

変位センサ

ZW-8000/7000/5000 シリーズ

ファイバ同軸変位センサ

通信ライブラリリファレンスマニュアル

形 ZW-8000□/7000□/5000□

Technology
Introduction
Guide

目次

1	改訂履歴	4
2	ソフトウェア使用許諾契約について	4
3	はじめに	5
3.1	はじめに	5
3.2	ご承諾事項	5
3.3	安全上の注意	8
3.4	安全上の要点	8
3.5	使用上の注意	8
3.6	法規と規格	8
3.7	著作権・商標について	8
3.8	関連マニュアル	9
4	動作環境	10
4.1	Windows	10
4.1.1	実行環境	10
4.1.1.1	Microsoft .NET Framework 4 Client Profile	10
4.2	MacOS	11
5	ファイル構成について	11
5.1.1	Windows	11
5.1.2	MacOS	11
6	組み込み方法	11
6.1	ファイル構成	11
6.1.1	C#	11
6.1.2	Swift	12
6.2	リンク	12
6.2.1	C#	12
6.2.1.1	参照	12
6.2.2	Swift	14
6.2.2.1	参照	14
7	型について	16
7.1	Windows	16
7.2	MacOS	16
8	定数、データクラスの構造定義	17
8.1	定数定義	17
8.2	データクラスの構造定義	22
8.3	デリゲートメソッドの I/F について	29
8.4	プロパティについて	30
9	関数	31

9.1	メソッド一覧	31
9.1.1	クラスに関するメソッド	31
9.1.2	センサコントローラとの通信経路確立／切断	31
9.1.3	システム制御	31
9.1.4	測定制御	32
9.1.5	設定変更／読み出し関連	32
9.1.6	測定結果の取得	32
9.1.7	内部ロギング機能関連	34
9.1.8	高速データ通信関連	34
9.2	メソッドリファレンス	35
9.2.1	クラスに関する操作	35
9.2.2	センサセンサコントローラとの通信経路確立／切断	36
9.2.3	システム制御	38
9.2.4	測定制御	41
9.2.5	設定変更／読み出し関連	45
9.2.6	測定結果の取得	51
9.2.7	内部ロギング機能関連	55
9.2.8	高速データ通信関連	63
10	共通コード	65
10.1	共通エラーコード	65
11	データ構造(詳細)	66
11.1	計測波形データ	66
11.2	フローデータ	68
12	サンプルプログラム	70
12.1	ユーザーインタフェース仕様	70
12.1.1	IP アドレス入力画面	70
12.1.2	メイン画面	71
12.2	サンプルソース(Windows)	73
12.2.1	通信確立	73
12.2.2	測定値の取得	73
12.2.3	バンクデータの取得・設定	74
12.3	サンプルソース(MacOS)	75
12.3.1	通信確立	75
12.3.2	測定値の取得	75
12.3.3	バンクデータの取得・設定	76

1 改訂履歴

改訂記号	改訂年月	改訂理由と改訂ページ
A	2016 年 4 月 1 日	初稿
B	2017 年 4 月 1 日	MacOS 対応版を追加。ライブラリの一部追加。
C	2018 年 2 月 22 日	誤記修正、ZW-8000 対応
D	2025 年 4 月 1 日	動作環境を更新

2 ソフトウェア使用許諾契約について

本契約は、オムロン株式会社（以下オムロンといいます）がお客様（以下使用者といいます）にソフトウェアを使用許諾する条件を定めたものです。

1. 本契約にいう「ソフトウェア」とは、以下記載のコンピュータ・プログラムおよびその関連する技術資料等のすべてを含みます。ソフトウェアおよびソフトウェアの複製物についての権原、所有権ならびに知的財産権はオムロンまたはオムロンに使用許諾をしている第三者に帰属し、本契約により使用者に移転することはありません。

ソフトウェア名：ZW-8000/7000/5000 シリーズ通信ライブラリ

2. オムロンは使用者に対して、ソフトウェアに関し以下の非独占的権利を許諾します。

(1)ソフトウェアを複製する権利

(2)前項により複製されたソフトウェアを使用者がオムロンの製品と組み合わせた使用者の製品（以下、対象製品といいます）において自ら使用する権利

(3)本条第 1 項により複製されたソフトウェアを対象製品とともに販売する場合に限り、使用者の顧客に配布する権利

3. 使用者は、前条に定める場合を除き、ソフトウェアをオムロンの事前の書面による同意なしに第三者に再使用許諾、譲渡、貸与またはリースすることはできません。

4. 使用者はソフトウェアの逆コンパイル、逆アセンブル、リバースエンジニアリングおよびそれに類する行為を行うことはできません。

5. 使用者は、ソフトウェアの内容について秘密として保持し、第三者に開示しないものとします。

6. オムロンはソフトウェアが付属のマニュアルと主要な点で一致して作動することを保証します。

7. オムロンは、ソフトウェアを現状有姿にして提供し、本契約またはソフトウェアの不具合等により発生した、使用者の直接的、間接的あるいは結果的損害に対して一切の責任を負いません。

8. オムロンはソフトウェアの使用によって生ずる第三者の知的財産権の侵害について一切の責任を負いません。

9. 使用者が本契約に違反した場合、オムロンは使用者に通知することによりソフトウェアの使用許諾を終了させることができます。その場合使用者はソフトウェアをオムロンに返却しなければなりません。

10. 本契約の成立・有効性・解釈・履行は日本法に基づいて行います。

11. 本契約に関連して生じる紛争・疑義は東京にて日本商事仲裁協会の規定に基づく仲裁によって終局的に解決するものとし、仲裁人の裁定は当事者を終局的に拘束します。

以 上

3 はじめに

3.1 はじめに

このたびは、ZW-8000/7000/5000シリーズをお買い上げいただきまして、まことにありがとうございます。

ZW-8000/7000/5000シリーズ通信ライブラリは、ユーザーアプリケーションからZW-8000/7000/5000シリーズを制御するための通信インターフェースを提供します(32bit/64bit DLL)。具体的な使用方法については、サンプルプログラムを参考にしてください。

このマニュアルでは、ZW-8000/7000/5000シリーズを使用する上で、必要な機能、性能、使用方法などの情報を記載しています。ZW-8000/7000/5000シリーズのご使用に際して、下記のことを守ってください。

- ・ZW-8000/7000/5000シリーズは電気の知識を有する専門家が扱ってください。
- ・このマニュアルをよくお読みになり、十分に理解のうえ、正しくご使用ください。
- ・このマニュアルはいつでも参照できるよう大切に保管ください。

このマニュアルの内容の一部または全部を無断で複写、複製、転載することを禁じます。

このマニュアルの内容に関しては、改良のため予告なしに仕様などを変更することがあります。あらかじめご了承ください。

このマニュアルの内容に関しては、万全を期しておりますが、万一ご不明な点や誤りなど、お気づきのことがありましたら、お手数ですが、弊社支店または営業所までご連絡ください。その際、巻末記載のMan.No.(マニュアルNo.)も併せてお知らせください。

3.2 ご承諾事項

「当社商品」について特別の合意がない場合には、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。

● 定義

本ご承諾事項中の用語の定義は次のとおりです。

- ・ 「当社商品」：「当社」のF Aシステム機器、汎用制御機器、センシング機器、電子・機構部品
- ・ 「カタログ等」：「当社商品」に関する、ベスト制御機器カタログ、電子・機構部品総合カタログ、その他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含まれます。
- ・ 「利用条件等」：「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取り扱い方法、利用上の注意、禁止事項その他
- ・ 「お客様用途」：「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み又は利用を含みます。
- ・ 「適合性等」：「お客様用途」での「当社商品」の（a）適合性、（b）動作、（c）第三者の知的財産の非侵害、（d）法令の遵守および（e）各種規格の遵守

● 記載事項のご注意

「カタログ等」の記載内容については次の点をご理解ください。

- ・ 定格値および性能値は、単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。
- ・ 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。
- ・ 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。
- ・ 「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

● ご利用にあたってのご注意

ご採用およびご利用に際しては次の点をご理解ください。

- ・ 定格・性能ほか「利用条件等」を遵守しご利用ください。
- ・ お客様ご自身にて「適合性等」をご確認いただき、「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」を一切保証いたしかねます。
- ・ 「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で、必ず事前に確認してください。
- ・ 「当社商品」をご使用の際には、（i）定格および性能に対し余裕のある「当社商品」のご利用、冗長設計などの安全設計、（ii）「当社商品」が故障しても、「お客様用途」の危険を最小にする安全設計、（iii）利用者に危険を知らせるための、安全対策のシステム全体としての構築、

(iv) 「当社商品」および「お客様用途」の定期的な保守、の各事項を実施してください。

- ・ 「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。ただし、次に掲げる用途であっても「当社」の意図した特別な商品用途の場合や特別の合意がある場合は除きます。
- a) 高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、娯楽設備、医用機器、安全装置、その他生命・身体に危険が及ぶ用途）
- b) 高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24 時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
- c) 厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
- d) 「カタログ等」に記載のない条件や環境での用途
- ・ 上記 a) から d) に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないでください。自動車搭載用商品については当社営業担当者にご相談ください。

● 保証条件

「当社商品」の保証条件は次のとおりです。

- ・ 保証期間：ご購入後 1 年間といたします。（ただし「カタログ等」に別途記載がある場合を除きます。）
- ・ 保証内容：故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
- a) 当社保守サービス拠点における故障した「当社商品」の無償修理（ただし、電子・機構部品については、修理対応は行いません。）
- b) 故障した「当社商品」と同数の代替品の無償提供
- ・ 保証対象外：故障の原因が次のいずれかに該当する場合は、保証いたしません。
- a) 「当社商品」本来の使い方以外のご利用
- b) 「利用条件等」から外れたご利用
- c) 本ご承諾事項「● ご利用にあたってのご注意」に反するご利用
- d) 「当社」以外による改造、修理による場合
- e) 「当社」以外の者によるソフトウェアプログラムによる場合
- f) 「当社」からの出荷時の科学・技術の水準では予見できなかった原因
- g) 上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

● 責任の制限

本ご承諾事項に記載の保証が、「当社商品」に関する保証のすべてです。

「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

● 輸出管理

「当社商品」または技術資料を、輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。

い。お客様が法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。

3.3 安全上の注意

安全上の注意の内容については、以下のマニュアルを参照してください。

参照：▶『ファイバ同軸変位センサZW-8000/7000/5000シリーズ ユーザーズマニュアル（SCEA-702）』の「安全上の注意」

3.4 安全上の要点

安全上の要点の内容については、以下のマニュアルを参照してください。

参照：▶『ファイバ同軸変位センサZW-8000/7000/5000シリーズ ユーザーズマニュアル（SCEA-702）』の「安全上の要点」

3.5 使用上の注意

使用上の注意の内容については、以下のマニュアルを参照してください。

参照：▶『ファイバ同軸変位センサZW-8000/7000/5000シリーズ ユーザーズマニュアル（SCEA-702）』の「使用上の注意」

3.6 法規と規格

法規と規格の内容については、以下のマニュアルを参照してください。

参照：▶『ファイバ同軸変位センサZW-8000/7000/5000シリーズ ユーザーズマニュアル（SCEA-702）』の「法規と規格」

3.7 著作権・商標について

・Sysmac は、オムロン株式会社製FA機器製品の日本およびその他の国における商標または登録商標です。

・Windows、Windows XP、Windows Vista、Windows 7、Windows 8、Windows8.1、Windows 10、Windows11、Windows Embeddedは米国 Microsoft Corporation の米国およびその他の国における登録商標です。・スクリーンショットはマイクロソフトの許可を得て使用しています。

- ・Mac OSは、米国および他の国々で登録されたApple Inc. の商標です。
- ・その他、本文中に記載しているシステム名および製品名は、各々各社の商標または登録商標です。

