IVI-COM 計測器ドライバ プログラミング・ガイド (C#.NET 編)

Dec 2003 Revision 1.1

1- 概要

1-1 C#.NET での IVI-COM ドライバの運用

C#.NET はマネージド環境なので、アンマネージド環境で実行される IVI-COM 計測器ドライバを直接使用する事はできません。一般に COM オブジェクトをマネージド環境から使う場合は、RCW(Runtime Callable Wrapper)と呼ばれるアセンブリが必要になります。幸いこのアセンブリは、Visual Studio.NET 統合環境内の Add Reference メニュー又はコマンドライン・ユーティリティ TLBIMP.EXE(Type Library Importer)を使用して、タイプライブラリから自動生成する事ができます。

IVI-COM 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。

1-2 アプリケーション・プロジェクトの作成

このドキュメントでは、C#.NET で最も一般的なフォーム志向のアプリケーションを例に説明します。Visual Studio.NET 統合環境を起動したら、**File | New | Project** メニューを選択して **New Project** ダイアログを表示します。**Project Types** から **C# Project**、**Templates** から **Windows Application** を選択し、更にプロジェクト名を指定して **OK** をクリックします。アプリケーションのプロジェクトが新規に作成されます。

> Notes:
>
> 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 DC 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。

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 タイプライブラリのインポート

新規プロジェクトを作成したあと最初にすべき事は、利用したい IVI-COM 計測器ドライバのタイプ・ライブラリからインタロップ・アセンブリを生成し、それを参照する事です。**Project | Add References** メニューを選択して **Add References** ダイアログを表示し、**COM** タブを選択します。

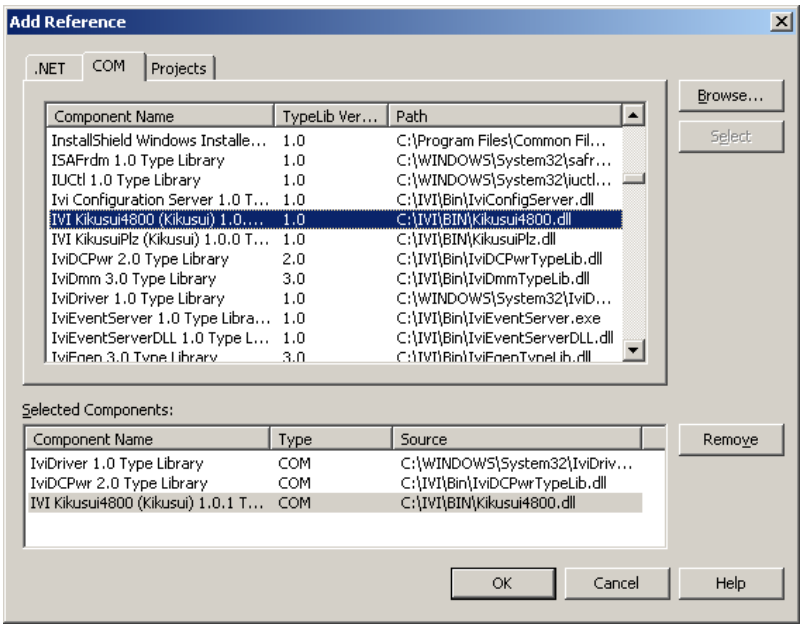

Figure 2-1 Add Reference ダイアログ

ここでは Kikusui4800 IVI-COM ドライバを使用する例を示すので、**Kikusui4800 (Kikusui) 1.0 Type Library** を選択します。更に **IviDriver 1.0 Type Library** 、及び **IviDCPwr 2.0 Type Library** も選択して下さい。

IviDriver は全ての IVI-COM 計測器ドライバに共通なので、使用する計測器ドライバに関係なく必ず選択します。**IviDCPwr** は、Kikusui4800 ドライバが IviDCPwr クラスなので選択が必要です。例えば、実際に使用する計測器ドライバが IviDmm クラスの場合には IviDmm Type Library を選択する事になります。**Select** ボタンで複数選択し、**OK** ボタンで決定します。

2-2 オブジェクト・ブラウザ

アセンブリへの参照が追加されると、Visual Studio.NET 統合環境からオブジェクト・ブラウザを通じて利用可能な構文を確認する事ができます。オブジェクト・ブラウザを起動するには、**View | Object Browser** メニューを選択します。

