IVI-COM 計測器ドライバ プログラミング・ガイド (LabWindows/CVI 編)

Sep 2005 Revision 2.0

1- 概要

1-1 IVI-C ドライバのサポート

LabWindows/CVI は C 言語を前提にした開発環境です。IVI-COM 計測器ドライバを直接利用することも可能ですが、IVI-C 又は VXI Plug&Play 計測器ドライバがあればそれを利用したほうがプログラミングは簡単になります。当社の IVI-COM 計測器ドライバは、IVI-COM インターフェースだけでなく IVI-C によるプログラミングインターフェースをサポートしています。IVI-C 仕様は VXIplug&play 計測器ドライバ仕様を進化させたもので、LabWindows/CVI で使うのには最も適したドライバタイプです。

Notes:

当社の IVI-COM 計測器ドライバでは、バージョン表記が 2.x.x.x 以上であれば IVI-COM と IVI-C の両方のインターフェースをサポートしています。バージョン表記が 1.x.x.x の場合は IVI-C をサポートしていないので注意してください。

IVI-C 計測器ドライバを使用するには、NI IVI Compliance Package 2.x を別途インストールする必要があります。このパッケージは当社製の IVI 計測器ドライバをインストールしても自動的にはインストールされません。また全てのバージョンの LabWindows/CVI がそれをインストールするわけでもありません。

本ガイドブックでは、IVI-COM Kikusui4800 計測器ドライバ(KIKUSUI PIA4800 シリーズ電源コントローラ)を使用する例を示します。他機種用の IVI-COM 計測器ドライバでも、ほぼ同様の手順で使用できます。

本ガイドブックでは、LabWindows/CVI 7.1 を使用した場合を例に説明しています。

IVI 計測器ドライバを利用する場合、スペシフィック・インターフェースを利用する方法とクラス・インターフェースを利用する方法の 2 種類があります。前者は計測器ドライバの固有インターフェースを利用するもので、使用する計測器の機能を最大限に利用する事ができます。後者は 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を利用するもので、インターチェンジャビリティ機能を利用する事ができますが、機種固有の機能を使うことは制限されます。

Notes:

計測器ドライバが所属する計測器クラスについては、ドライバ毎の Readme.txt に記載されています。Readme 文書は、Start ボタン➔Program➔IVI フォルダから開く事ができます。

計測器ドライバが如何なる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する事はできません。つまりこの場合、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事は出来ません。

2- スペシフィック・インターフェースを使用するサンプル

ここでは、スペシフィック・インターフェースを使用したサンプルを示します。スペシフィック・インターフェースを使用すると、計測器ドライバで提供される機能を最大限に利用する事ができますが、インターチェンジャビリティを実現する事はできません。

2-1 アプリケーションの作成

LabWindows/CVI 統合環境を起動すると新規アプリケーションのプロジェクトが作成されます。起動時に既存プロジェクトが読み込まれてしまう場合は、**File | New | Project (*.prj)**を選択してください。このガイドブックでは、新規アプリケーションで IVI-C ドライバを使う例を示しますが、既存のプロジェクトでも同様の手順が適用できます。

新規のプロジェクトを作成したら、まず **File | Save Untitled.PRJ As...**を選択して、先に保存をしておく事を推奨します。この例では Ex01.prj とします。また、新規のプロジェクトの作成直後には C 言語のソース・ファイルがありません。**File | New | Source (*.c)...**メニューでソース・ファイルを新規作成し、それを Ex01.c として保存します。また、**File | Add Ex01.c to Project** メニューを選択して、このソース・ファイルをプロジェクトに追加します。

2-2 計測器ドライバのロード

Instrument| Load メニューを選択し、**Program Files/IVI/Drivers/ki4800** ディレクトリに置かれている **ki4800.fp** を選択します。すると **Instrument | Kikusui PIA4800 Power Supply Controller** メニューが追加されます。

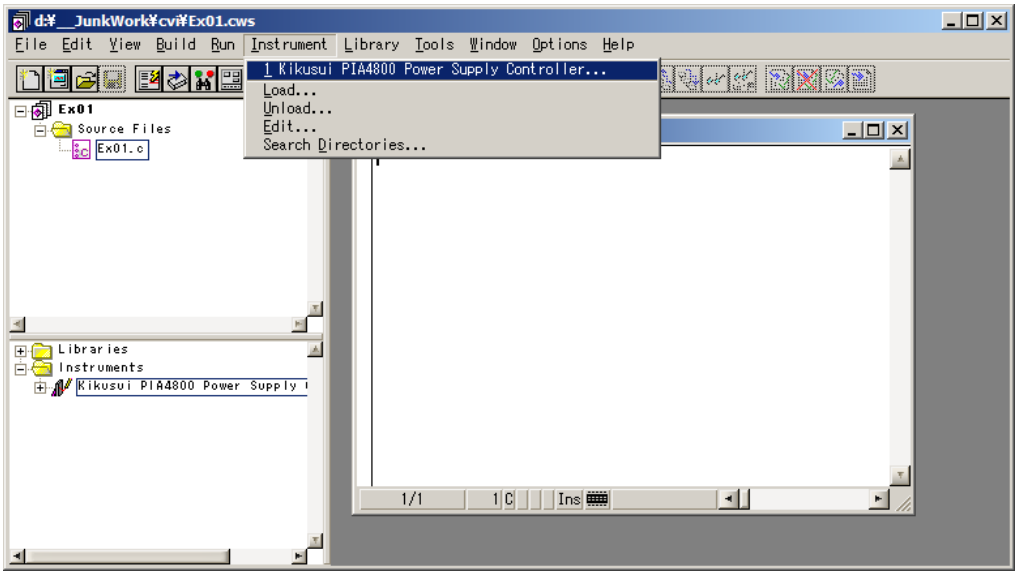


Figure 2-1 Instrument Menu

2-3 コードの記述

関数コールの挿入

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。現時点では何も書かれていません。今度は **Instrument | Kikusui PIA4800 Power Supply Controller** メニューを選択します。