3.8 関連マニュアル

関連するマニュアルは、下表の通りです。併せてご覧ください。

マニュアル No.	マニュアル名	内容	おもな用途
SBCA-362	Sysmac Studio Version 1 オペレーションマニュアル	Sysmac Studio の操作方 法について説明します。	Sysmac Studio の操 作方法、機能について知 りたいとき
SCEA-702	ファイバ同軸変位センサ ZW-8000/7000/5000 シリーズ ユーザーズマニュアル	ファイバ同軸変位センサ ZW-8000/7000/5000 シリー ズのセンサコントローラの設 定方法について説明します。	ファイバ同軸変位セン サ ZW-8000/7000/5000 シリーズのセンサコントロ ーラの設定方法を知りた いとき
SCEA-703	ファイバ同軸変位センサ ZW-8000/7000/5000 シリーズ ユーザーズマニュアル 通信設定編	ファイバ同軸変位センサ ZW-8000/7000/5000 シリー ズのセンサコントローラの通 信方法について説明します。	ファイバ同軸変位セン サ ZW-8000/7000/5000 シリーズのセンサコントロ ーラの通信方法を知りた いとき

4 動作環境

4.1 Windows

DLL ソフトウェア Ver.4.000:

オペレーティングシステム (OS)	Windows10 (32bit 版/64bit 版)/ Windows11 (64bit 版)
CPU	Intel® Core™ i5-3320M (2.60GHz) 以上の CPU を搭載した Windows パソコン。 Intel® Core™ Ultra 5 135U (1.60GHz)、または相当以上を推奨。
メインメモリ	4GB 以上(推奨:16GB 以上)
ハードディスク	1.6GB 以上の空き容量
通信ポート	Ethernet ポート
対応言語	日本語、英語

DLL ソフトウェア Ver.3.001 (*1):

オペレーティングシステム (OS)	Windows 7 (32bit 版/ 64bit 版)/ Windows 8 (32bit 版/64bit 版)/ Windows 8.1 (32bit 版/64bit 版)/ Windows Embedded Standard 7 (32bit 版/ 64bit 版)/ Windows Embedded 8 Standard (32bit 版/ 64bit 版)
CPU	Intel® Celeron® 540(1.8GHz) 以上の CPU を搭載した Windows パソコン。 Intel® Core™ i5 M520(2.4GHz)、または相当以上を推奨。
メインメモリ	2GB 以上(推奨:4GB 以上)
ハードディスク	1.6GB 以上の空き容量
通信ポート	Ethernet ポート
対応言語	日本語、英語

*1: DLL ソフトウェア Ver.3.000 と動作内容に違いはありません。

4.1.1 実行環境

ZW-8000/7000/5000シリーズ通信ライブラリを利用したアプリケーションを実行する場合に必要な環境について記述します。

4.1.1.1 Microsoft .NET Framework 4 Client Profile

DLL の動作に必要なランタイムです。

Microsoft .NET Framework 4.6.2 および 4.8 Advanced Services で動作確認をしております。

dotNetFx40_Client_x86_x64.exe を実行してインストールしてください。

4.2 MacOS

DLL ソフトウェア Ver.3.001 (*1):

オペレーティングシステム (OS)	OS X 12 (64bit 版)
CPU	Intel® Core™ i5 M520 (2.0GHz)、または相当以上を推奨。
メインメモリ	8GB 以上。
ハードディスク	1.6GB 以上の空き容量
通信ポート	Ethernet ポート
対応言語	日本語、英語

*1: DLL ソフトウェア Ver.3.000 と動作内容に違いはありません。

5 ファイル構成について

5.1.1 Windows

DSComm.dll	DLL 本体です。
Source	サンプルソースのフォルダです。 C#で作成したサンプルプログラムのソースが格納されています。
Document	ドキュメントのフォルダです。 C#で作成したサンプルプログラムに関するドキュメントが格納されています。

5.1.2 MacOS

DSComm.framework	DLL 本体です。
Source	サンプルソースのフォルダです。 Swift で作成したサンプルプログラムのソースが格納されています。
Document	ドキュメントのフォルダです。 Swift で作成したサンプルプログラムに関するドキュメントが格納されています。