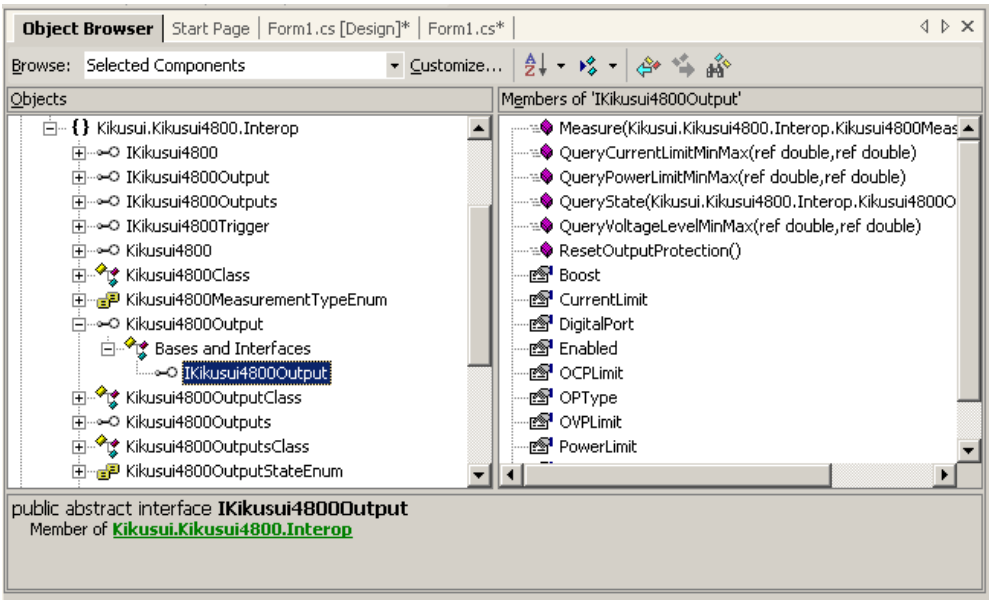

Figure 2-2 オブジェクト・ブラウザ

2-3 オブジェクトの作成とセッションのイニシャライズ

まず、フォーム設計画面の上をマウスでダブルクリックします。すると、Form1_Load イベント・ハンドラの骨格だけを持つソース・コードが表示されます。まずモジュールの先頭で下記の using 擬似命令を追加します。これは、以降のネーム・スペース参照を省略するものです。

```
using Kikusui.Kikusui4800.Interop;
```

更にフォームのデータ・メンバーとして変数 m_dcpwr を Kikusui4800Class 型で宣言してください。ここでは計測器ドライバのオブジェクトを作成するので new 演算子での初期化を忘れないように記述します。

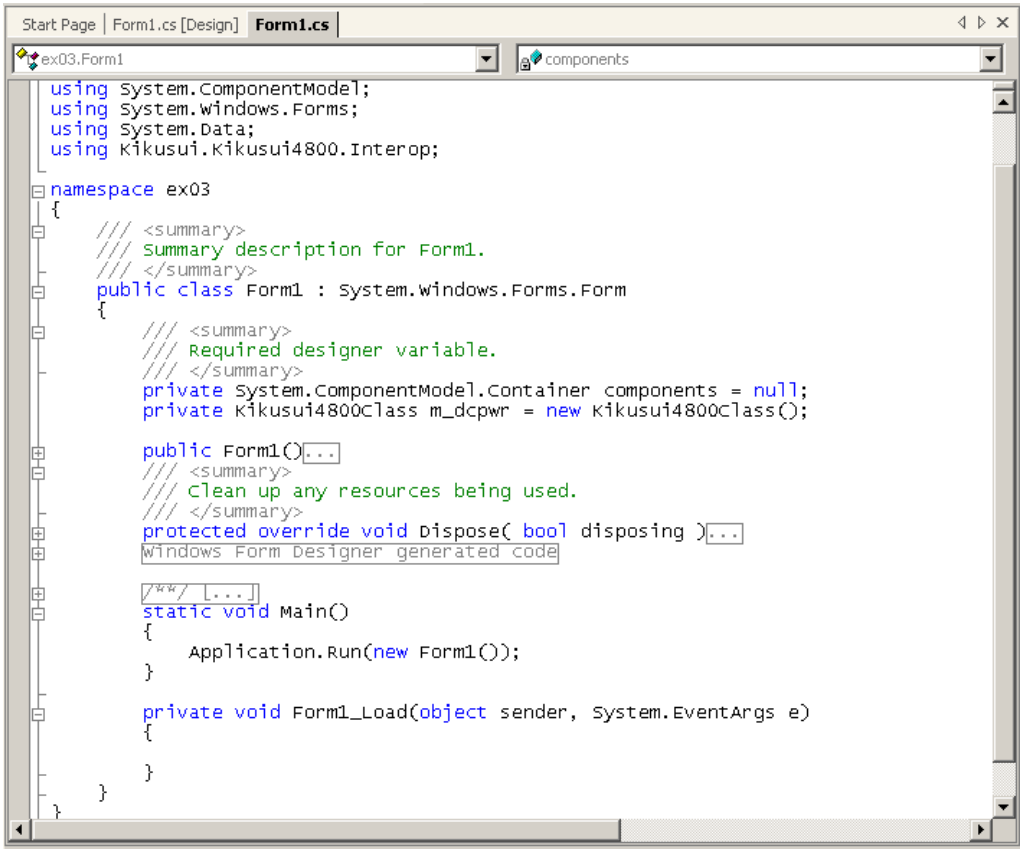

Figure 2-3 Form1_Load ハンドラ

オブジェクトを作成しただけでは計測器との I/O は行われていません。計測器との I/O を開始するには、`Initialize` メソッドを使用します。`Form1_Load` ハンドラ内に、`m_dcpwr.Ini` とタイプしてください。すると、C#.NET のインテリセンス機能により、`Kikusui4800Class` 型に関連するメソッドとプロパティのリストが現れます。ここでは Ini までタイプしているので、最も近い名前のメソッドとして `Initialize` メソッドがハイライトされます。

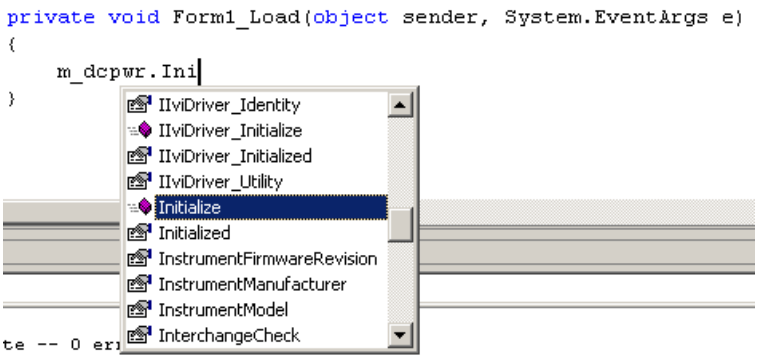

Figure 2-4 インテリセンス(メソッド・プロパティ・リスト)

