VEE Proアドバンスド・テクニック

マニュアル番号 E2110-90430

原　典

本書は "VEE Pro Advanced Techniques" (Part No. E2110-90430) (Printed in USA April 2000) を翻訳したものです。

詳細は上記の最新マニュアルを参照して下さい。

ご　注　意

- 本書に記載した内容は、予告なしに変更することがあります。
- 当社は、お客様の誤った操作に起因する損害については、責任を負いかねますのでご了承ください。
- 当社では、本書に関して特殊目的に対する適合性、市場性などについては、一切の保証をいたしかねます。
- また、備品、パフォーマンス等に関連した損傷についても保証いたしかねます。
- 当社提供外のソフトウェアの使用や信頼性についての責任は負いかねます。
- 本書の内容の一部または全部を、無断でコピーしたり、他のプログラム言語に翻訳することは法律で禁止されています。
- 本製品パッケージとして提供した本マニュアル、フレキシブル・ディスクまたはテープ・カートリッジは本製品用だけにお使いください。プログラムをコピーをする場合はバックアップ用だけにしてください。プログラムをそのままの形で、あるいは変更を加えて第三者に販売することは固く禁じられています。

納入後の保証について

- ハードウェア製品に対しては部品及び製造上の不具合について保証します。又、当社製品仕様に適合していることを保証します。
  ソフトウェアに対しては、媒体の不具合（ソフトウェアを当社指定のデバイス上適切にインストールし使用しているにもかかわらず、プログラミング・インストラクションを実行しない原因がソフトウェアを記録している媒体に因る場合）について保証します。又、当社が財産権を有するソフトウェア（特注品を除く）が当社製品仕様に適合していることを保証します。
  保証期間中にこれらの不具合、当社製品仕様への不適合がある旨連絡を受けた場合は、当社の判断で修理又は交換を行います。
- 保証による修理は、当社営業日の午前8時45分から午後5時30分の時間帯でお受けします。なお、保証期間中でも当社所定の出張修理地域外での出張修理は、技術者派遣費が有償となります。
- 当社の保証は、製品の動作が中断されないことや、エラーが皆無であることを保証するものではありません。保証期間中、当社が不具合を認めた製品を相当期間内に修理又は交換できない場合お客様は当該製品を返却して購入金額の返金を請求できます。
- 保証期間は、製品毎に定められています。保証は、当社が据付調整を行う製品については、据付調整完了日より開始します。但し、お客様の都合で据付調整を納入後31日以降に行う場合は31日目より保証が開始します。
  又、当社が据付調整を行わない製品については、納入日より保証が開始します。
- 当社の保証は、以下に起因する不具合に対しては適用されません。

  (1)不適当又は不完全な保守、校正によるとき
  (2)当社以外のソフトウェア、インターフェース、サプライ品によるとき
  (3)当社が認めていない改造によるとき
  (4)当社製品仕様に定めていない方法での使用、作動によるとき
  (5)お客様による輸送中の過失、事故、滅失、損傷等によるとき
  (6)お客様の据付場所の不備や不適正な保全によるとき
  (7)当社が認めていない保守又は修理によるとき
  (8)火災、風水害、地震、落雷等の天災によるとき
- 当社はここに定める以外の保証は行いません。又、製品の特定用途での市場商品価値や適合性に関する保証は致しかねます。
- 製品の保守修理用部品供給期間は、製品の廃止後最低５年です。

米国政府の権利の制限

本ソフトウェアおよびドキュメントは、すべて私費によって開発されています。これらの配布とライセンス承認条件は、DFAR 252.227-7014(1995年6月)に定義された"Commercial computer software"、FAR 2.101(a)に定義された"commercial item"、FAR 52.227-19(1987年6月)で定義された"Restricted computer software"(またはこれらに相当する政府機関の規則または契約条項)のうち当てはまるものに準拠します。ソフトウェアの使用、複製、公開はAgilentの標準商用ライセンス条項によって制限され、米国政府の国防省以外の省庁の権利はFAR 52.227-19(c)(1-2)(1987年6月)による制限を超えることはありません。すべての技術データに関する米国政府ユーザの権利は、FAR 52.227-14(1987年6月)またはDFAR 252.227-7015(b)(2)(1995年11月)のうち当てはまるものの制限を超えることはありません。

商標について

Microsoft®、MS-DOS®、Windows®、MS Windows®、Windows NT® は、Microsoft Corporationの米国における登録商標です。

MATLAB®はMathWorks, Inc.の登録商標です。

NetscapeはNetscape Communications Corporationの米国における登録商標です。

UNIX®はOpen Groupの登録商標です。

改版履歴

第1版- 2000年3月　　ソフトウェア・バージョン6.0を反映

本書で使用する表記規約

本書では、下記の表記規約を使用します。

Getting Started	イタリック体のテキストは、書名の表記、および強調に使用します。
Dialog Box	太字体のテキストは、用語集で定義されている語が最初に現れた場所で使用します。
`File`	コンピュータ・フォントは、メニュー名、機能、ボタン、ユーザが入力するテキストなど、画面に表示されるテキストを表します。
`dir` *filename*	この場合、コンピュータ・フォントのテキストは記載されたとおりに入力し、イタリック体の部分には実際の値を代わりに指定します。
`File` ⇒ `Open`	"⇒"は、Agilent VEEの機能のメニュー内での位置を示すための記号です。例えば、"`File` ⇒ `Open`"は`File`メニューを選択してから`Open`を選択することを示します。
`Sml`|`Med`|`Lrg`	コンピュータ・フォントの選択肢が縦線(|)で区切られている場合、オプションの1つを選択することを示します。
Press **Enter**	この場合、太字はキーボード上のキーを示します。
Press **Ctrl + O**	キーボード上のキーの組合わせを示します。これら2つのキーを同時に押してください。

目次

1. はじめに

2. 機器制御の基本

3. 機器の構成

4. トランザクションI/Oの使用

5. I/Oに関する高度なトピックス

8. データ伝搬

9. 演算機能

10. 変数

11. レコードとデータセットの使用

12. ユーザ定義関数/ライブラリ

13. ActiveXオートメーション・オブジェクトおよびコントロールの使用

14. シーケンサ・オブジェクトの使用

A. I/O トランザクション・リファレンス

図

表

1

はじめに

はじめに

この章では、本書とVEEに関する概要を説明します。下記の内容があります。

- 本書について
- VEEの構成
- VEEサンプル・プログラムの使用
- ライブラリ・オブジェクトの使用
- サポートされるI/Oインタフェース
- VEE実行モードの使用
- 参考文献

本書について

本書には、VEEの高度な機能に関する詳細な情報が記載されています。表1-1に本書の内容を簡単に記します。

表1-1. 本マニュアルの内容説明

章	説明
1 - はじめに	VEEサンプル・プログラムおよびライブラリ・オブジェクトの使い方を説明します。
2 - 機器制御の基本	機器と通信する5通りの方法について説明します。
3 - 機器の構成	機器との通信のためにVEEを構成する4通りの方法について説明します。
4 - トランザクション I/Oの使用	トランザクションを使用するすべてのVEE I/Oオブジェクトについて説明します。
5 - I/Oに関する高度なトピックス	I/Oの構成とアドレッシングについて説明します。
6 - パネル・ドライバおよびコンポーネント・ドライバ・オブジェクトの使用	パネル・ドライバおよびコンポーネント・ドライバ・オブジェクトをVEEで使用する方法を説明します。
7 - VXIplug&playドライバの使用	VXI*plug&play*ドライバを使用して機器と通信する方法を説明します。
8 - データ伝搬	オブジェクト間のデータ伝搬を使ったプログラムの作成方法を説明します。
9 - 演算機能	スカラと配列に対する演算機能について説明します。
10 - 変数	VEEの変数について説明します。
11 - レコードとデータセットの使用	レコード・データ型とデータセットについて説明します。
12 - ユーザ定義関数およびライブラリ	組込み関数の19のカテゴリとUserFunctionについて説明します。

表1-1. 本マニュアルの内容説明

13 - ActiveXオートメーション・オブジェクトおよびコントロールの使用	ActiveXオートメーションおよびコントロールをVEEで使用する方法を説明します。
14 - シーケンサ・オブジェクトの使用	シーケンサ・オブジェクトの使用に関する指針を示します。

VEEの構成

このセクションでは、環境に合わせてVEEの構成とカスタマイズを行うために、VEEオプション、X11オプション(UNIX®環境の場合)またはWindowsオプション(MS Windows®環境の場合)を変更するための指針を記します。

Windows版VEEの構成

Windows版VEEは、Windowsのレジストリを使ってVEEの環境情報を記憶します。VEEウィンドウのプロパティの多くは、VEE `Default Preferences`ダイアログ・ボックス(`File⇒Default Preferences`)で変更できます。これらのプロパティは、下記のディレクトリにあるデフォルト・ファイルVEE.RCに保存されます。

```
%userprofile%\LocalSettings\Application Data\Agilent\VEE Pro
```

または、`%HOME%`が定義されていればそこにあります。

カラーとフォントの設定

VEE 6.0からは、`Default Preferences`ダイアログ・ボックスに`Save colors & fonts with program`の選択肢がなくなりました。カラーとフォントをプログラムとともに保存するには、File ⇒ Save Asを使います。図1-1に新しい`Save File`ダイアログ・ボックスを示します。

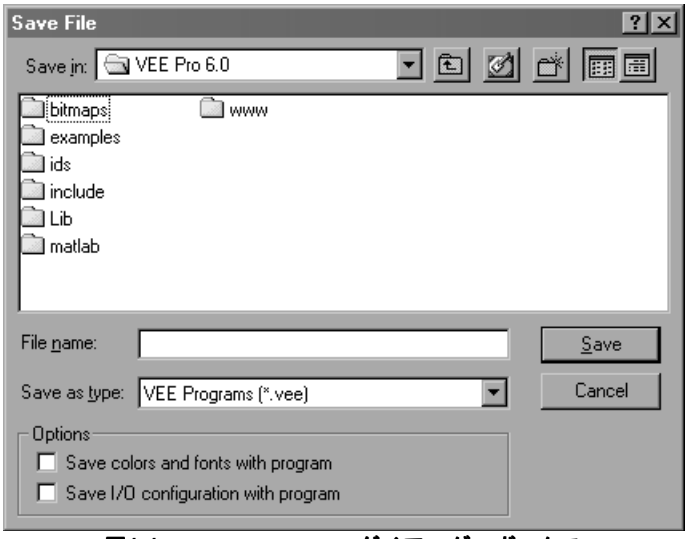


図1-1. File ⇒ Save As**ダイアログ・ボックス**

カラーとフォントに関しては、**変更**した設定だけがVEE.RCデフォルト・ファイルに保存されます。VEEでカラーとフォントを変更する方法の詳細については、VEE HelpでHow Do Iを参照してください。

アイコン・ビットマップのカスタマイズ

アイコン化されたオブジェクトに表示されるアイコンを、他のビットマップやpixmapに変更することができます。VEEに付属する多数のファイルが使用でき、自分でファイルを作成することもできます。Windows版VEEでは、24ビットの.BMPビットマップ・ファイル、.GIF87、.GIF89a、.PNG(Portable Network Graphics)、.WMF(Windows Meta Files)、.ICNアイコン・ファイルの各フォーマットがサポートされます。オブジェクトのアイコンを選択するには、オブジェクト・メニューのProperties機能を選択し、Propertiesダイアログ・ボックスのIconタブを使用します。

オブジェクト・アイコン用の独自のビットマップを作成するには、VEEがサポートするグラフィックス・フォーマットを出力する任意のエディタ(MS WindowsのPaintなど)を使用します。アイコンのサイズは48×48が適切です。あまり大きいアイコンを使うとVEEプログラム・エリアのスペースを消費し、あまり小さいアイコンを使うと識別しにくくなります。Print Screenなどの画面キャプチャ・ユーティリティとPaintを組み合わせて使うこともできます。

パネル・ビュー用のビットマップの選択

パネル・ビューのバックグランド・アイコンとして用いられるビットマップを選択できます。これは、UserObjectと、パネル・ビューに表示されたVEEプログラムに適

用されます。パネル・ビュー・アイコンのフォーマットはVEEがサポートするものである必要があります。前の項と同様に、独自に作成したアイコンも使用できます。

パネル・ビューのアイコンとしてビットマップを選択するには、**パネル・ビュー**をオンにして、Panelボタンと Detailボタンがタイトル・バーに表示されるようにします(このためにはオブジェクトをパネルに追加します)。オブジェクト・メニューをクリックし、Propertiesをクリックします。Propertiesダイアログ・ボックスのPanelタブを使ってビットマップを選択します。

UNIX版VEEの構成

カラーとフォントの設定

VEE 6.0からは、Default Preferencesダイアログ・ボックスにSave colors & fonts with programの選択肢がなくなりました。カラーとフォントをプログラムとともに保存するには、File ⇒ Save Asを使います。図1-1に新しいSave Fileダイアログ・ボックスを示します。

VEEで変更したカラーとフォントの設定は、$HOMEディレクトリにあるデフォルト・ファイル.veercに保存されます。カラーとフォントに関しては、変更した設定だけがデフォルト・ファイルに保存されます。VEEでカラーとフォントを変更する方法の詳細については、*VEE Help*でHow Do Iを参照してください。

UNIXプラットフォームでは、.bmp(ビットマップ)、.gif、.icn(アイコン)、.xwd(X11ビットマップ)の各ファイルがVEEでサポートされます。オブジェクト・アイコンのビットマップを独自に作成するには、VEEがサポートするグラフィックス・フォーマットを出力する任意のエディタ(HP-UXのIconEditorプログラムなど)を使用します。UNIX上のX11ウィンドウ・ダンプ(xwd)などの画面キャプチャ・ユーティリティも使用できます。

X11属性の変更(UNIX)

VEEのいくつかの属性をカスタマイズするため、Veeという名前の*app-defaults*ファイルが用意されています。HP-UX 10.20の場合、このファイルはopt/veetest/config/にあります。同じディレクトリにある*app-defaults*ファイルHelpviewは、ヘルプ・ウィンドウのカスタマイズに使います。

これらのファイルを使用するには、X11リソース・データベースにインストールする必要があります。xrdbを使用している場合、これらのファイルをインストールするには、VEEを起動する前に各ファイルに対してxrdb -merge *filename*と入力します。xrdbを使用しない場合は、ファイルをX11リソース・ファイルにマージします。X11リソース・ファイルは通常は$HOMEディレクトリにある.Xdefaultsですが、環境変数$XENVIRONMENTがあればそれで指定されるファイルです。

他のX11リソースを変更するには、X11リソース・ファイルに対して変更または追加を行います。例えば、VEEウィンドウのデフォルト・ジオメトリを変更して、常に

画面の右下隅に640×480ピクセルのサイズで起動するようにするには、下記の行をX11リソース・ファイル(通常は.Xdefaults)に追加します。

```
Vee*geometry: =640x480-0-0.
```

X11環境のカスタマイズ方法の詳細については、*Beginner's Guide to the X Window System*を参照してください。

画面のカラーの変化(UNIX)

ワークステーションには、一定の数のカラー・プレーン(通常は1、4、6、8)が備わっています。X11はこれらのカラー・プレーンの情報を使ってアプリケーションのウィンドウにカラーを表示します。

複数のアプリケーションが(それぞれ独立したウィンドウで)動作していて、ウィンドウを切り換えたときに画面のカラーが変化する場合、理由は2通り考えられます。1つは、すべてのアプリケーションが使っているカラー数の合計がディスプレイのカラー数を超えている場合、もう1つはどれかのアプリケーションが独自のプライベート・カラー・マップを使用している場合(Rocky Mountain Basic[1] など)です。

VEEでは最低39個のカラーが用いられます(カラーの指定と、実行中に実際に用いられるカラーによって異なります)。したがって、VEEを起動しているときにはこのような現象が見られることがあります。

症状としては、VEEウィンドウを選択している場合、VEEのカラーは正しく表示されますが、他のアプリケーションのウィンドウではカラーが正しく表示されないことがあります。他のアプリケーションに移ると、そのアプリケーションのカラーは正しくなりますが、VEEのカラーは正しくなくなります。**これはX11で普通に見られる動作です。この現象が発生してもVEEには特に問題は起きません。**

この動作は、VEEにも他のアプリケーションにも全く影響を与えません。ただし、いくつかの方法でこれを修正することは可能です。

使用するカラー数が多すぎる(UNIX)

ワークステーションで同時に表示できるカラー数は、ディスプレイのカラー・プレーン数に基づいて決まっています。この数は次のように求められます。

$$2^{\text{カラー・プレーン数}}$$

1. Rocky Montain Basicは、HP BASIC/UXとして知られています。

例えば、カラー・プレーン数が4の場合、16個のカラーを同時に表示できます。

$$2^4 = 16$$

この数を超えると、ウィンドウを切り換えたときに画面がちらつくことがあります。

表示可能なカラー数を超えた場合、ちらつきをなくすための第1歩は、ワークステーションが表示できる限界以内にカラー数を抑えることです。カラーを減らすためには下記のような方法が考えられます。

- 余分のカラーを削除します。2 つのアプリケーションで同一のカラー配列が使用できる場合、そのようにカスタマイズします。
- アプリケーションが使用するカラー配列のカラー数を減らします。File⇒Default Preferencesをクリックし、Default Preferencesダイアログ・ボックスでデフォルト・カラーを少数のカラーに変更します。
- 不要なアプリケーションを終了します。アプリケーションが独自のカラー配列を使用していると、要求されるカラーの数がカラー・マップの制限を簡単に超えてしまいます。他のアプリケーションを終了した場合、変更を確認するにはVEEを再起動しなければならない場合があります。
- xinitcolormap コマンドで割り当てられるカラーの数を減らします。これらのカラーは常にカラー・マップ上に置かれるため、テンポラリ・カラーのための領域を狭めています。

一部のX11ウィンドウ・マネージャには、カラーマップ・フォーカス・ディレクティブ(*colormapFocusPolicyなど)が用意されています。この値は画面上のカラーの使用方法を変更します。同時に表示できるカラーの総数を超えた場合、この値がexplicitに設定されていると、アプリケーションのウィンドウに正しいカラーが表示されない場合があります。

ローカル・カラー・マップを使用するアプリケーション (UNIX)

一部のアプリケーションは、ローカル・カラー・マップを使用します。この種のアプリケーションを実行すると、アプリケーションは現在のカラー・マップを保存し、独自のローカル・カラー・マップに切り換えます。この場合、ウィンドウの間にちらつきが生じることがあります。

これを避ける1つの方法は、xinitcolormapコマンドでVEEのカラーをあらかじめ割り当てておくことです。このためには、あらかじめ割り当てるカラーのリストを

ASCIIファイルとして作成します。このファイルについては、xinitcolormapのmanページに説明があります。

このファイルには空行が含まれていてはならず、先頭はBlackとWhiteでなければなりません。カラー・フォーマットは、既定義のワードで指定するか、#のあとに16進のRGB値を記述します。図1-2と図1-3に、黒、白、ライト・グレーを指定した例を示します。

```
Black
White
LightGray
```

図1-2. ワードを使ったカラー・マップ・ファイル

```
#000000
#ffffff
#a8a8a8
```

図1-3. 16進値を使ったカラー・マップ・ファイル

Rocky Mountain Basicはローカル・カラー・マップを使用するアプリケーションの1つであり、スタートアップ時にxinitcolormapコマンドを使ってRocky Mountain Basicのカラーをあらかじめ割り当てておくことが推奨されています。詳細については下記のファイルを参照してください。

```
/opt/rmb/newconfig/rgb.README
```

VEEのカラーをあらかじめ割り当てる手順:

1. 使用するVEEカラーをすべて記載したカラーマップ・ファイルを作成します。

2. $HOMEディレクトリに移ります。

   ```
   cd $HOME
   ```

3. Rocky Mountain BasicとVEEのカラーマップ・ファイルを結合します。

   ```
   cat /opt/rmb/newconfig/xrmbcolormap vee-colormapfile >
   .xveecolormap
   ```

Rocky Mountain Basicは、カラーマップの最初の16個のエントリが自分のカラーであると見なすので、Rocky Mountain Basicのカラーを最初に指定する必要があります。ワードと16進のカラー指定は1つのファイルに混在させることができます。

4. .x11startファイルの先頭近くにxinitcolormapコマンドを挿入します。他のアプリケーションのカラーを割り当てる前にこのコマンドが実行される必要があります。

   例えば、$HOME/.xveecolormapにカラーが記述されており、ファイルに55個のカラー (Rocky Mountain Basic用に16個、VEE用に39個)がある場合、下記の行を.x11startに追加します。

   ```
   /opt/X11/xinitcolormap -c 55 -f $HOME/.xveecolormap
   ```

5. X11を再起動します。このためには、**Shift**+**Ctrl**+**Break**を押してウィンドウ・マネージャを停止するか、ルート・メニューでResetを選択したあと、下記のように入力します。

   ```
   x11start
   ```

USASCII以外のキーボードの使用(UNIX)

USASCII以外のキーボードを使用する場合、X11環境の$LANG変数を変更する必要があります。ドイツ語キーボードを使用するには、下記のコマンド(Kornシェルの場合)を入力します。

```
export LANG=german.iso88591
```

LANG変数が設定されたら、File⇒Default Preferencesでフォントを変更します。

注記

Roman8文字セットで作成されたデータにアクセスする場合、特殊文字(ASCIIの127より上)を変換する必要があります。
ターミナル・ウィンドウでRoman8が用いられている場合があります。このため、stdoutに出力されるTEXT、ファイル名(To FileやFrom Fileで指定するもの)、プログラム名には、VEEでの指定と一致させるため、ASCII文字0～127を使用する必要があります。

HP-GLプロッタの使用(UNIX)

VEEでは、HP-GLを使ってプロッタやファイルにグラフィックスを出力できます。プロッタ(ローカルまたはネットワーク)にプロットを出力するためには、システム管理者にシステム上のスプール・デバイスとしてプロッタを追加してもらう必要があります。

標準のHP-GLプロッタであるHP 7475、HP ColorPro(HP 7440)、HP 7550などのほかに、PaintJet XLやLaserJet IIIなどの一部のプリンタもプロッタとして使用できます。HP ColorProプロッタの場合、極座標またはスミス・チャートのグラフと、Area-Fillライン・タイプの出力には、グラフィックス・エンハンスメント・カートリッジが必要です。PaintJet XLでプロットを出力するにはHP-GL/2カートリッジが必要です。

LaserJet IIIでプロットを作成するには、2メガバイト以上のオプション拡張メモリを使用し、Page Protection構成オプションを有効にする必要があります。ベクタを多く使用するプロット、特に極座標やスミス・チャートのグラフでは、さらに多くのオプション・メモリが必要になることがあります。プリンタからプロットを出力する場合、プロッタ・タイプをHP-GL/2に設定することにより、適切なHP-GL/2セットアップ・シーケンスをプロット情報に追加する必要があります。

下記の2次元グラフィック表示が、HP-GLまたはHP-GL/2プロッタ、またはファイルに出力できます。

- XYトレース
- ストリップ・チャート
- 複素平面
- X対Yプロット
- 極座標プロット
- 波形
- 振幅スペクトル
- 位相スペクトル
- 振幅対位相

デフォルトのプロッタ構成を指定するには、`File⇒Default Preferences`を選択し、`Default Preferences`ダイアログ・ボックスで`Printing`タブを選択して、`Plotter Setup`ボタンをクリックしたあと、`Plotter Configuration`ダイアログ・ボックスを編集します。

表示オブジェクトから直接プロットを生成するには、ディスプレイのオブジェクト・メニューから`Plot`を選択し、`Plotter Configuration`ダイアログ・ボックスで必要なパラメータを指定し、`OK`を押します。

また、制御入力として`Plot`を追加することにより、プログラムからプロットを生成することもできます。表示オブジェクトのビュー全体がプロットされ、元の表示オブジェクトのアスペクト比を維持しながら指定されたプロット・エリアいっぱいになるようにスケーリングされます。

表示オブジェクトのサイズを変更することにより、イメージのプロットのアスペクト比を制御することができます。表示オブジェクトを拡大すれば、プロット周辺のテキストや数値ラベルの相対的なサイズを小さくすることができます。

Plotter Configurationダイアログ・ボックスのプロッタ構成パラメータの説明については、*VEE Online Help*のReferenceにあるObjects and Menu ItemsのDefault Preferencesの項を参照してください。また、該当する2次元表示デバイスのリファレンス・セクションも参照してください。

VEEサンプル・プログラムの使用

VEEには多数のサンプル・プログラムが付属しており、VEEの動作を理解するために役立ちます。

サンプル・プログラムは、VEEのインストール・プロセスの一部としてインストールされます。

サンプル・ディレクトリ

サンプル用のデフォルト・ディレクトリは下記の通りです。

Windowsの場合:

```
C:\Program Files\Agilent\VEE Pro 6.0\examples\
```

HP-UX 10.20で動作するHP-UX版VEEの場合:

```
/opt/veetest/examples/
```

本書で紹介するサンプルは、Manualサブディレクトリにmanual01.VEEなどのファイル名で収録されています。本書で紹介していない他のサンプルが別のサブディレクトリに用意されており、VEEの諸概念やエンジニアリング上の問題の解決法を知るために役立ちます。