6 組み込み方法

6.1 ファイル構成

実行時に必要なファイルは次の通りです。

下記のファイルを実行ファイルと同じフォルダに配置してください。

6.1.1 C#

- DSCComm.dll

6.1.2 Swift

- DSCComm.framework

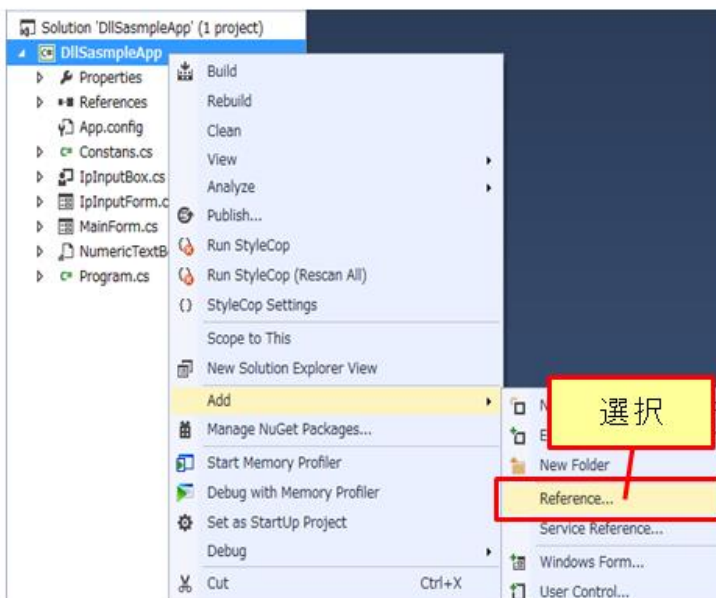
6.2 リンク

6.2.1 C#

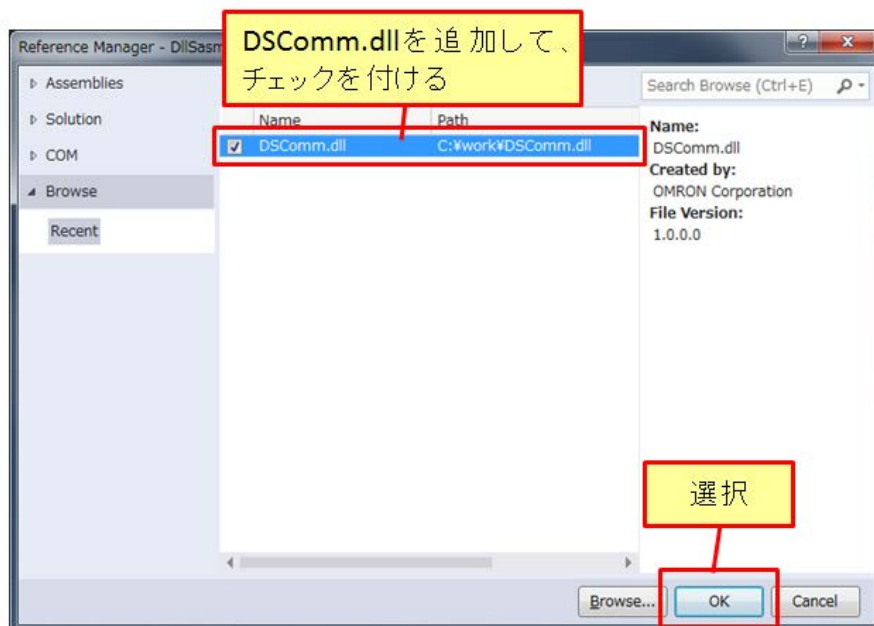
6.2.1.1 参照

プロジェクトの参照選定で、「DisplacementSensorSDK(DSCComm.dll)」を参照するようにしてください。

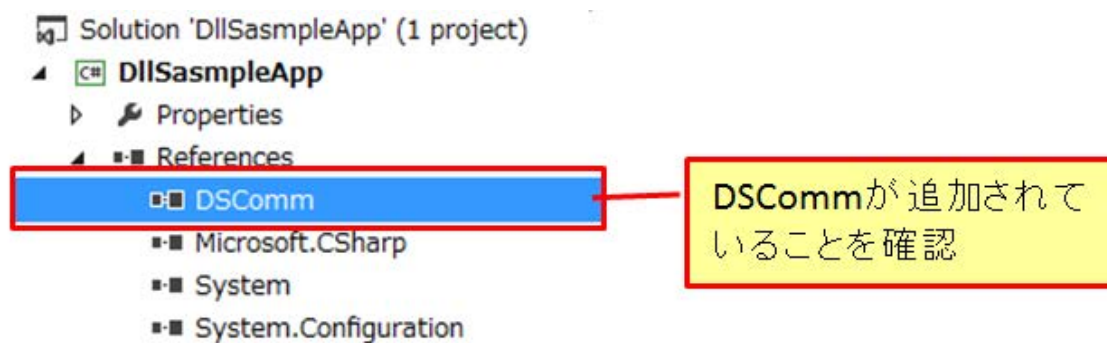
Step.1



Step.2



Step.3

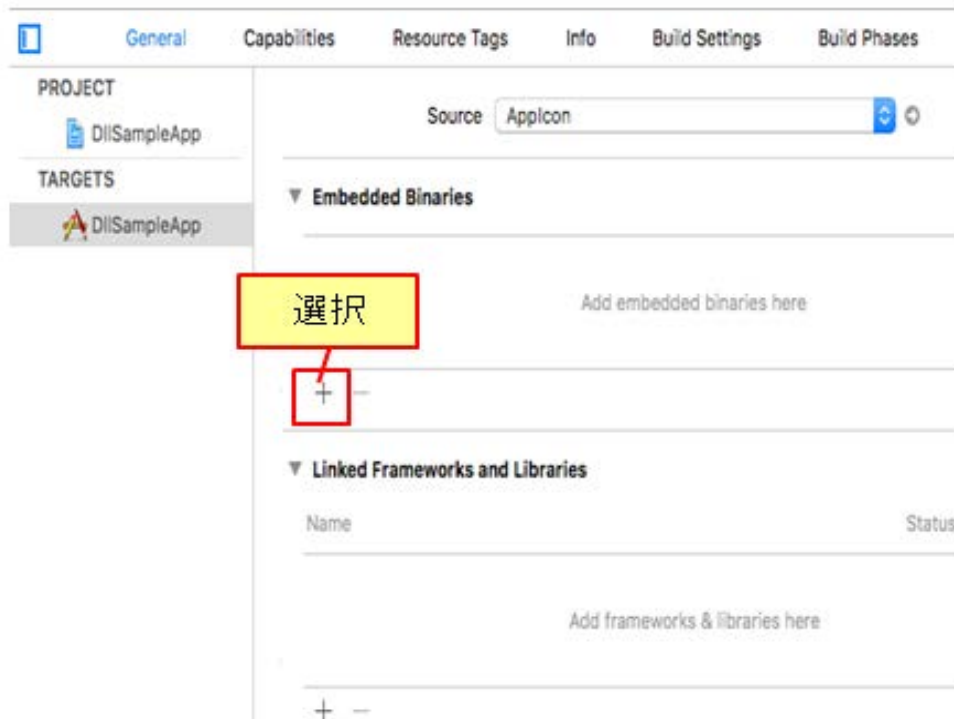


6.2.2 Swift

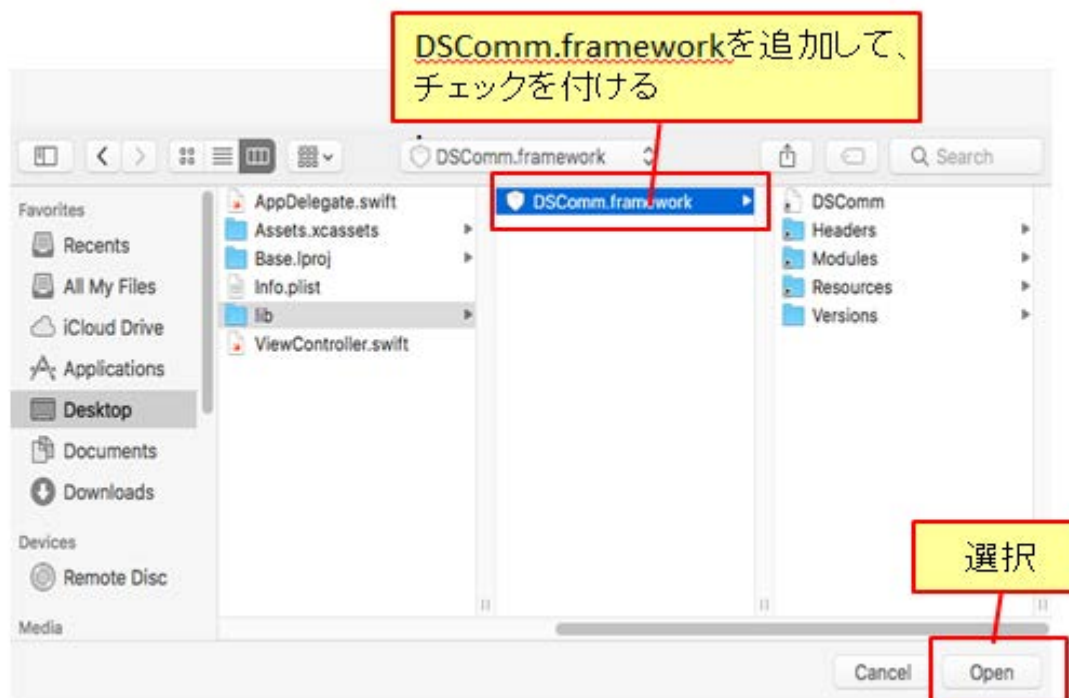
6.2.2.1 参照

プロジェクトの「General」 - 「Embedded Binaries」で、「DSComm.framework」を追加するようにしてください。

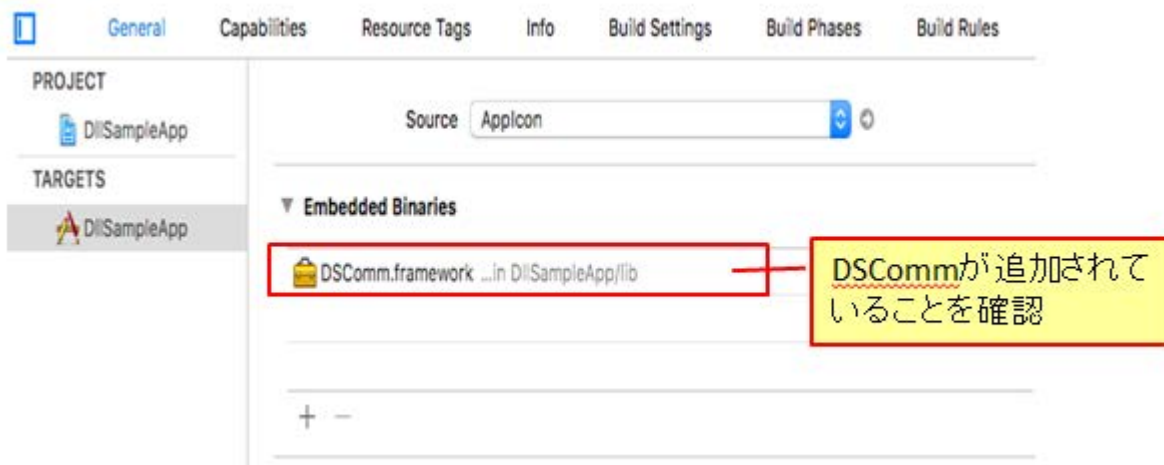
Step.1



Step.2



Step.3



7 型について

本文書内では、変数の型を以下の定義に沿って記載します。

7.1 Windows

bool	ブール値(true, false)
byte	符号なし8bit 整数
short	符号あり16bit 整数
ushort	符号なし16bit 整数
int	符号あり32bit 整数
uint	符号なし32bit 整数
string	Unicode 文字のシーケンス

7.2 MacOS

Bool	ブール値(true, false)
UInt8	符号なし8bit 整数
Int16	符号あり16bit 整数
UInt16	符号なし16bit 整数
Int32	符号あり32bit 整数
UInt32	符号なし32bit 整数
String	Unicode 文字のシーケンス

8 定数、データクラスの構造定義

8.1 定数定義

名前	バージョン番号の指定
定義(Windows)	<pre>enum Version { ZW2, };</pre>
定義(MacOS)	<pre>enum Version: Int32 { case ZW2 // ZW2 }</pre>
説明	DSCComm のインスタンスを作成する際に、対応するバージョンを指定するのに使用します。
備考	—

名前	タスク番号の指定
定義(Windows)	<pre>enum Task { T1 = 0, // TASK 1 T2 = 1, // TASK 2 T3 = 2, // TASK 3 T4 = 3, // TASK 4 ALL = 4, // TASK 1~4 };</pre>
定義(MacOS)	<pre>enum Task: Int32 { case T1 = 0, // TASK 1 case T2 = 1, // TASK 2 case T3 = 2, // TASK 3 case T4 = 3, // TASK 4 case ALL = 4, // TASK 1~4 }</pre>
説明	制御を行うメソッドで対象となるタスクを指定するのに使用します。
備考	—

名前	バンク番号の指定
定義(Windows)	<pre>enum Bank { B1 = 0, // BANK 1 B2 = 1, // BANK 2 B3 = 2, // BANK 3 B4 = 3, // BANK 4 }</pre>

	<pre> B5 = 4, // BANK 5 B6 = 5, // BANK 6 B7 = 6, // BANK 7 B8 = 7, // BANK 8 B9 = 8, // BANK 9 B10 = 9, // BANK 10 B11 = 10, // BANK 11 B12 = 11, // BANK 12 B13 = 12, // BANK 13 B14 = 13, // BANK 14 B15 = 14, // BANK 15 B16 = 15, // BANK 16 B17 = 16, // BANK 17 B18 = 17, // BANK 18 B19 = 18, // BANK 19 B20 = 19, // BANK 20 B21 = 20, // BANK 21 B22 = 21, // BANK 22 B23 = 22, // BANK 23 B24 = 23, // BANK 24 B25 = 24, // BANK 25 B26 = 25, // BANK 26 B27 = 26, // BANK 27 B28 = 27, // BANK 28 B29 = 28, // BANK 29 B30 = 29, // BANK 30 B31 = 30, // BANK 31 B32 = 31, // BANK 32 }; </pre>
定義 (MacOS)	<pre> enum Bank: Int32 { case B1 = 0, // BANK 1 case B2 = 1, // BANK 2 case B3 = 2, // BANK 3 case B4 = 3, // BANK 4 case B5 = 4, // BANK 5 case B6 = 5, // BANK 6 case B7 = 6, // BANK 7 case B8 = 7, // BANK 8 </pre>

	<pre> case B9 = 8, // BANK 9 case B10 = 9, // BANK 10 case B11 = 10, // BANK 11 case B12 = 11, // BANK 12 case B13 = 12, // BANK 13 case B14 = 13, // BANK 14 case B15 = 14, // BANK 15 case B16 = 15, // BANK 16 case B17 = 16, // BANK 17 case B18 = 17, // BANK 18 case B19 = 18, // BANK 19 case B20 = 19, // BANK 20 case B21 = 20, // BANK 21 case B22 = 21, // BANK 22 case B23 = 22, // BANK 23 case B24 = 23, // BANK 24 case B25 = 24, // BANK 25 case B26 = 25, // BANK 26 case B27 = 26, // BANK 27 case B28 = 27, // BANK 28 case B29 = 28, // BANK 29 case B30 = 29, // BANK 30 case B31 = 30, // BANK 31 case B32 = 31, // BANK 32 } </pre>
説明	バンクを操作する際に対象となるバンクを指定するのに使用します。
備考	—

名前	フラグの指定
定義(Windows)	<pre>enum Flag { OFF = 0, // OFF ON = 1, // ON };</pre>
定義(MacOS)	<pre>enum Flag: Int32 { case OFF = 0, // OFF case ON = 1, // ON }</pre>
説明	ON/OFF で制御する際に使用します。
備考	—

名前	領域の指定
定義(Windows)	<pre>enum Area { A1 = 0, // 領域 1 A2 = 1, // 領域 2 };</pre>
定義(MacOS)	<pre>enum Area: Int32 { case A1 = 0, // 領域 1 case A2 = 1, // 領域 2 }</pre>
説明	波形データ取得時に対象とする領域を指定するのに使用します。
備考	領域2 は、領域モードが「2領域モード」の場合に取得が可能です。

名前	出力データの指定
定義(Windows)	<pre>enum Out { 01 = 0, // OUT 1 02 = 1, // OUT 2 03 = 2, // OUT 3 04 = 3, // OUT 4 };</pre>
定義(MacOS)	<pre>enum Out: Int32 { case 01 = 0, // OUT 1 case 02 = 1, // OUT 2 case 03 = 2, // OUT 3 case 04 = 3, // OUT 4 }</pre>

説明	内部ロギング取得時の出力データ番号を指定する際に使用します。
備考	—

8.2 データクラスの構造定義

名前	計測波形情報
定義(Windows)	<pre> class MeasureWaveData { ushort BankNo; // バンク番号 byte AreaMode; // 2 領域モード ushort AreaNo; // 領域番号 int RecivedLight1; // 1 面目の受光量 int RecivedLight2; // 2 面目の受光量 int RecivedLight3; // 3 面目の受光量 int RecivedLight4; // 4 面目の受光量 ushort MeasurementValuePIX1; // 1 面目の計測値(PIX 値) ushort MeasurementValuePIX2; // 2 面目の計測値(PIX 値) ushort MeasurementValuePIX3; // 3 面目の計測値(PIX 値) ushort MeasurementValuePIX4; // 4 面目の計測値(PIX 値) ushort AreaStartPos; // 領域指定：開始座標 ushort AreaEndPos; // 領域指定：終了座標 ushort MaskAreaStartPos; // 領域指定：マスク領域開始 ushort MaskAreaEndPos; // 領域指定：マスク領域終了 ushort FlagAxisPos1; // グラフ軸座標 1(pix) ushort FlagAxisPos2; // グラフ軸座標 2(pix) ushort FlagAxisPos3; // グラフ軸座標 3(pix) ushort FlagAxisPos4; // グラフ軸座標 4(pix) ushort FlagAxisPos5; // グラフ軸座標 5(pix) uint MeasureRange; // 計測レンジ(nm) ushort MeasurementPeriod; // 測定周期 ushort LightPower; // 投光量 ushort RecivedLightAdjust; // 受光量 ushort CurrentOrVoltageValue; // 電流/電圧 DAC 値 byte CurrentOrVoltageValueState; // 電流/電圧状態 int AbsoluteDistance; // 絶対距離 int Task1Result; // TASK1 の測定結果(nm) int Task2Result; // TASK2 の測定結果(nm) int Task3Result; // TASK3 の測定結果(nm) int Task4Result; // TASK4 の測定結果(nm) int Task1Resolution; // TASK1 の分解能 int Task2Resolution; // TASK2 の分解能 int Task3Resolution; // TASK3 の分解能 </pre>

	<pre> int Task4Resolution; // TASK4 の分解能 int Task1UpperLimitValue; // TASK1 の判定上限値 int Task2UpperLimitValue; // TASK2 の判定上限値 int Task3UpperLimitValue; // TASK3 の判定上限値 int Task4UpperLimitValue; // TASK4 の判定上限値 int Task1LowerLimitValue; // TASK1 の判定下限値 int Task2LowerLimitValue; // TASK2 の判定下限値 int Task3LowerLimitValue; // TASK3 の判定下限値 int Task4LowerLimitValue; // TASK4 の判定下限値 byte ErrorNo; // エラー情報 int[] WaveDatas; // ラインブライトデータ }; </pre>
定義 (MacOS)	<pre> class MeasureWaveData { var BankNo: UInt16 // バンク番号 AreaMode: UInt8 // 2 領域モード AreaNo: UInt16 // 領域番号 RecivedLight1: Int32 // 1 面目の受光量 RecivedLight2: Int32 // 2 面目の受光量 RecivedLight3: Int32 // 3 面目の受光量 RecivedLight4: Int32 // 4 面目の受光量 MeasurementValuePIX1: UInt16 // 1 面目の計測値 (PIX 値) MeasurementValuePIX2: UInt16 // 2 面目の計測値 (PIX 値) MeasurementValuePIX3: UInt16 // 3 面目の計測値 (PIX 値) MeasurementValuePIX4: UInt16 // 4 面目の計測値 (PIX 値) AreaStartPos: UInt16 // 領域指定 : 開始座標 AreaEndPos: UInt16 // 領域指定 : 終了座標 MaskAreaStartPos: UInt16 // 領域指定 : マスク領域開始 MaskAreaEndPos: UInt16 // 領域指定 : マスク領域終了 FlagAxisPos1: UInt16 // グラフ軸座標 1 (pix) FlagAxisPos2: UInt16 // グラフ軸座標 2 (pix) FlagAxisPos3: UInt16 // グラフ軸座標 3 (pix) FlagAxisPos4: UInt16 // グラフ軸座標 4 (pix) FlagAxisPos5: UInt16 // グラフ軸座標 5 (pix) MeasureRange: UInt32 // 計測レンジ (nm) MeasurementPeriod: UInt16 // 測定周期 LightPower: UInt16 // 投光量 RecivedLightAdjust: UInt16 // 受光量 CurrentOrVoltageValue: UInt16 // 電流/電圧 DAC 値 } </pre>