そのまま Tab キーを押し、更に括弧"("をタイプすると、インテリセンスはこのメソッドのパラメータを表示します。

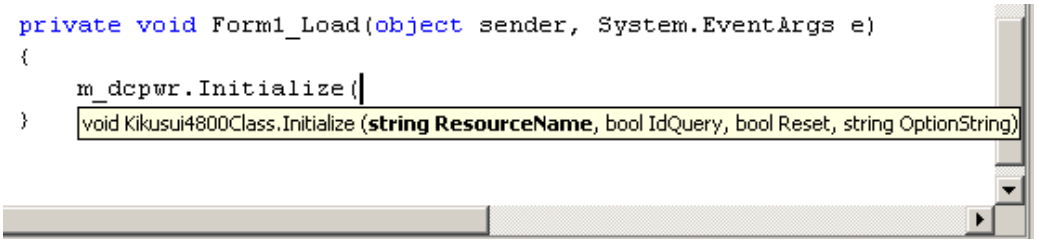

Figure 2-5 インテリセンス(パラメータ・リスト)

ここで、Initialize メソッドのパラメータについて説明しましょう。全ての IVI-COM 計測器ドライバは、IVI 仕様書で定義された Initialize メソッドを持っています。このメソッドには、以下のようなパラメータがあります。

Table 2-1 Initialize メソッドのパラメータ

パラメータ	タイプ	説明
ResourceName	string	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	bool	TRUE を指定した場合、計測器に対して ID クエリを行う。
Reset	boolean	TRUE を指定した場合、計測器の設定をリセットする。
OptionString	string	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。

ResourceName には VISA リソースを指定します。IdQuery に 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。Reset に 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 Boolean 型なので、TRUE、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 TRUE で、その他は全て 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 Initialize 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。

では具体的に、Initialize メソッドの呼び出しを記述してみましょう。OptionString パラメータはオプションなので、ここでは null を指定します。(Visual Basic 言語などでは OptionString は省略可能ですが、C#言語の場合は省略できません。)

```
uing Kikusui.Kikusui4800.Interop;

...(略)...

public class Form1 : System.Windows.Forms.Form
{
    /// <summary>
    /// Required designer variable.
    /// </summary>
    private System.ComponentModel.Container components = null;
    private Kikusui4800Class m_dcpwr = new Kikusui4800Class();

...(略)...

    private Form1_Load( object sender, System.EventArgs e)
    {
       m_dcpwr.Initialize("GPIB0::3::INSTR", true, true, null);
    }
}
```

2-4 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 メソッドを使います。ここでの例では、Initialize メソッドの呼び出しを Form1_Load ハンドラに記述したので、Close メソッドの呼び出しは Form1 クラスの Dispose メソッドをオーバーライドしてそこにに書くのが良いでしょう。Dispose メソッドをオーバーライドするには、Visual Studio.NET 統合環境の上部左側のコンボボックスで Form1 を選択し、右側のコンボボックスで Dispose(bool disposing)を選択します。

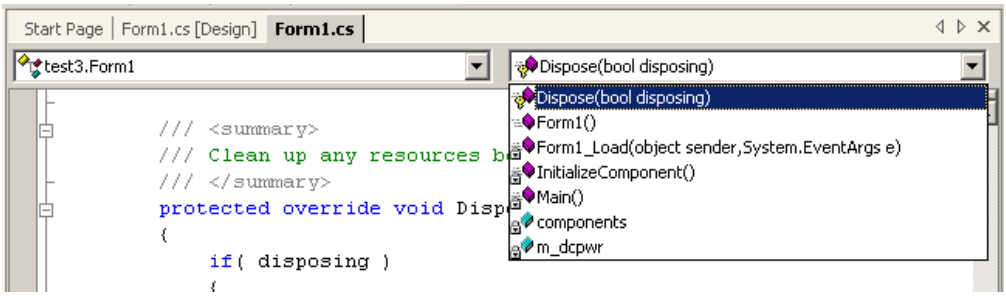

Figure 2-6 Dispose メソッドのオーバーライド

するとオーバーライドされた Dispose メソッドのコードが表示されます。そこでコードを書き足して下記のようにして下さい。

```
protected override void Dispose(bool disposing)
{
    if(disposing)
    {
        if (components != null)
        {
            components.Dispose();
        }
        m_dcpwr.Close();
        m_dcpwr = null;
    }
```

```
    base.Dispose( disposing );
}
```

2-5 実行

ここまでのコードだけで、とりあえず実行する事は可能です。この例では Form1_Load ハンドラ内で Initialize メソッドの呼び出しが行われるので、プログラムを実行すると、即座に計測器との通信が開始されます。実際に計測器が接続されていて Initialize メソッドが成功した場合は、フォーム画面が表示されます。通信に失敗した場合や、VISA ライブラリの設定が正しく行われていない場合などは、COM 例外(System.Runtime.InteropServices.COMException)を発生します。