サンプルの実行

サンプル・プログラムのロードと実行には、下記のようにHelpメニューを使います。

1. メニュー・バーでHelp⇒Open Exampleをクリックします。これにより、類似のサンプルをまとめているサブディレクトリのリストが表示されます(File⇒Open⇒ExamplesでVEEサンプルをロードすることもできます)。
2. 目的のサブディレクトリをダブルクリックして、そのグループのプログラムを表示させます。
3. 目的の例が見つかるまでリストをスクロールさせます。
4. サンプル名をクリックし、OKをクリックしてプログラムをオープンします。ワーク・エリアに既存のプログラムがある場合、保存のためのプロンプトが表示されます。
5. プログラムを実行するには、ツール・バーのRunボタンを押します。

ライブラリ・オブジェクトの使用

VEEにはオブジェクトのライブラリも用意されており、プログラムにマージして使うことができます。ライブラリ・オブジェクトは、VEEのインストール・プロセスの一部として下記のディレクトリにインストールされます。

Windowsの場合:

```
C:\Program Files\Agilent\VEE Pro 6.0\Lib\
```

HP-UX 10.20の場合:

```
/opt/veetest/lib/
```

大部分のライブラリ・オブジェクトはUserObjectであり、個々のオブジェクトをカプセル化しています。ライブラリ用のUserObjectを作成して保存することができます。

Windowsの場合、下記の場所にUserObjectを保存します。

```
C:\Program Files\Agilent\VEE Pro 6.0\Lib
```

HP-UXの場合、下記の場所にUserObjectを保存します。

```
/opt/veetest/lib/contrib/
```

HP-UXでは、`contrib`サブディレクトリに追加オブジェクトの独自のライブラリを作成できます。

注記

HP-UXプラットフォームで`lib`ディレクトリに書き込むには、rootユーザの権限が必要です。

Formulaオブジェクト

プログラムにマージできるFormulaオブジェクトも用意されています。これらのオブジェクトはそれぞれ便利な変換機能(度からラジアンへの変換など)を実行します。

Windowsの場合、ファイルは下記の場所にあります。

```
C:\Program Files\Agilent\VEE Pro 6.0\Lib\convert\
```

HP-UX 10.20の場合、ファイルは下記の場所にあります。

```
/opt/veetest/lib/convert/
```

サポートされるI/Oインタフェース

機器との通信をVEEから実行するためには、VEEを実行するコンピュータを正しく構成し、I/Oライブラリをインストールする必要があります。インストール手順については、*Installing the Agilent IO Libraries - VEE for Windows*または*Installing the Agilent IO Libraries - VEE for HP-UX*を参照してください。また、論理ユニットとI/Oアドレッシングについては、本書の212ページ「論理ユニットとI/Oアドレッシング」を参照してください。

表1-2に、各プラットフォームでサポートされるI/Oインタフェースの一覧を示します。

表1-2. 機器I/Oサポート

プラットフォーム	サポートされるI/Oインタフェース
Windows 95/98 (PC、HP 6232、HP 6233、EPC7/8)	GPIB[a] シリアル GPIO VXI[b]
Windows NT (PC, HP 6232, HP 6233, EPC7/8)	GPIB[a] シリアル GPIO VXIb
HP-UX (HP 9000 Series 700, V/743)	GPIB[a] シリアル GPIO VXI[c]

a. HP E1406コマンド・モジュールを使ってVXIデバイスと通信可能。

b. 組込みコントローラ(HP 6232またはHP 6233 VXI Pentium®コントローラ、HP RADI-EPC7/8 VXIコントローラ、RadiSys EPC7/8 VXIコントローラ)用の直接バックプレーン・アクセス。VXLinkを使用する外部PC用の直接バックプレーン・アクセス。

c. HP V/743 VXI組込みコントローラ用の直接バックプレーン・アクセス。HP E1489C EISA/ISA-MXIbusインタフェースを使用したシリーズ700外部コンピュータ用の直接バックプレーン・アクセス。

VEE実行モードの使用

このセクションでは、VEE実行モードに関する下記の事項について説明します。

- 実行モードの設定
- 実行モードの変更: VEE 3からVEE 4へ
- 実行モードの変更: VEE 4からVEE 5へ
- 実行モードの変更: VEE 5からVEE 6へ

実行モードの設定

VEEの各バージョンには、複数の実行モード(旧称は互換モード)が存在します。これにより、VEEの古いバージョンで作成されたプログラムを、新しいバージョンのVEEで古いVEEと全く同様に動作させることができます。これを後方互換性と呼び、実行モードによりVEEのすべてのバージョンは100%の後方互換性を示します。VEEのバージョン6.0ではVEE 6実行モードが追加されています。

実行モードとは何か

VEEバージョン4.0には、VEE 3.xとVEE 4の2つの実行モードがありました。これにより、VEEバージョン3.0(以前)で作成された古いプログラムをVEE 4.0で実行しても、VEEバージョン3と全く同じ動作をすることが保証されていました。VEE 3のプログラムをコンパイル・モードで実行したい場合、モードを切り換えるとVEE 4はコンパイラを使ってプログラムを実行しました。

同様に、VEEバージョン5.0にはVEE 3.x、VEE 4、VEE 5の3つの動作モードがありました。VEE 6.0では、これにVEE 6実行モードが加わっています。旧バージョンのVEEで作成されたデータをVEEで読み取ると、そのプログラムが使用していた実行モードが調べられます。プログラムをロードする際に、VEEは自動的にその実行モードに切り替わるので、プログラムは旧バージョンのVEEと全く同じように動作します。

プログラムを作成して実行モードとともに保存すると、ユーザが変更しない限りプログラムの実行モードは変わりません。バージョン5.0で開発して保存したプログラムには、実行モードVEE 5が保存されています。バージョン6.0でこのプログラムをロードして保存しても、プログラムの実行モードはVEE 5のままです。ユーザが実行モードを(Default Preferencesダイアログ・ボックスまたはその他の手段で)変更しない限り、どのバージョンのVEEでプログラムをロードしてもプログラムの実行モードは変化しません。

プログラムの実行モードを変更するには、ファイル・メニューをオープンして Default Preferencesを選択します。Default Preferencesダイアログ・ボックスがオープンしたら、選択する実行モードの隣の菱形をクリックします。プログラムを保存するか、OKをクリックします。

実行モードを変更する理由

実行モードを変更する必要があるのは、既存のプログラムに新しい機能を追加した場合です。例えば、VEEバージョン5.0で作成したプログラム(実行モードはVEE 5)に、VEE 6で使用可能になった新しいデータ型などの新機能を追加した場合、実行モードをVEE 6に変更する必要があります。プログラムを変更して実行モードを変更しないと、新しく追加された機能が正しく動作しないおそれがあります。

実行モードを変更すべきかどうかを知る方法

ほとんどの場合、旧バージョンのVEEで作成したプログラムは、実行モードを変更しない限り100%正しく動作します。しかし、新しい機能は最新の実行モードを使用しないと実行できない場合があります。

表1-3に、各バージョンと実行モードの組合わせで動作するプログラムを示します。古いプログラムは、実行モードを変更しない限りすべてのバージョンで動作することに注意してください。問題が起きる可能性があるのは、実行モードを切り換えた場合だけです。

表1-3. VEEのバージョンと実行モード

	作成時の実行モード:		
実行に使用するVEE:	VEE 4	VEE 5	VEE 6
バージョン4.0	動作	CNA*	CNA*
バージョン5.0	動作	動作	CNA*
バージョン6.0	動作	動作	動作

***CNAと記されているのは、互換性の保証なし(Compatibility Not Assured)という意味です。**新バージョンのVEEで作成されたプログラムは、新バージョン固有の機能を使用していなければ、旧バージョンでもロードして実行できる可能性があります。新たに追加された機能を使用しているプログラムは、旧バージョンのVEEでは正しく動作しません。場合によっては、旧バージョンのVEEにはロードできないことすらあります。

実行モードの意義は、既存のプログラムが新しいバージョンのVEEでも動作することを保証することにあります。新しい機能が旧バージョンのVEEで動作することは保証されません。

実行モード切替えの指針

VEE 6で新しいプログラムを開発する際には、VEE 6実行モードを使用してください。

既存のプログラムを実行する際には、適切な実行モード(VEE 3.xのプログラムではVEE 3、VEE 4.xのプログラムではVEE 4、VEE 5.xのプログラムではVEE 5)を使用します。古いプログラムをロードすると、適切な実行モードが自動的に設定されます。

VEE 6実行モードに切り換えると、古いプログラムは正しく動作するかもしれませんが、動作しない可能性もあります。ほとんどのプログラムは正しく動作します。「バグ・フィックス」のためにプログラムの動作が変わる例について次に説明します。

バージョン5.0のプログラム(実行モードVEE 5)で、To FileがWRITE BINARY BYTEトランザクションを実行しているとします。バージョン5.0(およびそれ以前のバージョン)では、これに"300"を送るとファイルには"44"が書き込まれました(これは技術的には誤りです。"300"は1バイトには入らないので、300を切り捨てて44にするのではなく、エラーを生成するのが正しい動作です)。

バージョン6.0でVEE 5およびそれ以前の実行モードを使っている場合、プログラムの互換性を保つために従来通り"44"が書き込まれます。一方、VEE 6.0でVEE 6実行モードを使っている場合、300をUInt8に変換できない(範囲外)というエラー・メッセージが表示されます。VEE 5実行モードとVEE 6実行モードとの違いについては、35ページ「実行モードの変更: VEE 5からVEE 6へ」を参照してください。

コンパイラについて

コンパイラを使用し、ActiveXオートメーションおよびコントロールを使用する場合、実行モードをVEE 6に設定してください。VEE 3のプログラムをVEE 6モードに変換する場合、まずVEE 4モードとVEE 5モードで動作することを確認してください。各モードの間でプログラムの動作に多少の違いがあるからです。

注記

この章の情報は、コンパイラを使用するためには必要ありません。このセクションでは、コンパイラの背景にある概念を参考のために説明しています。コンパイラに関する情報は、わずかな変更点を除いてVEE 4以上のバージョンに該当します。

コンパイラは、VEE 4、VEE 5、VEE 6の各モードで動作するプログラムに対して使用できます。VEEコンパイラはVEEプログラムをpコードに変換しますが、マシン語や実行可能ファイルは生成されません。

コンパイラを使うと、下記のことが可能となります。

- オブジェクトの実行順序を(実行時に決定するのではなく)コンパイル時に予測します。
- 特定のデータ・ラインにどのデータ型が流れるかを判定します。
- コード生成を最適化します。
- 与えられたVEEオブジェクトに対して最適なpコードを生成して実行します。

VEEプログラムのコンパイルは、Runボタンを押したときに自動的に実行されます。ステップ実行とブレークポイント、それにShow Execution Flow、Show Data Flow、Line Probeの各機能が完全にサポートされます。

同じプログラムを変更せずに再び実行する場合、再コンパイルは不要です。プログラムが変更された場合、再コンパイルが必要なコンテキストだけが再コンパイルされます(インクリメンタル・コンパイラと同様の動作)。ほとんどのプログラムはコンパイラの恩恵を受けますが、実際の結果はプログラムによって異なります。例えば、何レベルにもネストしたループを持つプログラムは、I/Oや画面更新(表示など)を多用するプログラムに比べてより高速化されます。

コンパイル・モードで最も高速化の効果が大きいのは、反復子とFormulaです。VEEの旧バージョンで作成されたプログラムは、コンパイラを使うと動作が多少異なる可能性があります。その理由としては、特定のプログラミング技法の使用、ドキュメントに記載されていない副作用の使用、ドキュメントに記載された動作のわずかな変更などが挙げられます。

実行モードの変更: VEE 3からVEE 4へ

VEE 4.0より前のバージョンで作成されたVEEプログラムは、VEE 3モードで実行すれば旧バージョンと**全く**同様に動作します。これを保証するため、古いプログラムをロードするとインタプリタが自動的に有効になります。このセクションでは、VEEバージョン4.0すなわちVEE 4モードの新機能と拡張について説明します。

コンパイラ・モードでのライン・カラー

コンパイラ・モードでは、オブジェクトの間を結ぶデータ・ラインが、そのラインを流れるデータの型に基づいて異なるカラーで表示されます。デフォルトのカラーとそのカラー・プロパティ名を下に示します。これらを変更するには、Fileメニューから Default Preferences ダイアログ・ボックスを選択します。変更したいラインをScreen Elementボックスで選択し、Color Valueボックスをクリックしてカラー・パレットをオープンして、選択するカラーをクリックします。OKをクリックすると選択したライン・タイプに新しいカラーが反映されます。

- ダーク・スカイ・ブルー :数値(IntegerまたはReal型)
- ダーク・スカイ・ブルー :複素数(ComplexおよびPComplex型)
- ミディアム・オレンジ:文字列(String型)
- ミディアム・ダーク・グレー :シーケンス出力(nil値、通常はシーケンス出力ラインから)
- マゼンタ:強調表示
- 黒:未知の型または最適化されない型(Record型など)

データ型が配列の場合、太いラインが表示されます。速度を上げるためには、プログラム内のラインのカラーに注目してください。黒のラインが少ないほど、プログラムは高速に実行されます。

互換性に関する潜在的問題

VEE 4.0より前のバージョンで作成されたプログラムは、自動的にVEE 3モードで実行されます。VEE 4.xで作成されたプログラムは自動的にVEE 4モードで実行されます。VEE 5.xで作成されたプログラムは自動的にVEE 5モードで実行されます。ただし、プログラムの実行モードはいつでも変更できます。

既存のプログラムをVEE 3からVEE 4に変更すると、一部の領域で互換性の問題が発生するおそれがあります。問題のおそれがある領域について以下で説明します。旧バージョンのVEEの使用に関する情報は、インタプリタ・モードまたはVEE 3モードを使用する場合にも当てはまります(プログラムを新規に作成する場合は、VEE 6実行モードを使用してください)。

UserFunctionのタイムスライス実行. VEE 4.0より前のバージョンでは、UserFunctionがプログラムの他の部分と並行にタイムスライス実行されることはありませんでした。コンパイル・モードの場合、別々のスレッドから呼び出されたUserFunctionはタイムスライス実行されます。並列実行が望ましくない場合、Callオブジェクトの間に必ずシーケンス・ピンを使用してください。

UserFunctionがタイムスライス実行されるのは、Call、Formula、If/Then/Else、Sequencerのいずれかのオブジェクトから呼び出された場合(Functionフィールドから呼び出された場合)だけです。また、Callを初め上記のオブジェクトから呼び出された場合、UserFunctionでブレークポイントが使用できるようになりました。

To File、To String、または類似のオブジェクトから呼び出された場合、または制御ピン経由で式が与えられた場合、UserFunctionはタイムスライス実行されず、ブレークポイントも動作しません。

UserFunctionが実行中にプログラムの別の部分から呼び出された場合、最初の呼出しが戻るまで2番目の呼出しはブロックされます。

UserObject. UserObjectは以前のバージョンでは常にタイムスライス実行されましたが、コンパイル・モードでは別々のスレッドから呼び出された場合のみタイムスライス実行されるようになりました。

関数の優先順位. Formulaオブジェクトから呼び出される関数の優先順序が下記のように変わりました。

1. 内部関数(sin()、totSize()など)
2. ローカルUserFunction
3. インポートされたUserFunction
4. コンパイルされた関数
5. リモート関数

VEE 3実行モードの場合、内部関数は優先順位が最低です。このため、totsize()やfft()などの内部関数を独自のものに置き換えることが可能でした。

Auto ExecuteとStart . 一部のオブジェクトのAuto Execute機能を使ったときの動作にわずかな変更があります。コンパイル・モードでの動作は、オブジェクトが

Startオブジェクトに直接フックされ、Startボタンが押された場合と同様です。ほとんどのプログラムはこの変化によって影響を受けません。

OKボタンとWait for Input. OKを初めとする非同期オブジェクトの大部分と、Wait for Inputを有効にしたすべてのオブジェクトの動作は、コンパイル・モードでは下記の2つの点で改良されています。

- ステップ実行: 以前のバージョンでは、これらのオブジェクトをステップ実行するとプログラムが終了することがありました。コンパイル・モードではステップ実行が正しく動作します。
- CPU使用率: 以前のバージョンでは、これらのオブジェクトを実行するとCPU使用率が上がりました。コンパイル・モードではCPUはアイドル状態のままです。

データのないコレクタ. 以前のバージョンでは、データが送られたことのないコレクタのXEQピンにデータを送ると、nilコンテナが出力されました。コンパイル・モードでは、コンパイル時にデータ型が知られている場合、そのデータ型の0要素の配列が得られます。データ型が知られていない場合、Integer型の0要素の配列が得られます。

この変更により、型の推定がより一貫したものになり、コレクタ・オブジェクトより下流で生成されるpコードの品質が改善されます。なお、nilにtotSize()を適用すると1が返りますが、0要素配列に対するtotSize()は0を返すことに注意してください。

データのないSample & Hold. 以前のバージョンでは、データが送られたことのないSample & HoldオブジェクトのXEQピンにデータを送るすると、nilコンテナが出力されました。コンパイル・モードでは、下記のエラー(エラー番号937)が生成されます。

Sample & Hold was not given any data.(Sample & Holdにデータが与えられていません)

この変更により、型の推定がより一貫したものになり、Sample & Holdオブジェクトより下流で生成されるpコードの品質が改善されます。

タイマ・オブジェクト. 以前のバージョンでは、Timerオブジェクトの Time2ピン(下のデータ入力ピン)にTime1ピンよりも先にデータが送られた場合、未定義の結果が出力されていました。コンパイル・モードでは、Timerオブジェクトのピンの実行順序が間違っているとエラーが生成されます。

フィードバック・サイクル. コンパイル・モードでは、フィードバック・サイクルの内部にJunctionオブジェクトが必要です。Startオブジェクトは必要でなくなりました。フィードバックにJunctionがないことが検出された場合、下記のエラー (エラー番号935)が生成されます。

`A Junction is required inside of feedback cycles.`(フィードバック・サイクル内部にJunctionが必要です)図1-4および図1-5を参照してください。

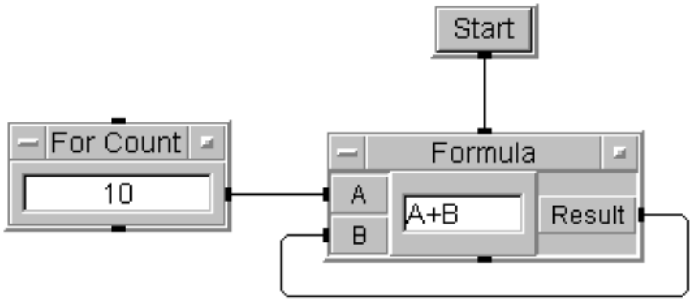


図1-4. 以前のバージョンのフィードバック

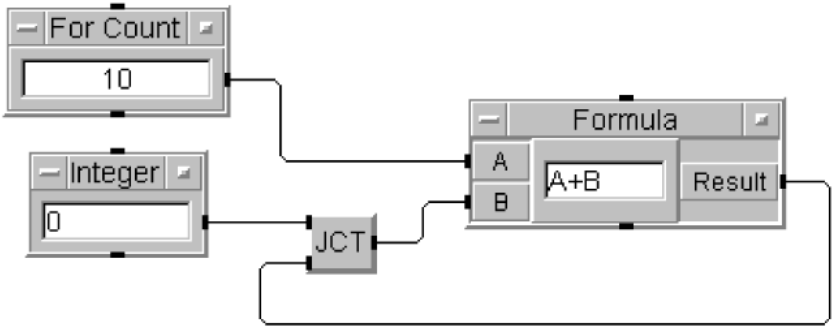


図1-5. コンパイル・モードのフィードバック

VEEバージョン4.0以上では、無効な接続をすることはできません。例えば、オブジェクトのデータ入力ピンを同じオブジェクトのデータ出力ピンに接続することはできず、大部分のオブジェクトではシーケンス出力ピンをデータ入力ピンに接続することができません。

並列スレッド. VEE 3実行モードの場合、独立したスレッドは順番にラウンドロビン実行されます。すなわち、あるスレッドのオブジェクトが実行されたあと、別のスレッドのオブジェクトが実行されます。コンパイル・モードの場合、この動作は保証されません。

ループの限界. ループの性能を改善するため、反復子の限界(For Rangeオブジェクトのstepフィールドなど)は毎回調べられるのではなく最初の反復の開始時にだけ調べられます。実行時には、値が変更できないことを示すため、オブジェクトのフィールドが淡色表示になります。ループの実行中に値を変更しても、反復子へのデータ入力は無視されます。

例えば、For Rangeオブジェクトのstep値をループ実行中にデータ入力ピンから変更しても、VEE 4以上の実行モードでは無視されます。以前のバージョンでは、反復のたびにステップ値がチェックされていました。

UserObjectとXEQピンによる呼出し. 4.0より前のバージョンでは、UserObjectやCallオブジェクトのXEQピンを使って、全データ入力ピンが満足される前にUserObjectやUserFunctionを実行することができました。UserObjectまたはUserFunction内部のオブジェクトのうち、これら満足されていないデータ入力に接続されたものの動作は未定義でした。VEE 4以上の実行モードではこれは許容されません。これらのオブジェクトにXEQピンがあるとエラーが発生します。これらのオブジェクトにXEQピンを追加することはできなくなりました。

XEQピンを持つOKボタン. 4.0より前のバージョンでは、XEQピンを持つOKオブジェクトは、OKボタンが押されたときか、XEQピンにデータが送られたときのどちらか1回だけしか実行されませんでした。VEE 4以上の実行モードでは、XEQピンにデータが送られるたびにOKボタンは実行されます。OKオブジェクトにXEQピンを追加することはできなくなりました。

EOFピンを持つFrom File. 4.0より前のバージョンでは、From Fileオブジェクトのデータ出力ピンはVEEの他のデータ出力ピンと異なる扱いを受けていました。From

Fileがループ内にある場合、EOFデータ出力ピンが実行されても出力ピンのデータは有効なままでした。

VEE 4以上の実行モードでは、ループが実行されるたびに(他のすべてのオブジェクトと同様)From Fileオブジェクトからのデータ出力は無効になります。このため、EOFピンが実行された時点で、データ出力はすでに無効であり、伝搬することはできません。

図1-6にこの状況を示します。4.0より前のバージョンでは、FormulaのAに入力されたデータはループの次の反復が実行される間も有効です。FormulaのBに有効なデータを送るため、EOFピン(下)が実行され、それによりFormulaが実行されます。

VEE 4以上の実行モードでは、ループの次の反復が始まると同時にAに入力されたデータは無効になります。ループの同じ反復でFormulaが有効な入力を受け取らないため、Formulaは実行されません。

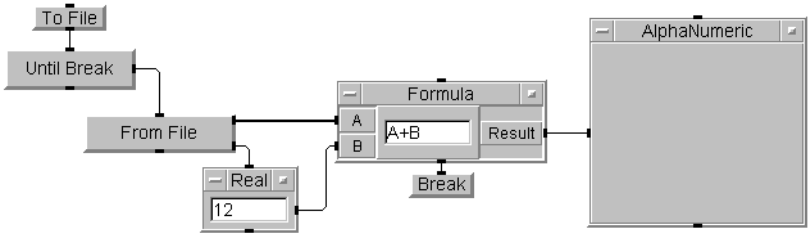


図1-6. EOFの違い

並列ジャンクション. VEE 4.0より前のバージョンでは、Junctionオブジェクトと並列に拘束条件のないオブジェクトが接続されている場合、実行の順序は接続を行った順序によって決まっていました。VEE 4以上の実行モードでは、図1-7のように接続の順序は影響しません。

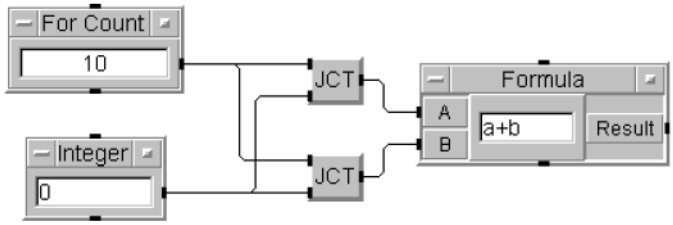


図1-7. 並列ジャンクション

交差ループ. 4.0より前のバージョンでは、反復オブジェクトを交差することができました。実行の順序は未定義であり、接続の順序によって決まっていました。VEE 4以上の実行モードでは、ループが交差できるのはJunctionオブジェクトを介した場合に限られます。これ以外の交差ループはエラー 938(VEE was unable to compile this part of the program、VEEはプログラムのこの部分をコンパイルできません)を生成します。図1-8にこの状況を示します。

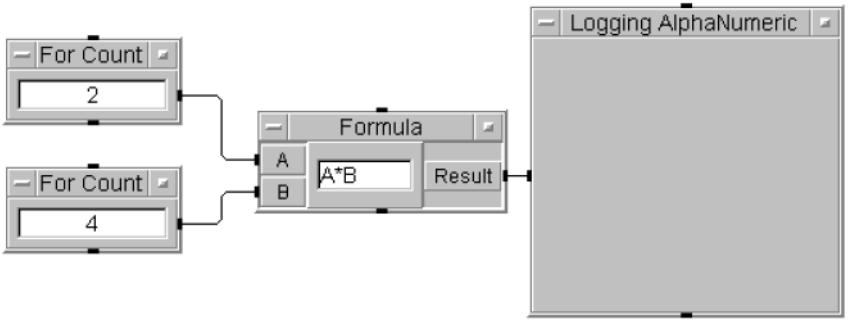


図1-8. 交差ループ

Junctionsを介した交差ループ. VEE 4.0より前のバージョンの場合、図1-9の例ではInteger が最初に実行され、Breakに達するとプログラムは停止します。VEE 4以上の実行モードの場合、Breakでプログラムが停止しないため、下の例ではIntegerオブジェクトのあとにFor Countオブジェクトが実行されます。

