IVI 計測器ドライバ プログラミングガイド LabWindows/CVI 編

Oct 2020 Revision 2.2

目次

はじめに 4
LabWindows/CVI で使用する計測器ドライバ 4
使用できるインターフェース 4
スペシフィック・インターフェースを使用したプログラミング 5
プログラミングの準備 5
新規のプロジェクトを作成する 5
計測器ドライバをロードする 6
プログラムを構成する 7
Initialize With Options 関数を設定する 7
Close 関数を設定する 9
そのほかの関数を追加する 10
プロジェクトをビルドする 11
プログラムの実行 12
ブレークポイントを設定する 12
プログラムを実行する 12
変数 vi と vs に格納される値 12
関数の解説 13
セッションの開始 13
チャンネル名の設定 15
セッションのクローズ 15
エラー処理 16
クラス・インターフェースを使用したプログラミング 17
プログラミングの準備 17
仮想インストルメントを作成する 17
新規のプロジェクトを作成する 23
計測器ドライバをロードする 24
プログラムを構成する 25

Initialize With Options 関数を設定する 25
Close 関数を設定する 27
そのほかの関数を追加する 28
プロジェクトをビルドする 28
プログラムの実行 29
ブレークポイントを設定する 29
プログラムを実行する 29
変数 vi と vs に格納される値 29
関数の解説 30
セッションの開始 30
チャンネル名の設定 31
セッションのクローズ 31
エラー処理 32
計測器を交換する 33

はじめに

本ガイドでは、KikusuiPwr IVI 計測器ドライバ（KIKUSUI PWR-01 シリーズ直流電源）を使用する例を示します。他社メーカーおよび他機種用の IVI 計測器ドライバでも、ほぼ同様の手順で使用できます。

本ガイドでは、LabWindows/CVI 2019 を使用し、Windows10（x64）で動作する 32bit（x86）プログラムを作成する場合を例に説明します。

LabWindows/CVI で使用する計測器ドライバ

本ガイドでは、IVI-C 計測器ドライバを推奨し、IVI-C 計測器ドライバを使用してプログラミングする手順を例に説明しています。

LabWindows/CVI は C 言語を前提にした開発環境なので、COM DLL より関数ベースの C DLL を使うほうが簡単です。IVI-C 計測器ドライバは、LabWindow/CVI 計測器ドライバのアーキテクチャを拡張して作成されているので、LabWindows/CVI との親和性があります。

使用できるインターフェース

IVI 計測器ドライバは、以下の 2 つのインターフェースに対応しています。

- **スペシフィック・インターフェース**
  計測器ドライバの固有インターフェースです。使用する計測器の機能を最大限に利用できます。
- **クラス・インターフェース**
  IVI 仕様書で定義されている計測器クラスのインターフェースです。
  インターチェンジャビリティを利用できますが、機種固有の機能を使うことは制限されます。

本ガイドでは、それぞれのインターフェースを使用してプログラミングする方法について説明します。

Memo

- 計測器ドライバが所属する計測器クラスは、Windows スタートメニューの［Kikusui］＞［KikusuiPwr IVI Driver 1.0.0 Documentation］から確認できます。
- 計測器ドライバがどの計測器クラスにも属していない場合、クラス・インターフェースを利用できないため、インターチェンジャビリティを利用するアプリケーションは作成できません。

スペシフィック・インターフェースを使用したプログラミング

スペシフィック・インターフェースを使用してプログラムする手順を説明します。スペシフィック・インターフェースを使用すると、計測器ドライバの機種固有機能を最大限に利用できます。

Memo

スペシフィック・インターフェースの場合、インターチェンジャビリティ機能は利用できません。インターチェンジャビリティ機能を利用する場合は、クラス･インターフェースを使用してください。（p.17）

プログラミングの準備

新規のプロジェクトを作成する

新規アプリケーションで IVI-C ドライバを使う例を説明します。

1 **LabWindows/CVI を起動します。**

Welcome ページが表示されます。

2 **［New］＞［Project］をクリックします。**

Welcome ページが表示されずに既存プロジェクトが読み込まれる場合には、［File］＞［New］＞［Project（*.prj)］をクリックしてください。

新規プロジェクトが作成されます。

3 **［File］＞［Save Untitled.prj As］をクリックしてプロジェクトを保存します。**

ここでは、"Ex01.prj" として保存します。

4 **［File］＞［New］＞［Source (*.c)］をクリックします。**

ソースファイルが作成されます。

5 **［File］＞［Save Untitled1.c As］をクリックしてソースファイルを保存します。**

ここでは、"Ex01.c" として保存します。

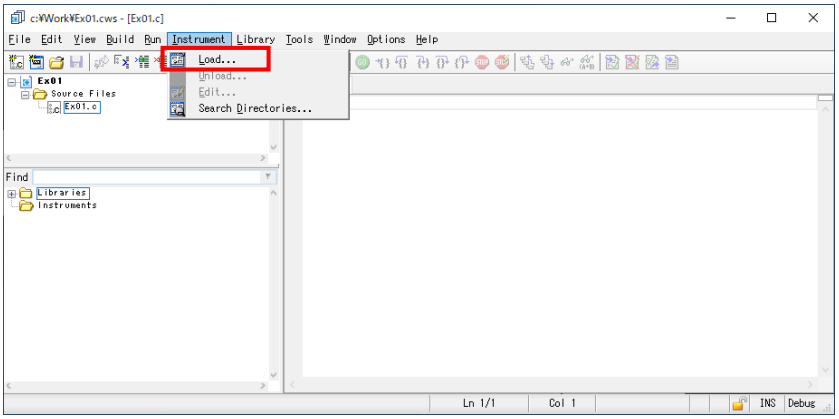
6 **［File］＞［Add Ex01.c to Project］をクリックします。**

ソースファイルがプロジェクトに追加されます。

以上で、新規プロジェクトの作成は完了です。

計測器ドライバをロードする

1 **［Instrument］＞［Load］をクリックします。**



2 **"C:/Program Files (x86)/IVI Foundation/IVI/Drivers/kipwr" ディレクトリに保存されている "kipwr.fp" をロードします。**