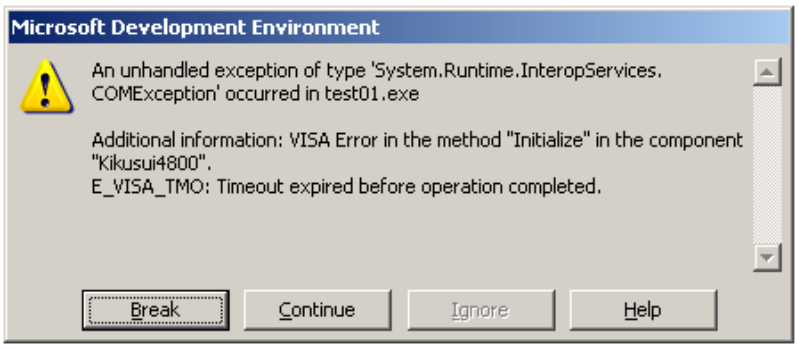

Figure 2-7 COM 例外

2-6 リピーテッド・キャパビリティ

Kikusui4800 IVI-COM ドライバの場合、IviDCPwr クラスで定義されているコンセプトと同様に、DC 電源装置の出力設定は Output インターフェースを通じて行われます。Kikusui4800 ドライバで提供されるスペシフィック・インターフェースの場合、IKikusui4800Output と IKikusui4800Outputs がそうです。IviDCPwr クラスに属する計測器ドライバは、複数の出力チャンネルを持つマルチ・トラック電源装置を前提に設計されています。

これらの COM インターフェースは、単数形と複数形の違いを除いて、同じ名前になっています。このように複数形の名前を持つインターフェースは、IVI 仕様書では一般にリピーテッド・キャパビリティと呼ばれます。リピーテッド・キャパビリティとは、機能が全く同じ又は類似している複数のオブジェクトを扱うために定義されたコンテナのようなもので、IKikusui4800Outputs のような複数形の名前を持つ COM インターフェースは通常 Count、Name、Item プロパティ(いずれもリード・オンリー)を持ちます。また、Item プロパティを通じて単品のオブジェクトを参照する事もできます。

まずは、下記の例を見てください。これは、Kikusui4800 IVI-COM ドライバがサポートする電源装置(実際には Kikusui PIA4800 シリーズ電源コントローラ)の"N5!C1"で識別される出力チャンネルを制御するものです。この例では、フォームにボタン(button1)を貼り付けた場合の、イベント・ハンドラとして記述しています。

```
private void button1_Click( object sender, System.EventArgs e)
{
    IKikusui4800Output output;
    output = m_dcpwr.Outputs.get_Item("N5!C1");
    output.VoltageLevel = 10.5;
    output.CurrentLimit = 1.2;
    output.Enabled = true;
}
```

一旦 IKikusui4800Output インターフェースを取得してしまえば、あとは難しい事はありません。VoltageLevel プロパティは電圧レベル設定を、CurrentLimit プロパティは電流リミット設定を、それぞれ行います。Enabled プロパティは出力の ON/OFF 設定を行います。

IKikusui4800Output インターフェースを取得する際の記述に注意してください。ここでは、IKikusui4800 インターフェースの Outputs プロパティを通じて IKikusui4800Outputs を取得し、直ぐに Item プロパティを使って IKikusui4800Output インターフェースを取得しています。

```
IKikusui4800Output output;
output = m_dcpwr.Outputs.get_Item("N5!C1");
```

このコードは次のように書くこともできます。

```
IKikusui4800Outputs outputs = m_dcpwr.Outputs;
IKikusui4800Output output = outputs.get_Item("N5!C1");
```

ここで、Item プロパティに渡しているパラメータに注意する必要があります。このパラメータは参照したい単品の Output オブジェクトの名前を指定しています。しかしここで使える名前(Output Name)は、ドライバごとにそれぞれ違います。例えば Kikusui4800 IVI-COM ドライバでは、"N1!C1"のような、NODE と CH を指定する表現になっていますが、他のドライバでは(たとえ、同じ IviDCPwr クラスであっても)違ったものになります。例えば、他の計測器ドライバでは、"Track1" のような表現かも知れません。特定の計測器ドライバで使用可能は名前は、通常はドライバのオンライン・ヘルプなどに記載されていますが、下記のようなテスト・コードを書くことでそれらを調べる事も可能です。

```
IKikusui4800Outputs outputs = m_dcpwr.Outputs;
int cnt = outputs.Count;
int ndx;
for( ndx=1; ndx<=cnt; ndx++)
{
    string strName;
    strName = outputs.Name(ndx);
    System.Diagnostics.Debug.WriteLine(strName);
}
```

Count プロパティは、リピーテッド・キャパビリティが持つ単品オブジェクトの個数を返します。Name プロパティは、与えられたインデックス番号の単品オブジェクトが持つ名前を返します。この名前こそが、Item プロパティに渡す事のできるパラメータになるのです。上記の例では、for 文を使って、インデックス 1 から Count までを反復処理しています。Name パラメータに渡すインデックス番号は 0 ベースではなく 1 ベースである事に注意してください。

3- 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-COM 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。

3-1 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-COM 計測器ドライバに依存した記述(例えば Kikusui4800 型で直接オブジェクトを生成)したり、"GPIB0::3::INSTR"のような特定 VISA 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-COM 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 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選択を間接的に行い、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・インターフェース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-COM 計測器ドライバと計測器ドライバのベンダーによって提供される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。弊社の場合はコンフィグレーション･ツールとして、**Kikusui IVI Config Utility** を提供しています。これを利用する事で、仮想インストルメントの設定を行うことが出来ます。

> Notes:
>
> Kikusui IVI Config Utility を使用して仮想インストルメントの設定を行う手順に関しては、「プログラミング・ガイド(IVI Config Utility 編)」を参照してください。