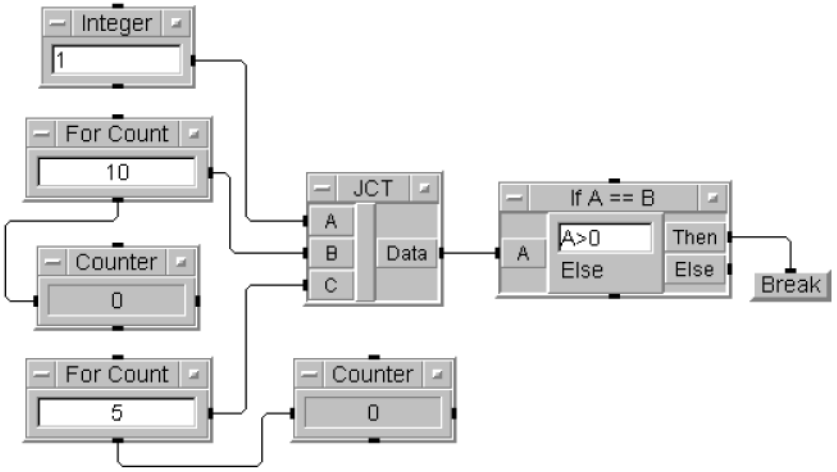


図1-9. Junctionsを介した交差ループ

オープン・ビュー・オブジェクトの変更. VEE 4.0より前のバージョンでは、プログラムの実行中または休止中に、オープン・ビュー・フィールドのデータを変更することができました。このような変化はプログラムの動作に影響し、結果は保証されません。VEE 4以上の実行モードでは、多くのオブジェクトでプログラムの実行中または休止中にこのような変更を行うことができなくなりました(入力フィールドが淡色表示されます)。下にその例を示します。

- FormulaおよびIf/Then
- Collector
- すべてのトランザクション・オブジェクトのトランザクション
- Get MappingsおよびSet Mappings
- Get ValuesおよびSet Values
- Constantのプロパティ、例えばScalarまたは1D Array、Wait for Input、Auto Executeの設定など
- Clear at PreRunなどのプロパティの設定
- UserObjectおよびUserFunctionのTrig Mode

■ Dialog Boxesプロパティ

オブジェクトの入出力端子の追加と削除は、実行時には淡色表示になります(休止時にはなりません)。休止時にこの動作を実行すると、プログラムは停止します(これはVEE 4.0より前のバージョンと同じです)。

式の配列シンタックス. 式で配列シンタックスを[1 2 3]のようにカンマなしで入力した場合、プログラムのロード時に再解釈され、[1,2,3]のようにカンマを使った形に自動的に変更されます。これはVEE 3モードとVEE 4モードのどちらのプログラムにも当てはまります。

実行モードの変更: VEE 4からVEE 5へ

VEE 6.0でも、VEE 4モードとVEE 3モードはVEE 4.0で設定された互換性定義(21ページ「実行モードの変更: VEE 3からVEE 4へ」参照)に従います。元の実行モード(VEE 3またはVEE 4)で実行されるプログラムには影響しない小さな変更がいくつかあります。古いモードから新しいモードにプログラムを変換する場合、これらの変更について知っておく必要があります。

VEE 5実行モードについて

VEE 5実行モードは、VEE 4モードのスーパーセットです。VEE 5モードにはすでに説明したコンパイラ機能があり、プログラムの互換性に影響するいくつかの重要な変更が加えられています。変更のほとんどは、ActiveXオートメーションおよびコントロールのサポートを有効にするためのものです。

他の変更のうちには、ActiveXを使用しなくても、このセクションで説明する機能を使用する場合にプログラミング方法に影響するものがあります。VEEでActiveXを使用する方法の詳細については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

VEE 5実行モードへのプログラムの変換

古いプログラムは適切な古い実行モードで自動的にオープンされます。古いプログラムを新しいモードに変更する場合、FileメニューのDefault Preferencesから手動で行う必要があります。プログラムをVEE 5モードに変更すると、エラーが発生する可能性があります。この場合、問題を記述したリストが表示されます。これらのエラーを修正することによって、初めてVEE 5モードへの切替えが完了します。エラーを修正するためにプログラムが自動的に変更されることはありません。

VEE 5実行モードの変更

エラーの修正方法を知るため、VEE 5モードの互換性に関する変更を下に記します。

注記

VEE 3.xのプログラムをVEE 5モードに変更する場合、VEE 4モードで動作することをまず確認してから、VEE 5モードに変更してください。この変換に関しては、21ページ「実行モードの変更: VEE 3からVEE 4へ」を参照してください。

メニューの変更. VEE 5.0で追加されたActiveXサポートに伴って、Deviceメニューが多少変更されました。下記の新しいメニュー・アイテムが追加されています。

```
ActiveX Automation References*
ActiveX Control References*
ActiveX Controls
```

また、以前にSelect Functionsダイアログ・ボックスをオープンする役割を果たしていたメニュー・アイテムMath & Functionsは、Function & Object Browserに変更され、Function & Object Browserをオープンするようになりました。これは従来と同様に、プログラムで使用する演算子と関数の選択に使用でき、ActiveXをサポートするための機能が追加されています。

式. Formulaなど、式を含むオブジェクトに影響する下記の変更が行われています。

- ActiveXオートメーションに用いられるキーワードとして、SETとByRefが追加されました。これらの語は予約されており、端子名としては使用できません。
- ActiveXオートメーション用の新しいシンタックスとして、excel.worksheets(1).cells(1,2) = 2などがサポートされるようになりました。
- VEE 3およびVEE 4モードでは、式で配列シンタックスを[1 2 3]のようにカンマなしで入力した場合、プログラムのロード時に再解釈され、[1,2,3]のようにカンマを使った形に自動的に変更されます。VEE 5以上のモードでは、[1 2 3]のようにカンマなしで配列シンタックスを入力すると、Formulaがフォーカスを失った時点でエラーが発生します。
- 1のような値はINT32を返し、1.0はREAL64を返します。従来はどちらもREAL64を返していました。

- ActiveX オートメーション用の組込み関数として、CreateObject() および GetObject()の2つが新たに追加されています。

- ActiveXオートメーション用の組込み定数として、trueおよびfalseの2つが新たに追加されています。

変数. 変数に関して下記の変更があります。

- Delete Variables at PreRunが(Default Preferencesで)オンになっていても、ActiveXコントロールを参照するグローバル変数は削除されません。

- Declare Variableオブジェクトに、ActiveXオートメーション用の新しい変数型Objectが追加されました。

- 新しいObject変数型は、入力端子でRequired Typeとしても使用できます。ただし、他の型との間の強制型変換はできません。

グローバル名前空間. グローバル名前空間の規則が変わりました。これにより、変数、関数、ライブラリの名前に対して下記の影響があります。

- ローカルUserFunction、ライブラリ名、宣言済みおよび未宣言のグローバル変数、ライブラリ・ローカル宣言済み変数は、すべて同じ名前空間に存在し、一意の名前を持たなければならなくなりました。これにより、1つの名前に複数のインスタンスが存在する既存のプログラムは影響を受けます。例えば、UserFunctionと宣言済みグローバル変数の両方にdaily_resultsという名前を付けることはできません。この場合、VEE 5モードに切り換えたときにエラーが発生します。

- ライブラリ内で、ローカルUserFunctionとライブラリ・ローカル宣言済み変数は同じ名前空間に存在し、一意の名前を持たなければなりません。名前が衝突していると、プログラムをVEE 5モードに切り換えたとき、またはVEE 5モードのプログラムにライブラリをインポートしたときにエラーが発生します。

■ 下記のような新しいシンタックスがすべてのモードで Formula オブジェクトに使用できるようになりました。

```
lib.func(a,b) = RightHandExpr
```

これはすべてのモードで正しく解釈されます。ただし、正しく実行されるのはVEE 5以上のモードのときだけで、VEE 3およびVEE 4モードではランタイム・エラーが発生します。

グローバル名前空間の規則の変更に伴い、Formulaで用いられる変数名および関数名の探索の優先順位がVEE 5以上のモードでは下記のように変更されています。

1. ローカル入出力端子
2. 宣言済みのコンテキスト・ローカル変数
3. UserFunctionコンテキストにネストされたUserObjectコンテキスト内部では、宣言済みのライブラリ・ローカル変数
4. 宣言済みおよび未宣言のグローバル変数、ローカルUserFunction、ライブラリ名。これらはすべて一意でなければなりません。
5. sin()、totSize()などの組込み関数
6. ActiveXコントロールおよびオートメーション定数。これは、Instrument⇒ActiveX Automation ReferencesまたはActiveX Control Referencesで参照されたライブラリに依存します。例えば、Excelのオートメーション・ライブラリにはxlMaximizedなどの多くの定数が存在します。
7. インポートされた UserFunction、コンパイル関数、リモート関数はランダムな順序で現れます。正しい順序を保証するには、インポートされたライブラリ名をmyLib.func()のように指定してください。

この新しい順序のために古いプログラムがうまく動作しなくなる例としては、あまり現実的ではありませんが、sin(90)という式を持つFormulaにsinという名前のデータ入力端子(変数)が存在する場合が挙げられます。VEE 3およびVEE 4モードの場合、入力端子名は無視され、組込み関数sin()が呼び出されます。

一方、VEE 5以上のモードでは、新しい優先順位によって関数名と変数名が探索されます。このため、VEE 6.0は端子名を探索し、入力にActiveXオブジェクトがあると仮定し、オブジェクトのデフォルト・メソッドを呼び出そうとします。ActiveXオブジェクトのデフォルト・メソッドを呼び出すcells(1,1)のような式は、sin(90)と類似しています。ActiveXに関しては、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

READ TEXTトランザクション. VEE 3およびVEE 4モードでは、READ TEXTトランザクションでTOKENフォーマットをEXCLUDE CHARS付きで使用した場合、指定された文字の前で読取りポインタが止まり、指定された文字は除去されません。図1-10にVEE 4でのこの例を示します。

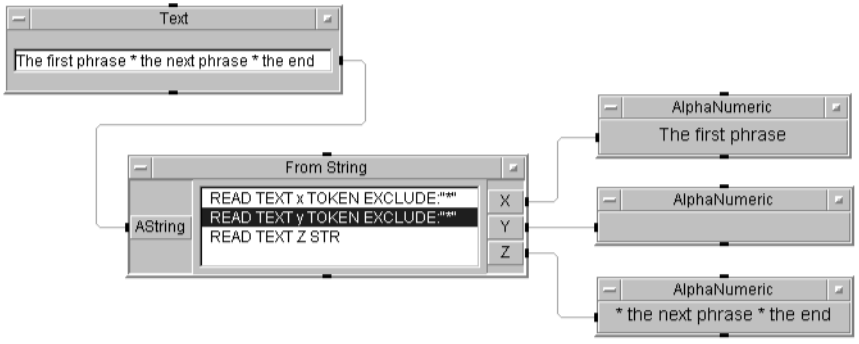


図1-10. VEE 4**モードでのTOKENを使ったREAD TEXTトランザクション**

このように予期しない結果が得られます。予期される結果としては、図1-11のVEE 5の例のように、各フレーズが除去された文字"*"で区切られ、別々のAlphaNumeric表示に現れるはずです。

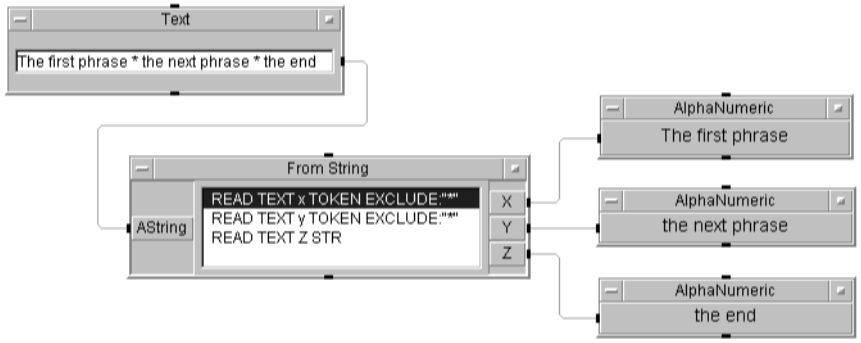


図1-11. VEE 5モードでのTOKENを使ったREAD TEXTトランザクション

To/From FileとTo/From DataSetの相互作用. VEE 3およびVEE 4モードでは、To FileまたはFrom Fileオブジェクトを EXECUTE REWINDトランザクションとともに使用して、To DataSetまたはFrom DataSetと同じデータ・ファイルにアクセスすると、予期しない相互作用が発生します。具体的には、プログラムがFrom Fileを(EXECUTE REWINDとともに)使用してファイルからデータを**読み取った**あとで、To DataSetを使用してデータを同じファイルに**書き戻す**と、データが間違って書き込まれることがあります。

同様の相互作用が、From DataSetとTo Fileを組み合わせた場合にも発生します。VEE 5以上のモードでは、この予期しない相互作用が修正されているため、データは正しくファイルに書き込まれます。ただし、同じファイルに対してTo/From Fileと To/From DataSetを組み合わせて使用することは、VEE 5以上のモードでも推奨されません。

HP-UXでのVEE 5モードの使用

VEE 5モードでActiveXがサポートされるのはWindowsに対してだけなので、いくつか注意しなければならないことがあります。HP-UX版VEEでプログラムをVEE 5モードにした場合、上記のグローバル名前空間の変更は発生しますが、ActiveXがHP-UXではサポートされないため、ActiveXオートメーション関係のメニュー・アイテムは追加されません。

Windows版VEEでActiveX機能を使って開発したプログラムをHP-UXシステムで動作させると、エラーが発生したり、正しく動作しなかったりするおそれがあります。ActiveX オートメーションをサポートするVEE 関数(CreateObject() および GetObject())をプログラムが呼び出している場合、エラーが発生します。Object変

数型を宣言しているプログラムの場合、HP-UX版VEEにロードすることはできますが、正しく実行されません。

実行モードの変更: VEE 5からVEE 6へ

VEE 6.0でも、VEE 4モードとVEE 3モードはVEE 4.0で設定された互換性定義(21ページ「実行モードの変更: VEE 3からVEE 4へ」参照)に従います。元の実行モード(VEE 3またはVEE 4)で実行されるプログラムには影響しない小さな変更がいくつかあります。古いモードから新しいモードにプログラムを変換する場合、これらの変更について知っておく必要があります。これらについて以下で説明します。

VEE 5実行モードについて

VEE 5実行モードは、VEE 4モードのスーパーセットです。VEE 5モードにはすでに説明したコンパイラ機能があり、プログラムの互換性に影響するいくつかの重要な変更が加えられています。変更のほとんどは、ActiveXオートメーションおよびコントロールのサポートを有効にするためのものです。

他の変更のうちには、ActiveXを使用しなくても、このセクションで説明する機能を使用する場合にプログラミング方法に影響するものがあります。VEEでActiveXを使用する方法の詳細については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

新しいデータ型

VEE 6.0の新しいデータ型として、Int16、Real32、Variant、UInt8があります。新しいデータ型と新しいトランザクション(WRITE TEXT INT16など)は、すべての実行モードに現れます。ただし、古いモードでは新しいトランザクションは従来の方法で動作します。

例えば、VEE 5モードでWRITE BINARY INT16を使用すると、実際にはWRITE BINARY INT32が実行され、データがInt16に変換されることはありません。VEE 6モードでは、WRITE BINARY INT16はデータをInt16に変換します。VEE実行モードによるプログラムの動作の違いについては、17ページ「実行モードの設定」を参照してください。

VariantからVEEデータ型への変換 - 配列の扱いの改善

ActiveXオートメーション・サーバ(Excelなど)またはActiveXコントロールからデータが返された場合、VEEはオートメーションのデータ型をVEEのデータ型に変換します。VEE 5.0では、Variantの配列はVEEのRecordに変換されていました。VEE 6.0(VEE 6実行モード)では、すべての要素のデータ型が一致するVariantの配列はVEEの配列に変換されます(データ型が混在する場合、VEE 5.0と同じ動作になります)。

配列のすべての要素が同じデータ型の場合、Variantデータ型からVEEの配列へのマッピングは下記のように行われます。

注記

この機能はVEE 6実行モードでのみ使用できます。

Variant配列のメンバの型	VEE 6.0データ型
VT_UI1	UInt8
VT_BOOL	Int16
VT_I2	Int16
VT_UI2	Int16
VT_I4	Int32
VT_UI4	Int32
VT_R4	Int32
VT_R8	Real64
VT_DATE	Real64
VT_CY	Real64
VT_BSTR	Text
VT_DISPATCH	Object

ただし、新しいVEE 6データ型とActiveXオートメーション・サーバとのこの互換性には、まだいくつかの穴があります。

- 論理値(VT_BOOL)、日付(VT_DATE)、通貨(VT_CY)、エラー(VT_ERROR)の各データ型に対しては、対応するVEEのデータ型がありません。これらのデータ型をActiveXで"ByRef"付きで使用できるのは、下記のセット関数とクエリ関数の場合だけです。

- 一部の特殊なVariant値、例えば空値(VT_EMPTY)やヌル(VT_NULL)などは、対応するものがVEEになく、一意に識別することができません。

セット関数. セット関数は、これらの関数が返すコンテナがActiveXオートメーション動作中に特別な扱いを受けることをVEEに通知する役割を果たします。セット関数には下記のものがあります。

asVariantBool()
asVariantCurrency()
asVariantDate()

asVariantError()
asVariantEmpty()
asVariantNull()

クエリ関数. クエリ関数は、オートメーション・メソッドおよびプロパティの戻り値から作成されたコンテナに対して用いられます。クエリ関数には下記のものがあります。

isVariantBool()
isVariantCurrency()
isVariantDate()
isVariantError()
isVariantEmpty()
isVariantNull()

アップデートされた関数

下記の関数はVEE 6.0でアップデートされました。

whichOS() - 戻り値に"Windows_98"と"Windows_2000"が加わりました。

createObject() - リモート・ホスト・コンピュータの名前を指定する任意指定の2番目のパラメータが追加されました。

参考文献

本書で紹介した機器制御に関する話題の詳細については、下記の文書を参照してください。

■ *Tutorial Description of the Hewlett-Packard Interface Bus* (Hewlett-Packard Company, 1987), part number 5021-1927.

この文書には、IEEE 488.1およびIEEE 488.2に含まれる重要な諸概念の簡潔な説明が記載されています。IEEE 488.1インタフェースになじみのない方には最良の入門書です。

■ *IEEE Standard 488.1-1987*, *IEEE Standard Digital Interface for Programmable Instrumentation* (The Institute of Electrical and Electronics Engineers, 1987).

この標準は、GPIB(IEEE 488.1)インタフェースの設計と製造に必要な技術的細部を定義しています。この標準に記載されている電気的仕様やプロトコル情報は、大部分のプログラマが必要とする範囲を超えています。

■ *IEEE Standard 488.2-1987*, *IEEE Standard Codes*, *Formats*, *Protocols*, *and Common Commands For Use with ANSI/IEEE Std 488.1-1987* (The Institute of Electrical and Electronics Engineers, 1987).

この文書は、SCPI(Standard Commands for Programmable Instruments)を実装した機器が使用する基本的なメッセージ・フォーマットとデータ型について説明しています。

■ *IEEE Standard 728-1982*, *IEEE Recommended Practice For Code and Format Conventions For Use with ANSI/IEEE Std 488-1978, etc.* (The Institute of Electrical and Electronics Engineers, 1983).

■ *VMEbus Extensions for Instrumentation*, including: "VXI-0, Rev. 1.0: Overview of VXIbus Specifications" and "VXI-1, Rev. 1.4: System Specification," VXIbus Consortium, Inc., 1992.

■ *HP VISA User's Guide* (Hewlett-Packard Company, 1998), part number E2090-90035.

この文書は独自のVXI*plug&play*ドライバを作成するユーザに役立つもので、VXI*plug&play*ドライバとの通信とドライバの使用に関する詳細な情報が記載されています。

参考文献

2

機器制御の基本

機器制御の基本

VEEでは、機器を制御するために5種類のオブジェクトがサポートされています。図2-1に各オブジェクトのオープン・ビューを示します。これらの例は、PCプラグイン・カード・ドライバ・オブジェクトを除いて、HP E1410A VXIマルチメータと通信するためのものです。

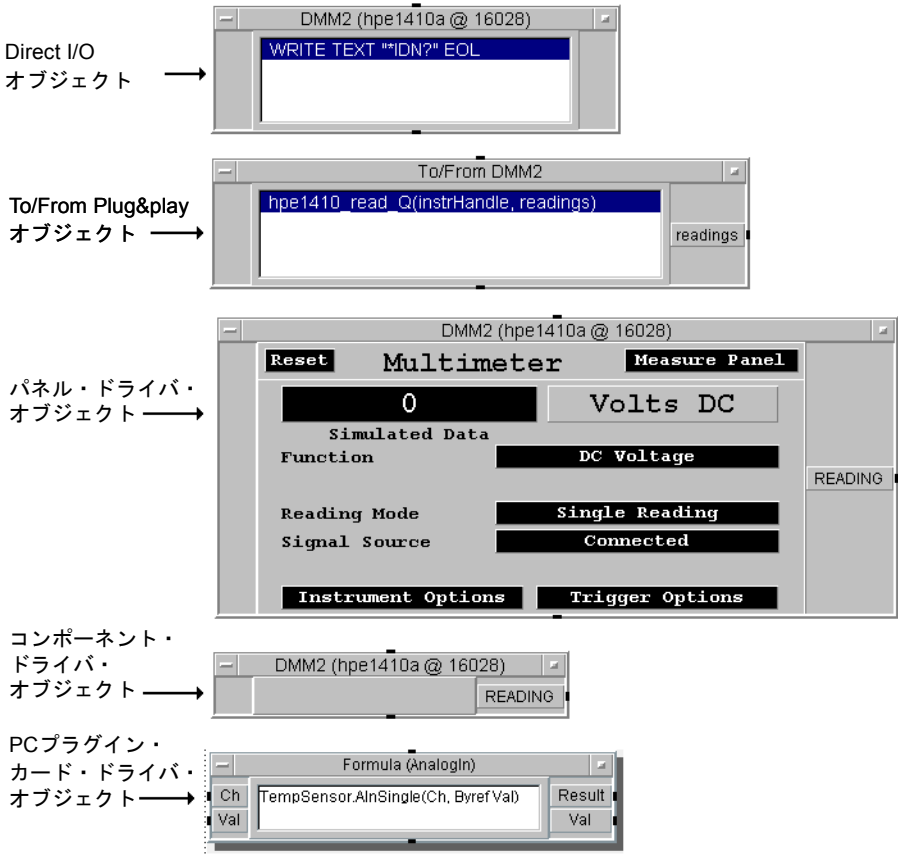


図2-1. VEE機器制御オブジェクト

表2-1は、これらの機器制御オブジェクトの間の違いをまとめたものです。

表2-1. VEEの機器制御オブジェクトの比較

VEE オブジェクト	機器アクセス方法	主な利点	サポートされるインタフェース[a]
`Direct I/O`	任意の機器と直接通信	高速なI/O。任意の機器を制御可能。	GPIB、シリアル、GPIO、VXI、LAN
`To/From VXIplug&play`	機器メーカから提供される各プラットフォームに固有のVXI*plug&play*ドライバが必要。VISAがインストールされていること。	高速なI/O。複数のソフトウェア・アプリケーションからドライバを使用可能。	GPIB、VXI、シリアル
`Panel Driver`	VEEに付属する機器パネル・ドライバが必要。[b]	使いやすさ	GPIBおよびVXI
`Component Driver`	VEEに付属する機器パネル・ドライバが必要。	パネル・ドライバより高速なI/O	GPIBおよびVXI
`PC Plugin Card Driver`	機器メーカから提供されるODASドライバが必要。	高速なI/O。複数のソフトウェア・アプリケーションからドライバを使用可能。	PCプラグイン・スロット

a. HP-IBはIEEE-488インタフェース・バス標準のHewlett-Packardによる実装です。他の実装はGPIBと呼ばれます。LANインタフェース・サポートは、純粋にLANベースの機器を対象としません。

b. パネル・ドライバは「VEEドライバ」と呼ばれることもあります。

`To/From VXIplug&play`、`Panel Driver`、`Component Driver`、`PC PlugIn Driver`の各オブジェクトを使えば、機器のプログラミング・ニーモニックやシンタックスの詳細を知らなくても機器を制御することができます。ローレベル・ニーモニックを送ることによって機器と通信したい場合、または機器用のドライバが用意されていない場合は、`Direct I/O`を使います。

注記

1つのVEEプログラムで、これら5種類の方法すべてを組み合わせて異なる機器と通信することができます。ただし、同じプログラムで同じ機器と通信するために、VXI*plug&play*ドライバと他の方法とを組み合わせて使わないでください。予期しない結果が生じるおそれがあります。