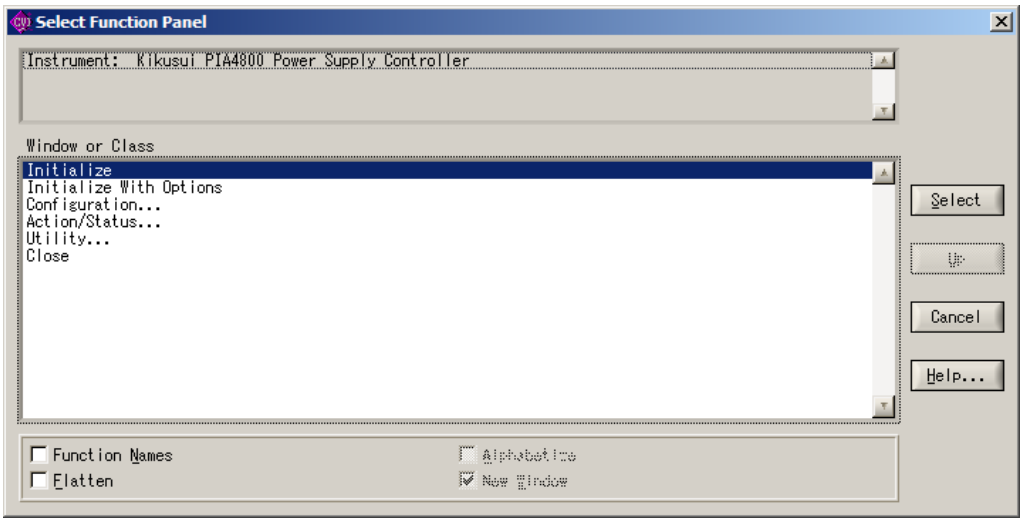


Figure 2-2 Select Function Panel

Initialize With Options を選択して **Select** ボタンをクリックします。すると Initialize With Options のファンクション・パネルが表示されます。ここでは **Instrument Handle** に`&vi` とタイプし、**Status** には vs とタイプします。**Option String** はデフォルトの状態です。また **Resource Name** には実際の計測器が接続されている VISA アドレスを指定します。そして **Code | Insert Function Call** メニュー選択して、`ki4800_InitWithOptions()`関数の呼び出しコードをソースコード(Ex01.c)に挿入します。

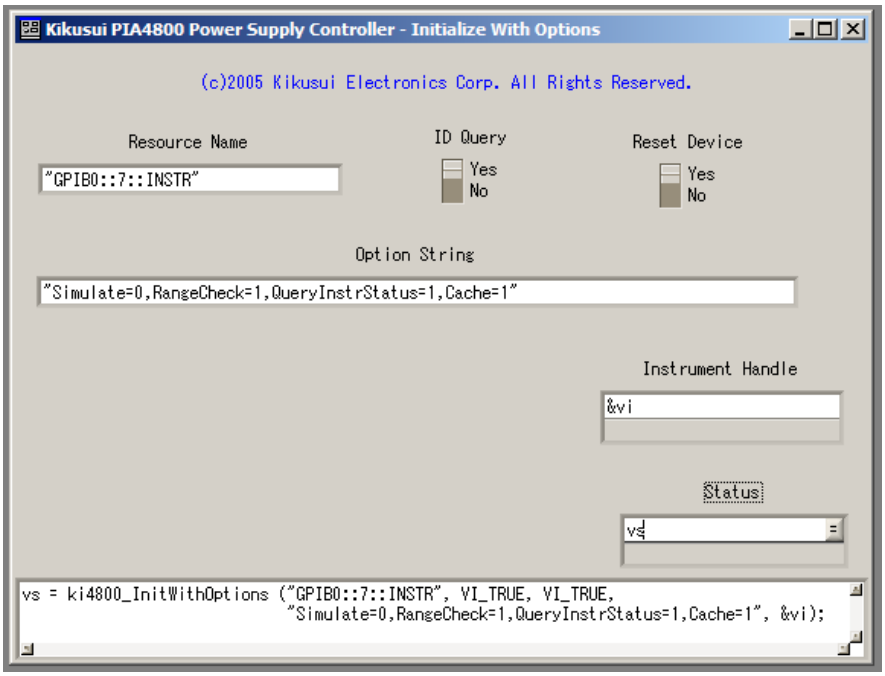


Figure 2-3 Initialize With Options Function Panel

同様に Select Function Panel ダイアログから **Close** を選択して Close ファンクション・パネルを表示します。ここでは **Instrument Handle** に `vi` とタイプし、**Status** には vs とタイプします。そして **Code | Insert Function Call** メニュー選択して、`ki4800_close()`関数の呼び出しコードをソースコード(Ex01.c)に挿入します。

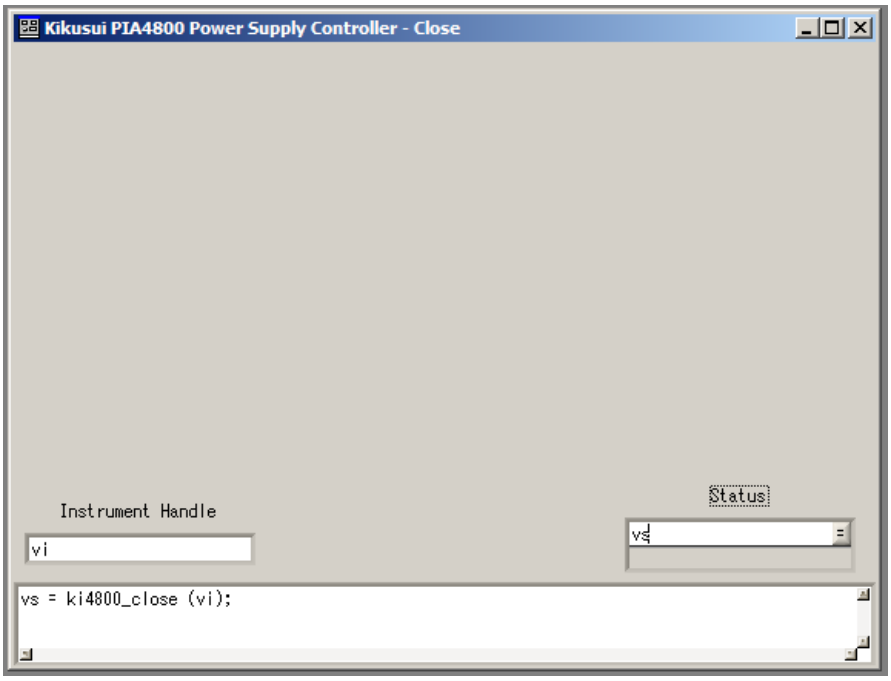


Figure 2-4 Close Function Panel

この時点で C ソースコードは下記のようになっています。

```
vs = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
    "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);

vs = ki4800_close (vi);
```

しかしこれでは実行可能プログラムとしての形になっていないので、全体を main()関数で囲み、更に計測器ドライバのインクルード・ファイルの読み込み指定を記述します。更に、変数 vi と vs の宣言が必要になるので追加します。