［Instrument］メニューの中に、［KikusuiPwr01］が追加されます。

プログラムを構成する

Initialize With Options 関数を設定する

Initialize With Options 関数を設定し、変数を宣言してソースコードに挿入します。

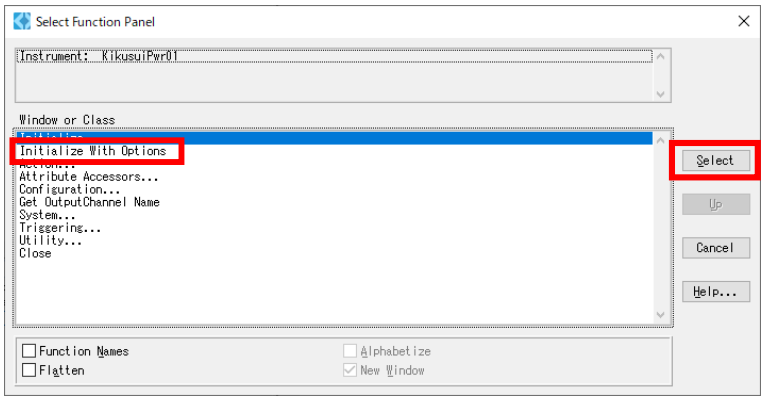
パラメータを設定する

1 **プロジェクトに追加された C のソースコード（Ex01.c）を開きます。**

2 **［Instrument］＞［KikusuiPwr01］をクリックします。**

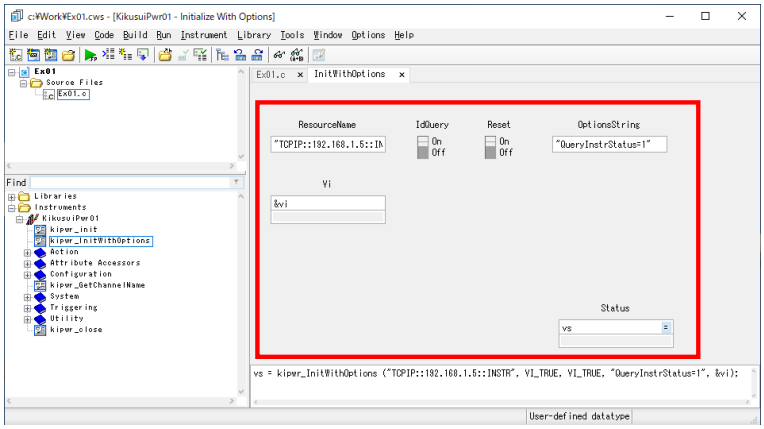
Select Function Panel が表示されます。

3 **［Initialize With Options］をクリックし、［Select］をクリックします。**



Initialize With Options のファンクションパネルが表示されます。

4 **各パラメータに値を設定します。**



計測器が LAN インターフェースで接続され、IP アドレスが 192.168.1.5 である場合を例にします。ResourceName と OptionString のパラメータは文字列なので、引用符で囲んでください。

パラメータ	値
ResourceName	"TCPIP::192.168.1.5::INSTR"
IdQuery	On（関数コール記述では VI_TRUE）
Reset	On（関数コール記述では VI_TRUE）
OptionString	"QueryInstrStatus=1"
Vi	&vi
Status	vs

変数を宣言する

Vi と Status に入力した変数を宣言します。

1 **ファンクションパネルでパラメータ Vi をクリックします。**

2 **［Code］＞［Declare Variable］をクリックします。**

Declare Variable ダイアログが表示されます。

3 **Execute declaration in Interactive Window と Add declaration to top of target file "Ex01.c" の両方をチェックします。**

4 **［OK］をクリックします。**

5 **ファンクションパネルでパラメータ Status をクリックします。**

6 **手順 2 ～手順 4 と同様に操作します。**

宣言が完了しました。

Initialize With Options 関数の呼び出しコードを挿入する

1 **［Code］＞［Insert Function Call］をクリックします。**

kipwr_InitWithOptions 関数の呼び出しコードがソースコード（Ex01.c）に挿入されます。

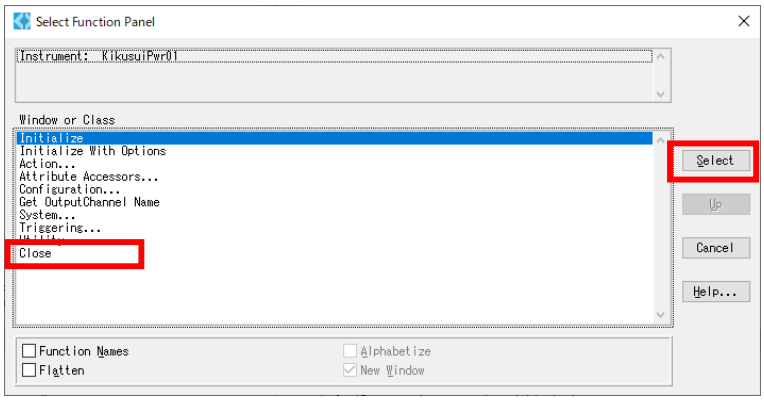
Close 関数を設定する

Initialize With Options 関数と同様に Close 関数を設定し、ソースコードに挿入します。

1 **［Instrument］＞［KikusuiPwr01］をクリックします。**

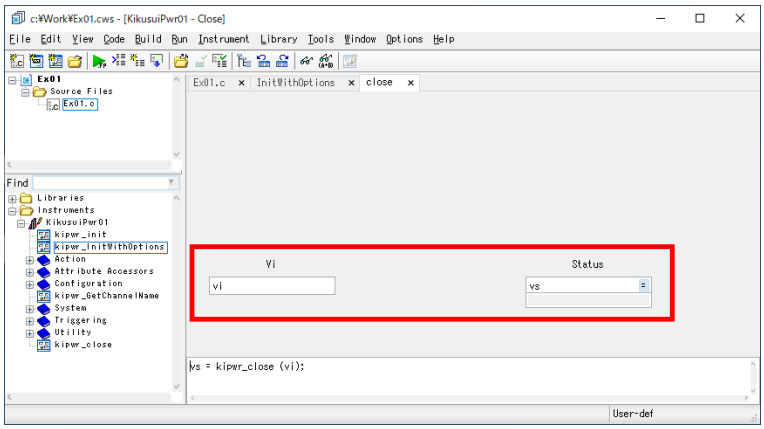
Select Function Panel が表示されます。

2 **［Close］をクリックし、［Select］をクリックします。**



Close のファンクションパネルが表示されます。

3 **パラメータ［Vi］に vi と入力し、［Status］に vs と入力します。**



4 **［Code］＞［Insert Function Call］をクリックします。**

kipwr_close 関数の呼び出しコードがソースコード（Ex01.c）に挿入されます。

そのほかの関数を追加する

［Ex01.c］タブを選択すると、ソースコードが表示されます。p.7 ～ p.9 を実行すると、ソースコードは下記のようになります。

```
static ViStatus vs;
static ViSession vi;
vs = kipwr_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_
     TRUE, "QueryInstrStatus=1", &vi);
vs = kipwr_close (vi);
```

このソースコードを実行可能なプログラムにするため、次の内容を記述します。

- 計測器ドライバのインクルードファイルの読み込みを指定します。
- InitWithOptions と close の呼び出しを main 関数で囲みます。
- InitWithOptions と close の呼び出しの間に、電圧と電流を設定する関数、出力 ON/OFF を制御する関数の呼び出しコードを追加します。ファンクションパネルを使って挿入する方法のほかに、下記のようにソースコードをエディタで直接入力する方法もあります。

下記は、計測器ドライバセッションをオープンし、電圧、電流、出力を設定して、すぐにクローズするサンプルです。

```
#include <kipwr.h>
static ViStatus vs;
static ViSession vi;
void main()
{
vs = kipwr_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_
     TRUE, "QueryInstrStatus=1", &vi);
vs = kipwr_ConfigureVoltageLevel (vi, "", 20);
vs = kipwr_ConfigureCurrentLimit (vi, "", KIPWR_VAL_CURRENT_REGULATE,
     2.0);
vs = kipwr_ConfigureOutputEnabled (vi, "", 1);
vs = kipwr_close (vi);
}
```