Direct I/Oについて

Direct I/Oオブジェクトは、ファイルの読み書きと同様の方法で任意の機器データを読み書きするためのものです。これにより、あらゆる機器のプログラム可能なすべての機能に自由にアクセスできます。Direct I/Oを使うのに機器ドライバ・ファイルは不要ですが、機器のプログラミング・コマンドに関する詳細な知識が必要です。GPIB、VXI、Serialの各デバイスとDirect I/Oによって通信する場合、I/Oライブラリをインストールしておく必要があります。このための手順については、*Installing the Agilent I/O Libraries (VEE for Windows)* または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。

Direct I/Oオブジェクトは、**ラーン・ストリング**を簡単にサポートできるという特長もあります。ラーン・ストリングとは、ある種の機器がサポートする特殊な機能です。これを使う際には、まず物理機器のフロントパネルから測定ステートをセットアップします。機器の構成が済んだら、Direct I/Oオブジェクト・メニューからUploadを選択するだけで、機器の測定ステートがすべてVEEにアップロードされます。測定ステートをプログラムからリコールすることもDirect I/Oオブジェクトからできます。

Direct I/Oの例

図2-2に示すのは、HP 34401Aマルチメータから識別文字列を取得するためのDirect I/Oオブジェクトのセットアップです。

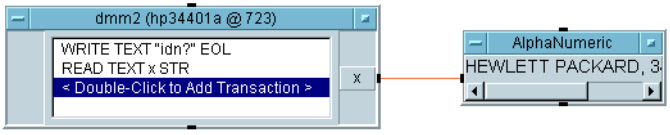


図2-2. Direct I/Oを使った機器の識別

Direct I/Oオブジェクトの最初のトランザクションでは、GPIBアドレス722にあるHP 34401Aに対してテキスト文字列*IDN?が出力されます。これに対してHP 34401Aが識別文字列を送り返し、それが 2 番目のトランザクションで読み取られてAlphaNumericオブジェクトに出力されます。

Direct I/Oを使用するためにVEEを**構成**する方法については、第3章「機器の構成」を参照してください。Direct I/Oオブジェクトの使い方の詳細については、第4章「トランザクションI/Oの使用」を参照してください。

MultiInstrument Direct I/O

MultiInstrument Direct I/Oオブジェクトを使うと、1個のオブジェクトからダイレクトI/Oトランザクションを使って複数の機器を制御できます。このオブジェク

トはDirect I/Oオブジェクトに似ていますが、オブジェクト内のトランザクションがそれぞれ異なる機器を相手にできる点が異なります。

このオブジェクトは標準のトランザクション・オブジェクトであり、VEEがサポートするすべてのインタフェースに対して動作します。MultiInstrument Direct I/Oオブジェクトの制御対象は1台の機器とは限らないため、機器名、アドレス、ライブ・モード条件はタイトルに表示されません。

MultiInstrument Direct I/Oを使うことにより、プログラムに存在する特定機器専用のDirect I/Oオブジェクトの数を減らすことができます。GPIBに比べて機器制御が高速なVXIインタフェースでは、これによる性能の向上が特に顕著です。

図2-3に示すのは、MultiInstrument Direct I/Oオブジェクトとそ のI/Oトランザクション・ダイアログ・ボックスです。オブジェクトはHP E1413B、HP E1328、HP 3325の3台の機器と通信するように設定されています。

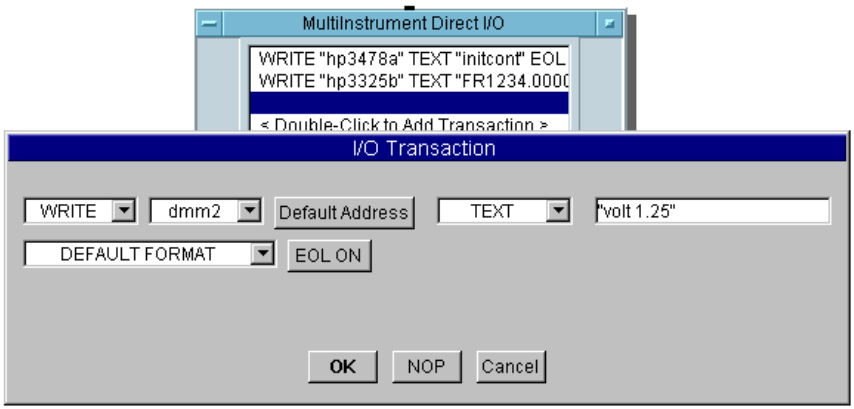


図2-3. MultiInstrument Direct I/Oによる複数機器の制御

MultiInstrument Direct I/Oオブジェクトの使用方法の詳細については、181ページ「MultiInstrument Direct I/Oオブジェクトの使用」を参照してください。

VXI*plug&play*について

VXI*plug&play*は、複数のベンダが互換性のあるハードウェアおよびソフトウェアを提供することを可能にするインタフェース仕様です。VXI*plug&play*ドライバは、特定の機器を制御する関数のライブラリです。ドライバはハードウェア・ベンダによって作成され、機器とともに提供されます。

非VXI機器用にVXI*plug&play*ドライバを作成することもできます。

VEEバージョン3.2以降では、WIN95/98、WINNT、HP-UXのいずれかのフレームワークに適合する、VXI*plug&play*仕様バージョン3.0以降のドライバがサポートされます。HP-UXフレームワークはHP-UXバージョン10.xをサポートします。

準備

VXI*plug&play*を使い始める前に、下記の手順を実行しておく必要があります。

1. インタフェース(GPIBまたはVXI)をインストールします。
2. VISA をインストールします。Agilent インタフェース・カードを使用している場合、VEEに付属しているVISAを使ってください。詳細については、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。他のインタフェース・カードを使用している場合、カードに付属しているVISAをインストールしてください。
3. 各ハードウェア・インタフェースに合わせてVISA を構成します。Agilentインタフェース・カードを使用している場合、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*に記載された手順に従ってください。他のインタフェース・カードを使用している場合、インタフェース・メーカが指定する方法でVISAを構成してください。

注記

VISA(Virtual Instrument Software Architecture)は、機器制御のためにVXI*plug&play*ドライバが使用するI/Oライブラリです。VISAはVXI*plug&play*の動作に必須であり、VXI*plug&play*ドライバが使用するVISA関数呼出しを提供します。

必要なもの

VEEでは各VXI*plug&play*ドライバに対して下記の4つのファイルが必要です。

- ライブラリ・ファイル
- 関数パネル・ファイル
- ヘッダ・ファイル
- ヘルプ・ファイル

VXI*plug&play*ドライバをインストールすると、必ずこれらのファイルがインストールされます。他のファイルも同時にインストールされます。

注記

すべてのVXI*plug&play*ドライバがすべてのフレームワーク(プラットフォーム)をサポートするとは限りません。また、VISAの一部のバージョンは一部のフレームワークでサポートされない可能性があります。ベンダに確認してください。

VXI*plug&play*ドライバ・ソフトウェアのインストール

各ドライバに必要なファイル群をインストールするには、機器メーカがドライバに添付している説明を参照してください。

ファイルの位置(WIN95およびWINNTフレームワーク)

VXI*plug&play*のファイルは、`WIN95\`、`WIN98\`、`WINNT\`のいずれかのディレクトリにあります。この位置は、VISAのインストール時にレジストリに記憶されたルート・ドライブおよびディレクトリを基準とした相対指定です。ルート・ドライブおよびディレクトリのデフォルト値は`C:\VXIPNP`です。

表2-2に、VEEに必要なVXI*plug&play*ドライバ・ファイルを示します。

表2-2. WIN95およびWINNTフレームワークのドライバ・ファイルの位置

ファイル名[a]	位置	目的
*PREFIX*_32.DLL	BIN	機器ドライバ・ライブラリ
PREFIX.FP	*PREFIX*	機器ドライバ関数パネル・ファイル
PREFIX.H	INCLUDE	機器ドライバ・ヘッダ・ファイル
PREFIX.HLP	*PREFIX*	機器ドライバ・ヘルプ・ファイル

a. PREFIXはHPE1410などの機器名を表します。

ファイルの位置 (HP-UXフレームワーク)

VXI*plug&play*のファイルは、vxipnp/hpux/ディレクトリにあります。この位置は、環境変数VXIPNPPATHで示されるルート・ディレクトリを基準とした相対指定です。この環境変数はデフォルトでは/optに設定されるので、通常のディレクトリは/opt/vxipnp/hpux/となります。

表2-3に、VEEに必要なVXI*plug&play*ドライバ・ファイルを示します。

表2-3. HP-UXフレームワークのドライバ・ファイルの位置

ファイル名[a]	位置	目的
PREFIX.sl	bin	機器ドライバ・ライブラリ
PREFIX.fp	*PREFIX*	機器ドライバ関数パネル・ファイル
PREFIX.h	include	機器ドライバ・ヘッダ・ファイル
PREFIX.hlp	*PREFIX*	機器ドライバ・ヘルプ・ファイル

a. *PREFIX*はHPE1410などの機器名を表します。

用語一覧

VXI*plug&play*ドライバの使用方法は、VEEの他の種類のI/Oとは異なります。各部品の間の関係を以下に説明します。

- VEEプログラムはVXI*plug&play*関数を呼び出します。
- 関数(関数パネルで設定可能なパラメータを持つ場合もある)はVXI*plug&play*ドライバの一部です。関数はVISAソフトウェアを通じて機器と通信します。
- 機器はVISAを通じて関数のパラメータにデータを返します。

VXIplug&playサンプル・プログラム

図2-4に示すのは、To/From VXIplug&playオブジェクトを使ってHP E1410Aマルチメータの電圧測定を始動し、測定値を取得するサンプル・プログラムです。

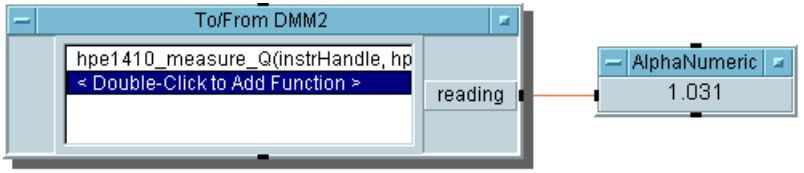


図2-4. To/From VXIplug&playドライバ・オブジェクトの使用

詳細情報

VXI*plug&play*を使用するためにVEEを**構成**する方法については、第3章「機器の構成」を参照してください。VEEでVXI*plug&play*を使用する方法の詳細については、第7章「VXIplug&playドライバの使用」を参照してください。

パネル・ドライバおよびコンポーネント・ドライバについて

`Panel Driver`オブジェクトと`Component Driver`オブジェクトが使用できるのは、サポートのための**ドライバ・ファイル**が用意されている特定の機器に対してだけです。HP-UX版VEEのインストール手順では、使用可能なすべてのドライバ・ファイルがシステム・ディスクに自動的にコピーされます。Windows 95/98およびWindows NT版VEEのインストール手順では、どのドライバをインストールするかを選択できます。第3章「機器の構成」で、機器に合わせて適切なドライバ・ファイルを選択して構成する方法について説明しています。このほか、I/Oライブラリもインストールしておく必要があります。手順については、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。

パネル・ドライバ

VEEの`Panel Driver`には2つの目的があります。

- すべての機器機能の設定を指定する測定ステートを定義します。`Panel Driver`が動作すると、対応する物理機器が`Panel Driver`の設定に合わせて自動的にプログラムされます。
- 機器を対話的に制御するための操作パネルとして動作します。これはプログラムの開発やデバッグの段階で役立ちます。また、物理フロントパネルを持たない機器を操作するのに便利です。

図2-1に示すように、Panel Driverのオープン・ビューには対応する物理機器の操作パネルのグラフィックスが表示されます。物理機器がコンピュータに正しく接続されていれば、操作パネルのグラフィックスのフィールドをクリックすることにより、機器を制御できます。また、数値表示やXY表示をクリックすることにより、測定を実行して結果を表示することもできます。

機器がコンピュータに接続されていなくても、パネルのグラフィックスを使って測定ステートを定義することは可能です。これを利用すれば、機器の購入前や機器が別の場所にある場合でもプログラムを開発できるので便利です。

例えば、HP 3325Bファンクション・ジェネレータをプログラムして下記の2つの出力信号を供給させたいとします。

1. 周波数20 kHz、振幅20 mV rmsの方形波

2. 周波数 50 kHz、振幅50 mV rmsの正弦波

図2-5に示すのが、上記の信号を生成するための2個のPanel Driverです。

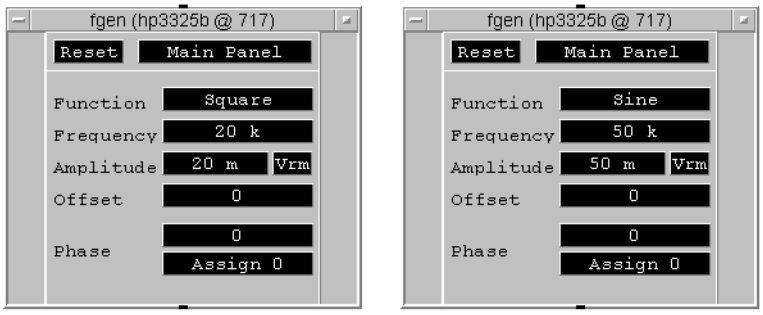


図2-5. 2個のHP3325B用Panel Driver

コンポーネント・ドライバ

機器ドライバでは、個々の機器機能と測定値を**コンポーネント**と呼びます。コンポーネントはドライバ内部の変数のようなもので、機能設定や測定値を記録する役割を果たします。Component Driverとは、指定したコンポーネントだけを入出力端子として読み書きするオブジェクトのことです。これに対してPanel Driverは、多数またはすべてのコンポーネントの値を自動的に書き込みます。

Component Driverは、プログラムの実行速度を向上させるために用意されています。Component DriverがPanel Driverに比べて優れているのは、実行速度が速い点だけです。一般に、プログラムの実行速度に最も影響があるのは、機器制御オブジェクトが反復子オブジェクトに接続され、何回も実行される場合です。この場合、変化するのは1つまたは2つのコンポーネントだけであることが珍しくありません。このような場合にComponent Driverが真価を発揮します。

Component Driverによってどれだけ実行速度が向上するかは、状況によって異なります。向上率は主に使用するドライバ・ファイルに左右されます。実際にどれだけ実行速度が向上するかを正確に予測するのは困難です。

例えば、HP 3325Bファンクション・ジェネレータをプログラムして下記のことをさせたいとします。

1. 初期周波数が10 kHzで振幅がオペレータ入力によって決まる正弦波を出力

2. 10 kHzから1 MHzまで、5ステップで1桁の割合で周波数出力を掃引

この場合、初期セットアップにはPanel Driverを使い、出力周波数を繰り返し設定するのにはComponent Driverを使うのが適切です。図2-6にこのためのプログラムを示します。

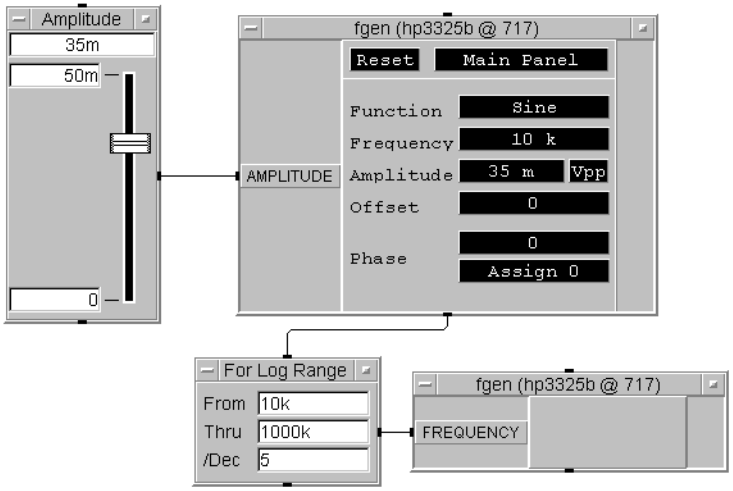


図2-6. Panel DriverとComponent Driverの組合わせ

詳細情報

VEEの構成方法については、第3章「機器の構成」を参照してください。`Panel Driver`および`Component Driver`オブジェクトの使用法については、第6章「パネル・ドライバおよびコンポーネント・ドライバ・オブジェクトの使用」を参照してください。

レジスタ・ベースVXIデバイスのサポート

機器制御オブジェクトを使ってVXIバックプレーン上のVXIデバイスと直接通信する場合、デバイスが**メッセージ・ベース**と**レジスタ・ベース**のどちらなのかを知る必要があります。メッセージ・ベースのデバイスとVEEで通信するには、SCPI(Standard Commands for Programmable Instruments)メッセージを使います。

VEEにはまた、HPとAgilentのほとんどのレジスタ・ベース・デバイスに対するI-SCPI(Interpreted SCPI)サポートも用意されています。I-SCPIドライバを使うと、レジスタ・ベースのデバイスに対してメッセージ・ベースのデバイスと同じ方法で通信できます。すなわち、特定のレジスタ・ベース・デバイス用のI-SCPIドライバが用意されていれば、標準のSCPIメッセージを使ってVEEプログラムでそのデバイスと通信できます。相手のレジスタ・ベース・デバイス用のI-SCPIドライバが用意されていない場合、VEEで通信するためには相手のレジスタに直接アクセスする必要があります。

I-SCPIドライバを使えば、任意の機器制御オブジェクトが使用できて便利です。プログラミングを容易にするにはPanel Driverを使い、実行速度を高速にするにはDirect I/OでSCPIコマンドを使用します。SCPIメッセージを使ってレジスタ・ベース・デバイスと通信するプログラムをVEEで作成した場合、必要なI-SCPIドライバが存在しなければ警告が出ます。この場合、Direct I/OまたはMultiInstrument Direct I/Oを使ってデバイスのレジスタに直接アクセスする必要があります。

3

機器の構成

機器の構成

この章では、下記の手段で機器と通信するためのVEEの構成方法について説明します。

1. Direct I/Oオブジェクト経由(機器ドライバ不要)
2. To/From VXIplug&playオブジェクト経由でVXI*plug&play*ドライバを使用
3. Panel DriverまたはComponent Driverオブジェクト経由でAgilentパネル・ドライバ("ID")を使用
4. ODAS PCプラグイン・カード・ドライバ経由でFormulaオブジェクトを使用
   VEE 6.0はODAS(Open Data Acquisition Standard)準拠のソフトウェア・ドライバを持つPCプラグイン・カードをサポートします。

VEEのInstrument Managerダイアログには、これらの機器制御オブジェクトの選択と構成のための統一的な手段が用意されています。

VEEで機器と通信するためには、あらかじめAgilent I/Oライブラリをインストールしておく必要があります。手順については、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。Panel Driver、Component Driver、Direct I/Oの各オブジェクトには、Agilent SICLライブラリが必要です。To/From VXIplug&playオブジェクトにはVISAライブラリが必要です。

Panel DriverまたはComponent Driverオブジェクトを使用するには、対応するパネル・ドライバをインストールする必要があります。HP-UX版VEEでは、VEEのインストール時に自動的にドライバもインストールされます。Windows版VEEでは、VEEのインストール時に必要な機器ドライバを選択してインストールすることができます(Direct I/Oオブジェクトには機器ドライバは不要です)。

多くのVXI機器にはメーカ製のVXI*plug&play*ドライバが付属しています。To/From VXIplug&playオブジェクトを使用するには、ドライバの説明書に従って、対応するVXIplug&playドライバ・ファイルをインストールする必要があります。VXI*plug&play*ドライバの詳細については、第7章「VXIplug&playドライバの使用」を参照してください。

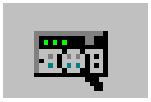
多くのPC機器にはメーカ製のODAS PCPIカード・ドライバが付属しています。ODAS PCPIカード・ドライバ・オブジェクトを使用するには、ドライバの説明書に従って、対応するODAS PCPIドライバ・ファイルをインストールする必要があります。

Instrument Managerの使用

このセクションでは、Instrument Managerと構成ダイアログ・ボックスを使ってVEEで機器を検索し、構成する方法の概要を説明します。いくつかの例を示します。多くのアプリケーションでは、大多数のパラメータがデフォルト値のままでかまいません。これらのダイアログ・ボックスの構成フィールドの詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。

概要

機器を構成するには、I/O ⇒ Instrument Managerを選択するか、ツールバーのInstrument Managerボタンをクリックします。

ボタンの外観: 

Instrument Managerダイアログ・ボックスが表示されます。機器を検索して追加するまでは、図3-1のようにダイアログ・ボックスは空です。

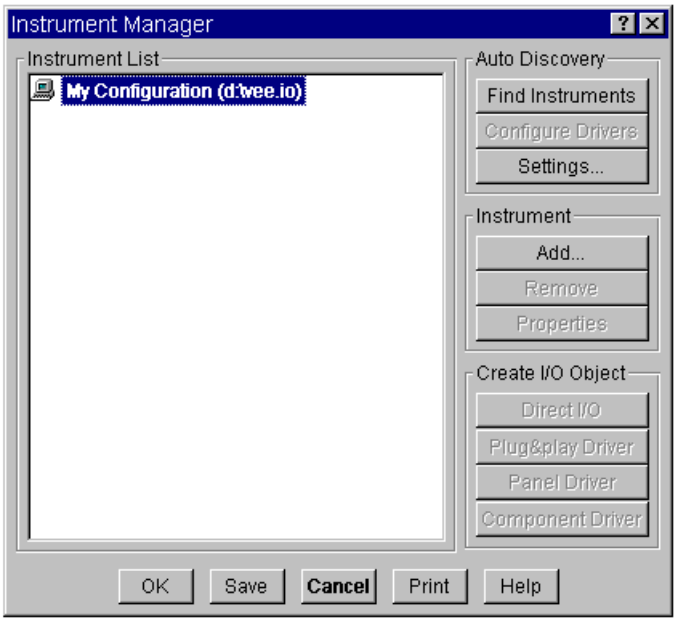


図3-1. Instrument Managerダイアログ・ボックス

Instrument Managerには4つのセクションがあります。

- Auto Discoveryボタンは、機器を検索してそのドライバを構成するためのものです。Find Instrumentsボタンをクリックすると、構成済みのGPIBおよびVXI機器が自動的にアップデートされ、コンピュータに接続されている他のGPIBおよびVXI機器が表示されます。Settingsボタンをクリックすると、次のセクションで説明するAuto Discovery Settingsダイアログ・ボックスが表示されます。

- Instrument Listには、現在構成済みの機器が表示されます。この構成は、I/O構成ファイル(詳細については189ページ「I/O構成ファイル」参照)で定義されます。デフォルトの構成は空白(空)です。

- Instrument ボタンは、機器構成を変更するためのものです。Instrument ボタンの動作については、この章のあとのほうで詳しく説明します。

■ Create I/O Objectボタンは、Direct I/O、Plug&play Driver、Panel Driver、Component Driverの各オブジェクトを選択してプログラムに配置するために使います。

Auto Discovery

Auto Discoveryエリアには、Find Instruments、Configure Drivers、Settingsの3つのボタンがあります。

■ Find Instrumentsボタンは、既存のGPIBおよびVXI機器の構成をアップデートし、コンピュータに接続されている未構成のGPIBおよびVXI機器をInstrument Listに追加します。Find InstrumentsボタンではシリアルおよびインタフェースもInstrument Listに追加されますが、それらに接続された機器は表示されません。

■ Configure Driversボタンは、検索されてInstrument Listに表示された機器のドライバ構成のために使います。

■ Settingsボタンは、機器とドライバを構成する方法を決めるために使います。

Instrument ListでMy Configurationが強調表示された状態でFind Instrumentsをクリックすると、既存のすべてのGPIBおよびVXI機器構成がアップデートされ、未構成のGPIBおよびVXI機器がリストに追加されます。見つかった機器のうち電源が入っているものに対してはライブ・モードがオンになります(構成済みの機器が見つからない場合でも、ライブ・モード設定がオンからオフに切り替わることはありません)。

次にSettingsをクリックすると、機器とドライバの検出と構成の方法を制御するダイアログ・ボックスが表示されます。このボックスには、Find InstrumentsとInstrument Identificationの2つのセクションが存在します。

Find Instrumentsセクションには下記の2個のラジオ・ボタンがあります。

- ❑ Detect only
- ❑ Detect, identify, and configure drivers for each instrument.