最後に、InitWithOptions()と close()の呼び出しの間に、電圧や電流を設定する関数、更には出力 ON/OFF 制御を行う関数呼び出しコードも追加してみます。ソースコードはソースコード・エディタに直接入力してもかまいません。

```
#include <ki4800.h>

void main()
{
ViSession vi = 0;
ViStatus vs = 0;

vs = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
    "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);

vs = ki4800_ConfigureVoltageLevel (vi, "N5!C1", 20);
vs = ki4800_ConfigureCurrentLimit (vi, "N5!C1",
KI4800_VAL_CURRENT_REGULATE, 2.0);
vs = ki4800_ConfigureOutputEnabled (vi, "N5!C1", 1);

vs = ki4800_close (vi);
}
```

プロジェクトのビルド

Build | Create Debuggable Executable メニューを選択し、プロジェクトをビルドしてみてください。特にエラーがなければ直ぐにビルドは完了します。

2-4 プログラムの実行

上記のサンプル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ものです。いきなり実行しても、プログラムが対話形式でないため、何がどう実行されたのかわかりません。そこで `ki4800_InitWithOptions()`の呼び出し行にブレークポイントを貼ります。ブレークポイントは **Run | Toggle Breakpoint** (又は F9)メニューで設定することができます。

Run | Debug Ex01_dbg.exe メニューを選択するとプログラムが実行され、ブレークポイントが設定されている `ki4800_InitWithOptions()`で停止します。**Run | Step Over** メニューを選択(又は F10)すると、その行を実行します。

`ki4800_InitWithOptions()`が呼び出された後の `vs` と `vi` に注目して下さい。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 0x00000001 以上)、`vs` にはエラーコード(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。

更に F10 を操作して、`ki4800_ConfigureVoltageLevel()`、`ki4800_ConfigureCurrentLimit()`等を順に実行します。いずれの場合も戻り値 `vs` にはエラーコードが格納されます。

`vs` が負の値の場合、関数の呼び出しによってエラーが発生した事を示しています。下記の様なコードを必要に応じて追加すれば、エラーコードを読みやすい英文メッセージに変換することもできます。

```
char buf[256];
...
ki4800_error_message (VI_NULL, vs, buf);
```

Note:

KIKUSUI PIA4800 シリーズ電源コントローラは、接続されている DC 電源の接続状態の認識(TP-BUS サーチ)を行うのに数分かかります。この処理を Initialize 時に毎回行うわけには行きません。その為、Kikusui4800(ki4800)ドライバを使用する場合は、Scan Utility を使って制御対象 DC 電源装置の接続状態を事前にコンフィグレーションしておく必要があります。ドライバを始めて使う場合は、必ず Scan Utility を実行してください。実行するには、[Start]ボタン→All Programs→IVI→Kikusui4800→Scan Utility メニューを選択します。

DC 電源装置の台数、ノード・アドレス、通信インターフェース(RS232/GPIB)、及びそれらのポート番号やアドレスを変更した場合は、再度 Scan Utility を実行する必要があります。

Scan Utility によるコンフィグレーションは、IVI-COM Kikusui4800(ki4800)ドライバ固有のものです。他の計測器ドライバでは必要ありません。

3- 解説

3-1 セッションの開始

セッションの開始には `ki4800_InitWithOptions()`関数を使用します。関数名に付く `ki4800_`というプレフィックスは計測器ドライバ毎に異なりますが、全ての IVI-C 計測器ドライバにはこのような名前ルールによる関数が用意されています。

```
vs = ki4800_InitWithOptions ("GPIB0::7::INSTR", VI_TRUE, VI_TRUE,
        "Simulate=0,RangeCheck=1,QueryInstrStatus=1,Cache=1", &vi);
```

Notes:

IVI-C 及び VXI Plug&Play 計測器ドライバの専門用語として<prefix>という表記が良く使われます。これは各計測器ドライバが固有に持つ識別用の名前で、本書の例では ki4800 がそれに該当します。例えば、<prefix>_init()という一般的表現は、ki4800 計測器ドライバでは ki4800_init()関数の事を指します。

<prefix>_init()及び<prefix>_InitWithOptions()を除く全てのドライバ関数は、第 1 パラメータに ViSession が付き、戻り値は全て ViStatus になります。

<prefix>_init()関数は VXI Plug&Play ドライバ仕様との互換性の為に残されています。OptionString パラメータを指定できない点を除き、<prefix>_InitWithOptions()と同じ動作をします。

ここで、ki4800_InitWithOptions 関数のパラメータについて説明しましょう。全ての IVI 計測器ドライバは、IVI 仕様書で定義された InitWithOptions 関数を持っています。この関数には、以下のようなパラメータがあります。

Table 3-1 InitWithOptions 関数のパラメータ

パラメータ	タイプ	説明
ResourceName	ViRsrc (const char*)	VISA リソース名の文字列。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。例えば、GPIB ボード 0 に接続されたプライマリ･アドレス 3 の計測器であれば、GPIB0::3::INSTR となる。
IDQuery	ViBoolean	VI_TRUE を指定した場合、計測器に対して ID クエリを行う。
resetDevice	ViBoolean	VI_TRUE を指定した場合、計測器の設定をリセットする。
optionString	ViConstString (const char*)	RangeCheck Cache Simulate QueryInstrStatus RecordCoercions Interchange Check に関する設定を、デフォルト以外に指定できる。更に、計測器ドライバが DriverSetup 機能をサポートする場合、その設定を行うことができる。
newVi	ViSession*	計測器セッションを受け取るパラメータ(ポインタ渡し)。

ResourceName には VISA アドレス(リソース名)を指定します。IDQuery に VI_TRUE を指定した場合は、計測器に対して"*IDN?"クエリなどを発行して機種情報を問い合わせます。resetDevice に VI_TRUE を指定した場合は、"*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 には、2 つの機能があります。1 つは RangeCheck, Cache, Simulate, QueryInstrStatus, RecordCoercions, Interchange Check,などの IVI 定義の動作を設定します。もう 1 つは、計測器ドライバ毎に独自に定義される DriverSetup を指定します。OptionString は文字列パラメータなので、これらの設定は下のサンプルのような書式でなければなりません。

```
QueryInstrStatus = TRUE , Cache = TRUE , DriverSetup=12345
```

設定したい機能の名称及び設定値はケース･インセンシティブ(大文字と小文字の区別なし)です。設定値は ViBoolean 型なので、VI_TRUE、VI_FALSE、1、0 の何れかが有効です。複数の項目を設定する場合は、コンマで区切ります。optionString パラメータで特に設定値を指定しない場合、

IVI 仕様書で定義されたデフォルト値が適用されます。IVI 仕様書で定義されたデフォルト値は、RangeCheck と Cache だけが VI_TRUE で、その他は全て VI_FALSE です。