Memo

直流電源装置用の IVI 計測器ドライバは、通常は多チャンネルを考慮した作りになっています。そのため、装置設定ファンクションの多くには ChannelName パラメータがあります。

プロジェクトをビルドする

1 **[Build] > [Build] をクリックします。**

ビルドが完了します。

プログラムの実行

プログラム実行時のデバッグに役立つブレークポイントの紹介と、プログラムの実行方法、実行後に変数に格納される値について説明します。

ブレークポイントを設定する

p.10 のサンプルは、プログラムが対話形式ではないため実行したときの経緯がわかりません。呼び出し行にブレークポイントを挿入すると、呼び出し前にプログラムが一時停止するため、実行中のプログラムの状態が確認できます。

1 **ブレークポイントを挿入したい行にカーソルを移動します。**

2 **［Run］＞［Toggle Breakpoint］をクリックします。**

ブレークポイントが設定されます。

プログラムを実行する

1 **［Run］＞［Debug Ex01.exe］をクリックします。**

プログラムが実行されます。ブレークポイントを設定した行を呼び出す前に一時停止します。

2 **［Run］＞［Continue］をクリックします。**

プログラムの実行が再開されます。

変数 vi と vs に格納される値

計測器ドライバのセッションをオープンできた場合には、vi にはセッションハンドル（通常は IVI ハンドルとして 0x00000001 以上）が格納されます。vs には実行結果（成功した場合は 0x00000000、失敗した場合は負の値）が格納されます。

関数の解説

プログラムを構成する関数について、その内容や設定値などを説明します。

Memo

IVI-C 計測器ドライバおよび VXI Plug&Play 計測器ドライバには、関数名のルールとして、<prefix> という表記があります。この表記は、各計測器ドライバの識別名を表します。

たとえば「<prefix>_init()」という表記は、kipwr 計測器ドライバでは「kipwr_init()」となります。

セッションの開始

スペシフィック・インターフェースのセッションを開始するには、<prefix>_InitWithOptions を使用します。

```
vs = kipwr_InitWithOptions ("TCPIP::192.168.1.5::INSTR", VI_TRUE, VI_
     TRUE, "QueryInstrStatus=1", &vi);
```

<prefix>_InitWithOptions で設定できるパラメータは以下のとおりです。

パラメータ	タイプ	説明
ResourceName	ViRsrc (const char*)	計測器が接続されている I/O インターフェース、アドレスなどによって決定される VISA リソース名の文字列です。 たとえば、IP アドレス 192.168.1.5 の計測器を LAN で接続し、VXI-11 インターフェースで制御している場合には、「TCPIP::192.168.1.5::INSTR」となります。
idQuery	ViBoolean	VI_TRUE を指定した場合、計測器に対して "*IDN?" などの ID クエリを発行して機種情報を問い合わせます。
Reset	ViBoolean	VI_TRUE を指定した場合、"*RST" コマンドなどを発行して計測器の設定をリセットします。
OptionString	ViConstString (const char*)	IVI 定義の設定を変更できます。（p.14）
Vi	ViSession*	計測器セッションを受け取ります（ポインタ渡し）。

OptionString を設定するには

OptionString では、下記の IVI 定義を設定できます。

IVI 定義	デフォルト値
RangeCheck	VI_TRUE
Cache	VI_TRUE
Simulate	VI_FALSE
QueryInstrStatus	VI_FALSE
RecordCoercions	VI_FALSE
Interchange Check	VI_FALSE

OptionString は文字列のパラメータです。下記に、サンプルを記載します。

```
QueryInstrStatus = VI_TRUE, Cache = VI_TRUE, DriverSetup=12345
```

書式について、次の点に注意してください。

- 設定値を指定しない場合にはデフォルト値が適用されます。
- 設定値は ViBoolean 型です。「VI_TRUE」、「VI_FALSE」、「1」、「0」のいずれかが設定できます。
- 機能名称および設定値はケース・インセンシティブです。大文字と小文字は区別されません。
- 複数の項目を設定する場合には、カンマで区切ります。
- 計測器ドライバによっては、「DriverSetup」パラメータがサポートされています。
  「DriverSetup」は、IVI 仕様書では定義されない項目を InitWithOptions の呼び出し時に指定するパラメータです。利用目的や書式はドライバに依存します。そのため、「DriverSetup」を設定する場合には、「Option String」の最後の項目として指定します。「DriverSetup」の指定内容については、ドライバの Readme またはオンライン・ヘルプなどを参照してください。

チャンネル名の設定

電源装置やオシロスコープなどの場合、IVI 計測器ドライバは複数のチャンネルが装備されていることを前提に設計されています。そのため、計測器のパネル設定について操作をするドライバ関数の多くは、下記のように第 2 パラメータにチャンネルを指定する必要があります。

```
vs = kipwr_ConfigureVoltageLevel( vi, "", 20.0);
```

この例では、チャンネル名にブランク（""）を指定しています。チャンネル数が 1 個の場合はブランクで問題ありませんが、チャンネル数が複数ある場合には、明示的に指定する必要があります。

使用できるチャンネル名は計測器ドライバによって異なります。詳細は、各ドライバのヘルプを参照してください。

セッションのクローズ

計測器ドライバのセッションをクローズするには、<prefix>_close を使います。

```
vs = kipwr_close (vi);
```

エラー処理

範囲外の値をパラメータに渡したり、サポートされていない機能を呼び出したりすると、計測器ドライバにエラーが発生します。IVI-C 計測器ドライバでは、計測器ドライバ内で発生したエラーはすべて下記のような ViStatus 型の戻り値としてクライアントプログラムに伝えられます。

値の範囲	説明
vs = 0	成功
vs > 0	警告
vs < 0	エラー

これまで示したサンプルではエラー処理を行っていませんでしたが、error_message 関数を使って ViStatus 型の戻り値を読みやすいメッセージに変換することができます。error_message は例外的に ViSession として VI_NULL を受け入れます。受け取りのバッファサイズは 256Bytes 以上にしてください。

```
char buf[256];
...
kipwr_error_message (VI_NULL, vs, buf);
```

クラス・インターフェースを使用したプログラミング

クラス・インターフェースを使用してプログラムする手順を説明します。クラス・インターフェースでは、IVI 仕様で定義された計測器クラスのインターフェースを使用してプログラミングすることで、計測器クラス・インターフェースを利用したインターチェンジャビリティを実現できます。インターチェンジャビリティを利用すると、アプリケーションを再度コンパイル・リンクすることなく計測器を交換できます。

Memo

- インターチェンジャビリティを利用するには、交換前後の両機種に対して同じ計測器クラスの IVI-C 計測器ドライバが提供されている必要があります。異なる計測器クラス間でのインターチェンジャビリティは実現できません。
- クラス・インターフェースを使用したプログラミングでは、使用できる機種固有の機能が制限されます。機種固有の機能を最大限に使用するには、スペシフィック・インターフェースを使用してプログラミングしてください（p.5）。