"Detect only"をチェックすると、Find Instrumentsボタンをクリックしたときに、すべてのライブ・バス・アドレスが検出されます。"Detect, identify, and configure drivers for all instruments"をチェックすると、すべてのライブ・バス・アドレスが検出され、

検出されたすべての機器に対して"*IDN?"が送信され、各機器に対するドライバの構成が行われます。

下のセクションは、Configure Driversボタンを制御します。"Ask before sending "*IDN?" to each instrument?"ボックスがチェックされていると、各ドライバの構成の前にVEEはいったん停止し、構成を行うかどうかをユーザに問い合わせます。このボックスがチェックされていないと、すべてのドライバが自動的に構成されます。

Instrument List

コンピュータに接続された機器がFind Instrumentsで見つかったばあい、Instrument Managerは図3-2のようになります。この例では、シリアル・インタフェースがFind Instrumentsで見つかっていますが、それに接続された機器は表示されていません。新規に見つかった機器には、"newInstrument"、"newInstrument1"といった名前が付きます。もっとわかりやすい名前を付ける方法についてはあとで説明します。

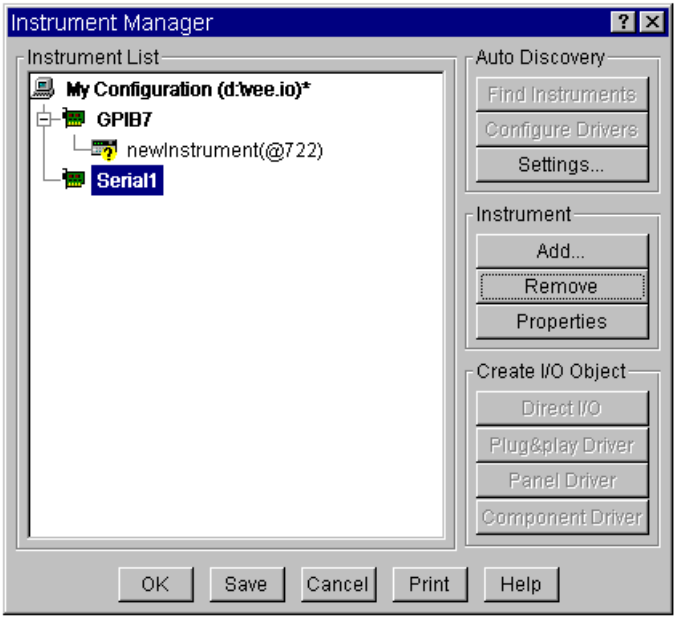


図3-2. Instrument List

Instrument Managerを使用するには、GPIB7インタフェースを選択します。インタフェースが強調表示され、Propertiesボタンがアクティブになります。GPIB7の前にある[-]アイコンをクリックすると、その下の選択肢が畳み込まれます。図3-3は畳み込んだ状態の構成を示します。

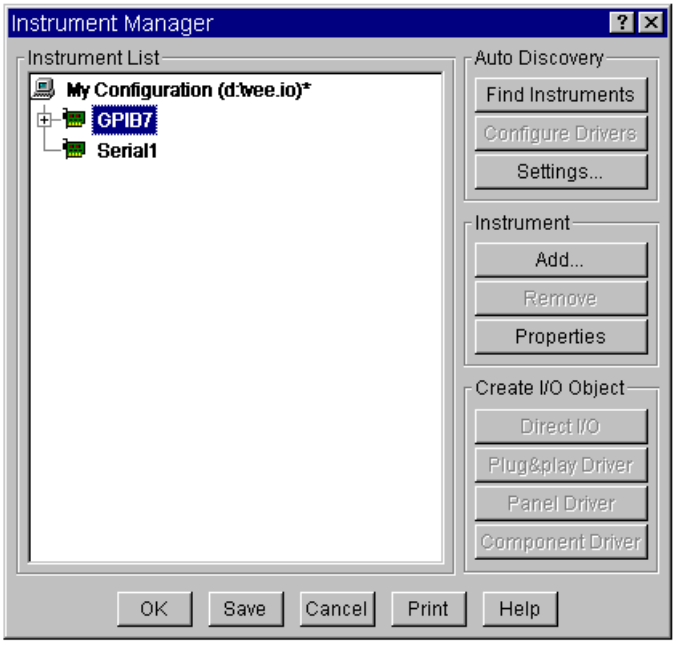


図3-3. GPIB7インタフェース構成を畳み込んだところ

もう一度選択肢を展開するには、GPIB7の前にある[+]アイコンをクリックします(ツリー全体を展開するには、My configurationを選択して*キーを押します)。次に、newInstrument@722またはその前の機器アイコンをクリックして強調表示します。図3-4にウィンドウのようすを示します。

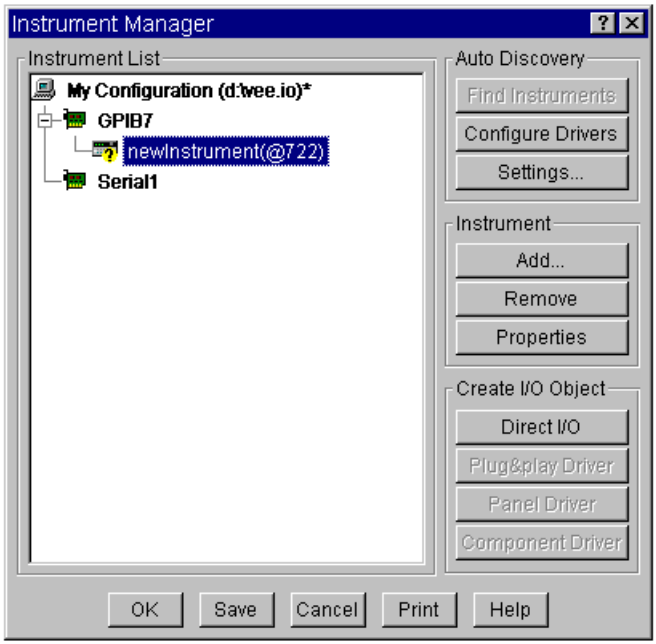


図3-4. 構成する機器の選択

機器構成

Instrumentの下のボタンが、Propertiesを初めすべてアクティブになっています。これにより、既存の機器の構成を削除したり、編集したり、手動で構成したりすることができます。また、新しい機器をリストに追加することもできます。

また、Create I/O Objectの下のボタンの1つがアクティブになっているはずです。これにより、この機器に対するDirect I/Oオブジェクトを選択して配置することができます。他の機器構成では、インストールされているドライバに応じて、Plug&play Driver、Panel Driver、Component Driverの各ボタンがこの時点でアクティブになります。

Configure Driversボタンをクリックすると機器構成がアップデートされます。Identify Instrumentダイアログ・ボックスが表示され、*IDN?(識別)メッセージを機器に送るかどうかを尋ねてきます。図3-5にこのダイアログ・ボックスを示します。

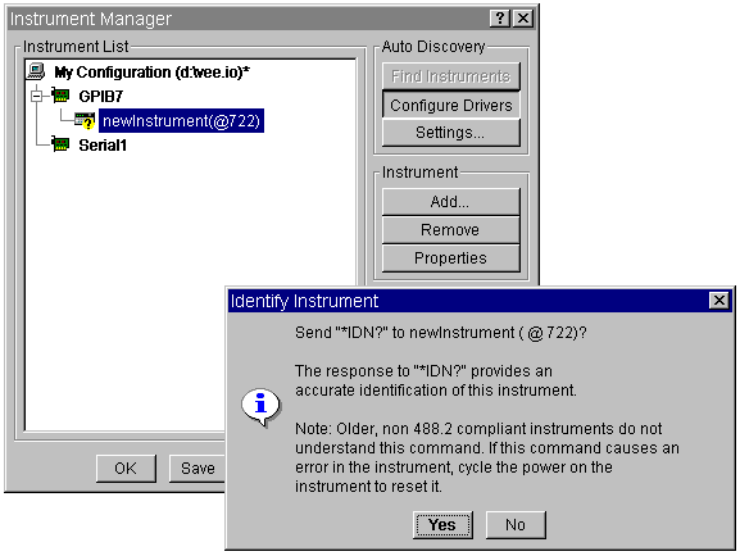


図3-5. 機器構成のアップデート

OKをクリックします。GPIBインタフェースに接続された機器がオンになっている場合、機器が応答します。この例では、機器はHP 34401Aで、オンになっています。図3-6に、この時点での`Instrument List`を示します。

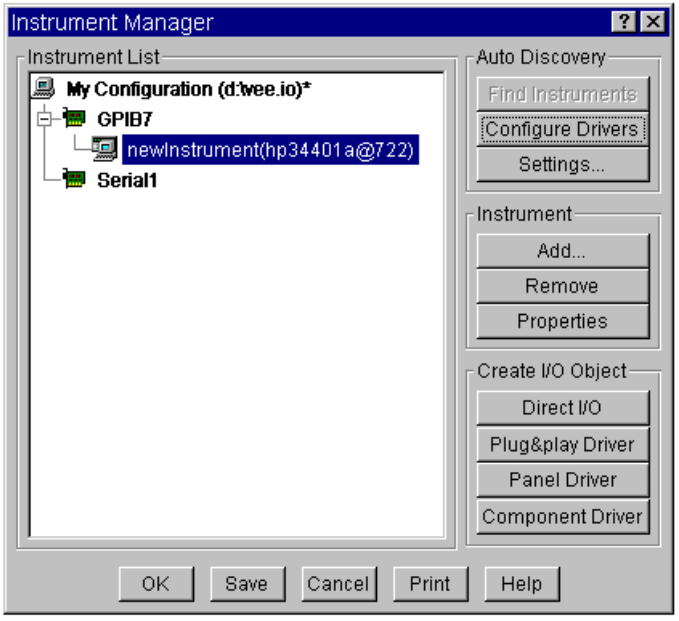


図3-6. ドライバ構成後のInstrument List

下記の2つの点が変化していることに注目してください。

1. 機器識別子が`newInstrument(hp34401a@722)`に変わっています。

2. `newInstrument(hp34401a@722)`の前の機器アイコンが、コンピュータに接続されていることを示すアイコンに変わっています。

(機器の電源が投入されていない場合、識別子とアイコンは変化しません)

機器のリネーム

機器が識別されたら、意味のある名前でInstrument Listに表示させることができます。このためには`Properties`ボタンをクリックします。`Properties`ダイアログ・ボックスが表示されたら、Nameフィールドをクリックし、新しい名前を入力します。図3-7は、HP 34401Aに対して"newInstrument"の代わりに"dmm"という名前を入力したところです。

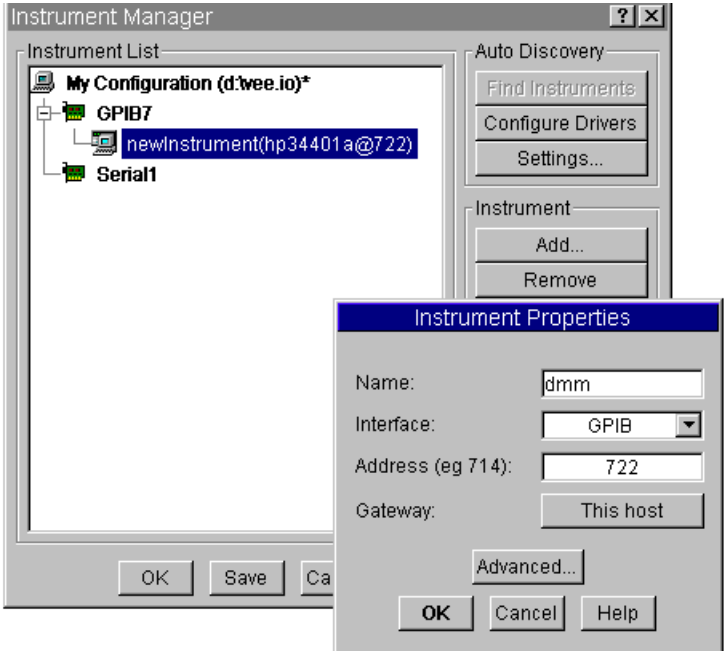


図3-7. 機器名の変更

OKをクリックすると変更が確定します。図3-8は、HP 34401Aの新しい名前が表示されたInstrument Listです。

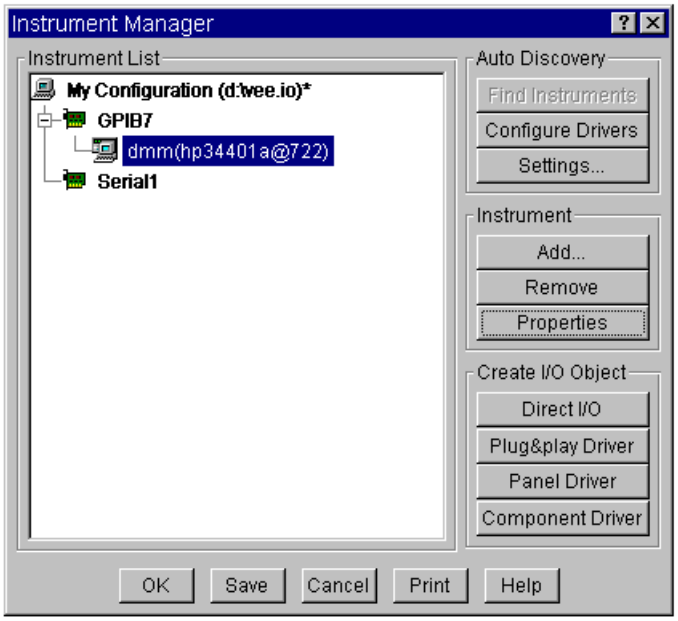


図3-8. リネームされた機器

機器構成の追加

機器を追加するには、Add...ボタンをクリックします。図3-9のようにInstrument Propertiesダイアログ・ボックスが表示されます。

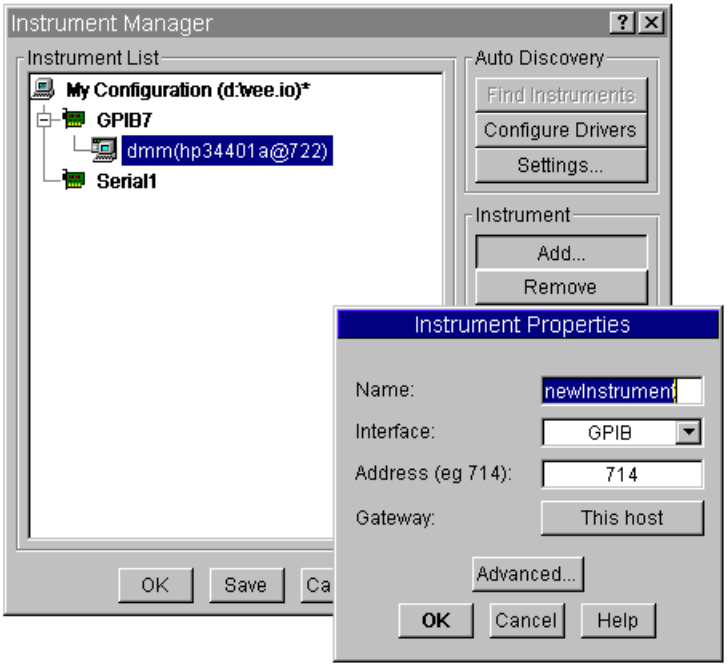


図3-9. 機器の追加

デフォルトでは、新しい構成にはnewInstrumentという名前が表示されます。新しい名前(dmm2など)を入力することもできます。Interfaceフィールドは GPIBが選択されたままにしておきます(インタフェースのタイプを変更したい場合、GPIBの右にある矢印をクリックすると、ドロップダウン・リストが表示されます)。次に、Addressフィールドをクリックし、アドレスを723に変更します。図3-10は、これらの変更を行ったInstrument Propertiesダイアログ・ボックスを示します。

注記

ダイアログ・ボックスのフィールドの間を移動するには、目的のフィールドをクリックするか、**Tab**キーを使います。**Enter**または**Return**を押すと、ダイアログ・ボックスが終了します。

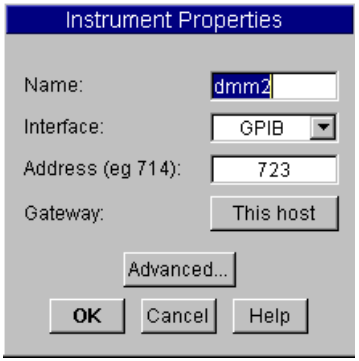


図3-10. NameフィールドとAddressフィールドの変更

次に、Advanced...ボタンをクリックしてAdvanced Instrument Propertiesダイアログ・ボックスを表示させます(図3-11)。

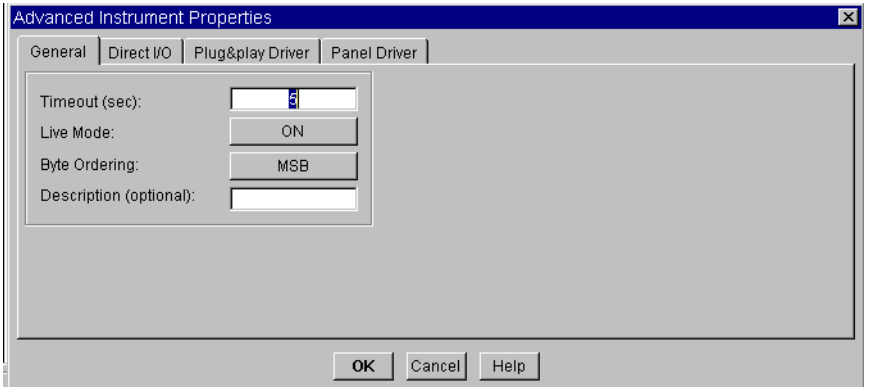


図3-11. Advanced Instrument Propertiesダイアログ・ボックス

このダイアログ・ボックスのGeneralタブを使うと、ライブ・モードをオン/オフするタイムアウト値を指定したり、バイト順序を選択したり、説明を追加したりすることができます。Descriptionフィールドをクリックし、hp34401aと入力します。

注記

Instrument Propertiesダイアログ・ボックスとAdvanced Instrument Propertiesダイアログ・ボックスの各フィールドの詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。

Advanced Instrument Propertiesダイアログ・ボックスに表示されるタブとフィールドは、選択したインタフェースに依存します。

次に、Panel Driverタブを選択して、図3-12のダイアログ・ボックスを表示させます。

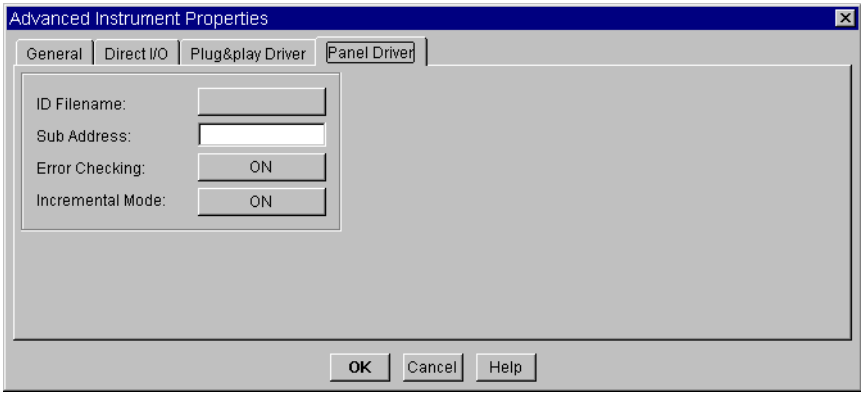


図3-12. Panel Driverタブ

ID Filenameフィールドをクリックします。機器ドライバ・ファイルを選択するためのプロンプトが出ます(図3-13にWindows版のダイアログ・ボックスを示します。HP-UX版のダイアログはこれとは異なりますが、やはりファイルの選択が可能です)。

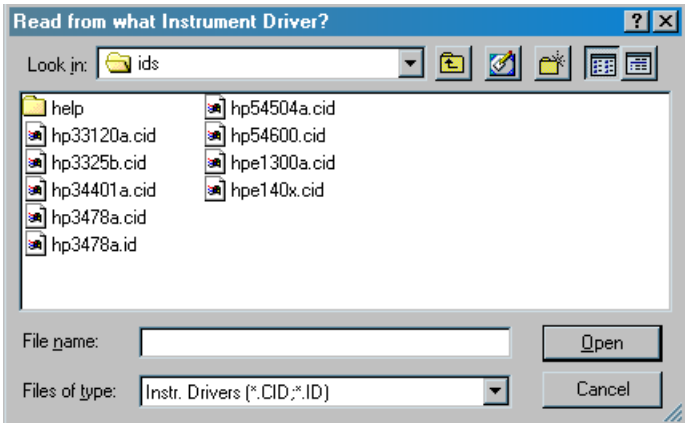


図3-13. 機器ドライバ・ファイルの選択

hp34401a.cidをダブルクリックして、図3-14のようにファイルを選択します。

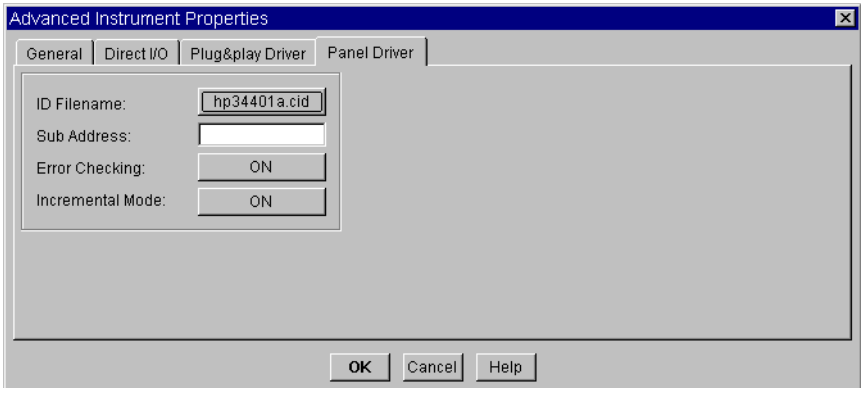


図3-14. 選択されたIDファイル名

各ダイアログ・ボックスでOKをクリックして、図3-15のようにInstrument Managerに戻ります。

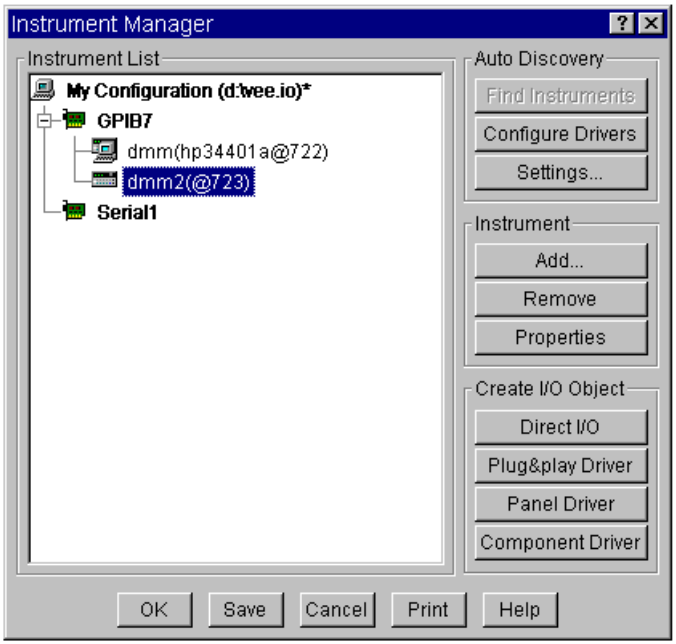


図3-15. 新しい構成

この時点で、Saveボタンをクリックすることにより新しい構成を保存できます。

パネル・ドライバまたはコンポーネント・ドライバの追加

新しい構成を保存したら、dmm2に対するPanel Driverオブジェクトまたは Component Driverオブジェクトを追加できます。I/O ⇒ Instrument Manager を選択して、図3-15のようにInstrument Managerを再表示します。dmm2(@723)が強調表示されていなければクリックして選択し、Component Driverボタンをクリックします。ワーク・エリアの適切な位置に輪郭を移動し、マウス・ボタンをクリックしてComponent Driverオブジェクトを配置します。オブジェクトは図3-16のようにアイコンとして表示されます。

図3-16. Component Driverオブジェクト

同様にして、Panel Driverボタンをクリックすれば、Panel Driverオブジェクトが表示されます。

機器構成の編集

既存の機器構成を、やはりInstrument Propertiesダイアログ・ボックスとAdvanced Instrument Propertiesダイアログ・ボックスを使って編集できます。HP 34401Aディジタル・マルチメータの構成を編集するには、Instrument Listでdmm(hp34401a@722)を選択し、Properties...ボタンをクリックします。Instrument Propertiesダイアログ・ボックスが図3-17のように表示されます。

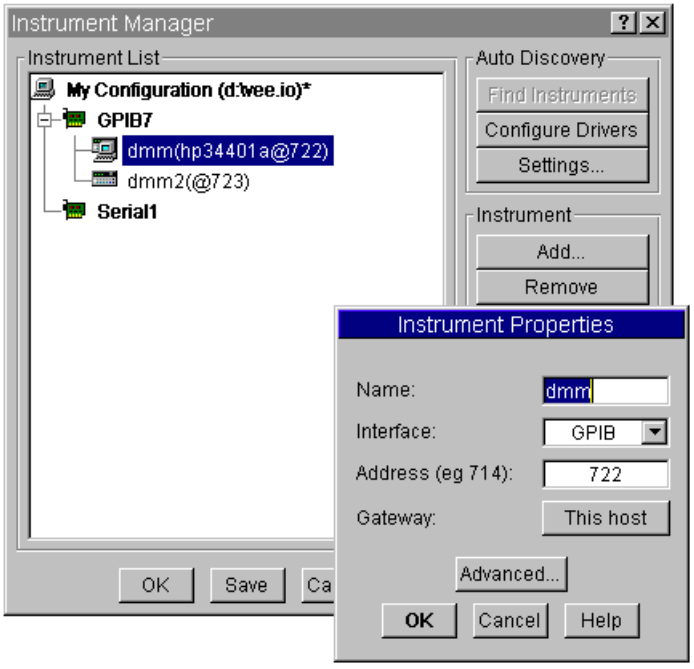


図3-17. dmm構成の編集

構成を変更するには、67ページ「機器構成の追加」で説明した方法で、プロパティ・ダイアログ・ボックスのフィールドを変更します。

インタフェース構成の編集

インタフェース構成全体を編集することにより、複数の機器を変更することができます。このためには、Instrument Listでインタフェースを選択し、Propertiesボタンをクリックします。例えば、GPIB7を選択し、Propertiesボタンをクリックすると図3-18の画面が表示されます。