計測器ドライバによっては、DriverSetup パラメータが意味を持つ場合もあります。これは、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もので、利用目的や書式はドライバ依存です。従って DriverSetup の指定を行う場合、それは optionString の最後の項目として指定される必要があります。DriverSetup の指定内容はドライバ毎に異なるので、ドライバの Readme 文書又はオンライン・ヘルプなどを参照してください。

3-2 チャンネルへのアクセス

IVI 計測器ドライバでは一般に、電源装置や電子負荷装置などの計測器の場合、複数のチャンネルが装備されている事を前提に設計されています。従って、計測器のパネル設定に関する操作を行うドライバ関数は、第 2 パラメータにチャンネルを指定するケースが多く見られます。

例:

```
vs = ki4800_ConfigureVoltageLevel( vi, "N5!C1", 20.0);
```

この例では直流電源装置を操作する Kikusui4800 (ki4800)ドライバを使用するので、チャンネル名には NODE と CHANNEL を含ませた表現を使用します。上記例の"N5!C1"というチャンネル名はこの計測器ドライバ固有のものであり、ドライバ毎に異なる命名法が使用されます。実際に使用できるチャンネル名の詳細は、各ドライバのオンライン・ヘルプなどを参照してください。

ここでは、PIA4800 シリーズ電源コントローラの NODE5、CHANNEL1 に接続された直流電源を 20V に設定します。

3-3 セッションのクローズ

計測器ドライバによるセッションをクローズするには、close 関数を使います。

```
vs = ki4800_close (vi);
```

4- エラー処理

これまで示したサンプルでは、エラー処理を何も行っていませんでした。しかし実際には、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、計測器ドライバがエラーを発生する事があります。また、どんなに堅牢に設計・実装されたアプリケーションでも、計測器との I/O 通信エラーは避けることが出来ません。

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全て 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。

Table 4-1 ViStatus の大まかな分類

値の範囲	説明
vs=0	成功
vs >0	警告
vs <0	エラー

ViStatus 型戻り値からエラーの内容を知ることも可能ですが、error_message( )関数を使って読みやすいメッセージに変換することもできます。この関数は例外的に ViSession として VI_NULL を受け入れます。また受け取りバッファーのサイズは必ず 256Bytes 以上にして下さい。

```
char buf[256];
...
ki4800_error_message (VI_NULL, vs, buf);
```

5- クラス・インターフェースを使用するサンプル

ここでは、クラス・インターフェースを使用したサンプルを示します。クラス・インターフェースを使用すると、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、計測器を交換する事ができます。但しその場合、交換前後の両機種に対して IVI-C 計測器ドライバが提供されており、且つそれらのドライバが同じ計測器クラスに属し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。

5-1 仮想インストルメント

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の作成を行う前にやっておかなければならない事は、仮想インストルメントの作成です。インターチェンジャビリティ機能を実現するには、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(例えば ki4800_init( )関数の呼び出し)をしたり、"GPIB0::3::INSTR"のような特定 VISA アドレス(リソース名)の記述などをするべきではありません。これらの事柄をアプリケーション内に直接記述すると、インターチェンジャビリティを損ないます。

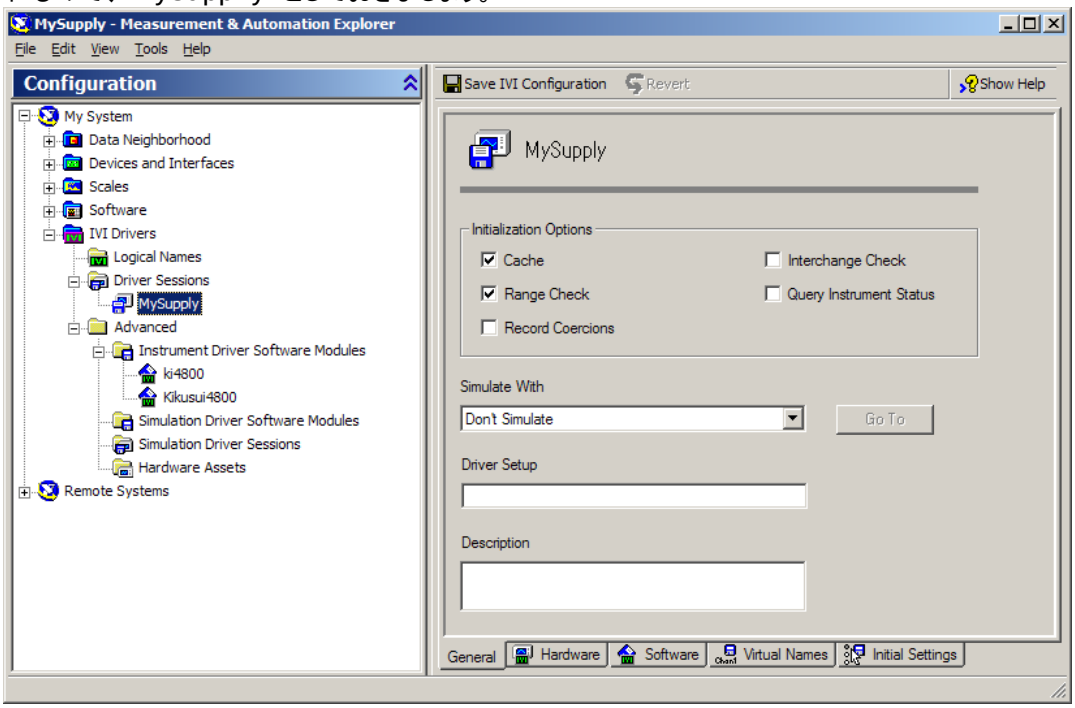
その代わりに、IVI 仕様では、計測器ドライバとアプリケーションの外部に IVI コンフィグレーション・ストアを置く事によってインターチェンジャビリティを実現します。アプリケーションは特定機種用の計測器ドライバを直接使うのではなく、クラスドライバと呼ばれる計測器ドライバを通じて計測器を制御します。その際、IVI コンフィグレーション・ストアの内容に従って計測器ドライバ DLL の選択を行い、間接的にロードされた計測器ドライバを特定機種に依存しないクラスドライバの関数を通じてアクセスします。

IVI コンフィグレーション・ストアは通常、/Program Files/IVI/Data/IviConfigurationStore.XML ファイルで、IVI Configuration Server DLL を通じてアクセスされます。この DLL を利用するのは、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・ツールであって、アプリケーションからは通常は使いません。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。