プログラミングの準備

仮想インストルメントを作成する

インターチェンジャビリティ機能を利用するアプリケーションを作成するには、あらかじめ仮想インストルメントを作成する必要があります。

Memo

インターチェンジャビリティ機能を損なうため、アプリケーション・コード内に特定の IVI-C 計測器ドライバに依存した記述（例：kipwr_init 関数の直接コール）や、特定 VISA アドレス（リソース名）の記述（例："TCPIP::192.168.1.5::INSTR"）などを行わないでください。

IVI 仕様では、計測器ドライバとアプリケーションの外部に IVI コンフィグレーション・ストアを置くことでインターチェンジャビリティ機能を実現します。アプリケーションは機種固有の計測器ドライバを直接使うのではなく、計測器クラス・インターフェースと呼ばれる特別な計測器ドライバを通じて制御します。

制御の際は、IVI コンフィグレーション・ストアの内容に従って計測器ドライバの DLL を選択し、間接的にロードされた計測器ドライバに、機種に依存しないクラス・インターフェースの関数を通じてアクセスします。

IVI コンフィグレーション・ストアには、XML ファイル（C:/ProgramData/IVI Foundation/IVI/IviConfigurationStore.xml）が使用されます。主に IVI 計測器ドライバや一部の VISA/IVI コンフィグレーション・ツールが、IVI Configuration Server DLL を通じてアクセスします。アプリケーションが利用することは通常はありません。LabWindows/CVI を使用する場合は、National Instruments 社製のソフトウェア NI-MAX（NI Measurement and Automation Explorer）を使用して IVI ドライバのコンフィグレーションを行います。

以降、NI-MAX を使用して仮想インストルメントを作成する手順について説明します。

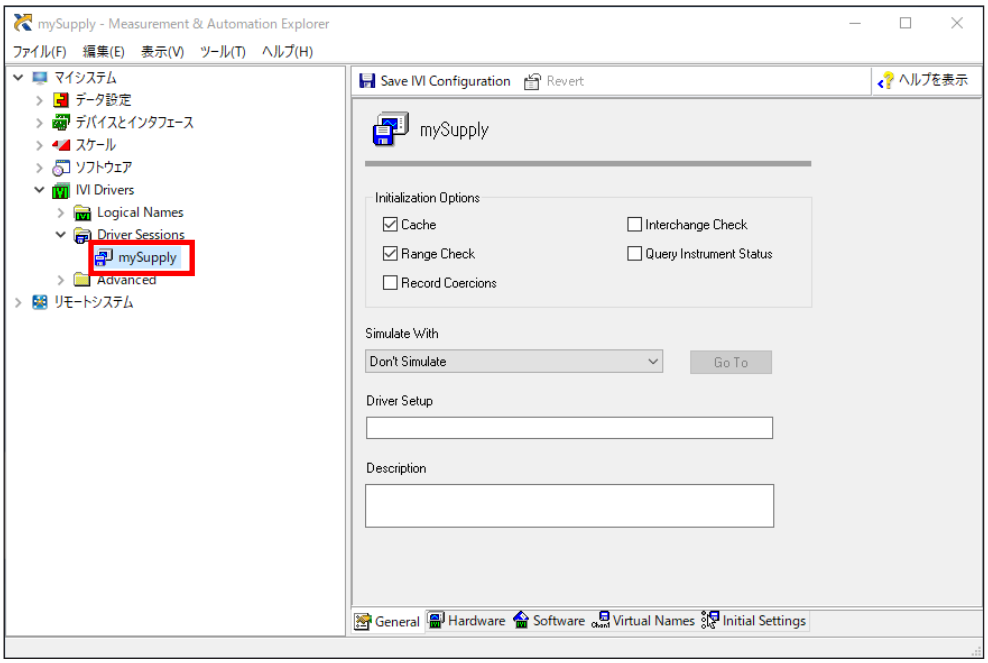
Driver Sessions を作成する

1 **NI-MAX を起動し、画面左のツリー表示で［IVI Drivers］以下の階層を確認します。**

2 **［Driver Sessions］を右クリックし、［Create New（case-sensitive）］を選択します。**

3 **Driver Sessions の名前を設定します。**

ここでは［mySupply］とします。

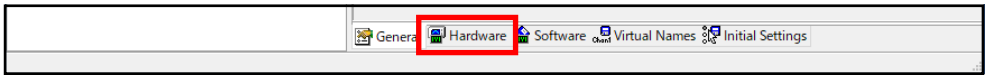


Driver Sessions が作成されました。

Hardware Asset を作成する

Hardware Asset（ハードウェア・アセット）では、使用する計測器をどのような経路で接続するかを指定します。

1 **［Hardware］タブを選択します。**



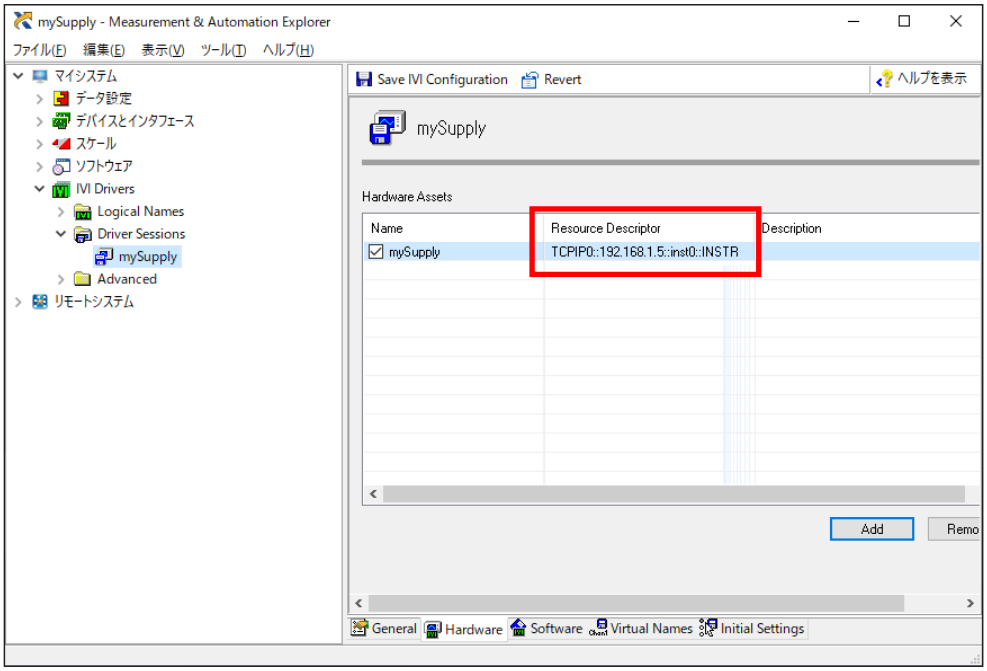
2 **［Add］をクリックして Hardware Asset を新規作成します。**