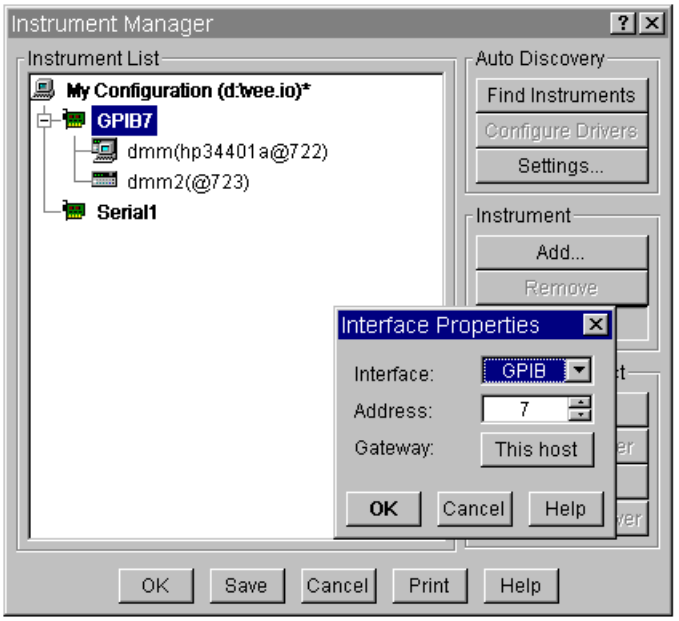


図3-18. GPIB7構成の編集

ここでは、GPIB7の元の構成をサンプルで使用するため、Cancelを押して変更を行わずに終了します。

注記

Interface Propertiesダイアログ・ボックスでは、インタフェース・タイプをGPIBからVXIに変更したり、アドレスを7から未使用の他の論理ユニットに変更したり、LANゲートウェイを構成したりできます。この変更は、現在GPIB7の下にあるすべての機器(dmm、dmm2など)に影響します。詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。

Direct I/Oオブジェクトの構成

下の例は、Direct I/Oオブジェクトの構成方法を示します。この例では、論理ユニット1(COM1)のシリアル機器をダイレクトI/O用に構成します。

1. My Configurationを選択します。
2. Find Instrumentsをクリックします。
3. Find Instrumentsが終了したら、Serial1を選択し、Add...をクリックします。
4. 図3-19のダイアログ・ボックスが表示されるはずです。

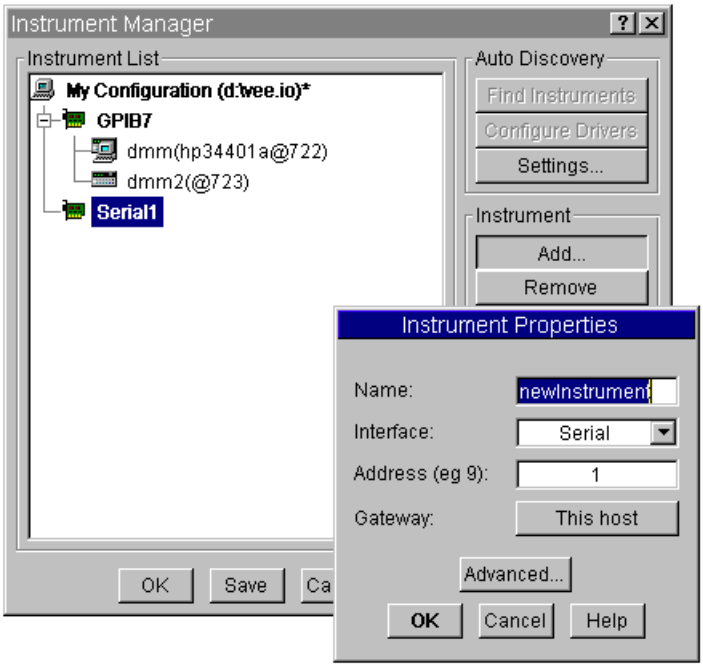


図3-19. シリアル・デバイスの構成

Instrument Propertiesダイアログ・ボックスで、新しい機器の名前とアドレスを選択できます。名前をSerial1に変更します。

Advanced...をクリックして、図3-20のAdvanced Instrument Propertiesダイアログ・ボックスを表示します。必要なタブは2つあります。

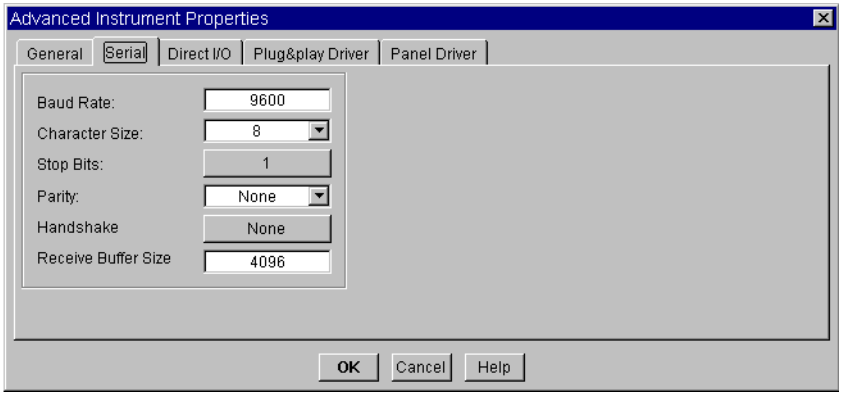


図3-20. Serialタブ

Serialタブでは、ボーレートなどのシリアル・パラメータを指定します。パラメータとフィールドの詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。多くのアプリケーションではデフォルトを使用できます。

図3-21に示すDirect I/Oタブでは、ダイレクトI/Oのパラメータ(EOLシーケンスなど)を指定します。ほとんどのアプリケーションではデフォルトを使用できます。

注記

Direct I/Oタブに表示されるフィールドの選択は、選択したインタフェースに依存します。またVXIの場合のみ、A16 SpaceとA24/A32 Spaceの2つのタブが余分に存在します。

これらのタブでは、Direct I/OオブジェクトのWRITEまたはREADトランザクション用にVXIデバイスのレジスタを構成します。各タブのパラメータとフィールドの詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。

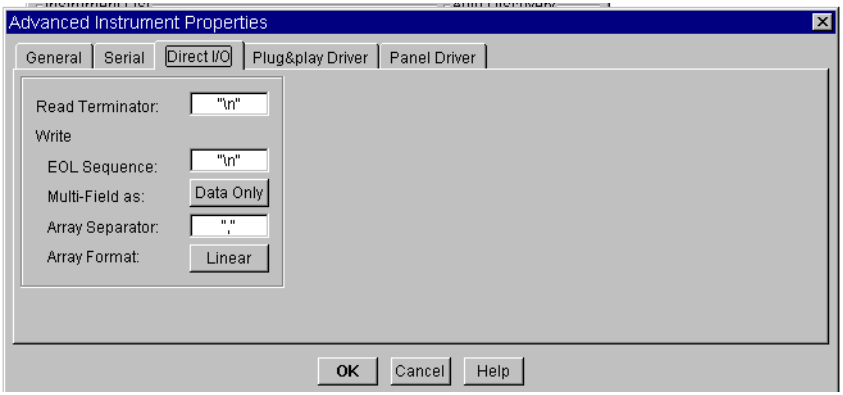


図3-21. Direct I/Oタブ

各ダイアログ・ボックスでOK(変更しない場合はCancel) をクリックして、Instrument Managerに戻ります。この例では、Serial1インタフェースの下に新しい機器Serial1が追加されています。ワーク・エリアにDirect I/Oオブジェクトを追加するには、Direct I/Oボタンをクリックし、オブジェクトを配置してから、もう一度クリックして図3-22の画面を表示させます。

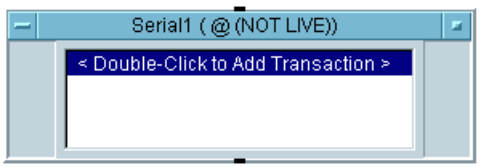


図3-22. Direct I/Oオブジェクト

注記

Direct I/Oオブジェクトはトランザクション・ベースのI/Oを使用して機器と通信し、機器ドライバは使用しません。詳細については、第4章「トランザクションI/Oの使用」を参照してください。

VXI*plug&play*ドライバの構成

To/From VXIplug&playオブジェクトの構成手順は、Panel Driver、Component Driver、Direct I/Oの各オブジェクトの場合とよく似ています。ただし、適切なVXI*plug&play*ドライバ・ファイルをあらかじめインストールしておく必要があります。47ページ「VXIplug&playドライバ・ソフトウェアのインストール」を参照してください。

注記

Windowsオペレーティング・システムを使用している場合、VISAアシスタント・ユーティリティを使ってVXI*plug&play*ドライバに関する有用な情報を得ることができます。この情報を使って、VXI*plug&play*ドライバ構成に必要な正しいアドレスを知ることができます。VISAアシスタントはWindowsのStartメニューのProgram Files ⇒ Agilent I/O Libraries ⇒ VISA Assistantにあります。

例えば、HP E1410A 6.5桁VXIマルチメータのVXI*plug&play*構成を追加する場合、I/O ⇒ Instrument Managerを選択し、Add...をクリックします。Instrument Propertiesダイアログ・ボックスが表示されます。名前をvxiDeviceに変更し、インタフェース・タイプとしてVXIを選択します(図3-23)。

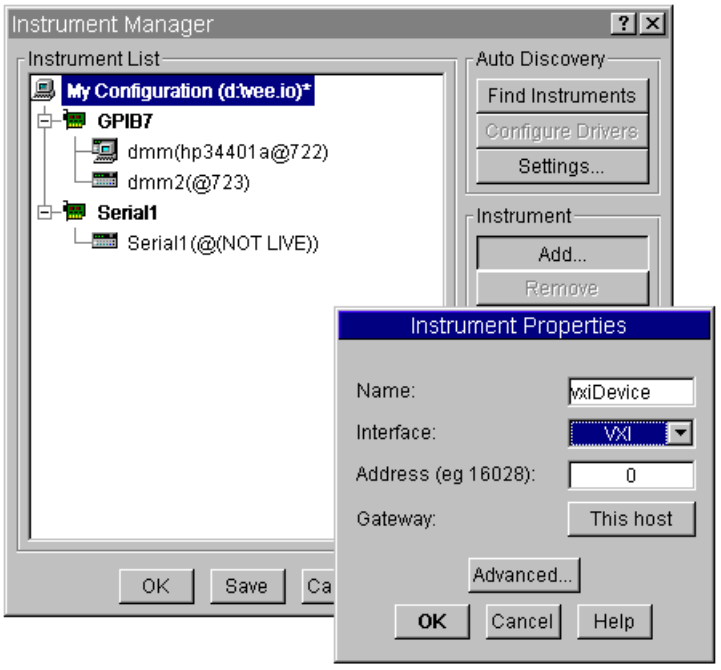


図3-23. VXIデバイスの追加

VXI*plug&play*ドライバではAddressフィールドは用いられません。Advanced...をクリックしてAdvanced Instrument Propertiesダイアログ・ボックスを表示し、Plug&play Driverタブを選択します。

次に、Plug&play Driver Nameドロップダウン・リストからHPE1410という名前のドライバを選択します(図3-24)。選択できるVXI*plug&play*ドライバは、あらかじめインストールされているものだけです。インストール手順については47ページ「VXIplug&playドライバ・ソフトウェアのインストール」を参照してください。

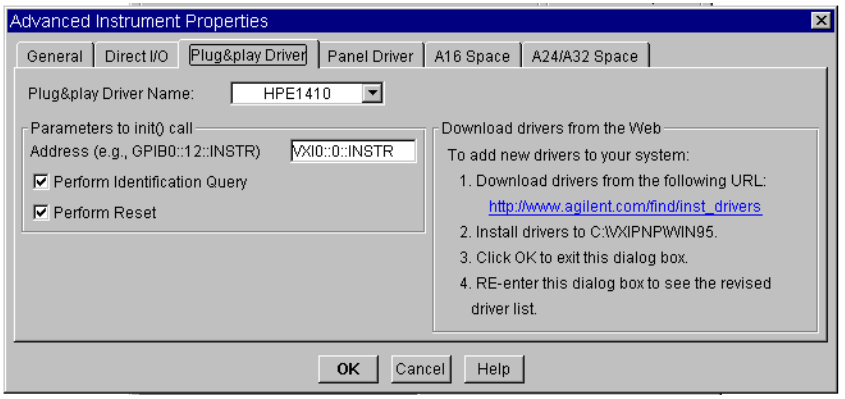


図3-24. Plug&play Driverタブ

デフォルトでは、Addressフィールドには`VXI0::0::INSTR`が表示されます。これは機器のVXI論理アドレスが0であると仮定しています。一般には、正しい論理アドレスを指定する必要があります。例えば、HP E1410Aの論理アドレスが24の場合、Addressフィールドを`VXI0::24::INSTR`に変更します。`Plug&play Driver`タブのフィールドの詳細については、85ページ「プロパティ・ダイアログ・ボックスの詳細」を参照してください。

注記

VXI*plug&play*ドライバの構成に関係するのは`Plug&play Driver`タブだけです。`General`、`Direct I/O`、`Panel Driver`、`A16 Space`、`A24/A32 Space`の各タブはVXI*plug&play*構成には無関係です。

例えば、VXI*plug&play*デバイスは常にライブであると見なされるので、`General`タブの`Live Mode`設定は無視されます。

機器の構成が済んだら、各ダイアログ・ボックスでOKをクリックしてInstrument Managerに戻ります。追加された機器が図3-25のように表示されます。

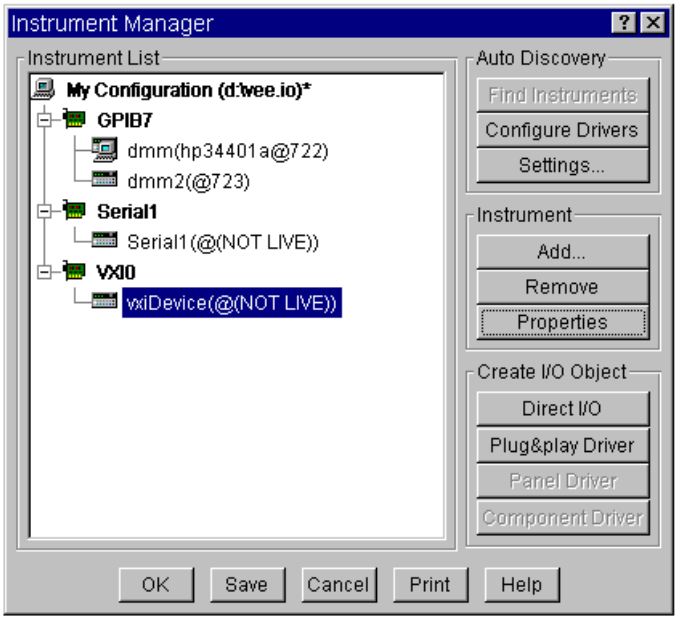


図3-25. VXI構成

Plug&play Driverボタンをクリックして、To/From VXIplug&playオブジェクトを図3-26のように追加します。

図3-26. To/From VXIplug&playオブジェクト

To/From VXIplug&playオブジェクトの使用法については、235ページ「To/From VXIplug&playオブジェクトの使用」を参照してください。

PCプラグイン・カードの構成

VEEでは、ODAS(Open Data Acquisition Standard)互換のPCプラグイン・カードがActiveXオートメーションによってサポートされます。カードのインストールと構成の手順については、メーカの説明書を参照してください。

Instrument ManagerでFind Instrumentsをクリックします。PCプラグイン・ハードウェアおよびソフトウェアが正しく構成されていれば、図3-27のような構成が表示されます。

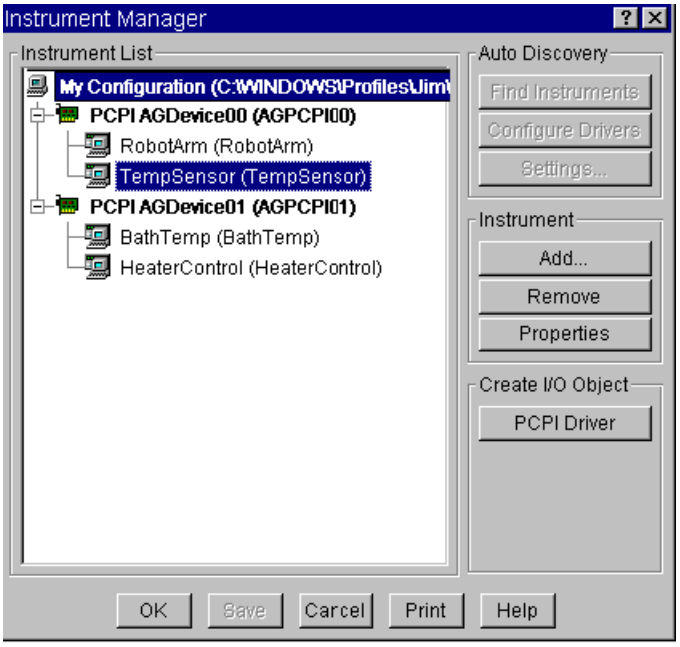


図3-27. PCプラグイン構成の例

`PCPI Driver`ボタンをクリックすると、図3-28のようなFormulaオブジェクトが表示されます。

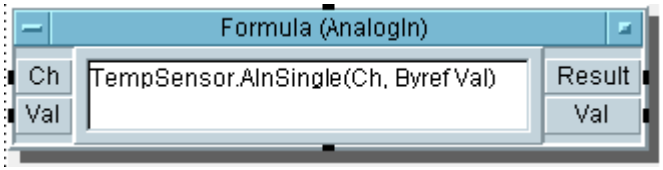


図3-28. VEEが作成したFormulaオブジェクト

このFormulaオブジェクトには、AInSingleメソッドへの呼出しが入っています。VEEはこのメソッドのためのオブジェクト(この例では'TempSensor')を自動的に作成するので、CreateObject()を呼び出して作成する必要はありません。このオブジェクトがサポートするすべてのプロパティとメソッドは、ActiveXオブジェクト下のFunction & Object browserに表示されます。

プロパティ・ダイアログ・ボックスの詳細

このセクションでは、Instrument Propertiesダイアログ・ボックス、Advanced Instrument Propertiesダイアログ・ボックスの各タブ、Interface Propertiesダイアログ・ボックスについて詳しく説明します。Instrument Managerおよびこれらのダイアログ・ボックスの使用法の概要については、58ページ「Instrument Managerの使用」を参照してください。

Instrument Propertiesダイアログ・ボックス

Instrument Propertiesダイアログ・ボックスは、Instrument Managerで機器を選択してAdd...ボタンまたはPropertiesボタンをクリックしたときに表示されます。このダイアログ・ボックスの例については図3-29を参照してください。

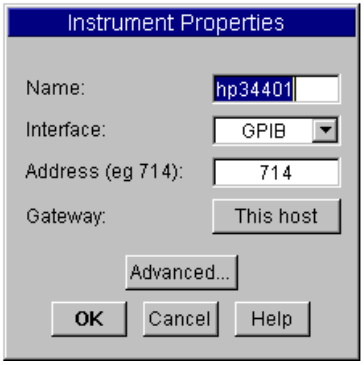


図3-29. Instrument Propertiesダイアログ・ボックス

以下の部分では各フィールドについて説明します。

Nameフィールド

VEEの機器制御オブジェクトでは、Nameフィールドで機器構成を一意に識別できる必要があります。機器のNameは、機器制御オブジェクトの各インスタンスと、そのNameに対応するすべての構成情報との間のシンボリック・リンクです。通常は、このフィールドを使ってOscilloscopeやPower_Supplyなどのわかりやすい名前を機器に付けます。

プログラムでプロパティの取得や設定を行う場合、NameはVEEの有効な変数名でなければなりません。名前の先頭は英字で、そのあとは英数字または下線文字です。

Nameは一意でなければなりません。例えば、ScopeというNameを持つ機器は1つしか構成することができません。同じ物理機器を参照する2つの異なるNameを作成することは可能ですが、同じプログラムで両方の名前をPanel DriverまたはVXI*plug&play*ドライバで使用すると問題が生じるおそれがあります。

機器のNameと、機器制御オブジェクトのタイトルに表示されるテキストとを混同しないでください。機器制御オブジェクトのデフォルト・タイトルはNameですが、タイトルはNameと独立に変更することができます。機器制御オブジェクトの特定のインスタンスのNameを知りたい場合は、機器コントロール・オブジェクトのメニュー(Direct I/O、MultiInstrument I/Oなど)でPropertiesを選択します。

注記

Nameを正しく使うことは大変重要です。このセクションで説明しているのは一般的な例に過ぎません。VEEによる名前の使用の詳細については、227ページ「名前の重要性」を参照してください。

Interfaceフィールド

Interfaceフィールドは、機器との通信に用いられるハードウェア・インタフェースのタイプ(GPIB、VXI、GPIO、Serial)を指定します。

Addressフィールド

Addressフィールドは、機器のアドレスを指定します。GPIOまたはSerialインタフェースを使用する機器の場合、アドレスはインタフェースの論理ユニットと一致します。インタフェースの論理ユニットとは、特定のインタフェースを識別するためにコンピュータが使用する番号です。

GPIBインタフェースを使用する機器の場合、アドレスは*xxyyzz*という形式です。ここで、

- *xx*は1桁または2桁のインタフェース論理ユニットです。ほとんどのGPIBインタフェースには論理ユニット7が出荷時に設定されています。
- *yy*は機器の2桁のバス・アドレスです。バス・アドレスが10より小さい場合は先頭に0を付けます。例えば、9の場合は09と指定します。
- *zz*は機器の2次アドレスです。2次アドレスは、複数のプラグイン・モジュールを使用するカードケージ形の機器で主に用いられます。CサイズVXIメインフレームのコマンド・モジュールを通じてデバイスにアクセスする場合や、BサイズVXIメインフレームのデバイスと通信する場合には2次アドレスが用いられます。

注記

この2次アドレスは、IEEE 488.1で定義された2次アドレスです。これは機器ハードウェアのインタフェース仕様の一部です。2次アドレスが必要かどうかは、機器**ハードウェア**のデザインによって決まります。2次アドレスは**ドライバ**構成とは無関係です。

Advanced Instrument Propertiesダイアログ・ボックスのSub Addressフィールドと2次アドレスとを混同しないでください。サブアドレスは**ドライバに関係する**機能であり、**まれに**しか用いられません。

VXIインタフェースを使用する機器(組込みコントローラまたはVXIバックプレーンに直接アクセスできるコントローラに接続されたもの)の場合、アドレスは*xxyyy*という形式です。ここで、

- *xx*は、組込みまたは外部コントローラのVXIバックプレーン・インタフェースの1桁または2桁の論理ユニットです。
- *yyy*はVXIデバイスの論理アドレスです。論理アドレスが100より小さい場合は先頭に0を付けます。例えば、8の場合は008と指定します。

注記

Addressフィールドを0にすると特別な意味になります。Addressフィールドを0に設定すると(すべてのインタフェースに対して)、このデバイス記述に一致する物理機器がコンピュータに接続されていないという意味になります。アドレスを0にするとLive Modeは自動的にOFFになります。

GPIBアドレスの例1. 論理ユニット7のGPIBインタフェース・カードを使ってバス・アドレス9のGPIB機器を制御する場合、機器のAddressフィールドを709に設定します。推奨される論理ユニットについては、212ページ「論理ユニットとI/Oアドレッシング」を参照してください。

GPIBアドレスの例2. 論理ユニット14のGPIBインタフェース・カードを使ってバス・アドレス12の機器を制御するには、Addressフィールドの設定を1412にします。

VXIアドレスの例1. 論理ユニット16の組込みVXIコントローラを使って論理アドレス28のVXI機器を制御するには、Addressフィールドの設定を16028にします。推奨される論理ユニットについては、212ページ「論理ユニットとI/Oアドレッシング」を参照してください。VXI機器の論理アドレスは1～255です。

VXIアドレスの例2. 論理アドレス24のVXI機器と、バス・アドレス9のHP E1406 GPIBコマンド・モジュールを使って、論理ユニット7のGPIBインタフェース経由で通信するには、Addressフィールドの設定を70903にします。

HP E1406コマンド・モジュールの場合、VXI機器の論理アドレスを8で割ったものを2次アドレスとして使います。論理アドレスが24なら2次アドレスは3です。したがって、アドレス全体では70903となります。

シリアル・アドレスの例. 論理ユニット9のCOM1シリアル・ポートを使って機器を制御するには、Addressフィールドの設定を9にします。
推奨される論理ユニットについては、212ページ「論理ユニットとI/Oアドレッシング」を参照してください。

GPIOアドレスの例. 論理ユニット13のGPIOインタフェースを使ってカスタム機器を制御するには、Addressフィールドの設定を13にします。推奨される論理ユニットについては、212ページ「論理ユニットとI/Oアドレッシング」を参照してください。

Gatewayフィールド

Gatewayフィールドは、リモート・プロセス中に使用されるLANゲートウェイの名前を設定するために使います。詳細については、193ページ「LANゲートウェイ」を参照してください。