このガイドブックでは、MySupply というロジカル・ネームで仮想インストルメントが既に作成されていて、Kikusui4800 ドライバを使用し、VISA リソース"GPIB0::3::INSTR"を使用する、という設定が行われているものとします。

3-2 タイプ･ライブラリのインポート

新規プロジェクトを作成したあと最初にすべき事は、利用したい IVI-COM 計測器ドライバのタイプ・ライブラリからインタロップ・アセンブリを生成し、それを参照する事です。**Project | Add References** メニューを選択して **Add References** ダイアログを表示し、**COM** タブを選択します。

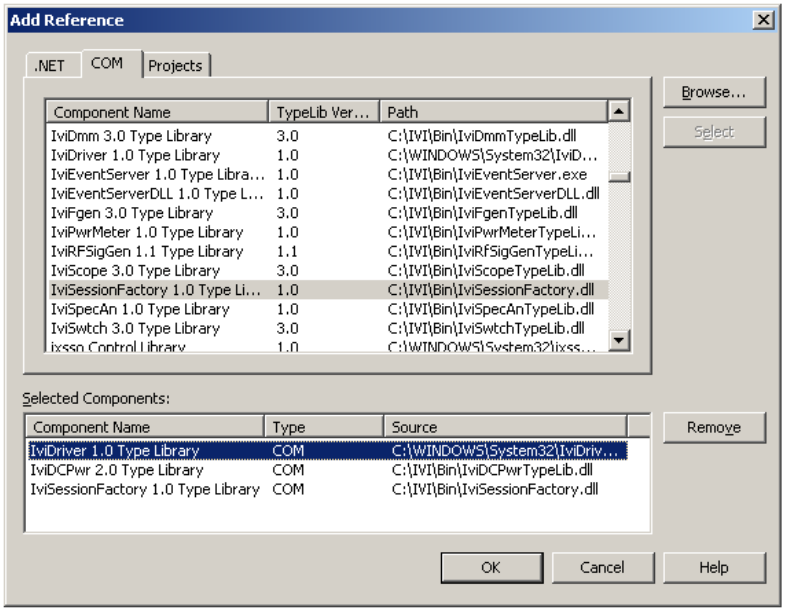

Figure 3-1 Add Reference ダイアログ

ここでは IviDCPwr クラス・インターフェースを使用するので、**IviDCPwr 2.0 TypeLibrary** を選択します。また、**IviDriver 1.0 Type Library** と **IviSessionFactory 1.0 TypeLibrary** の 2 つは、使用する計測器クラス・インターフェースに関係なく必ず選択してください。**Select** ボタンで複数選択し、**OK** ボタンで決定します。

これでタイプ・ライブラリのインポート作業は完了したので、あなたのアプリケーションからは任意の計測器ドライバを IviDCPwr クラス・インターフェースを通じて使うことができます。

3-3 オブジェクト・ブラウザ

アセンブリへの参照が追加されると、Visual Studio.NET 統合環境からオブジェクト・ブラウザを通じて利用可能な構文を確認する事ができます。オブジェクト・ブラウザを起動するには、**View | Object Browser** メニューを選択します。

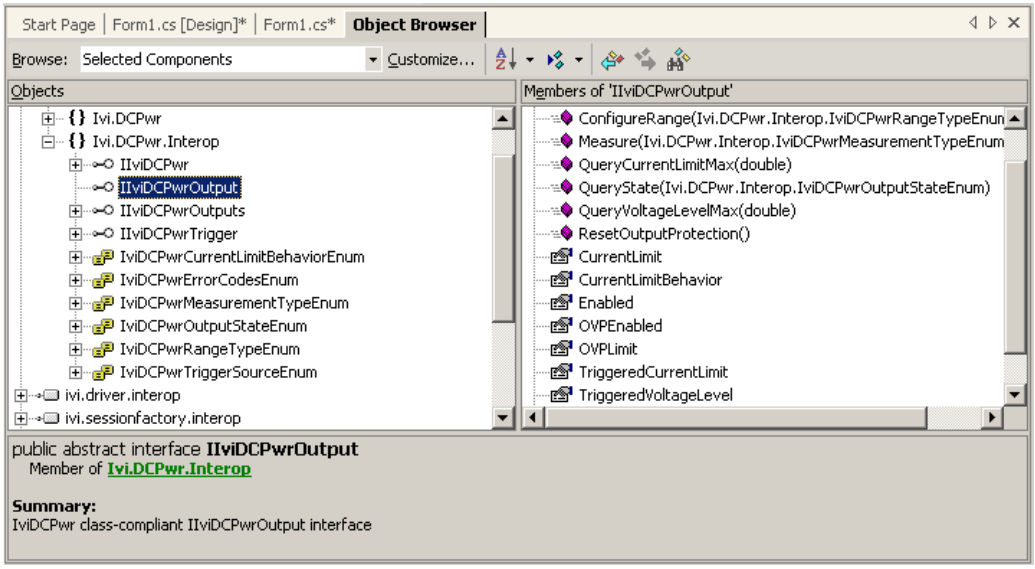

Figure 3-2 オブジェクト・ブラウザ

3-4 オブジェクトの作成とセッションのイニシャライズ

まず、フォーム設計画面の上をマウスでダブルクリックします。すると、Form1_Load イベント・ハンドラの骨格だけを持つソース・コードが表示されます。まずモジュールの先頭で下記の using 擬似命令を追加します。これは、以降のネーム・スペース参照を省略するものです。

```
using Ivi.SessionFactory.Interop;
using Ivi.Driver.Interop;
using Ivi.DCPwr.Interop;
```