3 **Hardware Asset の名前を設定します。**

ここでは［mySupply］とします。

4 **［Resource Descriptor］に、使用する計測器が接続されている VISA アドレスを指定します。**

この例では、「TCPIP::192.168.1.5::inst0::INSTR」を指定しています。

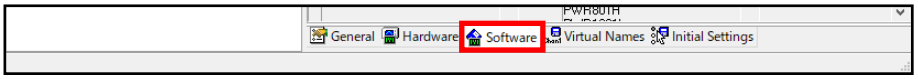


Hardware Asset が作成されました。

Software Module を設定する

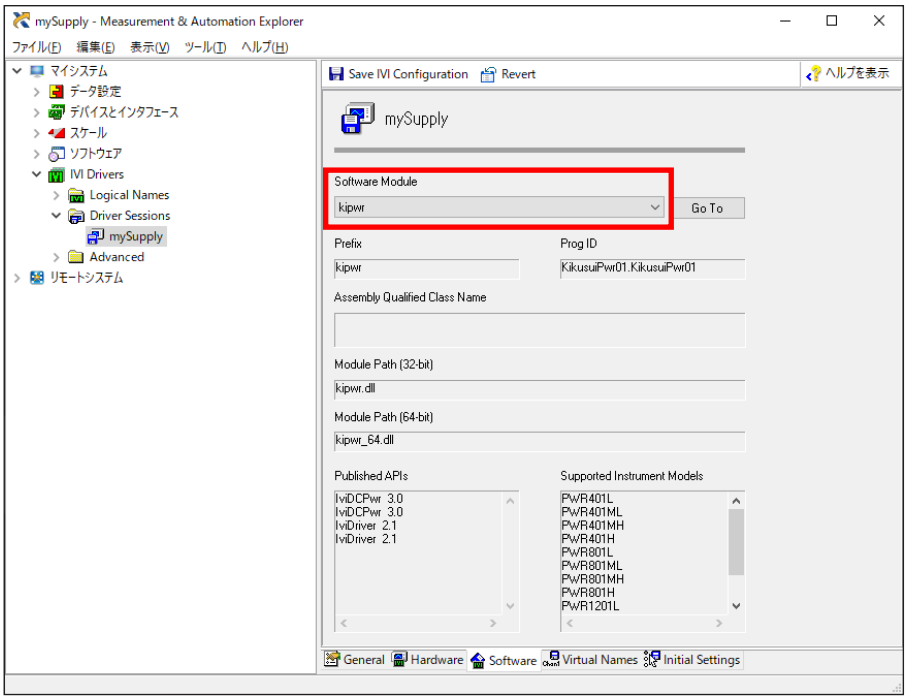
Software Module（ソフトウェア・モジュール）では、計測器ドライバモジュール（DLL モジュール）を設定します。

1 ［Software］タブを選択します。



2 ［Software Module］のリストから、使用する計測器ドライバモジュールを選択します。

この例では、［kipwr］を選択しています。

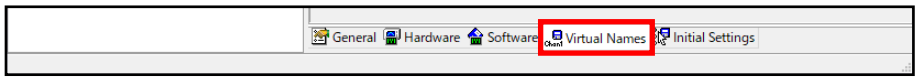


Software Module が設定されました。

Virtual Name を作成する

Virtual Name（バーチャルネーム）では、計測器ドライバのチャンネル名を仮想化した名称を作成します。チャンネルの指定が必要な計測器ドライバの場合、有効なチャンネル名は計測器ドライバによって異なるためです。

1 **［Virtual Names］タブを選択します。**



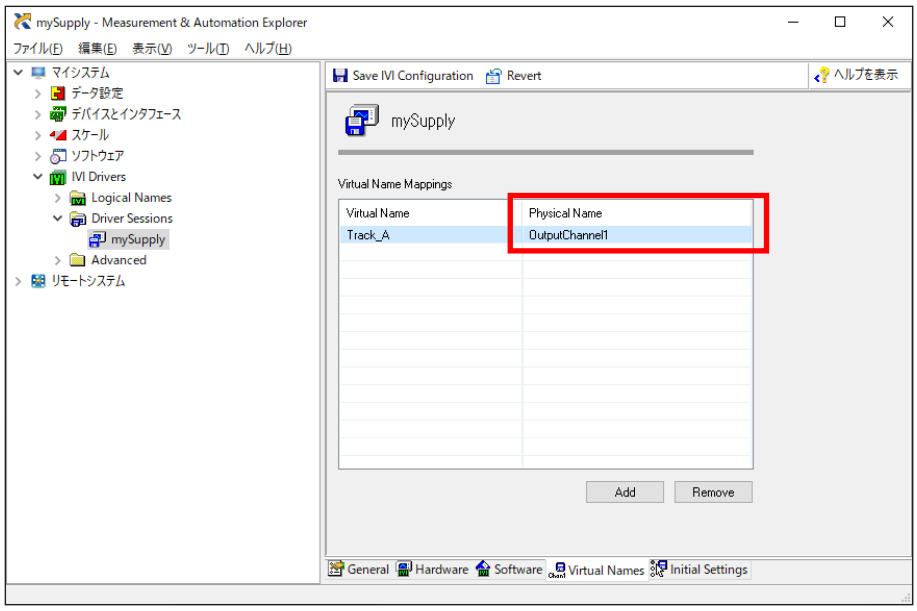
2 **［Add］をクリックしてバーチャルネームを追加し、［Virtual Name］に［Track_A］を入力します。**

［Physical Name］リストには、計測器が稼動するチャンネルが表示されます。

3 **［Physical Name］の列で、リストに表示されているチャンネル名を選択するか、または有効なチャンネル名を入力します。**

この例では、［OutputChannel1］を選択または入力します。

> **Memo**
>
> ドライバの実装状態や、多チャンネル電源装置での構成によっては、すべてのチャンネル名が表示されない場合があります。ドライバの利用可能なチャンネル名については、ドライバごとの Readme 文書またはオンライン・ヘルプなどを参照してください。



Virtual Name が作成されました。

Logical Name を設定する

Logical Name（ロジカル・ネーム）とは、NI-MAX で設定される仮想計測器の名称です。

1 **画面左のツリー表示で［IVI Drivers］以下の階層を確認します。**

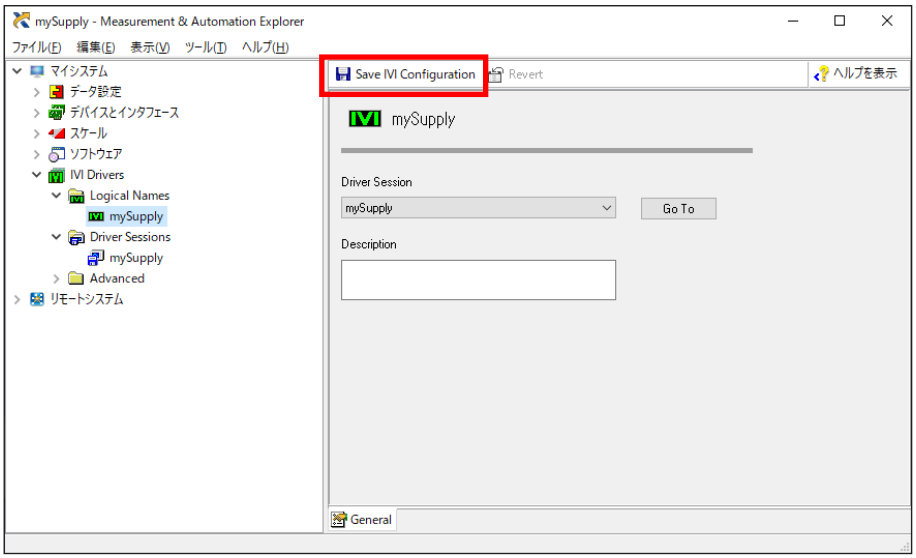
2 **[Logical Name]を右クリックして[Create New(case-sensitive)]を選択します。**