Driver Session の作成

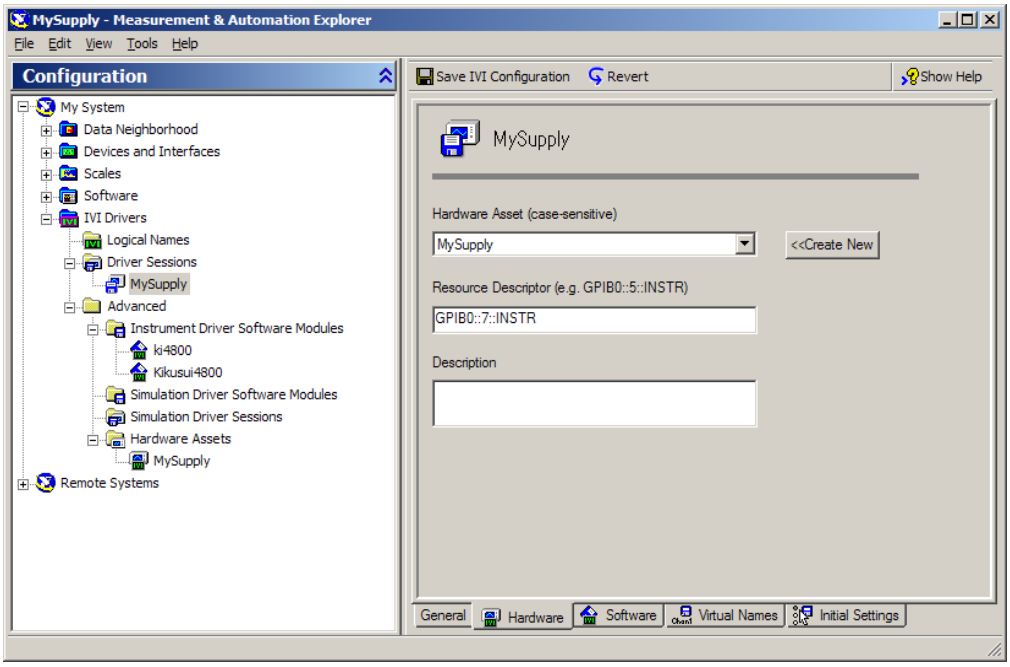
NI-MAX を起動したら、**IVI Drivers** ノードの階層を参照して下さい。**Driver Seesion** の上で右クリックして **Create New** メニューを選択し、Driver Session の新規作成を行います。名前を尋ねら

れるので、"MySupply"としておきましょう。



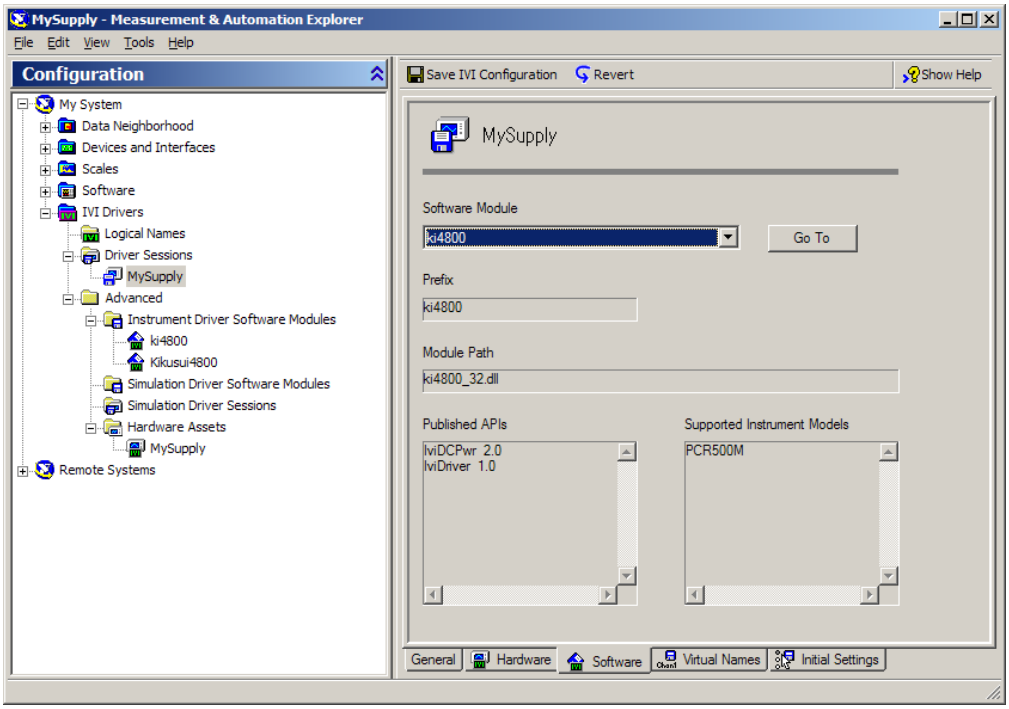
Hardware Asset の作成

引き続き **Hardware** タブを選択すると、ハードウェア・アセットの管理画面になります。ハードウェア・アセットとは、実際の計測器がどのような経路に接続されているかを示すものです。ここで **Create New** ボタンをクリックして Hardware Asset を新規作成します。新しい名前を尋ねられるので、ここでも"MySupply"として **Create** ボタンをクリックします。更に **Resource Descriptor** として、実際の計測器が接続されている VISA アドレスを指定します。