Advanced...ボタン

Advanced...ボタンをクリックすると、Advanced Instrument Propertiesダイアログ・ボックスが表示されます。

Advanced Instrument Propertiesダイアログ・ボックス: Generalタブ

図3-30は、Advanced Instrument Propertiesダイアログ・ボックスのGeneralタブの例を示します。

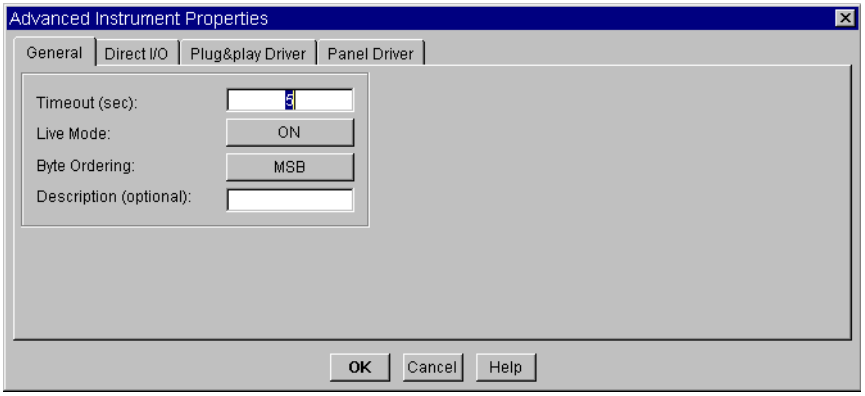


図3-30. Generalタブ

以下の部分では各フィールドについて説明します。

注記

Generalタブのパラメータは、Direct I/O、Panel Driver、Component Driverの各オブジェクトに当てはまりますが、To/From VXIplug&playオブジェクトには当てはまりません。

Timeout (sec) フィールド

Timeoutフィールドは、通信において機器が要求に応答しないときに、VEEがエラーを生成するまでの秒数を指定します。ほとんどのアプリケーションはデフォルト値の5秒で動作します。

一般に、このフィールドを0に設定すべきではありません。0に設定すると、VEEはタイムアウトを検出しなくなります。レジスタに対するDirect I/Oトランザクションや VXIデバイスのメモリ・アクセスの一部は、タイムアウトをサポートしません。

Live Modeフィールド

Live Modeフィールドは、指定されたアドレスの機器に対してVEEが通信を試みるかどうかを決定します。コンピュータに接続された機器と通信するためには、Live ModeをONに設定する必要があります。

特定の機器に対してLive ModeがOFFになっていても、その機器に対して読み書きを行うPanel Drivers、Component Drivers、Direct I/Oオブジェクトを含むプログラムを実行することは可能です。ただし、機器との通信は実際には行われません。この動作が役立つのは、機器が使用できない状態でプログラムの一部を開発したりデバッグしたりする場合です。

Byte Orderingフィールド

このフィールドは、デバイスがバイナリ・データを読み書きする順序を指定するために使います。このフィールドの値を元に、バイト・スワッピングが必要かどうかをVEEは判断します。このフィールドをクリックすると、MSB(上位バイトを先に送信)とLSB(下位バイトを先に送信)が切り替わります。IEEE 488.2準拠のデバイスはすべてデフォルトでMSB順序を使用しなければなりません。具体的な情報についてはデバイスのマニュアルを参照してください。

Description (optional)フィールド

Descriptionフィールドは一般に、メーカのモデル番号を記録するために用いられます。例えば、HP 54504Aオシロスコープの Descriptionはhp54504aのようになります。このフィールドはユーザの便宜のためのもので、VEEからは用いられません。

Advanced Instrument Propertiesダイアログ・ボックス: Direct I/Oタブ

図3-31は、Advanced Instrument Propertiesダイアログ・ボックスのDirect I/Oタブの例を示します(GPIBインタフェースの場合)。

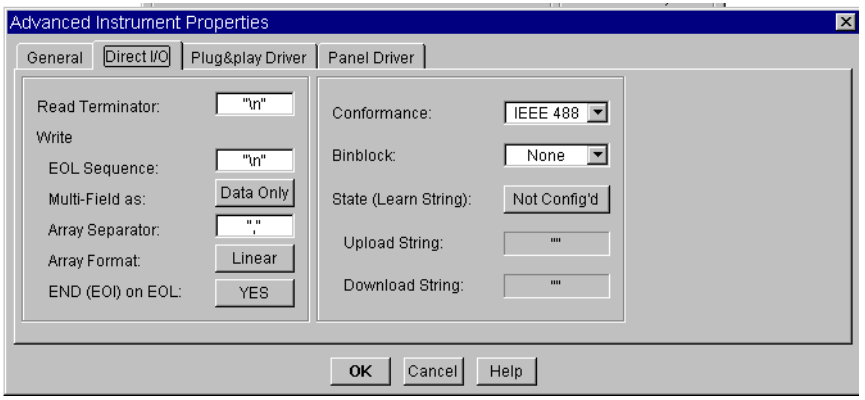


図3-31. Direct I/Oタブ

以下の部分では各フィールドについて説明します。

注記

VXIバックプレーン上のVXIデバイスと直接通信する場合、デバイス用のI-SCPIドライバが存在すれば、SCPIメッセージを使ってレジスタ・ベースのデバイスを制御することができます。必要なI-SCPIドライバが存在しない場合、VEEが警告を発します。

I-SCPIドライバが使用できない場合、レジスタ・ベース・デバイスを制御するには、デバイス・レジスタまたはデバイス・メモリを直接読み書きする必要があります。詳細については、104ページ「Advanced Instrument Propertiesダイアログ・ボックス: A16 Space(VXIのみ)タブ」または108ページ「Advanced Instrument Propertiesダイアログ・ボックス: A24/A32 Space(VXIのみ)タブ」を参照してください。

Read Terminator フィールド

Read Terminatorフィールドは、READトランザクションを終了する文字を指定します。このフィールドのエントリは、1個の文字を2重引用符で囲んだものでなければなりません。2重引用符とは、ASCIIの10進34のことです。任意のASCII文字と、表3-1に示すエスケープ文字が、VEEによってRead Terminatorとして認識されます。

どの文字を指定したらよいかは､機器のデザインによって決まります｡大部分のGPIB機器は、コンピュータにデータを送ったあとで改行文字を送信します。詳細については、機器のプログラミング・マニュアルを参照してください。

表3-1. エスケープ文字

エスケープ文字	ASCIIコード(10進)	意味
\n	10	改行
\t	9	水平タブ
\v	11	垂直タブ
\b	8	バックスペース
\r	13	キャリッジ・リターン
\f	12	改ページ
\"	34	2重引用符
\'	39	シングル引用符
\\	92	バックスラッシュ
ddd		3桁の8進数*ddd*に対応するASCII文字

Write EOL Sequenceフィールド

EOL Sequenceフィールドは、EOL ONを使用したWRITEトランザクションの最後に送信される文字を指定します。このフィールドのエントリは、0個以上の文字を2重引用符で囲んだものです。2重引用符とは、ASCIIの10進34のことです。EOLシーケンス内では、任意のASCII文字と、表3-1に示すエスケープ文字が認識されます。

Write Multi-field Asフィールド

Multi-field Asフィールドは、WRITE TEXTトランザクションのマルチフィールド・データ型のフォーマット・スタイルを指定します。VEEのマルチフィールド・データ型とは、Coord、Complex、PComplex、Spectrumです。他のデータ型と他のフォーマットは、この設定によって影響されません。

マルチフィールド・フォーマットとして(...) Syntaxを指定すると、マルチフィールド・アイテムのそれぞれが括弧で囲まれます。Data Onlyを指定すると、括弧がなくなりますが、区切りのカンマは残されます。例えば複素数2+2*j*は、(...) Syntaxでは(2,2)と書かれ、Data Onlyシンタックスでは2,2と書かれます。

Write Array Separatorフィールド

Array Separatorフィールドは、WRITE TEXTトランザクションで出力される配列の要素を区切る文字列を指定します。このフィールドのエントリは、1個の文字を2重引用符で囲んだものでなければなりません。2重引用符とは、ASCIIの10進34のことです。任意のASCII文字と、表3-1に示すエスケープ文字が、VEEによってArray Separatorとして認識されます。

Direct I/OオブジェクトのWRITE TEXT STRトランザクションで配列を出力する場合は特別です。この場合、Array Separatorフィールドの値は無視され、配列の要素はラインフィード文字(ASCIIの10進10)によって区切られます。この動作は、大部分の機器の要求に適合しています。

VEEではマルチフィールド・データ型の配列が使用できます。例えば、Complexデータの配列を作成できます。この場合、Multi-Field Formatを(...) Syntaxに設定すると、配列は次のように出力されます。

```
(1,1)array_sep(2,2)array_sep ...
```

ここで、*array_sep*はArray Separatorフィールドに指定された文字です。

Write Array Formatフィールド

Array Formatフィールドは、多次元配列を出力する方法を決定します。例えば、数学では行列を次のように表記します。

```
1  2  3 4  5  6 7  8  9
```

VEEはこの行列を、Array Formatの設定に応じて2つの方法のどちらかで出力します。以下の2つの例では、EOL Sequenceが"\n"(改行文字)に設定され、Array Separatorが" "(スペース)に設定されているものとします。

```
1 2 3    Block Array Format
4 5 6
7 8 9

1 2 3 4 5 6 7 8 9    Linear Array Format
```

どちらの配列フォーマットでも、配列の要素の間はArray Separator文字で区切られます。Block Array Formatでは、配列の各行をEOL Sequence文字で区切る操作が加わっています。

より一般的な例(3次元以上の配列)では、Block Array Formatの場合、右端以外の添字が変化するたびにEOL Sequence文字が出力されます。例えば、3次元配列A[x,y,z]を下記のトランザクションでBlock配列フォーマットで出力した場合、

```
WRITE TEXT A
```

xまたはyの値が変化するたびにEOL Sequenceが出力されます。Aの各次元のサイズが2の場合、要素は下記の順序で出力されます。

```
A[0,0,0]  A[0,0,1]<EOL Sequence>
A[0,1,0]  A[0,1,1]<EOL Sequence>
<EOL Sequence>
A[1,0,0]  A[1,0,1]<EOL Sequence>
A[1,1,0]  A[1,1,1]<EOL Sequence>
```

A[0,1,1]が出力されたあとでは、xとyが同時に変化するため、2個の<EOL Sequence>が出力されます。

Direct I/Oによる配列の出力. WRITE TEXT STRトランザクションでダイレクトI/Oパスに配列を出力する場合、Direct I/Oオブジェクトの**Array Separator**設定は無視されます。このトランザクションは、配列を出力する際に常にラインフィード(ASCIIの10進10)を使って各要素を区切ります。この動作は、大部分の機器の要求に適合しています(**配列に対するこの特別な動作は、他のトランザクション・タイプには当てはまりません**)。

Write END (EOI) On EOLフィールド (GPIBのみ)

END on EOLは、EOI (End Or Identify)の動作を制御します。END on EOLがYESの場合、下記のどれかの条件で最後のデータ・バイトが出力された時点でバスのEOIラインがアサートされます。

1. WRITEトランザクションがEOL ONで実行された
2. Direct I/Oオブジェクトの最後のトランザクションでWRITEトランザクションが実行された
3. 1つまたは複数のWRITEトランザクションがEOIをアサートせずに実行されたあとで、WRITE以外のトランザクション(READなど)が実行された

多くの機器は、EOIと改行文字のいずれも有効なメッセージ・ターミネータとして受け付けます。ブロック転送の場合はEOIが必要なことがあります。詳細については、機器のプログラミング・マニュアルを参照してください。

Conformanceフィールド

Conformanceは、IEEE 488.1またはIEEE 488.2標準に機器が適合するかどうかを指定します。機器のプログラミング・マニュアルを見て、どの標準に機器が適合するかを調べ、それに基づいてConformanceフィールドを設定します。

これらの標準はそれぞれ、GPIBインタフェースの通信プロトコルを定義しています。ただし、IEEE 488.1で未定義だったブロック・ヘッダとラーン・ストリングの規則がIEEE 488.2には規定されています。メッセージ・ベースのVXI機器と、I-SCPIドライバでサポートされるレジスタ・ベースのVXI機器は、すべてIEEE 488.2に適合しています。

ConformanceをIEEE 488(IEEE 488.1の意味)に設定した場合、ブロック・ヘッダとラーン・ストリングを処理するための追加の設定(このあとで説明)を指定できます。

Binblockフィールド

Binblockフィールドは、WRITE BINBLOCKトランザクションで用いられるブロック・データ・フォーマットを指定します。Binblockに指定できるのは、IEEE 728の#A、#T、#Iのいずれかのブロック・ヘッダです。BinblockがNoneの場合、WRITE BINBLOCKはIEEE 488.2の固定長任意ブロック応答データ・ブロックを出力します。

IEEE 728ブロック・ヘッダの形式は下記のとおりです。

```
#A<Byte_Count><Data>
#T<Byte_Count><Data>
#I<Data><END>
```

ここで:

<Byte_Count>は16ビットの符号なし整数で、<Data>部分のバイト数を示します。

<Data>は任意のバイトのストリームです。

<END>は、最後のデータ・バイトの送信とともにEOIがアサートされることを示します。

State (Learn String) フィールド

Stateフィールドは、ラーン・ストリングのアップロードとダウンロードが可能なように機器が構成されているかどうかを示します。Stateのエントリが Not Config'd で、ラーン・ストリングが使えるように機器を構成したい場合、Stateフィールドをクリックすると Upload String と Download の2つのフィールドが表示されます。Stateのエントリが Not Config'd の場合、Upload Stringフィールドと Download Stringフィールドはヌル文字列に設定されています。

Upload Stringフィールド

Upload String フィールドは、Direct I/O オブジェクト・メニューで Upload State を選択したときに機器に送信されるコマンドを指定します。機器にラーン・ストリングを送信させるためのコマンドを指定します。詳細については機器のプログラミング・マニュアルを参照してください。コマンドは2重引用符で囲みます。

Download String フィールド

Download Stringフィールドは、Direct I/O オブジェクトの WRITE STATE トランザクションの結果として、ラーン・ストリングの直前に機器に送信される文字列を指定します。このフィールドの目的は、ラーン・ストリングをダウンロードする際にコマンド・プレフィックスが必要な機器をサポートすることです。詳細については機器のプログラミング・マニュアルを参照してください。

Advanced Instrument Propertiesダイアログ・ボックス: Plug&play Driverタブ

図3-32は、Advanced Instrument Propertiesダイアログ・ボックスの Plug&play Driverタブの例を示します(GPIBインタフェースの場合)。

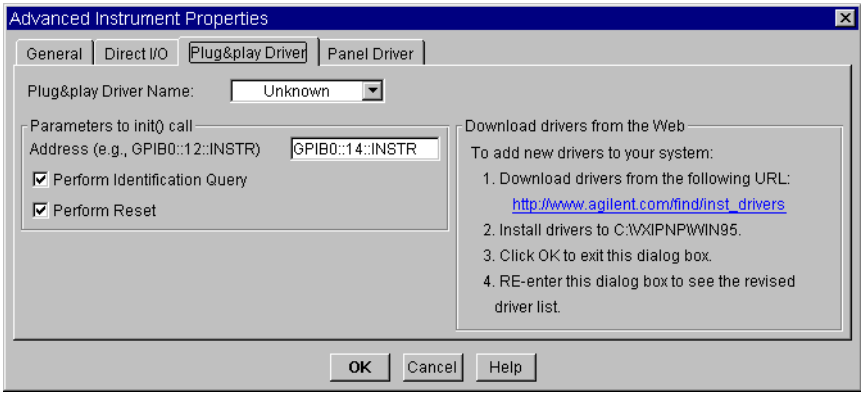


図3-32. Plug&play Driverタブ

Plug&play Driverタブは、Advanced Instrument Propertiesダイアログ・ボックスのタブのうちVXI*plug&play*ドライバ構成に関連するただ1つのものです。

以下の部分では各フィールドについて説明します。

Plug&play Driver Nameフィールド

このフィールドは、VXI*plug&play*ドライバの名前を指定します。このパラメータは必須なので、ドライバ名を必ず選択する必要があります。インストールされているすべてのVXI*plug&play*ドライバがドロップダウン・リストに表示されます。このリストにエントリがない場合、VXI*plug&play*ドライバがインストールされていないか、レジストリ・エントリまたは環境変数が正しく設定されていない可能性があります。詳細については、46ページ「VXIplug&playについて」を参照してください。

Parameters to init() callフィールド

Address. 機器を識別するアドレスを入力します。アドレスのフォーマットは、機器が接続されているインタフェースに依存します。

■ VXIアドレス文字列(組込みVXI、VXLink、MXIbusコントローラ)

組込み、VXLink、またはMXIbusコントローラを持つVXI機器の場合、アドレス文字列は下記の形式を取ります。

`VXI[board]::VXI logical address[::INSTR]`

例えば、論理アドレス24の機器の場合、`VXI::24::INSTR`となります。

board(ボード番号)は最初のボードの場合には省略可能です(`VXI::24::INSTR`は`VXI0::24::INSTR`と同じ)。ただし、2番目以降のボードではボード番号が必須です(`VXI1`、`VXI2`など)。

■ GPIB-VXIアドレス文字列(コマンド・モジュール)

コマンド・モジュールに接続されたGPIBカードから制御されるVXI機器の場合、アドレス文字列は下記の形式を取ります。

`GPIB-VXI[board]::VXI logical address [::INSTR]`

例えば、VXI論理アドレスが24の機器の場合、`GPIB-VXI::24::INSTR`(または`GPIB-VXI0::24::INSTR`)となります。

■ GPIBアドレス文字列(GPIB機器)

GPIBカードから制御される非VXI機器の場合、アドレス文字列は下記の形式を取ります。

`GPIB[board]::GPIB primary address::[GPIB secondary address][::INSTR]`

例えば、1次アドレス23のGPIB機器の場合、`GPIB::23::INSTR`(または`GPIB0::23::INSTR`)となります(任意指定の2次アドレスは、まれにしか用いられません)。

Perform Identification Query. このチェック・ボックスを選択すると、このドライバのファンクション・パネルが最初に実行されたときに、ドライバが機器に識別クエリを送ります。機器がこの動作をサポートしないまれな場合を除いて、一般にはこのチェック・ボックスを選択しておきます。

Perform Reset. このチェック・ボックスを選択すると、このドライバのファンクション・パネルが最初に実行されたときに、機器にリセットが送られます。機器がこの動作をサポートしないまれな場合を除いて、一般にはこのチェック・ボックスを選択しておきます。なお、VXI機器は必ずこの動作をサポートします。

Download Drivers. 新しいドライバを入手したり、ドライバをアップデートしたりするには、`Advanced Instrument Properties`ダイアログ・ボックスのURLをクリックします。

Advanced Instrument Propertiesダイアログ・ボックス: Panel Driverタブ

図3-33は、`Advanced Instrument Properties`ダイアログ・ボックスの`Panel Driver`タブの例です。

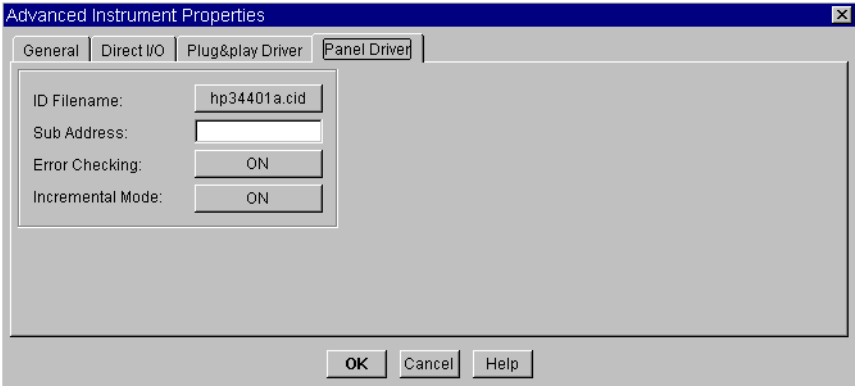


図3-33. The Panel Driverタブ

注記

レジスタ・ベースのVXIデバイスは、I-SCPIドライバでサポートされる場合に限ってメッセージ・ベースとして構成できます。

このタブは、Panel Driverオブジェクトと Component Driverオブジェクトの構成に用いられます。以下の部分では各フィールドについて説明します。

ID Filenameフィールド

ID Filenameフィールドは、目的のパネル・ドライバが入っているファイルを指定します。このフィールドをクリックすると、Read from what Instrument Driver?ダイアログ・ボックスが表示され、ファイルを選択できます。ファイル名は機器のモデル番号に基づいています。

似たようなファイル名(hp3325a.cidとhp3325b.cidなど)があるので、使用するモデル番号と正確に一致する名前を選ぶように注意してください。

Sub Addressフィールド

Sub Addressフィールドは、カードケージ型の機器(データ収集システムやスイッチなど)のプラグイン・モジュールを識別するために一部のドライバが使用するサブアドレスを指定します。この種のプラグインのドライバを構成する場合以外は、このフィールドは""(NULL文字列)にしておきます。

注記

サブアドレスを使用するドライバは**ごく**少数なので、ほとんどの場合はデフォルト設定の""(NULL文字列)が適切です。

これらのプラグインのドライバを構成する場合は、サブアドレスが用いられるかどうかをドライバのオンライン・ヘルプで確認してください。

注記

GPIB機器の2次アドレスとSub Addressフィールドとを混同しないでください。サブアドレスは**ドライバ**構成の一部であり、ハードウェア・アドレスの一部ではありません。

Error Checkingフィールド

Error Checkingフィールドは、コンポーネント値を設定したあとでVEEが機器にエラーを問い合わせるかどうかを指定します。実行速度が極端に低下する場合を除いて、このフィールドはONに設定しておきます。

Incremental Modeフィールド

Incremental Modeフィールドは、Panel Driverオブジェクトでインクリメンタル・ステート・リコールが用いられるかどうかを指定します。

注記

ほとんどの場合、Incremental ModeはONに設定するのが適切です。

Incremental ModeがONに設定されていると、機器のステートを変更する際にVEEから送信されるコマンドの数が自動的に最小化されます。このために、VEEは自分が記録している物理機器の現在のステートと、Panel Driverで指定された新しいステートとを比較します。

VEEはどのコンポーネント設定が異なっているかを判定し、目的のステートに一致しないコンポーネントを変更するコマンドだけを送信します。ほとんどの場合、実行を高速にするためにIncremental ModeはONにしておくのが適切です。

Incremental ModeがOFFに設定されていると、対応するPanel Driverが動作したときに、VEEはすべてのコンポーネントの値を明示的に設定します。このモードを使うのは一般に、VEEが記録している機器のステートが実際のステートと異なる可能性がある場合に限られます。

Incremental Modeの設定はPanel Driverオブジェクトの動作に影響しますが、Component Driverオブジェクトには影響しません。下記のような場合にはIncremental ModeをOFFにします。

- VEEプログラムで機器を制御しながら、フロントパネルからも機器を操作できるようにする場合
- VEEプログラムで機器を制御しながら、Cプログラム、Rocky Mountain Basicプログラム、シェル・コマンドなどを使ってVEE環境の外から機器設定を変更する場合

Component Drivers、Panel Drivers、Direct I/Oオブジェクトを1つのプログラムで組み合わせて使用する場合は、Incremental ModeをOFFにする必要はありません。

Advanced Instrument Propertiesダイアログ・ボックス: Serialタブ

図3-34は、Advanced Instrument Propertiesダイアログ・ボックスのSerialタブの例です(シリアル・インタフェースに対してのみ有効)。

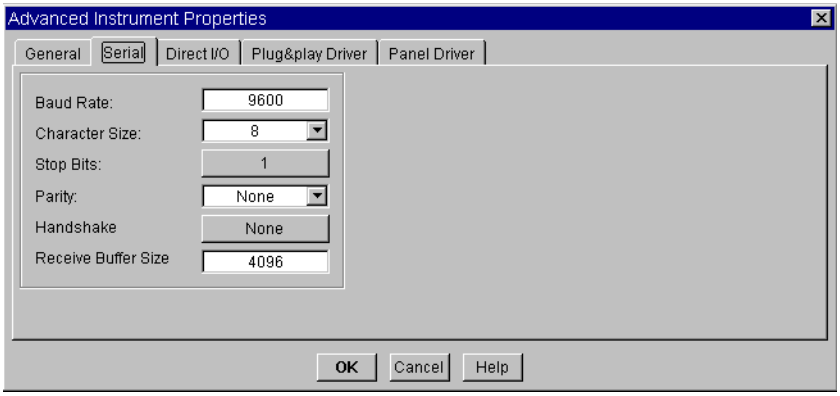


図3-34. Serialタブ

シリアル(RS-232)インタフェースに対して下記のフィールドを設定できます。

- Baud Rate - ボーレート。デフォルトは9600(ビット毎秒)
- Character Size - キャラクタ・サイズ。デフォルトは8(ビット)。使用できる値は5、6、7、8、None。
- Stop Bits - ストップ・ビット。デフォルトは1。使用できる値は1と2。
- Parity - パリティ。デフォルトはNone。使用できる値はNone、Odd、Even、Mark、Space。
- Handshake - ハンドシェーク。デフォルトはNone。使用できる値はNoneとXon/Xoff。
- Receive Buffer Size - 受信バッファ・サイズ。デフォルトは4096(バイト)。