	<pre> CurrentOrVoltageValueState: UInt8 // 電流/電圧状態 AbsoluteDistance: Int32 // 絶対距離 Task1Result: Int32 // TASK1 の測定結果 (nm) Task2Result: Int32 // TASK2 の測定結果 (nm) Task3Result: Int32 // TASK3 の測定結果 (nm) Task4Result: Int32 // TASK4 の測定結果 (nm) Task1Resolution: Int32 // TASK1 の分解能 Task2Resolution: Int32 // TASK2 の分解能 Task3Resolution: Int32 // TASK3 の分解能 Task4Resolution: Int32 // TASK4 の分解能 Task1UpperLimitValue: Int32 // TASK1 の判定上限値 Task2UpperLimitValue: Int32 // TASK2 の判定上限値 Task3UpperLimitValue: Int32 // TASK3 の判定上限値 Task4UpperLimitValue: Int32 // TASK4 の判定上限値 Task1LowerLimitValue: Int32 // TASK1 の判定下限値 Task2LowerLimitValue: Int32 // TASK2 の判定下限値 Task3LowerLimitValue: Int32 // TASK3 の判定下限値 Task4LowerLimitValue: Int32 // TASK4 の判定下限値 ErrorNo: UInt8 // エラー情報 WaveDatas: [Int32] // ラインブライトデータ } </pre>
説明	測定波形に関する情報。 ※ZW-7000/5000 用
備考	エラー情報 以下の情報を各ビットに付加する b0-2: 受光量エラー (0: 光量安定 1: 光量調整中 2: 光量飽和 3: 光量不足 4: LIGHT OFF 状態 5: 相互干渉防止 OFF) b3: システムエラー b4: 負荷短絡検知(0 固定) b5: 面数エラー b6: STAB 状態 b7: センサヘッド校生エラー ラインブライトデータ 配列数は、256 である

名前	計測波形情報2
定義(Windows)	<pre> class MeasureWaveData2 { ushort BankNo; // バンク番号 byte AreaMode; // 2 領域モード ushort AreaNo; // 領域番号 int RecivedLight1; // 1 面目の受光量 int RecivedLight2; // 2 面目の受光量 int RecivedLight3; // 3 面目の受光量 int RecivedLight4; // 4 面目の受光量 ushort MeasurementValuePIX1; // 1 面目の計測値(PIX 値) ushort MeasurementValuePIX2; // 2 面目の計測値(PIX 値) ushort MeasurementValuePIX3; // 3 面目の計測値(PIX 値) ushort MeasurementValuePIX4; // 4 面目の計測値(PIX 値) ushort AreaStartPos; // 領域指定：開始座標 ushort AreaEndPos; // 領域指定：終了座標 ushort MaskAreaStartPos; // 領域指定：マスク領域開始 ushort MaskAreaEndPos; // 領域指定：マスク領域終了 ushort FlagAxisPos1; // グラフ軸座標 1(pix) ushort FlagAxisPos2; // グラフ軸座標 2(pix) ushort FlagAxisPos3; // グラフ軸座標 3(pix) ushort FlagAxisPos4; // グラフ軸座標 4(pix) ushort FlagAxisPos5; // グラフ軸座標 5(pix) uint MeasureRange; // 計測レンジ(nm) ushort MeasurementPeriod; // 測定周期 ushort LightPower; // 投光量 ushort RecivedLightAdjust; // 受光量 ushort CurrentOrVoltageValue; // 電流/電圧 DAC 値 byte CurrentOrVoltageValueState; // 電流/電圧状態 int AbsoluteDistance; // 絶対距離 int Task1Result; // TASK1 の測定結果(nm) int Task2Result; // TASK2 の測定結果(nm) int Task3Result; // TASK3 の測定結果(nm) int Task4Result; // TASK4 の測定結果(nm) int Task1Resolution; // TASK1 の分解能 int Task2Resolution; // TASK2 の分解能 int Task3Resolution; // TASK3 の分解能 int Task4Resolution; // TASK4 の分解能 int Task1UpperLimitValue; // TASK1 の判定上限値 </pre>

	<pre> int Task2UpperLimitValue; // TASK2 の判定上限値 int Task3UpperLimitValue; // TASK3 の判定上限値 int Task4UpperLimitValue; // TASK4 の判定上限値 int Task1LowerLimitValue; // TASK1 の判定下限値 int Task2LowerLimitValue; // TASK2 の判定下限値 int Task3LowerLimitValue; // TASK3 の判定下限値 int Task4LowerLimitValue; // TASK4 の判定下限値 ushort ErrorNo; // エラー情報 ushort CenterPosition1; // 1面トラック領域中心位置 ushort CenterPosition2; // 2面トラック領域中心位置 ushort CenterPosition3; // 3面トラック領域中心位置 ushort CenterPosition4; // 4面トラック領域中心位置 int[] WaveDatas; // ラインブライต์データ }; </pre>
定義 (MacOS)	<pre> class MeasureWaveData { var BankNo: UInt16 // バンク番号 AreaMode: UInt8 // 2領域モード AreaNo: UInt16 // 領域番号 RecivedLight1: Int32 // 1面目の受光量 RecivedLight2: Int32 // 2面目の受光量 RecivedLight3: Int32 // 3面目の受光量 RecivedLight4: Int32 // 4面目の受光量 MeasurementValuePIX1: UInt16 // 1面目の計測値(PIX 値) MeasurementValuePIX2: UInt16 // 2面目の計測値(PIX 値) MeasurementValuePIX3: UInt16 // 3面目の計測値(PIX 値) MeasurementValuePIX4: UInt16 // 4面目の計測値(PIX 値) AreaStartPos: UInt16 // 領域指定: 開始座標 AreaEndPos: UInt16 // 領域指定: 終了座標 MaskAreaStartPos: UInt16 // 領域指定: マスク領域開始 MaskAreaEndPos: UInt16 // 領域指定: マスク領域終了 FlagAxisPos1: UInt16 // グラフ軸座標 1(pix) FlagAxisPos2: UInt16 // グラフ軸座標 2(pix) FlagAxisPos3: UInt16 // グラフ軸座標 3(pix) FlagAxisPos4: UInt16 // グラフ軸座標 4(pix) FlagAxisPos5: UInt16 // グラフ軸座標 5(pix) MeasureRange: UInt32 // 計測レンジ (nm) MeasurementPeriod: UInt16 // 測定周期 LightPower: UInt16 // 投光量 } </pre>

	<pre> RecvLightAdjust: UInt16 // 受光量 CurrentOrVoltageValue: UInt16 // 電流/電圧 DAC 値 CurrentOrVoltageValueState: UInt8 // 電流/電圧状態 AbsoluteDistance: Int32 // 絶対距離 Task1Result: Int32 // TASK1 の測定結果 (nm) Task2Result: Int32 // TASK2 の測定結果 (nm) Task3Result: Int32 // TASK3 の測定結果 (nm) Task4Result: Int32 // TASK4 の測定結果 (nm) Task1Resolution: Int32 // TASK1 の分解能 Task2Resolution: Int32 // TASK2 の分解能 Task3Resolution: Int32 // TASK3 の分解能 Task4Resolution: Int32 // TASK4 の分解能 Task1UpperLimitValue: Int32 // TASK1 の判定上限値 Task2UpperLimitValue: Int32 // TASK2 の判定上限値 Task3UpperLimitValue: Int32 // TASK3 の判定上限値 Task4UpperLimitValue: Int32 // TASK4 の判定上限値 Task1LowerLimitValue: Int32 // TASK1 の判定下限値 Task2LowerLimitValue: Int32 // TASK2 の判定下限値 Task3LowerLimitValue: Int32 // TASK3 の判定下限値 Task4LowerLimitValue: Int32 // TASK4 の判定下限値 ErrorNo: UInt8 // エラー情報 WaveDatas: [Int32] // ラインブライトデータ } </pre>
説明	測定波形に関する情報。 ※ZW-8000 用
備考	エラー情報 以下の情報を各ビットに付加する b0-2: 受光量エラー (0: 光量安定 1: 光量調整中 2: 光量飽和 3: 光量不足 4: LIGHT OFF 状態 5: 相互干渉防止 OFF) b3: システムエラー b4: 負荷短絡検知(0 固定) b5: 面数エラー b6: STAB 状態 b7: センサヘッド校正エラー b8: 1 面トラックエラー b9: 2 面トラックエラー b10: 3 面トラックエラー

	b11:4 面トラックエラー b12:1 面トラック受光波形エラー b13:2 面トラック受光波形エラー b14:3 面トラック受光波形エラー b15:4 面トラック受光波形エラー ラインブライต์データ 配列数は、1024 である
--	---

名前	フローデータ
定義(Windows)	<pre> class FlowData { uint OutNo; // OUT 番号 bool Timing; // パラレル入力 : TIMING bool Reset; // パラレル入力 : RESET bool LEDOff; // パラレル入力 : LEDOFF bool Zero; // パラレル入力 : ZERO bool Logging; // パラレル入力 : LOGGING bool Sync; // パラレル入力 : SYNC bool Busy; // パラレル出力 : Busy bool Enable; // パラレル出力 : Enable bool Low; // パラレル出力 : Low bool Pass; // パラレル出力 : Pass bool High; // パラレル出力 : High bool TaskStat; // パラレル出力 : TASKSTAT bool LogStat; // パラレル出力 : LOGSTAT bool LogErr; // パラレル出力 : LOGERR bool SyncFlg; // パラレル出力 : SYNCFLG bool Stability; // パラレル出力 : STABILITY bool BufferErr; // 高速データ通信あふれ bit bool FlowStop; // 高速データ通信停止 bit int MeasureData; // 計測データ }; </pre>
定義(MacOS)	<pre> class FlowData { OutNo: UInt32 // OUT 番号 Timing: Bool // パラレル入力 : TIMING Reset: Bool // パラレル入力 : RESET LEDOff: Bool // パラレル入力 : LIGHT OFF Zero: Bool // パラレル入力 : ZERO </pre>

	<pre> Logging: Bool // パラレル入力 : LOGGING Sync: Bool // パラレル入力 : SYNC Busy: Bool // パラレル出力 : Busy Enable: Bool // パラレル出力 : Enable Low: Bool // パラレル出力 : Low Pass: Bool // パラレル出力 : Pass High: Bool // パラレル出力 : High TaskStat: Bool // パラレル出力 : TASKSTAT LogStat: Bool // パラレル出力 : LOGSTAT LogErr: Bool // パラレル出力 : LOGERR SyncFlg: Bool // パラレル出力 : SYNCFLG Stability: Bool // パラレル出力 : STABILITY BufferErr: Bool // 高速データ通信あふれ bit FlowStop: Bool // 高速データ通信停止 bit MeasureData: Int32 // 計測データ </pre>
説明	フローデータに関する情報。
備考	計測データ 計測データの単位は、小数点位置情報(DecimalInfo)の値により変わります。 false: nm(ナノメートル) true: μm(マイクロメートル)

8.3 デリゲートメソッドの I/F について

書式 (Windows)	void DisconnectDelegate()
書式 (MacOS)	<pre> protocol DisconnectDelegate { func DisconnectDelegate () } </pre>
パラメータ	なし
戻り値	—
解説	センサコントローラとの通信が切断された時に呼び出すメソッド
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

書式 (Windows)	void LoggingDataDelegate(List<FlowData> flowDataList)
書式 (MacOS)	<pre> protocol LoggingDataDelegate { func LoggingDataDelegate (flowDataList[FlowData]) } </pre>

パラメータ	flowDataList タスク毎のフローデータ
戻り値	—
解説	定期的に出力された測定値を受け取った時に呼び出すメソッド
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