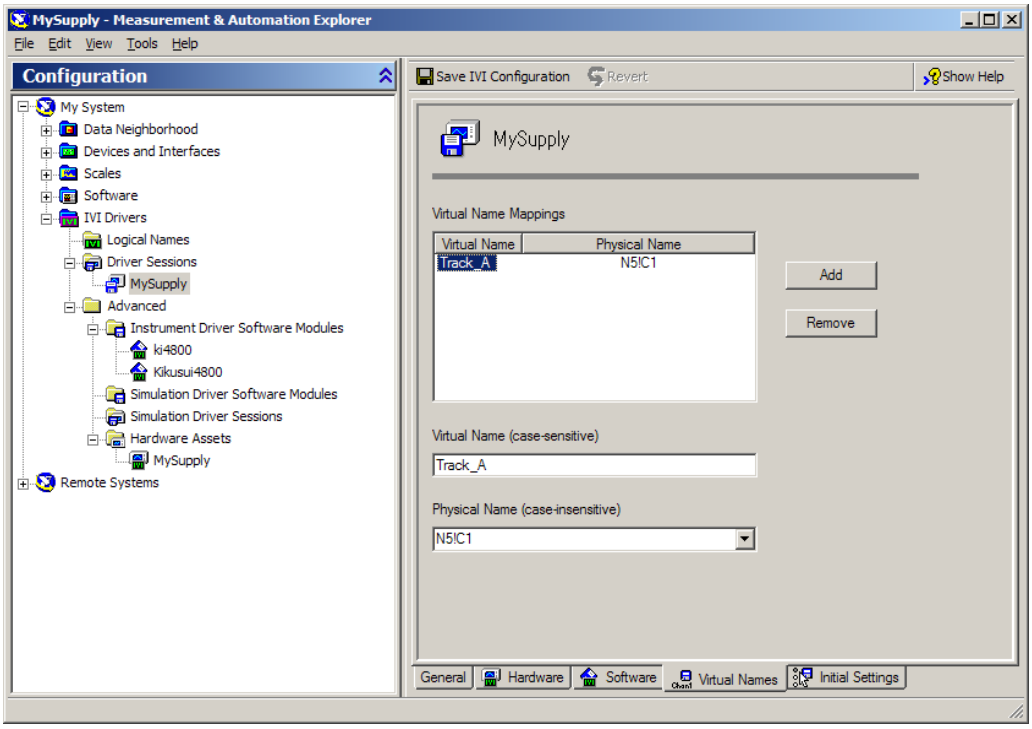
Software Module のリンク設定

引き続き **Software** タブを選択すると、ソフトウェア・モジュールの管理画面になります。ソフトウェア・モジュールとは計測器ドライバモジュール(DLL モジュール)の事を指します。ここで **Software Module** のリストから **ki4800** を選択します。



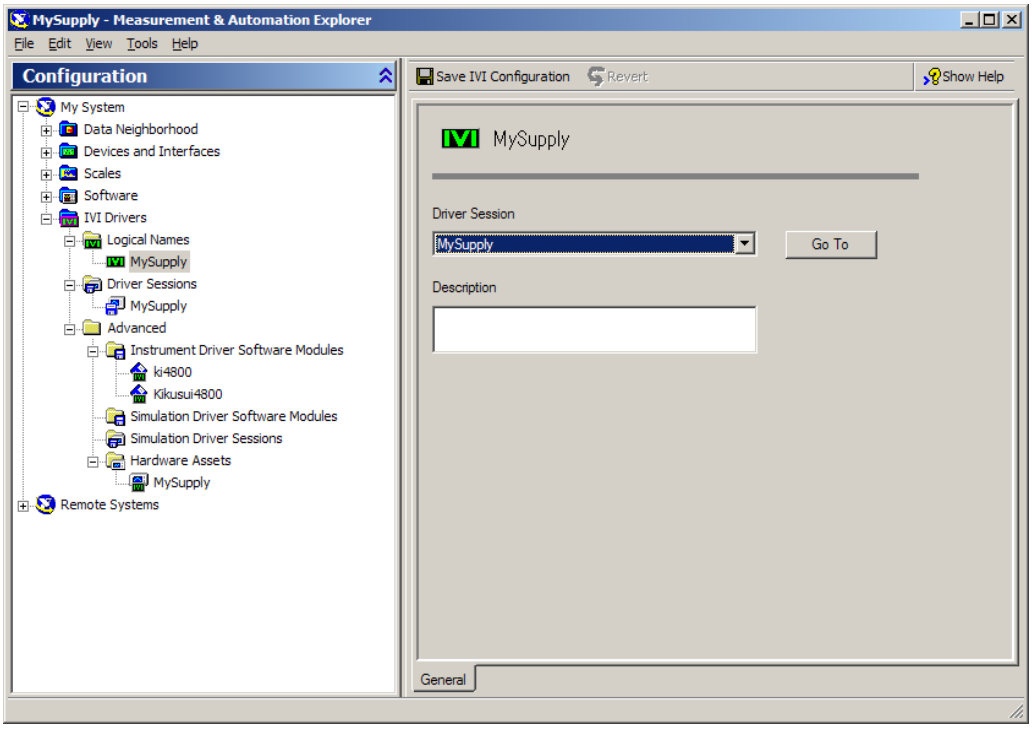
Virtual Name の作成

引き続き **Virtual Names** タブを選択すると、仮想チャンネル名の管理画面になります。通常、電源装置の計測器ドライバのようにチャンネルが関与する場合、有効なチャンネル名は計測器ドライバによって異なります。従ってそれらのチャンネル名も仮想化してやる必要があります。**Add** ボタンをクリックしてバーチャルネームを追加し、**Virtual Name** に"Track_A"と入力します。更に **Physical Name** リストから、実際の直流電源装置が接続されている NODE/CHANNEL に応じた名前を選択します。下の例では NODE 5, CHANNEL 1 に接続された場合で、**N5!C1** を選択しています。



Logical Name の作成とリンク設定

最後にロジカルネームを作成します。ロジカルネームとは、NI-MAX で設定される仮想計測器の名前に相当します。**IVI Drivers** ノードの階層を参照して下さい。**Logical Name** の上で右クリックして **Create New** メニューを選択し、Logical Name の新規作成を行います。名前を尋ねられるので、"MySupply"としておきましょう。更に、**Driver Session** リストから"MySupply"を選択します。



仮想計測器の設定はこれで終了です。NI-MAX 画面上部の **Save IVI Configuration** ボタンをクリックして設定を保存します。

5-2 アプリケーションの作成

LabWindows/CVI 統合環境を起動すると新規アプリケーションのプロジェクトが作成されます。起動時に既存プロジェクトが読み込まれてしまう場合は、**File | New | Project (*.prj)**を選択してください。このガイドブックでは、新規アプリケーションで IVI-C ドライバを使う例を示しますが、既存のプロジェクトでも同様の手順が適用できます。

新規のプロジェクトを作成したら、まず **File | Save Untitled.PRJ As...**を選択して、先に保存をしておく事を推奨します。この例では Ex02.prj とします。また、新規のプロジェクトの作成直後には C 言語のソース・ファイルがありません。**File | New | Source (*.c)...**メニューでソース・ファイルを新規作成し、それを Ex02.c として保存します。また、**File | Add Ex02.c to Project** メニューを選択して、このソース・ファイルをプロジェクトに追加します。

5-3 計測器ドライバのロード

Instrument| Load メニューを選択し、**Program Files/IVI/Drivers/ividcpwr** ディレクトリに置かれている **IviDCPwr.fp** を選択します。すると **Instrument | IviDCPwr Class Driver** メニューが追加されます。