3 **Logical Name の名前を設定します。**

ここでは［mySupply］とします。

4 **［Driver Session］で［mySupply］を選択します。**

5 **ツールバーの［Save IVI Configuration］をクリックし、設定を保存します。**



Logical Name が設定されました。

以上で、仮想インストルメントの作成は完了です。

新規のプロジェクトを作成する

新規アプリケーションで IVI-C ドライバを使う例を説明します。

1 **LabWindows/CVI を起動します。**

Welcome ページが表示されます。

2 **［New］＞［Project］をクリックします。**

Welcome ページが表示されずに既存プロジェクトが読み込まれる場合には、［File］＞［New］＞［Project（*.prj)］をクリックしてください。

新規プロジェクトが作成されます。

3 **［File］＞［Save Untitled.prj As］をクリックしてプロジェクトを保存します。**

ここでは、"Ex02.prj" として保存します。

4 **［File］＞［New］＞［Source (*.c)］をクリックします。**

ソースファイルが作成されます。

5 **［File］＞［Save Untitled1.c As］をクリックしてソースファイルを保存します。**

ここでは、"Ex02.c" として保存します。

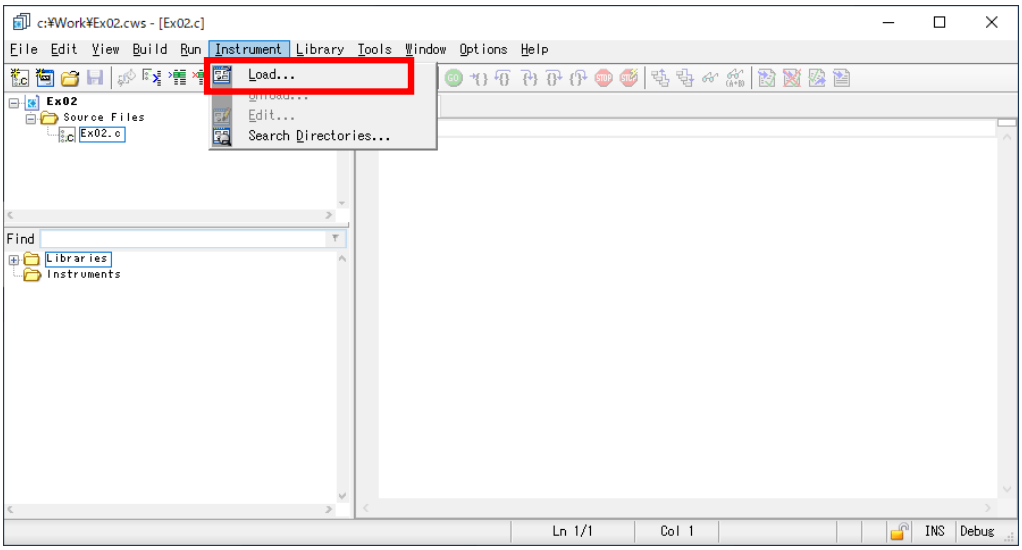
6 **［File］＞［Add Ex02.c to Project］をクリックします。**

ソースファイルがプロジェクトに追加されます。

以上で、新規プロジェクトの作成は完了です。

計測器ドライバをロードする

1 **［Instrument］＞［Load］をクリックします。**



2 **"C:/Program Files (x86)/IVI Foundation/IVI/Drivers/IviDCPwr" ディレクトリに保存されている "IviDCPwr.fp" をロードします。**

［Instrument］メニューの中に、［IviDCPwr Class Driver］が追加されます。

プログラムを構成する

Initialize With Options 関数を設定し、変数を宣言してソースコードに挿入します。

Initialize With Options 関数を設定する

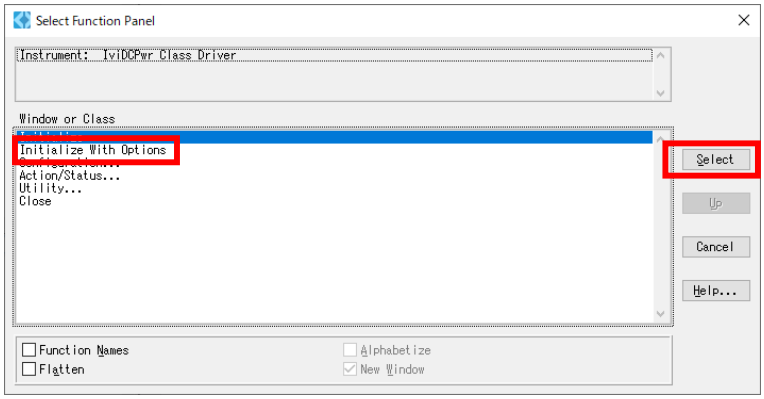
パラメータを設定する

1 **プロジェクトに追加された C のソースコード（Ex02.c）を開きます。**

2 **［Instrument］＞［IviDCPwr Class Driver］をクリックします。**

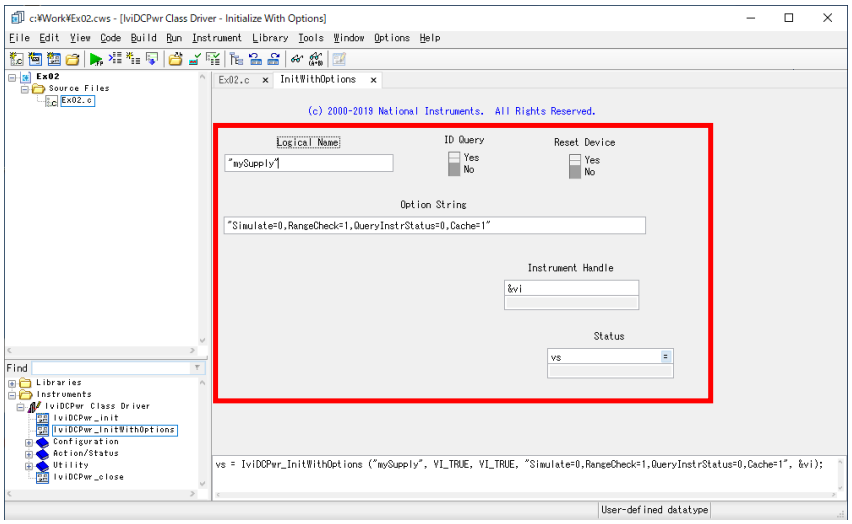
Select Function Panel が表示されます。

3 **［Initialize With Options］をクリックし、［Select］をクリックします。**



Initialize With Options のファンクションパネルが表示されます。

4 **各パラメータに値を設定します。**



Logical Name と OptionString のパラメータは文字列なので、引用符で囲んでください。

パラメータ	値
Logical Name	"mySupply"
ID Query	Yes 関数コール記述では VI_TRUE）
Reset Device	Yes 関数コール記述では VI_TRUE）
Option String	デフォルトのまま
Instrument Handle	&vi
Status	vs