8.4 プロパティについて

名前	HardwareType
説明	本プログラムで接続している実機の形式を取得するのに用いる。 オフライン時： “” オンライン時：接続中の実機の形式(例：“ZW-8000”)
備考	インスタンスでの設定は不可である。

9 関数

9.1 メソッド一覧

9.1.1 クラスに関するメソッド

センサコントローラがシステムエラー状態であっても正常に処理されます。

メソッド名	概要
DSComm	コンストラクタ
Dispose	オブジェクトの破棄

9.1.2 センサコントローラとの通信経路確立／切断

センサコントローラがシステムエラー状態であっても正常に処理されます。

メソッド名	概要
Open	Ethernet での接続を確立する
Close	接続を切断する

9.1.3 システム制御

ReturnToFactorySetting を除き、センサコントローラがシステムエラー状態であっても正常に処理されます。

ReturnToFactorySetting は、システムエラー状態では失敗することがあります（センサヘッドが接続されていない場合など）。

メソッド名	概要
RebootController	センサコントローラを再起動する
ReturnToFactorySetting	センサコントローラを工場出荷状態に戻す
GetSoftwareVersion	センサコントローラのバージョンを取得する
GetSensorSerialNumber	センサヘッドのシリアル情報を取得する
GetSensorName	センサコントローラ名を取得する
SetSensorName	センサコントローラ名を設定する
GetError	センサコントローラのシステムエラー番号を取得する

9.1.4 測定制御

センサコントローラがシステムエラー状態の場合、処理に失敗します。

メソッド名	概要
ZeroReset	ゼロリセットを発行する
Timing	タイミングを発行する
Reset	リセットを発行する
ClearMemory	内部メモリを初期化する
TurnLight	測定光を消灯/点灯する
CalibrationSensor	センサヘッドの校正を実行する

9.1.5 設定変更／読み出し関連

センサコントローラがシステムエラー状態の場合、処理に失敗します。

メソッド名	概要
GetSystemData	センサコントローラのシステムデータを取得する
SetSystemData	センサコントローラのシステムデータへ設定値を送信する
GetBankData	センサコントローラのバンクデータを取得する
SetBankData	センサコントローラのバンクデータへ設定値を送信する
GetBackupData	コントローラの設定データを一括で取得する。
SetBackupData	コントローラの設定データへ一括で送信する。
InitializeSetting	センサコントローラの設定値を初期化する
InitializeCurrentBankSetting	カレントバンクの設定値を初期化する
SaveSettings	設定書込領域の内容を、動作中設定領域や保存用領域へ反映させる
CopyBank	カレントバンクをコピーする
GetActiveBank	カレントバンクを取得する
ChangeActiveBank	カレントバンクを切り替える

9.1.6 測定結果の取得

センサコントローラがシステムエラー状態の場合、処理に失敗します。

メソッド名	概要
GetMeasurementValue	測定値を取得する
GetJudgementValue	判定結果を取得する
GetMeasureWaveData	計測波形を取得する

GetRawImageData	受光波形を取得する
-----------------	-----------

9.1.7 内部ロギング機能関連

センサコントローラがシステムエラー状態の場合、処理に失敗します。

メソッド名	概要
StartStorage	内部ロギングを開始する
StopStorage	内部ロギングを停止する
GetStorageStatus	ス内部ロギングの状態を取得する
GetStorageData	内部ロギングされた計測値を取得する

9.1.8 高速データ通信関連

センサコントローラがシステムエラー状態の場合、処理に失敗します。

メソッド名	概要
PreStartHighSpeedDataCommunication	高速データ通信開始前の準備をする
StartHighSpeedDataCommunication	高速データ通信を開始する
StopHighSpeedDataCommunication	高速データ通信を停止する
SingleHighSpeedDataCommunication	高速データ通信を開始する(単発)

9.2 メソッドリファレンス

9.2.1 クラスに関する操作

エラー発生のある可能性がある関数の戻り値は全てint型です。正常時は0 (OK)を返し、リターンコードは共通エラーコードとして表現されます。

関数に共通のリターンコードについては「9.1 共通エラーコード」を参照してください。

■ コンストラクタ

書式 (Windows)	DSCComm (Version version)
書式 (MacOS)	init (version: Version)
パラメータ	version (in) 接続する変位センサに対応するバージョン
戻り値	DSCComm のインスタンス
解説	コンストラクタ
対応バージョン	全て

■ オブジェクトの破棄

書式 (Windows)	void Dispose()
書式 (MacOS)	func Dispose()
パラメータ	—
戻り値	—
解説	オブジェクトの破棄 アンマネージリソースだけを解放する。
対応バージョン	全て

9.2.2 センサセンサコントローラとの通信経路確立／切断

■ Ethernet 通信

書式 (Windows)	int Open(byte[] ipAddress, DisconnectDelegate method)
書式 (MacOS)	func Open(ipAddress: [UInt8]) -> Int32
パラメータ	ipAddress (in) 接続先の IP アドレスを指定します。オクテット毎に設定する。 <Windows の場合> ※ 例) 192.168.250.50 ⇒ new byte[] {0xC0, 0xA8, 0xFA, 0x32}; <MacOS の場合> ※ 例) 192.168.250.50 ⇒ [UInt8] = [0xC0, 0xA8, 0xFA, 0x32] method (in) 通信切断時の呼び先メソッド <Windows の場合> delegate void DisconnectDelegate(); <MacOS の場合> func DisconnectDelegate()
戻り値	OK ERR_CONNECT ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION
解説	Ethernet で接続された変位センサとの通信が行えるように、接続を確立します。 計測波形取得メソッド (GetMeasureWaveData) の処理に 200ms 以上を要する場合には、TCP 遅延 ACK の影響を受けている場合があります。その際は、ソケットの受信バッファサイズを変更することで TCP 遅延 ACK の影響を回避することが可能です。 受信バッファサイズの変更は、DSComm.dll と同じ階層に DSComm.ini ファイルを作成して設定値を記載します。 例) rcvBuffSize_WaveData=1024 既定値: 512bytes(ZW-7000/5000)、752bytes(ZW-8000)
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 通信経路切断

書式 (Windows)	int Close()
書式 (MacOS)	func Close() -> Int32
パラメータ	—
戻り値	OK ERR_APPLICATION
解説	Ethernet の接続を切断します。 接続未確立の状態でも呼び出してもエラーとはなりません。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

9.2.3 システム制御

■ センサコントローラ再起動

書式 (Windows)	int RebootController()
書式 (MacOS)	func RebootController() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	センサコントローラを再起動します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 工場出荷状態へ戻す

書式 (Windows)	int RetrunToFactorySetting()
書式 (MacOS)	func ReturnToFactorySetting() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	センサコントローラの全ての設定値を、工場出荷状態へと戻します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ バージョン取得

書式 (Windows)	int GetSoftwareVersion(out string version)
書式 (MacOS)	func GetSoftwareVersion(version: inout String) -> Int32
パラメータ	version (out) センサコントローラのバージョン情報 (8bytes) “ 2 . 0 0 0 ”
戻り値	OK ERR_COMMUNICATION ERR_PARAM

	ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	バージョンのフル表記を取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ センサヘッドシリアル情報取得

書式 (Windows)	int GetSensorSerialNumber(out string serialNo)
書式 (MacOS)	func GetSensorSerialNumber(serialNo: inout String) -> Int32
パラメータ	serialNo (out) センサヘッド情報 (8bytes) “ 0 1 2 3 4 5 6 ”
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	センサヘッドのシリアル情報を取得します
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ センサコントローラ名取得

書式 (Windows)	int GetSensorName(out string sensorName)
書式 (MacOS)	func GetSensorName(sensorName: inout String) -> Int32
パラメータ	sensorName (out) 変位センサの名称 (最大 32bytes) “ Z W - 7 0 0 0 ”
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	センサコントローラ名を取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ センサコントローラ名設定

書式 (Windows)	int SetSensorName(string sensorName)
書式 (MacOS)	func SetSensorName(sensorName: String) -> Int32
パラメータ	sensorName (in) 変位センサの名称 (最大 32bytes の文字列)
戻り値	OK

	ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_APPLICATION ERR_RUN_MODE
解説	センサコントローラ名を設定します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ システムエラー番号取得

書式 (Windows)	int GetError(out ushort systemErrorNum)
書式 (MacOS)	func GetError(systemErrorNum: inout UInt16) -> Int32
パラメータ	systemErrorNum (out) システムエラー番号
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	センサコントローラのシステムエラー番号を取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

9.2.4 測定制御

■ ゼロリセット発行

書式 (Windows)	int ZeroReset(Flag flag, Task task)
書式 (MacOS)	func ZeroReset(flag: Flag, task: Task) -> Int32
パラメータ	flag (in) ON: ゼロリセット要求、OFF: ゼロリセットの解除要求 task (in) 処理対象とするタスク 有効値: T1～T4、ALL
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION

解説	ゼロリセット要求を発行します。 処理対象タスクが測定しない設定の場合であってもエラーとはなりません
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ タイミング発行

書式 (Windows)	int Timing(flag: Flag)
書式 (MacOS)	func Timing(flag: Flag) -> Int32
パラメータ	flag (in) ON: タイミング ON 要求、OFF: OFF 要求
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	タイミング要求を発行します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ リセット発行

書式 (Windows)	int Reset(flag: Flag)
書式 (MacOS)	func Reset(flag: Flag) -> Int32
パラメータ	flag (in) ON: リセット ON 要求、OFF: OFF 要求
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	リセット要求を発行します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 内部メモリクリア

書式 (Windows)	int ClearMemory()
書式 (MacOS)	func ClearMemory() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE

	ERR_APPLICATION
解説	内部メモリを初期化します
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 測定光点灯

書式 (Windows)	int TurnLight(Flag flag)
書式 (MacOS)	func TurnLight(flag: Flag) -> Int32
パラメータ	flag (in) ON: 測定光の点灯要求、OFF: 消灯要求
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	測定光が発射される LED を点灯/消灯します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ センサヘッド校正

書式 (Windows)	int CalibrateSensor()
書式 (MacOS)	func CalibrateSensor() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	測定センシングにより、センサヘッドを校正します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

9.2.5 設定変更／読み出し関連

■ システムデータの設定値取得

書式 (Windows)	int GetSystemData(int dataNo, out int value)
書式 (MacOS)	func GetSystemData(dataNo: Int32, value: inout Int32) -> Int32
パラメータ	dataNo (in) データ番号 value (out) 取得した値を返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	システムデータの指定した項目をセンサコントローラから取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ システムデータの設定値送信

書式 (Windows)	int SetSystemData(int dataNo, int value)
書式 (MacOS)	func SetSystemData(dataNo: Int32, value: Int32) -> Int32
パラメータ	dataNo (in) データ番号 value (in) 変更する値
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	システムデータの指定した項目に設定値を送信します。 電源が切断すると設定した値は保持されないため、設定データをセンサコントローラ内部メモリに保存する必要があります。(SaveSettings を利用してください)
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ バンクデータの設定値取得

書式 (Windows)	int GetBankData(int unitNo, int dataNo, out int value)
書式 (MacOS)	func GetBankData(unitNo: Int32, dataNo: Int32, value: inout Int32) -> Int32
パラメータ	unitNo (in) ユニット番号 dataNo (in) データ番号 value (out) 取得した値を返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	バンクデータの指定した項目をセンサコントローラから取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ バンクデータの設定値送信

書式 (Windows)	int SetBankData(int unitNo, int dataNo, int value)
書式 (MacOS)	func SetBankData(unitNo: Int32, dataNo: Int32, value: Int32) -> Int32
パラメータ	unitNo (in) ユニット番号 dataNo (in) データ番号 ※ ユニット番号、データ番号の詳細は「10.2 処理項目データ一覧」を参照してください。 value (in) 変更する値
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	バンクデータの指定した項目に設定値を送信します。 電源が切断すると設定した値は保持されないため、設定データをセンサコントローラ内部メモリに保存する必要があります。(SaveSettings を利用してください)

対応バージョン	ZW-7000/5000 シリーズ ver.200 以降
---------	------------------------------

■ 設定データの一括取得

書式 (Windows)	int GetBackupData(out byte[] binaryData)
書式 (MacOS)	func GetBackupData(binaryData: inout [UInt8]) -> Int32
パラメータ	binaryData (out) 取得した設定データのバイナリデータを返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	設定データのバイナリデータをコントローラーから一括で取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 設定データの一括送信