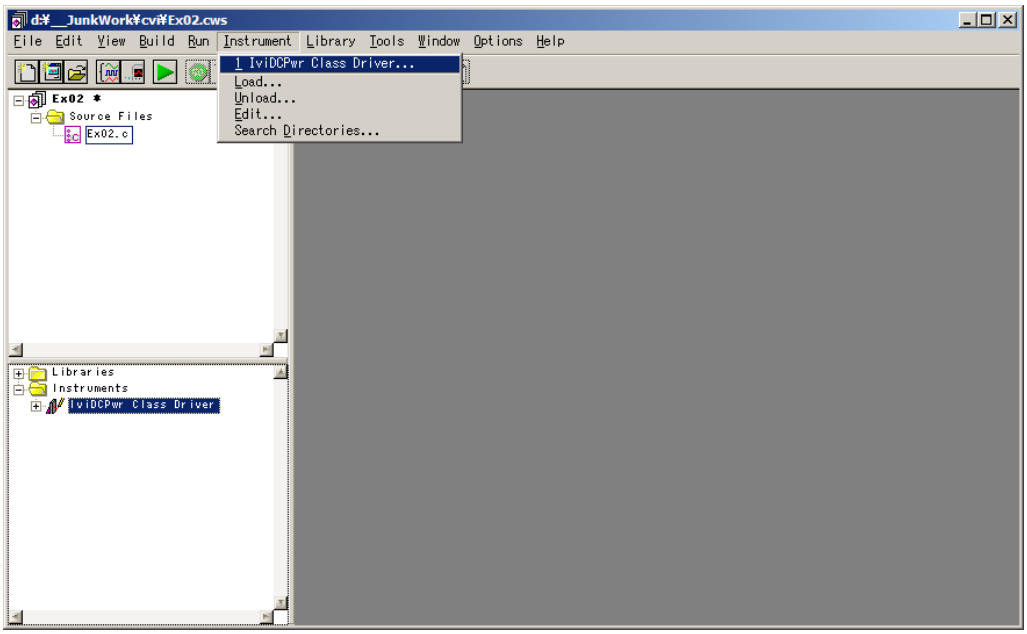


Figure 5-1 Instrument Menu

5-4 コードの記述

関数コールの挿入

プロジェクトに追加された C のソースコード(Ex02.c)開きます。現時点では何も書かれていません。今度は **Instrument | IviDCPwr Class Driver** メニューを選択します。

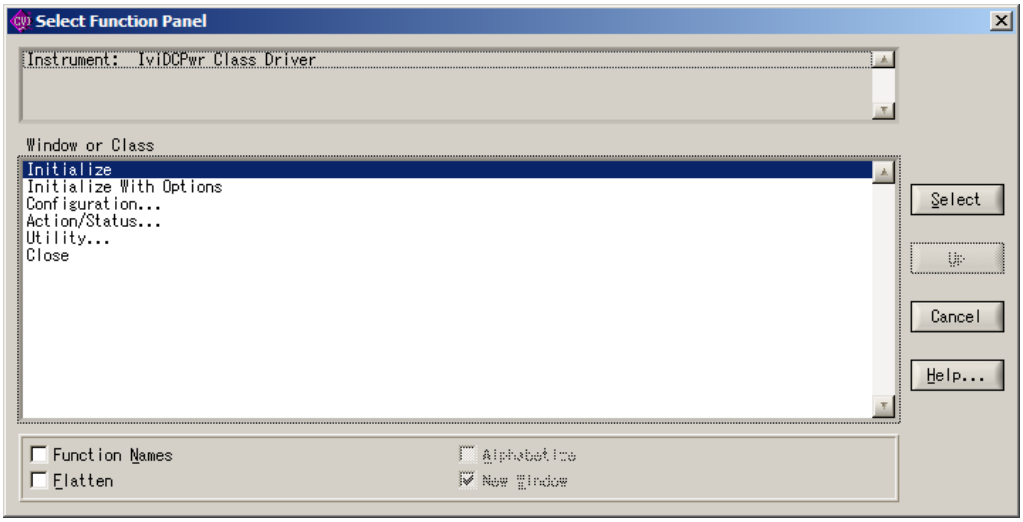


Figure 5-2 Select Function Panel

Initialize With Options を選択して **Select** ボタンをクリックします。すると Initialize With Options のファンクション・パネルが表示されます。ここでは **Instrument Handle** に`&vi` とタイプし、**Status** には `vs` とタイプします。**Option String** はデフォルトの状態です。**Logical Name** には先程の NI-MAX で設定した`"MySupply"`を指定します。そして **Code | Insert Function Call** メニュー選択して、`IviDCPwr_InitWithOptions()`関数の呼び出しコードをソースコード(Ex02.c)に挿入します。

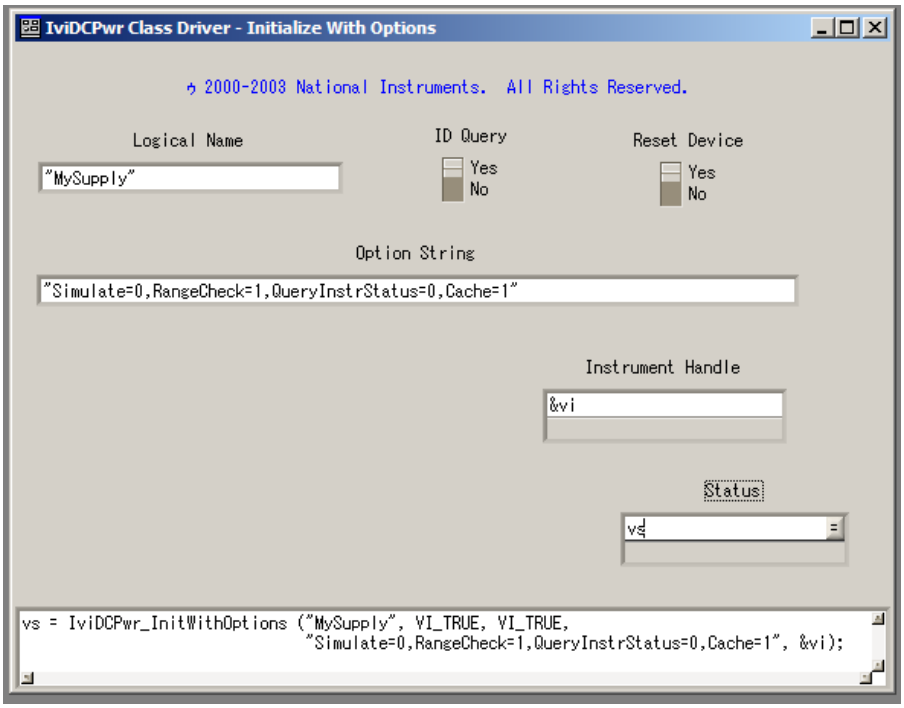


Figure 5-3 Initialize With Options Function Panel

同様に Select Function Panel ダイアログから **Close** を選択して Close ファンクション・パネルを表示します。ここでは **Instrument Handle** に `vi` とタイプし、**Status** には `vs` とタイプします。そして **Code | Insert Function Call** メニュー選択して、`IviDCPwr_close()`関数の呼び出しコードをソースコード(Ex02.c)に挿入します。