更にフォームのデータ・メンバーとして変数 m_dcpwr を IIviDCPwr 型で宣言してください。IIviDCPwr のように大文字の"I"で始まるタイプは一般に COM インターフェース型なので、New 演算子でオブジェクトを作成する事はできません。

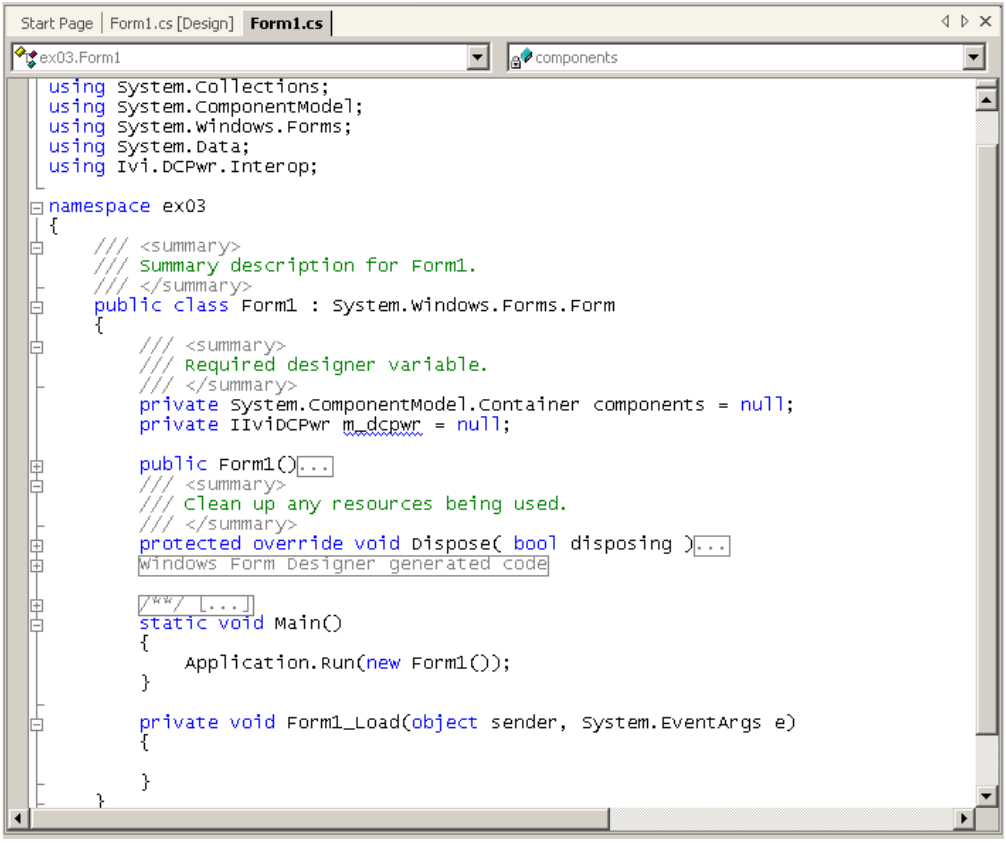

Figure 3-3 Form1_Load ハンドラ

次に、計測器ドライバのオブジェクトを作成しなければなりません。計測器ドライバ・オブジェクトの作成には、IVI Session Factory を使用します。IVI Session Factory は IVI Shared Components によって提供される COM DLL サーバーで、指定されたロジカル・ネームから該当する仮想インストルメントのコンフィグレーション情報を抽出し、適切な計測器ドライバ・ソフトウェアをロードしてオブジェクトを作成するものです。

```
private Form1_Load( object sender, System.EventArgs e)
{
    IviSessionFactoryClass sf = new IviSessionFactoryClass();
    m_dcpwr = (IIviDCPwr)sf.CreateDriver("MySupply");

}
```

ここでは、IviSessionFactory オブジェクトを作成し、CreateDriver メソッドで計測器オブジェクトを作成しています。パラメータに指定されている"MySupply"は、有効なロジカル・ネームとしてコンフィグレーションが行われている必要があります。また、この時にロードされる計測器ドライバ DLL は、仮想インストルメント MySupply で指定されたものが自動的に割り当てられます。

CreateDriver メソッドの戻り値を変数 m_dcpwr に割り当てる際、明示的なタイプキャストが必要である事に注意してください。CreateDriver メソッドが返す COM インターフェースは厳密には IUnknown 型と宣言されており、これに対応するインタロップ･アセンブリでの宣言は object 型になります。この場合、QueryInterface を呼び出して IIviDCPwr インターフェースを取得する必要があります。C#言語の場合は、明示的なタイプキャストを行うことで QueryInterface を水面下で呼び出すことができます。

ドライバ・オブジェクトが作成できたら、Initialize メソッドを呼び出します。Initialize メソッドのパラメータは、スペシフィック・インターフェースを使用した場合と全く同じですが、

ResourceName パラメータには、VISA リソースを直接指定するのではなく、ロジカル・ネームを指定します。

ここで、Initialize メソッドのパラメータについて再度説明しましょう。全ての IVI-COM 計測器ドライバは、IVI 仕様書で定義された Initialize メソッドを持っています。このメソッドには、以下のようなパラメータがあります。