書式 (Windows)	int SetBackupData(byte[] binaryData)
書式 (MacOS)	func SetBackupData(binaryData: [UInt8]) -> Int32
パラメータ	binaryData (in) 送信する設定データのバイナリデータ
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	設定データのバイナリデータをコントローラーへ一括で送信します。 電源が切断すると設定した値は保持されないため、設定データをコントローラー内部メモリに保存する必要があります。(SaveSettings を利用してください) 設定データのバージョンがコントローラーのバージョンより新しい場合、または設定データ型式がコントローラーの形式と違う場合は、設定パラメータのエラー (ERR_PARAM) となります。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 設定値初期化

書式 (Windows)	int InitializeSetting()
書式 (MacOS)	func InitializeSetting() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	すべての設定を初期化します
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ カレントバンク初期化

書式 (Windows)	int InitializeCurrentBankSetting()
書式 (MacOS)	func InitializeCurrentBankSetting() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	現在のバンクデータを初期化します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 設定値保存

書式 (Windows)	int SaveSettings()
書式 (MacOS)	func SaveSettings() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	全ての設定値を内部メモリに保存します。

対応バージョン	ZW-7000/5000 シリーズ ver.200 以降
---------	------------------------------

■ カレントバンクコピー

書式 (Windows)	int CopyBank(Bank srcBankNo, Bank dstBankNo)
書式 (MacOS)	func CopyBank(srcBankNo: Bank, dstBankNo: Bank) -> Int32
パラメータ	srcBankNo (in) コピー元バンク番号 有効値: B1～B32 dstBankNo (in) コピー先バンク番号 有効値: B1～B32
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	コピー元からバンク設定値をコピー先に上書します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ アクティブバンク取得

書式 (Windows)	int GetActiveBank(out Bank bankNo)
書式 (MacOS)	func GetActiveBank(srcBankNo: inout Bank) -> Int32
パラメータ	bankNo (out) カレントバンク番号
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	カレントバンク番号を取得します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ アクティブバンク切り替え

書式 (Windows)	int ChangeActiveBank(Bank bankNo)
書式 (MacOS)	func ChangeActiveBank(bankNo: Bank) -> Int32
パラメータ	bankNo (in) 切り換え後のバンク番号 有効値: B1～B32
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	現在のバンク番号を指定したバンク番号に切り替えます。 bankNo にカレントバンク番号と同じ番号を指定した場合は、カレントバンク番号は変更しません。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

9.2.6 測定結果の取得

■ 測定値取得

書式 (Windows)	int GetMeasurementValue(Task task, out int[] measureValue)				
書式 (MacOS)	func GetMeasurementValue(task: Task, measureValue: inout [Int32]) -> Int32				
パラメータ	task (in)				
	測定結果を取得するタスク番号				
	有効値: T1～T4,ALL				
	measureValue (out)				
	測定しないタスクも含め、4タスク分のデータが格納されます。				
	測定しないタスクは、0 を格納する。				
	測定不能のタスクは、Int32.MaxValue を格納する。				
	配列のインデックス	0	1	2	3
	測定結果の対象	タスク1	タスク2	タスク3	タスク4
戻り値	OK				
	ERR_COMMUNICATION				
	ERR_PARAM				
	ERR_TIME_OUT				
	ERR_RUN_MODE				
	ERR_APPLICATION				
解説	最新の測定結果(測定値)を取得します。				
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降				

■ 受光波形取得

書式 (Windows)	int GetRawImageData(Area area, out MeasureWaveData waveData)
書式 (MacOS)	func GetRawImageData(area: Area, waveData: inout MeasureWaveData) -> Int32
パラメータ	area (in) 取得を行う領域 waveData (out) 受光波形データ(ZW-7000/5000 用)。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION ERR_NONCOMPLIANT_METHODS
解説	最新の受光波形 (未処理) を取得します。 ※ZW-7000/5000 用 ZW-8000 に接続している状態で本メソッドを呼ぶと、ERR_NONCOMPLIANT_METHODS が返ります。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

書式 (Windows)	int GetRawImageData(Area area, out MeasureWaveData2 waveData)
書式 (MacOS)	func GetRawImageData(area: Area, waveData: inout MeasureWaveData2) -> Int32
パラメータ	area (in) 取得を行う領域 waveData (out) 受光波形データ(ZW-8000 用)。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION ERR_NONCOMPLIANT_METHODS
解説	最新の受光波形 (未処理) を取得します。 ※ZW-8000 用 ZW-7000/5000 に接続している状態で本メソッドを呼ぶと、ERR_NONCOMPLIANT_METHODS が返ります。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 判定結果取得

書式 (Windows)	int GetJudgementValue(Task task, out int[] judgementValue)														
書式 (MacOS)	func GetJudgementValue(task: Task, judgementValue: inout [Int32]) -> Int32														
パラメータ	task (in) 判定結果を取得するタスク番号 有効値: T1~T4,ALL judgementValue (out) 判定しないタスクも含め、4タスク分のデータが格納されます。 判定しないタスクの分は、0(PASS)を格納する。 <判定結果> PASS:0 HIGH:1 LOW:2 ERROR:3 <table><tr><td>配列のインデックス</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>判定結果の対象</td><td>タスク1</td><td>タスク2</td><td>タスク3</td><td>タスク4</td></tr></table>					配列のインデックス	0	1	2	3	判定結果の対象	タスク1	タスク2	タスク3	タスク4
配列のインデックス	0	1	2	3											
判定結果の対象	タスク1	タスク2	タスク3	タスク4											
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION														
解説	最新の測定結果(判定値)を取得します。														
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降														

計測波形取得

書式 (Windows)	int GetMeasureWaveData(Area area, out MeasureWaveData waveData)
書式 (MacOS)	func GetMeasureWaveData(area: Area, waveData: inout MeasureWaveData) -> Int32
パラメータ	area (in) 取得を行う領域 waveData (out) 測定波形データ。(ZW-7000/5000 用)
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION ERR_NONCOMPLIANT_METHODS
解説	最新の計測波形 (処理後) を取得します。 ※ZW-7000/5000 用 ZW-8000 に接続している状態で本メソッドを呼ぶと、ERR_NONCOMPLIANT_METHODS が返ります。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

書式 (Windows)	int GetMeasureWaveData(Area area, out MeasureWaveData2 waveData)
書式 (MacOS)	func GetMeasureWaveData(area: Area, waveData: inout MeasureWaveData2) -> Int32
パラメータ	area (in) 取得を行う領域 waveData (out) 測定波形データ(ZW-8000 用)。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION ERR_NONCOMPLIANT_METHODS
解説	最新の計測波形 (処理後) を取得します。 ※ZW-8000 用 ZW-7000/5000 に接続している状態で本メソッドを呼ぶと、ERR_NONCOMPLIANT_METHODS が返ります。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

9.2.7 内部ロギング機能関連

■ 内部ロギング開始

書式 (Windows)	int StartStorage(int cycle, int count)
書式 (MacOS)	func StartStorage(cycle: Int32, count: Int32, clear: Bool = true) -> Int32
パラメータ	cycle (in) ロギングの保存周期 有効値: 1～1000 count (in) ロギングの最大数 有効値: 1～2000000
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングを開始する
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 内部ロギング状態取得

書式 (Windows)	int GetStorageStatus(out int status, out int count)
書式 (MacOS)	func GetStorageStatus(status: inout Int32, count: inout Int32) -> Int32
パラメータ	status (out) ロギングの動作状態(0:停止、1:動作中)を返す。 count (out) ロギング保存数を返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギング情報を取得する。動作状態とロギング保存数を取得する。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

書式 (Windows)	int GetStorageStatus(out int status, out int count, out int labelCount)
書式 (MacOS)	func GetStorageStatus(status: inout Int32, count: inout Int32, labelCount: inout Int32) -> Int32
パラメータ	status (out) ロギングの動作状態(0:停止、1:動作中)を返す。 count (out) ロギング保存数を返す。 labelCount (out) ラベル数を返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギング情報を取得する。動作状態、ロギング保存数とラベル数を取得する。
対応バージョン	ZW-7000 シリーズ Ver.2.10 以降

書式 (Windows)	int GetStorageStatus(int labelNo, out int status, out int count)
書式 (MacOS)	func GetStorageStatus(labelNo: Int32, status: inout Int32, count: inout Int32) -> Int32
パラメータ	labelNo (in) 内部ロギング情報を取得するラベル番号 有効値: 1～16777215 status (out) ロギングの動作状態(0:停止、1:動作中)を返す。 count (out) ロギング保存数を返す。 labelCount (out) ラベル数を返す。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギング情報を取得する。 指定したラベル番号の動作状態とロギング保存数を取得する。
対応バージョン	ZW-7000 シリーズ Ver.2.10 以降

■ 内部ロギング停止

書式 (Windows)	int StopStorage()
書式 (MacOS)	func StopStorage() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングを終了する
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 内部ロギングデータ取得

書式 (Windows)	int GetStorageData(Out outNo, out int[] data)
書式 (MacOS)	func GetStorageData(outNo: Out, data: inout [Int32]) -> Int32
パラメータ	outNo (in) 内部ロギングデータを取得する出力データ番号 有効値: 01~04 data (out) outNo で指定した出力データ番号の内部ロギングデータを返す。 配列サイズは、ストレージ開始で設定した最大数、または、ロギング中にストレージ停止を行った場合はロギング済みの保存数(ストレージ状態取得で確認が可能)となる。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングのデータを取得する。
対応バージョン	ZW2

書式 (Windows)	int GetStorageData(Out outNo, int startNo, int outputNum, out int[] data)
書式 (MacOS)	func GetStorageData(outNo: Out, startNo: Int32, outputNum: Int32, data: inout [Int32]) -> Int32
パラメータ	outNo (in) 内部ロギングデータを取得する出力データ番号 有効値: 01~04 startNo (in) 取得する内部ロギングデータの先頭データ番号 有効値: 0~2000000 outputNum (in) 取得する内部ロギングデータの出力データ個数 有効値: 1~2000000 data (out) outNo で指定した出力データ番号の内部ロギングデータを返す。 配列サイズは、ストレージ開始で設定した最大数、または、ロギング中にストレージ停止を行った場合はロギング済みの保存数(ストレージ状態取得で確認が可能)となる。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングのデータを取得する。
対応バージョン	ZW2

書式 (Windows)	int GetStorageData(Out outNo, out int[] data)
書式 (MacOS)	func GetStorageData(outNo: Out, data: inout [[Int32]]) -> Int32
パラメータ	outNo (in) 内部ロギングデータを取得する出力データ番号 有効値: 01~04 data (out) outNo で指定した出力データ番号の全ラベルの内部ロギングデータを返す。 配列サイズは、1 次が 16777215、2 次がストレージ開始で設定した最大数、または、ロギング中にストレージ停止を行った場合はロギング済みの保存数(ストレージ状態取得で確認が可能)となる。 ラベル番号-1 の要素にラベル番号のロギングデータが格納されている。ロギングデータがない場合には null が格納されている。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングのデータを取得する。(全ラベル)
対応バージョン	ZW2

書式 (Windows)	int GetStorageData(Out outNo, int labelNo, out int[] data)
書式 (MacOS)	func GetStorageData(outNo: Out, labelNo: Int32, data: inout [Int32]) -> Int32
パラメータ	outNo (in) 内部ロギングデータを取得する出力データ番号 有効値: 01~04 labelNo (in) 内部ロギングデータを取得するラベル番号 有効値: 1~16777215 data (out) outNo と labelNo で指定した内部ロギングデータを返す。 配列サイズは、ストレージ開始で設定した最大数、または、ロギング中にストレージ停止を行った場合はロギング済みの保存数(ストレージ状態取得で確認が可能)となる。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングのデータを取得する。(ラベル指定)
対応バージョン	ZW2