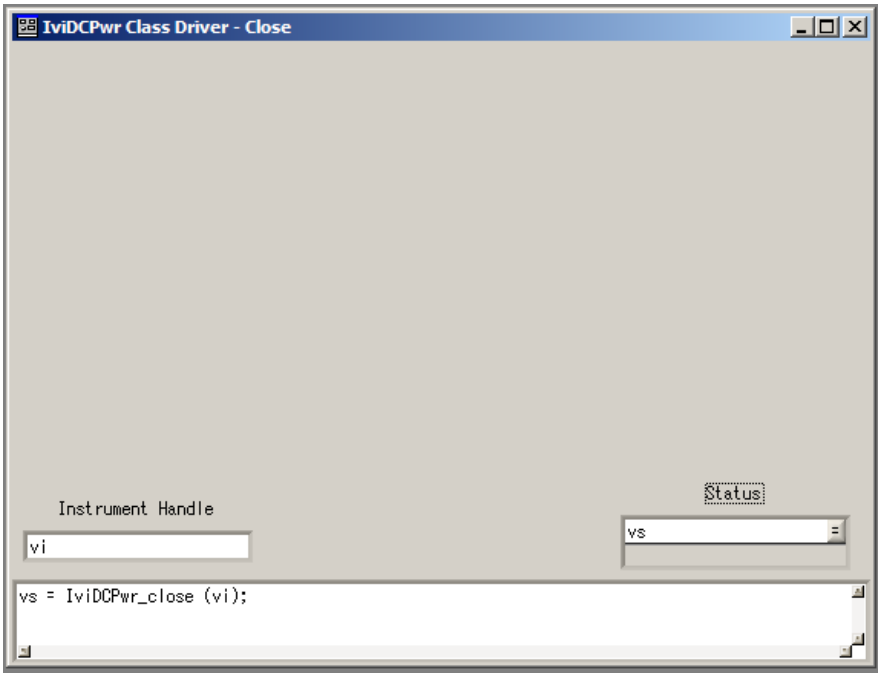


Figure 5-4 Close Function Panel

この時点で C ソースコードは下記のようになっています。

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

vs = IviDCPwr_close (vi);
```

しかしこれでは実行可能プログラムとしての形になっていないので、全体を main()関数で囲み、更にクラス・ドライバのインクルード・ファイルの読み込み指定を記述します。更に、変数 vi と vs の宣言が必要になるので追加します。

最後に、InitWithOptions()と close()の呼び出しの間に、電圧や電流を設定する関数、更には出力 ON/OFF 制御を行う関数呼び出しコードも追加してみます。ソースコードはソースコード・エディタに直接入力してもかまいません。

```
#include <IviDCPwr.h>

void main()
{
ViSession vi = 0;
ViStatus vs = 0;

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);

}
```

プロジェクトのビルド

Build | Create Debuggable Executable メニューを選択し、プロジェクトをビルドしてみてください。特にエラーがなければ直ぐにビルドは完了します。

5-5 プログラムの実行

上記のサンプル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ものです。いきなり実行しても、プログラムが対話形式でないため、何がどう実行されたのかわかりません。そこで IviDCPwr_InitWithOptions()の呼び出し行にブレークポイントを貼ります。ブレークポイントは **Run | Toggle Breakpoint** (又は F9)メニューで設定することができます。

Run | Debug Ex02_dbg.exe メニューを選択するとプログラムが実行され、ブレークポイントが設定されている IviDCPwr_InitWithOptions()で停止します。**Run | Step Over** メニューを選択(又は F10)すると、その行を実行します。

IviDCPwr_InitWithOptions()が呼び出された後の vs と vi に注目して下さい。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 0x00000001 以上)、vs にはエラーコード(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。

更に F10 を操作して、IviDCPwr_ConfigureVoltageLevel()、IviDCPwr_ConfigureCurrentLimit()等を順に実行します。いずれの場合も戻り値 vs にはエラーコードが格納されます。

6- 解説

6-1 セッションの開始

セッションの開始には IviDCPwr_InitWithOptions()関数を使用します。関数名に付く IviDCPwr_というプレフィックスは IviDCPwr クラスドライバ固有のものです。

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

クラスドライバは通常の計測器ドライバとは異なり、IviDCPwr_InitWithOptions()関数に直接 VISA アドレスを渡すことはできません。ここでは NI-MAX のロジカルネームに指定した "MySupply"を指定します。クラスドライバはこのロジカルネームを頼りに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 ki4800_InitWithOption()を間接的に呼び出します。

OptionString パラメータに渡す内容はスペシフィックドライバを直接使用する場合と同じですが、省略された場合のデフォルト値が異なります。スペシフィックドライバを直接使用する場合のデフォルトは IVI 仕様によって定義された値になりますが、クラスドライバを使用する場合のデフォルトは IVI コンフィグレーションストア内の **Driver Session** に指定された値です。

6-2 チャンネルへのアクセス

IVI 計測器ドライバでは一般に、電源装置や電子負荷装置などの計測器の場合、複数のチャンネルが装備されている事を前提に設計されています。従って、計測器のパネル設定に関する操作を行うドライバ関数は、第 2 パラメータにチャンネルを指定するケースが多く見られます。

例:

```
vs = IviDCPwr_ConfigureVoltageLevel( vi, "N5!C1", 20.0);
```

この例ではクラスドライバを使用していますが、チャンネル名には"N5!C1"という特定の計測器ドライバ(この場合は ki4800 ドライバ)でのみ使用可能な名前を指定しています。この指定方法でも計測器の制御を行うことはできますが、特定機種の計測器ドライバに依存した名前を使用するとインターチェンジャビリティを損ないます。

先の NI-MAX によるコンフィグレーションでは、バーチャルネームとして"Track_A"という名前を追加し、それが"N5!C1"というフィジカルネームに変換されるように設定しました。従ってここでは、チャンネル名にバーチャルネームを使うことができます。

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

6-3 セッションのクローズ

計測器ドライバによるセッションをクローズするには、IviDCPwr_close 関数を使います。

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

IVI-COM 計測器ドライバ・プログラミング・ガイド
本ガイドブックに登場する製品名・会社名等は各社の商標または登録商標です。