変数を宣言する

Instrument Handle と Status に入力した変数を宣言します。

1 **ファンクションパネルでパラメータ Instrument Handle をクリックします。**

2 **［Code］＞［Declare Variable］をクリックします。**

Declare Variable ダイアログが表示されます。

3 **Execute declaration in Interactive Window と Add declaration to top of target file "Ex02.c" の両方をチェックします。**

4 **［OK］をクリックします。**

5 **ファンクションパネルでパラメータ Status をクリックします。**

6 **手順 2 ～手順 4 と同様に操作します。**

宣言が完了しました。

Initialize With Options 関数の呼び出しコードを挿入する

1 **［Code］＞［Insert Function Call］をクリックします。**

IviDCPwr_InitWithOptions 関数の呼び出しコードがソースコード（Ex02.c）に挿入されます。

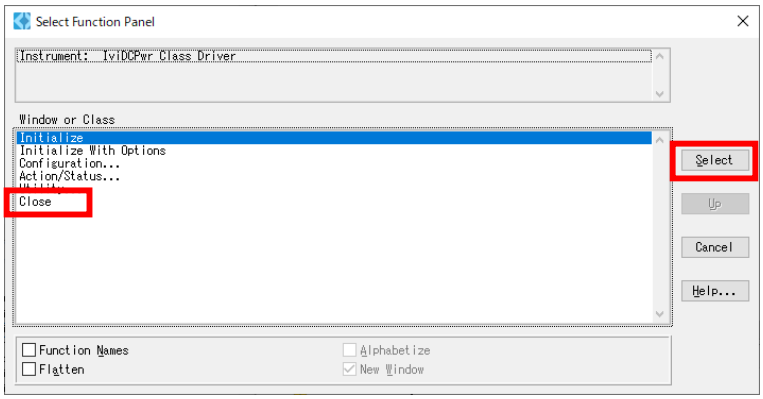
Close 関数を設定する

Initialize With Options 関数と同様に Close 関数を設定し、ソースコードに挿入します。

1 **［Instrument］＞［IviDCPwr Class Driver］をクリックします。**

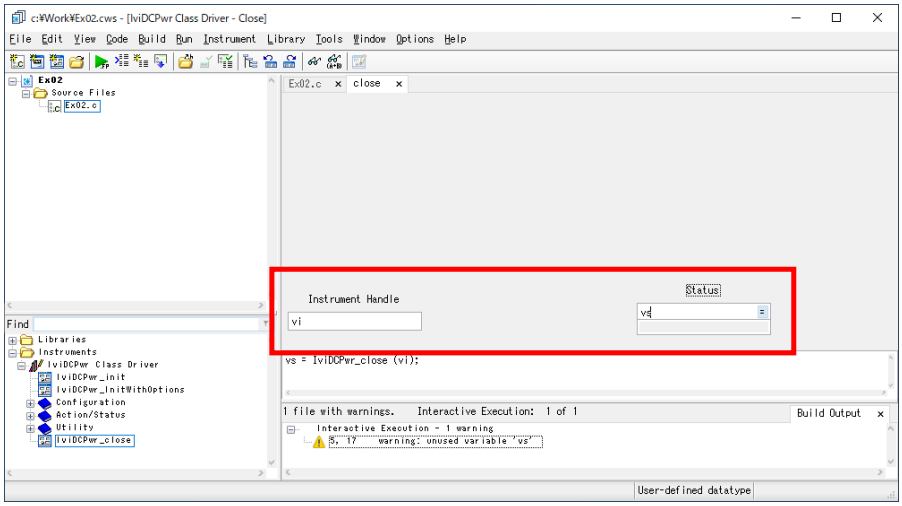
Select Function Panel が表示されます。

2 **［Close］をクリックし、［Select］をクリックします。**



Close のファンクションパネルが表示されます。

3 **パラメータ［Instrument Handle］に vi と入力し、［Status］に vs と入力します。**



4 **［Code］＞［Insert Function Call］をクリックします。**

IviDCPwr_close 関数の呼び出しコードがソースコード（Ex02.c）に挿入されます。

そのほかの関数を追加する

［Ex02.c］タブを選択すると、ソースコードが表示されます。p.25 ～ p.27 を実行すると、ソースコードは下記のようになります。

```
static ViStatus vs;
static ViSession vi;
vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
"Simulate=0,RangeCheck=1,QueryInstrStatus=0,Cache=1", &vi);
vs = IviDCPwr_close (vi);
```

このソースコードを実行可能なプログラムにするため、次の内容を記述します。

- 計測器ドライバのインクルードファイルの読み込みを指定します。
- 変数 vi と vs の初期値を 0 にします。
- InitWithOptions と close の呼び出しを main 関数で囲みます。
- InitWithOptions と close の呼び出しの間に、電圧と電流を設定する関数、出力 ON/OFF を制御する関数の呼び出しコードを追加します。ファンクションパネルを使って挿入する方法のほかに、下記のようにソースコードをエディタで直接入力する方法もあります。

下記は、計測器ドライバセッションをオープンし、電圧、電流、出力を設定し、すぐにクローズするサンプルです。

```
#include <IviDCPwr.h>
static ViSession vi = 0;
static ViStatus vs = 0;
void main()
{
vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
"Simulate=0,RangeCheck=1,QueryInstrStatus=0,Cache=1", &vi);
vs = IviDCPwr_ConfigureVoltageLevel (vi, "Track_A", 20);
vs = IviDCPwr_ConfigureCurrentLimit (vi, "Track_A", IVIDCPWR_VAL_CURRENT_
     REGULATE, 2.0);
vs = IviDCPwr_ConfigureOutputEnabled (vi, "Track_A", 1);
vs = IviDCPwr_close (vi);
}
```

プロジェクトをビルドする

1 **［Build］＞［Build］をクリックします。**

ビルドが完了します。

プログラムの実行

プログラム実行時のデバッグに役立つブレークポイントの紹介と、プログラムの実行方法、実行後に変数に格納される値について説明します。

ブレークポイントを設定する

p.28 のサンプルは、プログラムが対話形式ではないため、実行したときの経緯がわかりません。呼び出し行にブレークポイントを挿入すると、呼び出し前にプログラムが一時停止するため、実行中のプログラムの状態が確認できます。