書式 (Windows)	int GetStorageData(Out outNo, int labelNo, int startNo, int outputNum, out int[] data)
書式 (MacOS)	func GetStorageData(outNo: Out, labelNo: Int32, startNo: Int32, outputNum: Int32, data: inout [Int32]) -> Int32
パラメータ	outNo (in) 内部ロギングデータを取得する出力データ番号 有効値: 01～04 labelNo (in) 内部ロギングデータを取得するラベル番号 有効値: 1～16777215 startNo (in) 取得する内部ロギングデータの先頭データ番号 有効値: 0～2000000 outputNum (in) 取得する内部ロギングデータをの出力データ個数 有効値: 1～2000000 data (out) outNo と labelNo で指定した内部ロギングデータを返す。 配列サイズは、ストレージ開始で設定した最大数、または、ロギング中にストレージ停止を行った場合はロギング済みの保存数(ストレージ状態取得で確認が可能)となる。
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	内部ロギングのデータを取得する。(ラベル指定)
対応バージョン	ZW2

■

9.2.8 高速データ通信関連

■ 高速データ通信開始前の準備

書式 (Windows)	int PreStartHighSpeedDataCommunication(bool logCtrlFlag, int thinningNum, int saveNum)										
書式 (MacOS)	func PreStartHighSpeedDataCommunication(logCtrlFlag: inout [Bool], thinningNum: Int32, saveNum: Int32) -> Int32										
パラメータ	logCtrlFlag (in) true: 高速データ出力対象、false: 高速データ出力対象外 配列サイズ: 4 <table><tr><td>配列のインデックス</td><td>0</td><td>1</td><td>2</td><td>3</td></tr><tr><td>設定対象のタスク</td><td>タスク1</td><td>タスク2</td><td>タスク3</td><td>タスク4</td></tr></table> thinningNum (in) 間引き数 有効値: 0～65535 saveNum (in) 保存数 有効値: 0～128	配列のインデックス	0	1	2	3	設定対象のタスク	タスク1	タスク2	タスク3	タスク4
配列のインデックス	0	1	2	3							
設定対象のタスク	タスク1	タスク2	タスク3	タスク4							
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION										
解説	高速データ通信のための設定を行います。										
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降										

■ 高速データ通信開始

書式 (Windows)	int StartHighSpeedDataCommunication(LoggingDataDelegate method)
書式 (MacOS)	func StartHighSpeedDataCommunication(method: LoggingDataDelegate) -> Int32
パラメータ	method フローデータ取得デリゲートメソッド delegate void LoggingDataDelegate(List<FlowData> flowDataList) flowDataList : タスク毎のフローデータ
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	高速データ通信を開始します。 設定サンプリング数を計測し繰り返します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

■ 高速データ通信停止

書式 (Windows)	int StopHighSpeedDataCommunication()
書式 (MacOS)	func StopHighSpeedDataCommunication() -> Int32
パラメータ	—
戻り値	OK ERR_COMMUNICATION ERR_PARAM ERR_TIME_OUT ERR_RUN_MODE ERR_APPLICATION
解説	高速データ通信を停止します。
対応バージョン	ZW-7000/5000 シリーズ ver.200 以降

10 共通コード

10.1 共通エラーコード

「8 関数」で示す IF 関数の戻り値が下記示すように定義します。

定義名	データ	原因
OK	0x00000000	正常終了
ERR_PARAM	0x01080610	設定パラメータがエラー
ERR_RUN_MODE	0x01FF2204	動作モードがエラー
ERR_COMMUNICATION	0x02110100	受送信がエラー
ERR_TIME_OUT	0x02110101	受信でタイムアウトが発生
ERR_CONNECT	0x02110104	接続失敗
ERR_APPLICATION	0x12160802	アプリケーションエラー
ERR_NONCOMPLIANT_METHODS	0x1216FF01	非対応メソッド

11 データ構造(詳細)

11.1 計測波形データ

データ内容	最小	最大	単位	備考
バンク番号	1	32	–	
2 領域モード	0	1	–	0:1 領域モード、1:2 領域モード
領域番号	0	1	–	0:領域 0 の波形、1:領域 1 の波形
1 面目の受光量	0	4096	階調	
2 面目の受光量	0	4096	階調	
3 面目の受光量	0	4096	階調	
4 面目の受光量	0	4096	階調	
1 面目の計測値	0	255*256	画素	(1/256)[画素/div]
2 面目の計測値	0	255*256	画素	(1/256)[画素/div]
3 面目の計測値	0	255*256	画素	(1/256)[画素/div]
4 面目の計測値	0	255*256	画素	(1/256)[画素/div]
領域指定:開始座標	0	255	画素	
領域指定:終了座標	0	255	画素	
領域指定:マスク領域開始	0	255	画素	
領域指定:マスク領域終了	0	255	画素	
グラフ軸座標 1	0	255*256	画素	(1/256)[画素/div]
グラフ軸座標 2	0	255*256	画素	(1/256)[画素/div]
グラフ軸座標 3	0	255*256	画素	(1/256)[画素/div]
グラフ軸座標 4	0	255*256	画素	(1/256)[画素/div]
グラフ軸座標 5	0	255*256	画素	(1/256)[画素/div]
計測レンジ(nm)	0	999999999	nm	
測定周期	0	10000	μs	0.1[μs/div]
投光量	0	10000	%	0.01[%/div]
受光量	0	4095	階調	
電流/電圧 DAC 値	0	65535	–	以下の式で算出 電圧値=(20 – 4) / (59069 – 5069) * (DAC 値 – 5069) + 4 電流値=(10 – (–10)) / (59069 – 5069) * (DAC 値 – 5069) – 10
電流/電圧状態	0	1	–	0:電圧出力 1:電流出力
絶対距離	–999999999	999999999	nm	

TASK1 の測定結果(nm)	-999999999	999999999	nm	
TASK2 の測定結果(nm)	-999999999	999999999	nm	
TASK3 の測定結果(nm)	-999999999	999999999	nm	
TASK4 の測定結果(nm)	-999999999	999999999	nm	
TASK1 の分解能	-999999999	999999999	nm	
TASK2 の分解能	-999999999	999999999	nm	
TASK3 の分解能	-999999999	999999999	nm	
TASK4 の分解能	-999999999	999999999	nm	
TASK1 の判定上限値	-999999999	999999999	nm	
TASK2 の判定上限値	-999999999	999999999	nm	
TASK3 の判定上限値	-999999999	999999999	nm	
TASK4 の判定上限値	-999999999	999999999	nm	
TASK1 の判定下限値	-999999999	999999999	nm	
TASK2 の判定下限値	-999999999	999999999	nm	
TASK3 の判定下限値	-999999999	999999999	nm	
TASK4 の判定下限値	-999999999	999999999	nm	
エラー情報	0	255	-	以下の情報を各ビットに付加する b0-2:受光量エラー(0:光量安定 1:光量調整中 2:光量飽和 3:光量不足 4:LIGHT OFF 状態 5:相互干渉防止 OFF) b3:システムエラー b4:負荷短絡検知(0 固定) b5:面数エラー b6:STAB 状態 b7:センサヘッド校生エラー
ラインブライトデータ	0	4095		4Byte × 256 画素

11.2 フローデータ

データ内容	最小	最大	単位	備考
OUT 番号	0	3	–	0:OUT1 情報 1:OUT2 情報 2:OUT3 情報 3:OUT4 情報
TIMING 入力	0	1	–	0:TIMING 入力 OFF 1:TIMING 入力 ON
RESET 入力	0	1	–	0:RESET 入力 OFF 1:RESET 入力 ON
LIGHTOFF 入力	0	1	–	0:LIGHTOFF 入力 OFF 1:LIGHTOFF 入力 ON
ZERO 入力	0	1	–	0:ZERO 入力 OFF 1:ZERO 入力 ON
LOGGING 入力	0	1	–	0:LOGGING 入力 OFF 1:LOGGING 入力 ON
SYNC 入力	0	1	–	0:SYNC 入力 OFF 1:SYNC 入力 ON
BUSY 出力	0	1	–	0:BUSY 出力 OFF 1:BUSY 出力 ON
ENABLE 出力	0	1	–	0:ENABLES 出力 OFF 1:ENABLE 出力 ON
LOW 出力	0	1	–	0:LOW 出力 OFF 1:LOW 出力 ON
PASS 出力	0	1	–	0:PASS 出力 OFF 1:PASS 出力 ON
HIGH 出力	0	1	–	0:HIGH 出力 OFF 1:HIGH 出力 ON
TASKSTAT 出力	0	1	–	0:TASKSTAT 出力 OFF 1:TASKSTAT 出力 ON
LOGSTAT 出力	0	1	–	0:LOGSTAT 出力 OFF 1:LOGSTAT 出力 ON
LOGERR 出力	0	1	–	0:LOGERR 出力 OFF 1:LOGERR 出力 ON
SYNCFLG 出力	0	1	–	0:SYNCFLG 出力 OFF 1:SYNCFLG 出力 ON
STABILITY 出力	0	1	–	0:STABILITY 出力 OFF

				1: STABILITY 出力 ON
高速データ通信あふれ bit (BUFFER_ERR)	0	1	－	0: データあふれなし 1: データあふれあり ＜注意＞ データあふれが発生した場合は、高速データ通信開始前準備で設定した保存数単位でデータの抜け落ちが発生しています。
高速データ通信停止 bit	0	1	－	0: 高速データ通信継続 1: 高速データ通信停止
計測データ	－999999999	999999999	－	

12 サンプルプログラム

通信ライブラリを使用したアプリケーション作成例として添付している、サンプルプログラムに関して説明します。

12.1 ユーザーインターフェース仕様

12.1.1 IP アドレス入力画面

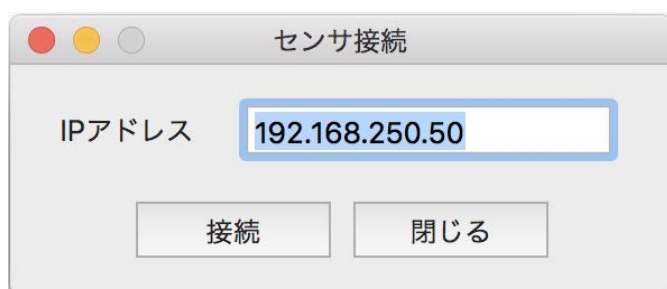
接続を行う ZW-8000/7000/5000 シリーズのセンサセンサコントローラの IP アドレスを入力します。

- ・画面レイアウト

<Windows>



<MacOS>



- ・機能

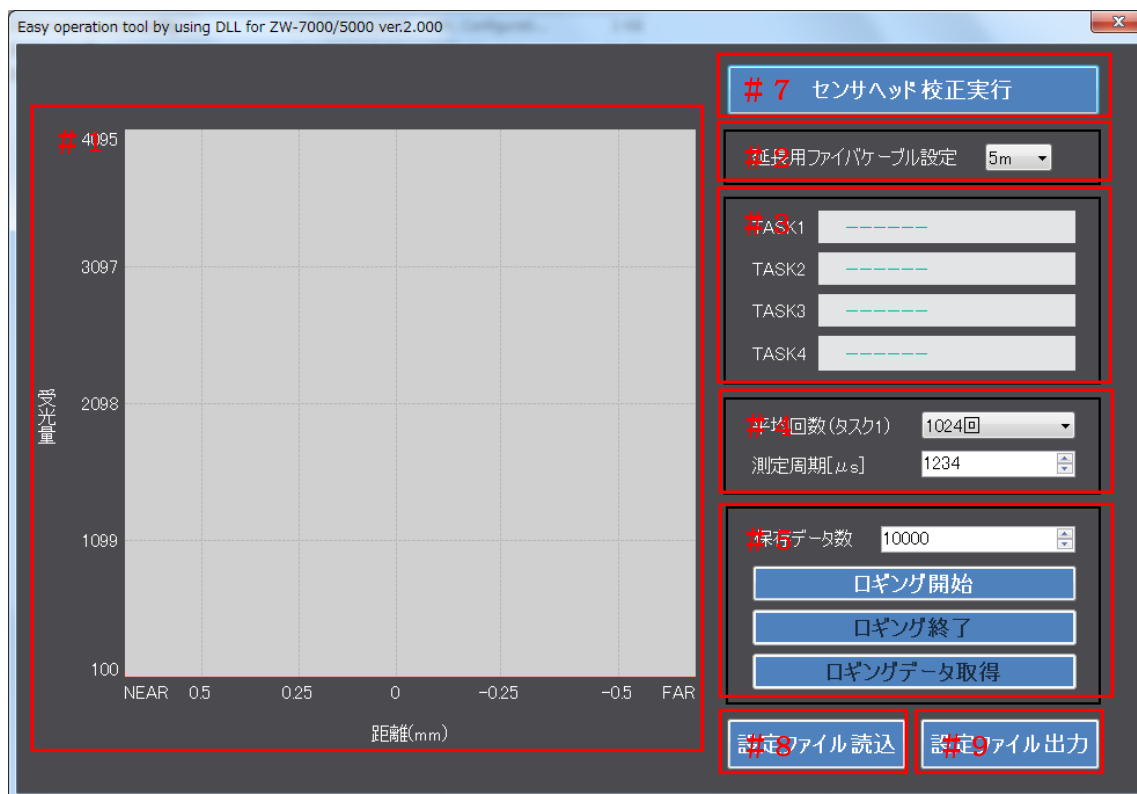
#	項目	説明
1	IP アドレス	IP アドレスの入力を行います。
2	接続	ZW-8000/7000/5000 シリーズのセンサセンサコントローラに接続して、メイン画面を表示します。
3	閉じる	本アプリケーションを終了します。