Table 3-1 Initialize メソッドのパラメータ

パラメータ	タイプ	説明
ResourceName	string	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。 ロジカル・ネームを指定した場合、そのロジカル･ネームの Hardware Asset コンフィグレーションに従った VISA リソースが間接指定される。
IdQuery	bool	TRUE を指定した場合、計測器に対して ID クエリを行う。
Reset	bool	TRUE を指定した場合、計測器の設定をリセットする。
OptionString	string	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。

インターチェンジャブル・アプリケーションでは通常、ResourceName には VISA リソースではなくロジカル･ネームを指定します。実際には VISA リソースを直接指定する事もできますが、仮想インストルメントの抽象性が少し損なわれます。

IdQuery に 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。Reset に 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 Boolean 型なので、TRUE、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に適用されるデフォルト値は、ResourceName にロジカル・ネームを指定した場合と VISA リソースを直接指定した場合とで異なります。ResourceName にロジカル・ネームを指定した場合のデフォルトは、そのロジカル・ネームで示される仮想インストルメントの Default Operation で指定されたものになります。一方 VISA リソースを直接指定した場合は、ロジカル・ネームをバイパスしてしまうので、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 TRUE で、その他は全て 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 Initialize 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。

では具体的に、Initialize メソッドの呼び出しを記述してみましょう。OptionString パラメータはオプションなので、ここでは null を指定します。(Visual Basic 言語などでは OptionString は省略可能ですが、C#言語の場合は省略できません。)

```
m_dcpwr.Initialize( "MySupply", true, true, null);
```

3-5 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 メソッドを使います。ここでの例では、Initialize メソッドの呼び出しを Form1_Load ハンドラに記述したので、Close メソッドの呼び出しは Form1 クラスの Dispose メソッドをオーバーライドしてそこにに書くのが良いでしょう。Dispose メソッドをオーバーライドするには、Visual Studio.NET 統合環境の上部左側のコンボボックスで Form1 を選択し、右側のコンボボックスで Dispose(bool disposing)を選択します。

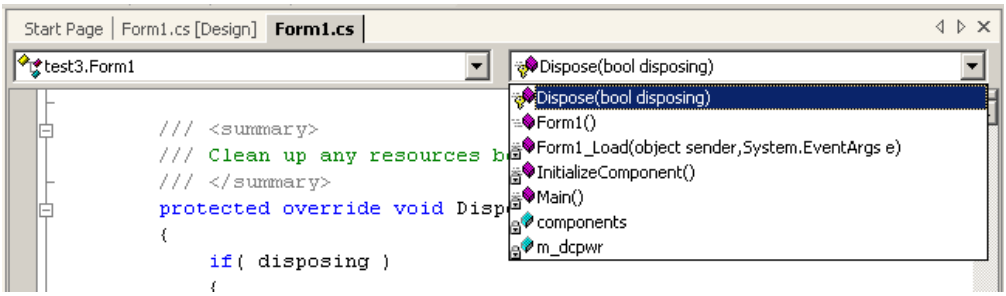

Figure 3-4 Dispose メソッドのオーバーライド

するとオーバーライドされた Dispose メソッドのコードが表示されます。そこでコードを書き足して下記のようにして下さい。

```
protected override void Dispose(bool disposing)
{
    if(disposing)
    {
        if (components != null)
        {
            components.Dispose();
        }
        m_dcpwr.Close();
        m_dcpwr = null;
    }
    base.Dispose( disposing );
}
```

3-6 実行

ここまでのコードだけで、とりあえず実行する事は可能です。この例では Form1_Load ハンドラ内で Initialize メソッドの呼び出しが行われるので、プログラムを実行すると、即座に計測器との通信が開始されます。実際に計測器が接続されていて Initialize メソッドが成功した場合は、

フォーム画面が表示されます。通信に失敗した場合や、VISA ライブラリの設定が正しく行われていない場合などは、COM 例外(System.Runtime.InteropServices.COMException)を発生します。

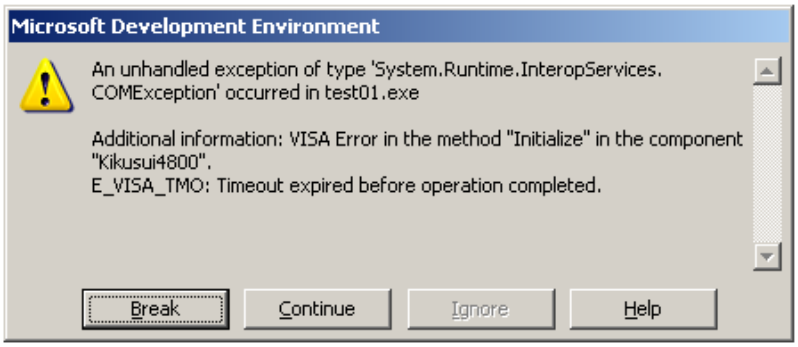

Figure 3-5 COM 例外

3-7 リピーテッド・キャパビリティ

IviDCPwr クラス・インターフェースでは、DC 電源装置の出力設定は Output インターフェースを通じて行われます。ここで使用するインターフェースは、IIviDCPwrOutput と IIviDCPwrOutputs です。IviDCPwr クラスに属する計測器ドライバは、複数の出力チャンネルを持つマルチ・トラック電源装置を前提に設計されています。

これらの COM インターフェースは、単数形と複数形の違いを除いて、同じ名前になっています。このように複数形の名前を持つインターフェースは、IVI 仕様書では一般にリピーテッド・キャパビリティと呼ばれます。リピーテッド・キャパビリティとは、機能が全く同じ又は類似している複数のオブジェクトを扱うために定義されたコンテナのようなもので、IIviDCPwrOutputs のような複数形の名前を持つ COM インターフェースは通常 Count、Name、Item プロパティ(いずれもリード・オンリー)を持ちます。また、Item プロパティを通じて単品のオブジェクトを参照する事もできます。