1 **ブレークポイントを挿入したい行にカーソルを移動します。**

2 **［Run］＞［Toggle Breakpoint］をクリックします。**

ブレークポイントが設定されます。

プログラムを実行する

1 **［Run］＞［Debug Ex02.exe］をクリックします。**

プログラムが実行され、ブレークポイントを設定した行を呼び出す前に一時停止します。

2 **［Run］＞［Continue］をクリックします。**

プログラムの実行が再開されます。

変数 vi と vs に格納される値

計測器ドライバのセッションをオープンできた場合には、vi にはセッションハンドル（通常は IVI ハンドルとして 0x00000001 以上）が格納されます。vs には実行結果（成功した場合は 0x00000000、失敗した場合は負の値）が格納されます。

関数の解説

プログラムを構成する関数について、IviDCPwr クラス・ドライバを使用した場合を例に説明します。VI（関数）に付くプレフィックスである「IviDCPwr」は、IviDCPwr クラス・ドライバ固有のものです。

セッションの開始

クラス･インターフェースのセッションを開始するには､IviDCPwr_InitWithOptions を使用します。

```
vs = IviDCPwr_InitWithOptions ("mySupply", VI_TRUE, VI_TRUE,
"Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);
```

IviDCPwr_InitWithOptions で設定できるパラメータは以下のとおりです。

パラメータ	説明
Logical Name	NI-MAX で作成した仮想インストルメントのロジカル･ネームを指定します。クラス･ドライバはロジカル・ネームから適切な計測器ドライバ DLL(Software Module) や VISA アドレス (Hardware Asset) を探し当て、最終的に kipwr_InitWithOption を間接的に呼び出します。クラス・ドライバは IviDCPwr_InitWithOptions 関数に直接 VISA アドレスを渡すことはできません。
ID Query	VI_TRUE を指定した場合、計測器に対して "*IDN?" などの ID クエリを発行して機種情報を問い合わせます。
Reset Device	VI_TRUE を指定した場合、"*RST" コマンドなどを発行して計測器の設定をリセットします。
Option String	IVI 定義の設定を変更できます。（p.31）
Vi	計測器セッションを受け取ります（ポインタ渡し）。

OptionString を設定するには

OptionString では、下記の IVI 定義を設定できます。

- RangeCheck
- Cache
- Simulate
- QueryInstrStatus
- RecordCoercions
- Interchange Check

OptionString は文字列のパラメータです。下記に、サンプルを記載します。

```
QueryInstrStatus = VI_TRUE, Cache = VI_TRUE, DriverSetup=12345
```

書式について、次の点に注意してください。

- 設定値を指定しない場合にはデフォルト値が適用されます。デフォルト値は、IVI コンフィグレーションの［Driver Session］＞［General］ページで指定された値となります。
- 設定値は ViBoolean 型です。「VI_TRUE」、「VI_FALSE」、「1」、「0」のいずれかが設定できます。
- 機能名称および設定値はケース・インセンシティブです。大文字と小文字は区別されません。
- 複数の項目を設定する場合には、カンマで区切ります。
- 計測器ドライバによっては、「DriverSetup」パラメータがサポートされています。「DriverSetup」は、IVI 仕様書では定義されない項目を InitWithOptions の呼び出し時に指定するパラメータです。利用目的や書式はドライバに依存します。そのため、「DriverSetup」を設定する場合には、「Option String」の最後の項目として指定します。「DriverSetup」の指定内容については、ドライバの Readme またはオンライン・ヘルプなどを参照してください。

チャンネル名の設定

電源装置やオシロスコープなどの場合、IVI 計測器ドライバは複数のチャンネルが装備されていることを前提に設計されています。そのため、計測器のパネル設定について操作をするドライバ関数の多くは、下記のように第 2 パラメータにチャンネルを指定する必要があります。

```
vs = IviDCPwr_ConfigureVoltageLevel( vi, "Track_A", 20.0);
```

この例では、チャンネル名に「Track_A」を指定しています。Track_A は IVI コンフィグレーション（p.17）で設定したバーチャルネームです。バーチャルネームを指定すると、特定機種の計測器ドライバに依存しないプログラミングができます。

IVI コンフィグレーションでは、「Track_A」というバーチャルネームを、「OutputChannel1」という特定の計測器ドライバ（この場合は kipwr ドライバ）だけで使用できるチャンネル名に変換するように設定しました。そのため、計測器を交換した場合には、プログラムを書き換えることなく IVI コンフィグレーションの設定を変更するだけで動作を継続できます（p.34）。

計測器ドライバに依存したチャンネル名を直接指定しても計測器を制御することはできますが、インターチェンジャビリティは損なわれます。たとえば、AgN57xx 計測器ドライバで有効なチャンネル名は「Output1」であるため、チャンネル名を「OutputChannel1」で指定している場合、計測器を AgN57xx に交換するにはプログラムを書き換える必要があります。

Memo

- IVI コンフィグレーションの設定情報が保存されている XML ファイル（IviConfigurationStore.xml）を手動で編集しないでください。
- IVI コンフィグレーションは、同一 PC 内のすべての 32bit/64bit 計測アプリケーション、およびすべてのログオンユーザで共有されます。

セッションのクローズ

計測器ドライバのセッションをクローズするには、IviDCPwr_close を使います。

```
vs = IviDCPwr_close (vi);
```

エラー処理

範囲外の値をパラメータに渡したり、サポートされていない機能を呼び出したりすると、計測器ドライバにエラーが発生します。IVI-C 計測器ドライバでは、計測器ドライバ内で発生したエラーはすべて下記のような ViStatus 型の戻り値としてクライアントプログラムに伝えられます。

値の範囲	説明
vs = 0	成功
vs > 0	警告
vs < 0	エラー

計測器を交換する

計測器を交換した場合は、仮想インストルメント（IVI コンフィグレーション）の Driver Session を変更するだけで、動作を継続できます。アプリケーション自体を変更する必要はありません。

変更する Driver Session の設定は、以下の 3 つです。

項目	設定内容
［Hardware］タブ＞［Hardware Assets］＞［Resource Descriptor］	計測器の接続先 VISA アドレス
［Software］タブ＞［Software Module］	使用する計測器ドライバ
［Virtual Names］タブ＞［Physical Names］	仮想チャンネル名のマップ先物理名

交換した計測器に合わせて正しく設定すれば、アプリケーションを再度コンパイル・リンクすることなく動作します。

本ガイドの例では、計測器を Kikusui PWR-01 シリーズ直流電源（kipwr 計測器ドライバでホストされる計測器）から Agilent N5700 シリーズ直流電源（AgN57xx ドライバでホストされる計測器）に交換した場合、［mySupply］の設定を以下のように変更します。

項目	変更内容
［Hardware］タブ＞［Hardware Assets］＞［Resource Descriptor］	Kikusui PWR-01 シリーズ直流電源の接続先 VISA アドレス ⇒ Agilent N5700 シリーズ直流電源の接続先 VISA アドレス
［Software］タブ＞［Software Module］	「kipwr」⇒「AgN57xx」
［Virtual Names］タブ＞［Physical Names］	「OutputChannel1」⇒「Output1」

Memo

IVI クラス・ドライバを利用したインターチェンジャビリティ機能は、計測器の交換前後での動作を保証するものではありません。交換後のシステムが正常に機能するかどうかを十分に検証してから運用してください。