- ・メッセージ表示

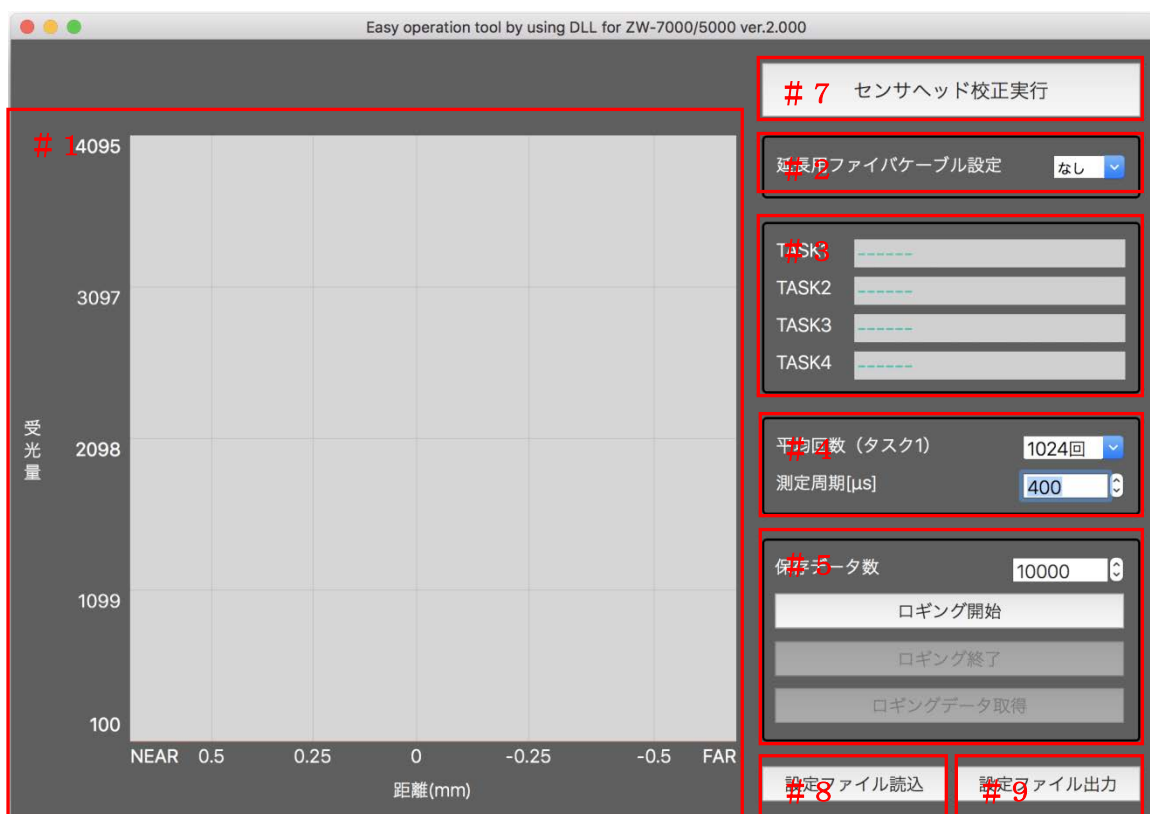
#	表示条件	メッセージ
1	接続失敗時	接続に失敗しました。

12.1.2 メイン画面

<Windows>



<MacOS>



・機能

#	項目	説明
1	受光波形モニタ	受光波形を表示します。
2	延長用ファイバケーブル設定	延長用ファイバケーブルの設定項目を表示します。
3	計測値モニタ	選択された計測データを表示します。
4	平均回数(タスク 1)	タスク1の平均回数を設定します。
5	測定周期[μ s]	測定周期の設定項目を表示します。
6	内部ロギング制御	内部ロギング制御を行います。
7	センサヘッド校正実行	センサヘッド校正を実行します。
	設定ファイル読込	設定ファイル読込を実行します。
	設定ファイル出力	設定ファイル出力を実行します。

メッセージ表示

#	表示条件	メッセージ
1	ロギング開始時	内部ロギングを開始します。よろしいですか？
2	内部ロギング終了時	内部ロギングが終了しました。

12.2 サンプルソース(Windows)

12.2.1 通信確立

```
using Omron.Cxap.Modules.DisplacementSensorSDK;
using Omron.Cxap.Modules.DisplacementSensorSDK.CommHelper;

namespace DllSampleApp
{
    public partial class IpInputForm : Form
    {
        // 通信DLLのインスタンス
        private DSComm dsComm = null;

        private void connectTextBox_Click(object sender, EventArgs e)
        {
            try
            {
                // 通信DLLのインスタンスを作成
                dsComm = new DSComm(DSComm.Version.ZW2);

                // 接続
                byte[] ipaddress = { 192, 168, 250, 50 };
                int ret = dsComm.Open(ipaddress, this.DisConnectDelegate);

                // 接続処理の結果を確認
                if (ret != CommErr.OK)
                {
                    // 接続に失敗
                    MessageBox.Show(this,
                        Resources.Msg_ConnectError,
                        Application.ProductName);
                }
            }
            catch (Exception ex)
            {
                throw (ex);
            }
        }

        /// <summary>
        /// センサとの接続が切断された時のデリゲートメソッド
        /// </summary>
        private void DisConnectDelegate()
        {
            // 切断通知
        }
    }
}
```

通信DLLの使用するライブラリを宣言

通信DLLのインスタンス変数を定義

通信DLLのインスタンスを作成

IPアドレス(デフォルト:192.168.250.50)と切断通知を受けるDelegate methodを指定してOpen関数を呼び出す

12.2.2 測定値の取得

```
MeasureWaveData waveData;
if (beforeWaveRadio.Checked == true)
{
    // 受光波形データを取得
    retApi = this.dsComm.GetRawImageData(DSComm.Area.A1, out waveData);
    if (retApi != CommErr.OK)
    {
        return;
    }
}
else
{
    // 計測波形データを取得
    retApi = this.dsComm.GetMeasureWaveData(DSComm.Area.A1, out waveData);
    if (retApi != CommErr.OK)
    {
        return;
    }
}

// 測定値を取得
int[] measureData;
retApi = this.dsComm.GetMeasurementValue(DSComm.Task.ALL, out measureData);
if (retApi != CommErr.OK)
{
    return;
}
```

指定したタスク番号の受光波形データを取得

指定したタスク番号の計測波形データを取得

指定したタスク番号の計測値データを取得

12.2.3 バンクデータの取得・設定

// センサから平均回数を取得

```
retApi = this.dsComm.GetBankData(Constants.UNIT_NO_AVERAGE,  
                                Constants.DATA_NO_AVERAGE,  
                                out value);  
if (retApi != CommErr.OK)  
{  
    MessageBox.Show(this, GetErrorMessage(retApi), Application.ProductName);  
    return;  
}
```

バンクデータの取得
取得するデータのユニット番号、
データ番号を指定する

// 平均回数をセンサに設定

```
retApi = this.dsComm.SetBankData(Constants.UNIT_NO_AVERAGE,  
                                Constants.DATA_NO_AVERAGE,  
                                value);  
if (retApi != CommErr.OK)  
{  
    MessageBox.Show(this, GetErrorMessage(retApi), Application.ProductName);  
    return;  
}
```

バンクデータの設定
設定するデータのユニット番号、
データ番号を指定する

12.3 サンプルソース(MacOS)

12.3.1 通信確立

```
import Cocoa
import DSComm

class ViewController: NSViewController, DisconnectDelegate {
    // 通信DLLのインスタンス
    var dsComm: DSComm? = nil

    @IBAction func connectClick(_ sender: Any) {
        // 通信DLLのインスタンスを作成
        self.dsComm = DSComm(version: DSComm.Version.ZW2)
        if (self.dsComm == nil) {
            return
        }

        // ZWに接続
        let ip: [UInt8] = [192, 168, 250, 50]
        let ret: Int32 = self.dsComm!.Open(ipAddress: ip, method: self)
        if (ret != CommErr.OK) {
            // 接続に失敗
            return
        }
    }

    // センサとの接続が切断された時のデリゲートメソッド
    public func Disconnect() {
        // 切断通知
    }
}
```

通信DLLの使用ライブラリを宣言

通信DLLのインスタンス変数を定義

通信DLLのインスタンスを作成

IPアドレス(デフォルト: 192.168.250.50)と切断通知を受けるDelegate methodを指定してOpen関数を呼び出す

12.3.2 測定値の取得

```
var waveData: MeasureWaveData = MeasureWaveData()
if(self.beforeWaveSelect == true)
{
    // 受光波形データを取得
    ret = self.dsComm!.GetRawImageData(area: DSComm.Area.A1, waveData: &waveData)
    if (ret != CommErr.OK)
    {
        return
    }
}
else
{
    // 計測波形データを取得
    ret = self.dsComm!.GetMeasureWaveData(area: DSComm.Area.A1, waveData: &waveData)
    if (ret != CommErr.OK)
    {
        return
    }
}

// 測定結果を取得
var measureData: [Int32] = [ ]
ret = self.dsComm!.GetMeasurementValue(task: DSComm.Task.ALL, measureValue: &measureData)
if (ret != CommErr.OK)
{
    return;
}
```

指定したタスク番号の受光波形データを取得

指定したタスク番号の計測波形データを取得

指定したタスク番号の計測値データを取得

12.3.3 バンクデータの取得・設定

// 変位センサから平均回数を取得

```
var value: Int32 = 0
ret = self.dsComm!.GetBankData(unitNo: Constans.UNIT_NO_AVERAGE, dataNo: Constans.
    DATA_NO_AVERAGE, value: &value)
if (ret != CommErr.OK)
{
    return
}
```

バンクデータの取得
取得するデータのユニット番号、
データ番号を指定する

// 平均回数を変位センサに設定

```
ret = self.dsComm!.SetBankData(unitNo: Constans.UNIT_NO_AVERAGE, dataNo: Constans.
    DATA_NO_AVERAGE, value: value)
if (ret != CommErr.OK)
{
    return
}
```

バンクデータの設定
設定するデータのユニット番号、
データ番号を指定する

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

製品に関するお問い合わせ先

お客様
相談室



0120-919-066

携帯電話の場合、

☎055-982-5015 (有料) をご利用ください。

受付時間：9:00～17:00 (土・日・12/31～1/3を除く)

クイック

オムロン



オムロンFAクイックチャット

www.fa.omron.co.jp/contact/tech/chat/

技術相談員にチャットでお問い合わせいただけます。(I-Webメンバーズ限定)

受付時間：平日9:00～12:00 / 13:00～17:00 (土日祝日・年末年始・当社休業日を除く)

※受付時間、営業日は変更の可能性がございます。最新情報はリンク先をご確認ください。



その他のお問い合わせ：納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。



オムロン制御機器の最新情報をご覧ください。緊急時のご購入にもご利用ください。 www.fa.omron.co.jp

- 本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
- 本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物 (又は技術) に該当するものを輸出 (又は非居住者に提供) する場合は同法に基づく輸出許可、承認 (又は役務取引許可) が必要です。

オムロン商品のご用命は

Man. No. SCEA-705D

2025年3月現在

©OMRON Corporation 2016-2025 All Rights Reserved.
お断りなく仕様などを変更することがありますのでご了承ください