut/* read count */;i++){
              UINT16 outDataY = ((UINT16*)param.dst)[i];
              double outDataX = (param.start + i) * hResolution + hOffset;
                 fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
        }
        break;
        case wdfDataTypeUINT16:
           for(int i=0;i<param.cntOut/* read count */;i++){
              double outDataX = (param.start + i) * hResolution + hOffset;
              UINT16 rawData = ((UINT16*)param.dst)[i];
              if (vEnableIllegalData == True && rawData == vIllegalData){
                 fprintf(fp, "%f, nan\r\n", outDataX); // Hidden Data
              }
              else{
                 double outDataY = rawData * vResolution + vOffset;
                 fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
              }
           }
           break;
        case wdfDataTypeSINT32:
           for(int i=0;i<param.cntOut/* read count */;i++){
              double outDataX = (param.start + i) * hResolution + hOffset;
              INT32 rawData = ((INT32*)param.dst)[i]; // no illegaldata on
              4byte type
                 double outDataY = rawData * vResolution + vOffset;
                 fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
              }
           break;
        case wdfDataTypeUINT32:
           for(int i=0;i<param.cntOut/* read count */;i++){
              double outDataX = (param.start + i) * hResolution + hOffset;
              UINT32 rawData = ((UINT32*)param.dst)[i]; // no illegaldata on
              4byte type
                 double outDataY = rawData * vResolution + vOffset;
                 fprintf(fp, "%f, %f\r\n", outDataX, outDataY);
              }
           break;
        default:
           break; // Unknown data type
     }
     fclose(fp);
     free(param.dst);
```

4.4 DLL Linking Method

There are following two methods to link DLL.

(1) Implicit DLL Linking
(2) Explicit DLL Linking

In case of Implicit DLL Linking, specify import library (.lib) as the link library to use API.
To execute API, call DLL export function as in the case of calling other functions.

In case of Explicit DLL Linking, load DLL by “LoadLibrary”function explicitly.
To execute API, call export function with function pointer.
Refer to the header file defining function name and type definition of the function.

Definition Rule for Explicit DLL Linking

In Explicit DLL Linking (dynamic DLL loading), API name and type definition are needed.
They are defined base on the following description rule.

API Name Definition : WDF_FUNCNAME_ + “API Name”
API Type Definition : WDF_FUNC_ + “API NAME”

For example, to call API WdfCloseFile, the symbol of function type is ”WDF_FUNC_WdfCloseFile” and the symbol of function name is ”WDF_FUNCNAME_WdfCloseFile”.

The following is the sample code calling WdfCloseFile by dynamic DLL loading.

```
#include "WDFAPIPublic.h"
• • •
WDFResult WINAPI WdfCloseFile(WDFHandle* handle)
{
  WDF_FUNC_WdfCloseFile func =
       (WDF_FUNC_WdfCloseFile)GetProcAddress(hModule,
             WDF_FUNCNAME_WdfCloseFile);
return func(handle);
}
```

Refer to the sample code (WDFAPIWrapper.cpp) about how to use API.