まずは、下記の例を見てください。これは、仮想インストルメントのバーチャル･ネームに登録されている"Track_A"で識別される出力チャンネルを制御するものです。この例では、フォームにボタン(Command1)を貼り付けた場合の、イベント・ハンドラとして記述しています。

```
private void button1_Click( object sender, System.EventArgs e)
{
    IKikusui4800Output output;
    output = m_dcpwr.Outputs.get_Item("Track_A");
    output.VoltageLevel = 10.5;
    output.CurrentLimit = 1.2;
    output.Enabled = true;
}
```

一旦 IIviDCPwrOutput インターフェースを取得してしまえば、あとは難しい事はありません。VoltageLevel プロパティは電圧レベル設定を、CurrentLimit プロパティは電流リミット設定を、それぞれ行います。Enabled プロパティは出力の ON/OFF 設定を行います。

IIviDCPwrOutput インターフェースを取得する際の記述に注意してください。ここでは、IIviDCPwr インターフェースの Outputs プロパティを通じて IIviDCPwrOutputs を取得し、直ぐに Item プロパティを使って IIviDCPwrOutput インターフェースを取得しています。

```
    IIviDCPwrOutput output;
    output = m_dcpwr.Outputs.get_Item("Track_A");
```

このコードは次のように書くこともできます。

```
    IIviDCPwrOutputs outputs = m_dcpwr.Outputs;
    IIviDCPwrOutput output = outputs.get_Item("Track_A");
```

ここで、Item プロパティに渡しているパラメータに注意する必要があります。このパラメータは参照したい単品の Output オブジェクトの名前を指定しています。スペシフィック・インターフェースを使用した例ではドライバごとにそれぞれ異なる名前(フィジカル・ネーム)を直接していしましたが、ここでは違います。ここでは特定の計測器ドライバに依存したフィジカル・ネームは使えないので、バーチャル・ネームを指定します。

3-8 計測器の交換

これまでの例では、仮想インストルメントのコンフィグレーションとして、Kikusui4800 計測器ドライバを設定しましたが、ここで計測器を例えば AgilentE36xx ドライバでホストされるものに交換するとどうなるでしょう。その場合には、アプリケーションを再度コンパイル・リンクする必要はありませんが、使用する仮想インストルメントのコンフィグレーションを変更する必要があります。

変更しなければならないコンフィグレーションは、基本的には、Driver Session タブにある Software Module の選択と、Virtual Names タブでのバーチャル・ネームのマップ変更(マップ先のフィジカル・ネームが変わるため)です。計測器が変わると、必ずしも同じ I/O インターフェースを使えるとは限らないので(GPIB オンリーの計測器から RS232 オンリーの計測器へのチェンジなど)、必要であれば、Hardware Asset タブにある IO Resource Descriptor も設定変更してください。

> Notes:
>
> 仮想インストルメントのコンフィグレーション方法については、「計測器ドライバ・プログラミング・ガイド(IVI Config Utility 編)」を参照してください。

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、範囲外の値をプロパティに設定したり、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-COM 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 COM 例外としてクライアント・プログラムに伝えられます。C#.NET の場合、COM 例外は try、catch、finally ブロックを使って処理する事が出来ます。

先ほど示した、電圧・電流を設定するコードを下記のように変更してみましょう。

```
private void button1_Click( object sender, System.EventArgs e)
{
    try
    {
        IKikusui4800Output output;
        output = m_dcpwr.Outputs.Item("N5!C1");
        output.VoltageLevel = 10.5;
        output.CurrentLimit = 1.2;
        output.Enabled = true;
    }
    catch(System.Runtime.InteropServices.COMException ex)
    {
        MessageBox.Show(
              ex.Message, "error 0x" + Convert.ToString(ex.ErrorCode, 16));
    }
    catch(Exception ex)
    {
        MessageBox.Show(ex.Message, "error");
    }

}
```

ここでは try 文を使ってエラー処理を行っています。例えば、Item プロパティに渡した名前が間違っている場合、VoltageLevel に設定する値が適正範囲から外れている場合、或いは計測器との通信に失敗した場合などはいずれも、計測器ドライバ内で COM 例外が発生します。上記の例では、例外が発生した場合に簡単なメッセージ・ボックスを表示しています。

エラー(COM 例外)の詳細は、catch ブロックのパラメータから取得できます。この例では、メッセージ・ボックスのキャプションを ErrorCode プロパティから取得したエラー・コード(16 進表記)に、Message プロパティから取得した文字列を本文に、それぞれ設定しています。

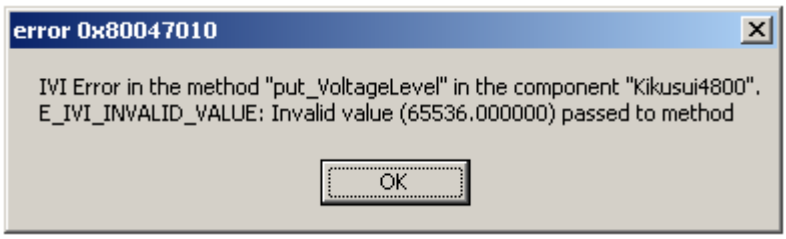

Figure 4-1 エラー処理によるメッセージ・ボックス