Advanced Instrument Propertiesダイアログ・ボックス: GPIOタブ

図3-35は、Advanced Instrument Propertiesダイアログ・ボックスのGPIOタブの例です(GPIOインタフェースに対してのみ有効)。

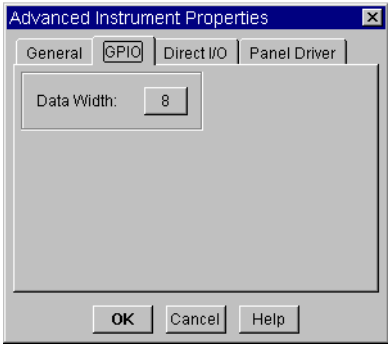


図3-35. GPIOタブ

GPIOタブにはData Widthフィールドが1つあるだけです。Data Widthフィールドは、GPIOインタフェースで1単位として伝送されるパラレル・データのビット数を指定します。このフィールドは、8ビットまたは16ビット幅でデータを読み書きするようにインタフェースを構成します。このフィールドに関連してハードウェア・スイッチを設定する必要はありません。

Advanced Instrument Propertiesダイアログ・ボックス: A16 Space(VXIのみ)タブ

図3-36は、Advanced Instrument Propertiesダイアログ・ボックスのA16 Spaceタブの例です。このタブはVXIインタフェースに対してのみ表示され、レジスタ・ベースのDirect I/Oトランザクションにのみ用いられます。

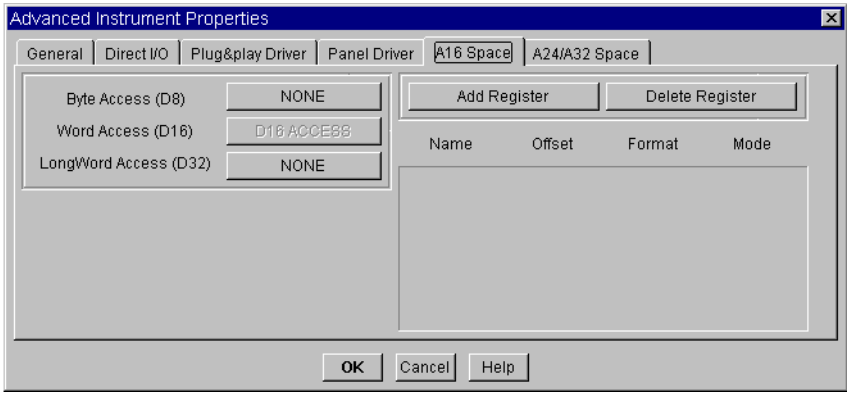


図3-36. A16 Spaceタブ

以下の部分では各フィールドについて説明します。

Byte Access (D8) フィールド

Byte Accessフィールドは、8ビットのA16メモリ・アクセスをVXIデバイスがサポートするかどうかを指定します。このフィールドでは下記の選択が可能です。

- NONE - デバイスはバイト・アクセスをサポートしません。
- ODD ACCESS - デバイスは奇数バイト境界でのみバイト・アクセスをサポートします(D08(O))。
- ODD/EVEN ACCESS - デバイスはすべての境界でバイト・アクセスをサポートします(D08(EO))。

Word Access (D16) フィールド

Word Accessフィールドは編集できません。すべてのVXIデバイスは16ビット・アクセス(D16)をサポートする必要があります。

LongWord Access (D32)フィールド

LongWord Accessフィールドは、32ビットのA16メモリ・アクセスをVXIデバイスがサポートするかどうかを指定します。下記の選択が可能です。

- NONE - デバイスは32ビット・アクセスをサポートしません。
- D32 ACCESS - デバイスは32ビットのA16メモリ・アクセスをサポートします。

Add Registerフィールド

Add Registerフィールドをクリックすると、新しいフィールド列がダイアログ・ボックスに追加されます。これらのフィールドでは、デバイスのA16メモリへのアクセス方法を設定します。下記の4つのフィールドがあります。

- Name - レジスタのシンボル名で、Direct I/Oオブジェクトの READ REGISTERまたはWRITE REGISTERトランザクションで特定のレジスタを参照するために用いられます。
- Offset - デバイスのA16メモリの**相対**ベースを基準とした構成対象レジスタのオフセット(バイト単位)。
- Format - 構成対象のレジスタとの間で読み書きされるデータのフォーマット。Offsetフィールドで指定されたバイトに対して読み書きが行われます。可能なフォーマットは下記の通りです。
  - BYTE - バイトを読み書きします。デバイスが8ビット・アクセスをサポートし、上記のBYTEフィールドが正しく設定されている必要があります。BYTEフィールドがODDの場合、Offsetフィールドで指定されたバイト位置は奇数でなければなりません。
  - WORD16 - 16ビット・ワードを読み書きします。16ビットは2の補数の整数で表現されます。すべてのVXIデバイスがこのフォーマットを明示的にサポートします。

- WORD32 - 32ビット・ワードを読み書きします。32ビットは2の補数の整数で表現されます。LongWord Accessフィールドが NONEに設定されていても、VEEはこのフォーマットをサポートします(この場合、2回のD16アクセスによって32ビットすべてが読み書きされます)。LongWord Accessフィールドが D32 ACCESSに設定されている場合、32ビットすべてが同時にアクセスされます。

- REAL32 - 32ビット・ワードを読み書きします。32ビットはIEEE 754の32ビット浮動小数点数で表現されます。LongWord Accessフィールドが NONEに設定されていても、VEEはこのフォーマットをサポートします(この場合、2回のD16アクセスによって32ビットすべてが読み書きされます)。LongWord Accessフィールドが D32 ACCESSに設定されている場合、32ビットすべてが同時にアクセスされます。

■ Mode - レジスタがサポートするI/Oモードを指定します。下記の選択が可能です。

- READ - このレジスタは、READ REGISTERトランザクションでのみ選択可能です。

- WRITE - このレジスタは、WRITE REGISTERトランザクションでのみ選択可能です。

- READ/WRITE - このレジスタは、READ REGISTERとWRITE REGISTERの両方のトランザクションで選択可能です。

Delete Register フィールド

Delete Registerフィールドをクリックすると、現在構成されているレジスタのシンボル名のリストが表示されます。選択したレジスタがダイアログ・ボックスから削除されます。

例

図3-37は、HP E1411B VXIマルチメータのレジスタを構成したA16 Spaceタブを示します。Add Registerでレジスタを追加していくと、レジスタのリストがスクロールするようになります。

注記

拡張(A24/A32スペース)メモリの構成もこれと似ていますが、「レジスタ」の代わりにメモリ「位置」が用いられます。

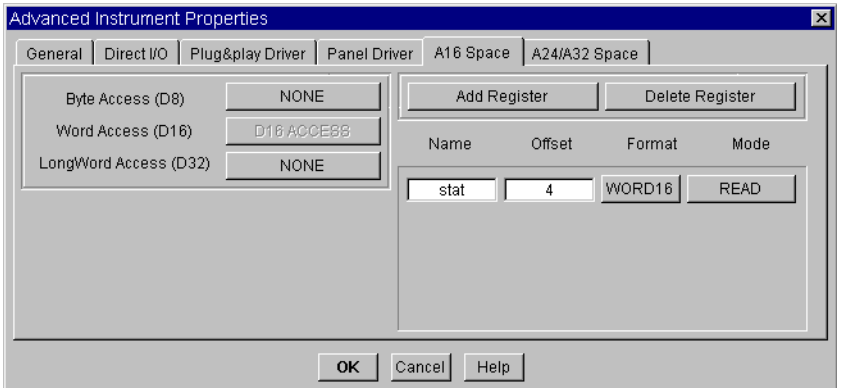


図3-37. HP E1411Bマルチメータ のA16構成

Offsetフィールドには、デバイスのA16スペースの**相対**ベースからの各レジスタのオフセット(バイト単位)が設定されます。ステータス・レジスタ(図のName:=stat)は4バイト・オフセットに設定されており、READモードに構成されています。

制御レジスタは図に示されていませんが、通常は4バイト・オフセットでWRITEモードに構成されます。2つの異なるレジスタ位置が同じモードになっていてもかまいませんが、Nameフィールドは一意でなければなりません。一方、バイト位置4のレジスタにstatuscontrolという名前を付けてREAD/WRITEモードに構成することもできます。

Advanced Instrument Propertiesダイアログ・ボックス: A24/A32 Space(VXIのみ)タブ

図3-38は、Advanced Instrument Propertiesダイアログ・ボックスのA24/A32 Spaceタブの例です。このタブはVXIインタフェースに対してのみ表示され、レジスタ・ベースのDirect I/Oトランザクションにのみ用いられます。

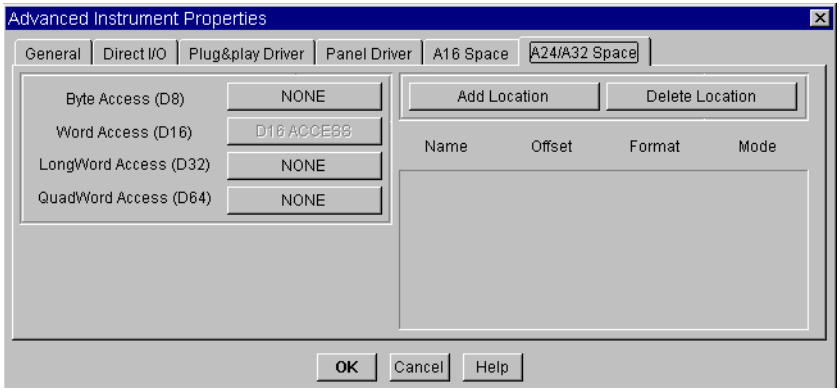


図3-38. A24/A32 Spaceタブ

以下の部分では各フィールドについて説明します。

注記

「拡張メモリ」と呼ばれるのは、VXIデバイスのA24またはA32メモリのことです(VXIデバイスはA24メモリとA32メモリのどちらかを実装できますが、両方はできません)。

Byte Access (D8) フィールド

Byte Accessフィールドは、8ビットの拡張メモリ・アクセスをVXIデバイスがサポートするかどうかを指定します。このフィールドでは下記の選択が可能です。

- NONE - デバイスはバイト・アクセスをサポートしません。
- ODD ACCESS - デバイスは奇数バイト境界でのみバイト・アクセスをサポートします(D08(O))。
- ODD/EVEN ACCESS - デバイスはすべての境界でバイト・アクセスをサポートします(D08(EO))。

Word Access (D16) フィールド

Word Accessフィールドは編集できません。すべてのVXIデバイスは全メモリ・スペースで16ビット・アクセス(D16)をサポートする必要があります。

LongWord Access (D32)フィールド

LongWord Accessフィールドは、32ビットの拡張メモリ・アクセスをVXIデバイスがサポートするかどうかを指定します。下記の選択が可能です。

- NONE - デバイスは32ビット・アクセスをサポートしません。
- D32 ACCESS - デバイスは32ビットの拡張メモリ・アクセスをサポートします。

QuadWord Access (D64)フィールド

QuadWord Accessフィールドは、64ビットの拡張メモリ・アクセスをVXIデバイスがサポートするかどうかを指定します。下記の選択が可能です。

- NONE - デバイスは64ビット・アクセスをサポートしません。
- D64 ACCESS - デバイスは64ビットのメモリ・アクセスをサポートします。

Agilent I/Oライブラリ G.02.02では、一部のVXI I/O機器のメモリ・スペースに対する64ビット・アクセスが可能になっています。この機能により、このモードをサポートするVXI機器に対してVEEのプログラムから64ビット単位のメモリ読み書きが可能になります。バージョンG.02.02がインストールされている場合、Advanced Instrument Configurationダイアログ・ボックスの**A24/A32 Space**タブを使ってこのアクセス・モードを有効にできます。

このモードを有効にするには、QuadWord access(D64)アクセスを有効にしてから、フォーマットとしてWORD32*2または**REAL64**を選択します。WORD32*2フォーマットを選択した場合、INT32配列内の2個の連続する数値に64ビット値が読み込まれます。

位置: I/O ⇒ Instrument Manager(VXI機器を選択) ⇒ Edit Instrument ⇒ Advanced I/O Config... **A24/A32 Space**タブ

Add Locationフィールド

Add Locationフィールドをクリックすると、新しいフィールド列がダイアログ・ボックスに追加されます。これらのフィールドでは、デバイスの拡張メモリへのアクセス方法を設定します。下記の4つのフィールドがあります。

- Name - 位置のシンボル名で、Direct I/Oオブジェクトの**READ MEMORY**または**WRITE MEMORY**トランザクションで特定のメモリ位置を参照するために用いられます。
- Offset - デバイスの拡張メモリの**相対**ベースを基準とした構成対象位置のオフセット(バイト単位)。

- Format - 構成対象の位置との間で読み書きされるデータのフォーマット。Offsetフィールドで指定されたバイトに対して読み書きが行われます。可能なフォーマットは下記の通りです。
  - BYTE - バイトを読み書きします。デバイスが8ビット・アクセスをサポートし、上記のBYTEフィールドが正しく設定されている必要があります。BYTEフィールドがODDの場合、Offsetフィールドで指定されたバイト位置は奇数でなければなりません。
  - WORD16 - 16ビット・ワードを読み書きします。16ビットは2の補数の整数で表現されます。すべてのVXIデバイスがこのフォーマットを明示的にサポートします。
  - WORD32 - 32ビット・ワードを読み書きします。32ビットは2の補数の整数で表現されます。LongWord AccessフィールドがNONEに設定されていても、VEEはこのフォーマットをサポートします(この場合、2回のD16アクセスによって32ビットすべてが読み書きされます)。LongWord AccessフィールドがD32 ACCESSに設定されている場合、32ビットすべてが同時にアクセスされます。
  - REAL32 - 32ビット・ワードを読み書きします。32ビットはIEEE 754の32ビット浮動小数点数で表現されます。LongWord AccessフィールドがNONEに設定されていても、VEEはこのフォーマットをサポートします(この場合、2回のD16アクセスによって32ビットすべてが読み書きされます)。LongWord AccessフィールドがD32 ACCESSに設定されている場合、32ビットすべてが同時にアクセスされます。
  - WORD32*32 - 64ビット・ワードを2個の32ビット・ワード(2個のInt32)として読み書きします。QuadWord Accessが有効になっている必要があります。
  - REAL64 - 64ビット・ワードをREAL64として読み書きします。QuadWord Accessが有効になっている必要があります。
- Mode - この位置がサポートするI/Oモードを指定します。下記の選択が可能です。
  - READ - この位置は、READ MEMORYトランザクションでのみ選択可能です。
  - WRITE - この位置は、WRITE MEMORYトランザクションでのみ選択可能です。

- READ/WRITE - この位置は、READ MEMORYとWRITE MEMORYの両方のトランザクションで選択可能です。

Delete Location フィールド

Delete Locationフィールドをクリックすると、現在構成されている位置のシンボル名のリストが表示されます。選択した位置がダイアログ・ボックスから削除されます。

Interface Properties

Interface Propertiesダイアログ・ボックスは、Instrument Managerの機器リストでインタフェースを選択してPropertiesボタンをクリックしたときだけ表示されます。図3-39にこのダイアログ・ボックスの例を示します。

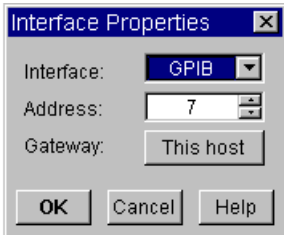


図3-39. Interface Propertiesダイアログ・ボックス

以下の部分では各フィールドについて説明します。

Interfaceフィールド Interfaceフィールドは、ハードウェア・インタフェースのタイプを指定します。GPIBとVXI(どちらも複数機器用のバス)、SerialとGPIO(どちらも単一機器用のインタフェース)は相互に交換可能です。

Addressフィールド Addressフィールドはインタフェースの論理ユニットを指定し、このインタフェースに接続されたすべての機器に影響します。上下の矢印を使ってAddressを変更します。衝突しない論理ユニットだけが表示されます。

Gatewayフィールド Gatewayフィールドは、リモート・プロセスで用いられるLANゲートウェイの名前を設定するために使います。詳細については、193ページ「LANゲートウェイ」を参照してください。

4

トランザクションI/Oの使用

トランザクションI/Oの使用

UNIX版VEEには、ファイル、プリンタ、名前付きパイプ、他のプロセスと通信するためのオブジェクトが含まれています。また、Rocky Mountain Basicとの通信や、ハードウェア・インタフェースとそれに接続された機器との通信の手段も用意されています。

このトランザクションを使った通信は、I/Oオブジェクトによって制御されます。この章では、**トランザクション**を使用するすべてのオブジェクトに共通の一般的概念について説明します。下記の内容があります。

- トランザクションの作成と読取り
- トランザクション・ベース・オブジェクトの 使用
- 正しいトランザクションの選択
- ファイルとの通信
- プログラムとの通信(UNIX)Rocky Mountain Basic
- プログラムとの通信(PC)
- ダイレクトI/Oおよびインタフェース操作でのトランザクションの使用

また、ダイレクトI/Oおよびインタフェース操作でトランザクションを使用する方法についても説明します。

注記

参考文献:
I. Haviland, Keith and Salama, Ben, *UNIX System Programming*. (Addison-Wesley Publishing Company, Menlo Park, California, 1987).

この本には、UNIXを使用するプログラマに役立つ一般的な情報が記載されています。特に、プロセス間通信とパイプに関するこの本の説明は、`To/From Named Pipe`、`To/From Socket`、`To/From Rocky Mountain Basic`、`Execute Program`を使う上で参考になります。

トランザクションの作成と読取り

この章で扱うすべてのI/Oオブジェクトにはトランザクションが含まれます。トランザクションは、データの読み書きの方法など、低レベルの入出力動作を指定します。トランザクションは、I/Oオブジェクトのオープン・ビューに1行のテキストとして表示されます。代表的なトランザクションを見るには、I/O⇒To⇒Stringをクリックして`To String`オブジェクトを作成します。図4-1にこのオブジェクトを示します。

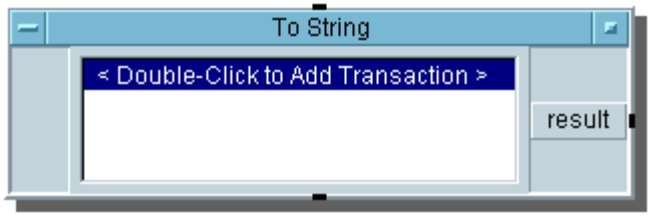


図4-1. `To String`**オブジェクトのデフォルト・トランザクション**

トランザクションを追加するには、オブジェクト内部をダブルクリックします。

図4-2に示すのは、`To String`オブジェクトを使ってトランザクションの動作を説明するための単純なプログラムです。このプログラムには2つのトランザクションが用いられています。1つは文字列リテラルを出力し、もう1つは固定10進フォーマットで数値を出力します。

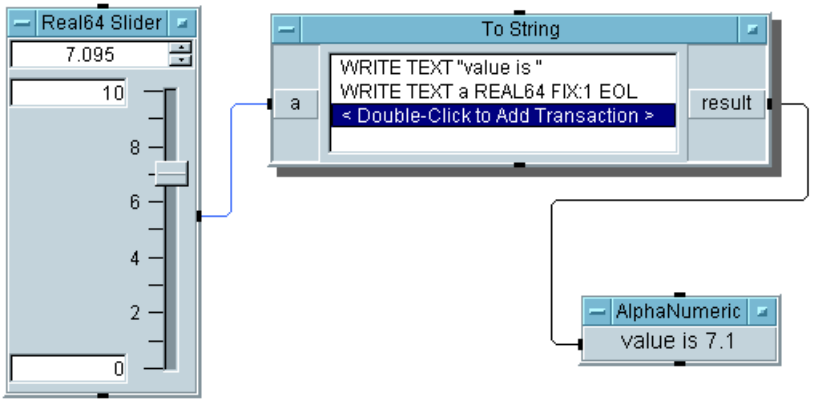


図4-2. `To String`**オブジェクトを使ったプログラム**

一般に、トランザクション・ベース・オブジェクトを扱うには最低でも下記の2つの手順が必要です。

1. 必要なトランザクションを追加します。

2. 入力端子、出力端子、またはその両方を追加します。ほとんどの端子は、トランザクションを追加または編集したときに自動的に追加されます。

トランザクションの作成と編集

マウスとキーボードを使った編集

表4-1は､マウスを使ってトランザクションを編集する方法を簡単に説明しています。

表4-1. マウスを使ったトランザクションの編集

目的	操作
リストの末尾にトランザクションを追加	オブジェクト内部をダブルクリックするか、オブジェクト・メニューのadd Transをクリック
強調表示バーを別のトランザクションに移動	強調表示されていないトランザクションをクリック
強調表示されたトランザクションの上にトランザクションを挿入	オブジェクト・メニューのInsert Transをクリック
強調表示されたトランザクションをカット(削除)し、トランザクションの「カット・アンド・ペースト」バッファに保存	オブジェクト・メニューのCut Transをクリック
強調表示されたトランザクションをトランザクションの「カット・アンド・ペースト」バッファにコピー	オブジェクト・メニューのCopy Transをクリック
現在バッファにあるトランザクションを強調表示されたトランザクションの上にペースト	オブジェクト・メニューのPaste Transをクリック
トランザクションを編集	トランザクションをダブルクリック

表4-2は、キーボードを使ってトランザクションを編集する方法を簡単に説明しています。

表4-2. キーボードを使ったトランザクションの編集

目的	押すキー
強調表示バーを次のトランザクションに移す	**CTRL**+**N**
強調表示バーを前のトランザクションに移す	**CTRL**+**P**
強調表示バーを別のトランザクションに移す	↑、↓、**Home**
強調表示されたトランザクションの上にトランザクションを挿入	**Insert line**または**CTRL**+**O**
強調表示されたトランザクションをカット(削除)し、トランザクションの「カット・アンド・ペースト」バッファに保存	**Delete line**または**CTRL**+**K**
現在バッファにあるトランザクションを強調表示されたトランザクションの上にペースト	**CTRL**+**Y**
強調表示されたトランザクションを編集	**スペース・バー**

トランザクション内部のフィールドを編集するには、トランザクションをダブルクリックして`I/O Transaction`ダイアログ・ボックス(図4-3)を表示させます。

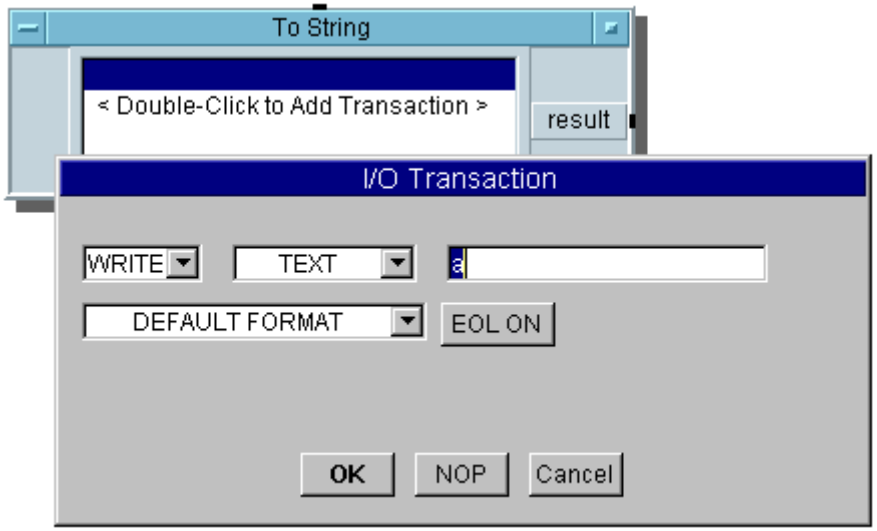


図4-3. `To String`**オブジェクトのデフォルト・トランザクションの編集**

I/O Transactionダイアログ・ボックスに表示されるフィールドは、I/O動作のタイプによって異なります。フィールドの情報を変更するには、矢印をクリックし、表示されるリストから選択します。矢印のないフィールドにはテキストを入力する必要があります。OKをクリックすると選択が確定し、I/Oオブジェクトに戻ります。

NOPをクリックすると、ダイアログ・ボックスに表示された最新の設定が保存され、このトランザクションが"NOP"(No Operation)になります。これは、テキスト・ベースのコンピュータ・プログラムでコード行をコメント・アウトするのと同じ意味です。

入出力端子は必要に応じて自動的に追加されます。また、オブジェクト・メニューを使って端子を追加したり削除したりすることもできます。

データ・フィールドの編集

データ・フィールドにはテキストを入力する必要があります。図4-4に示すのはREADトランザクションで、データ・フィールドへの入力の例が示されています。

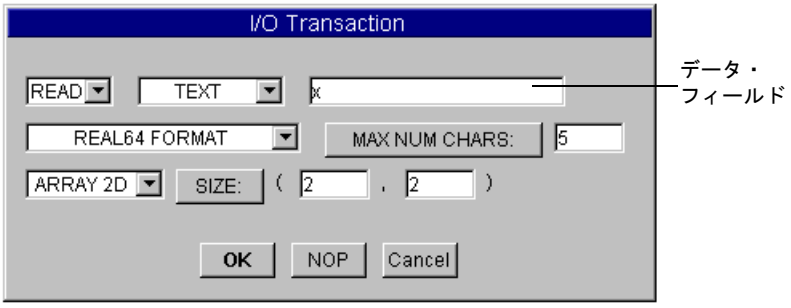


図4-4. データ・フィールドに変数を使ったREADトランザクション

図4-5に示すのはWRITEトランザクションで、データ・フィールドへの入力の例が示されています。

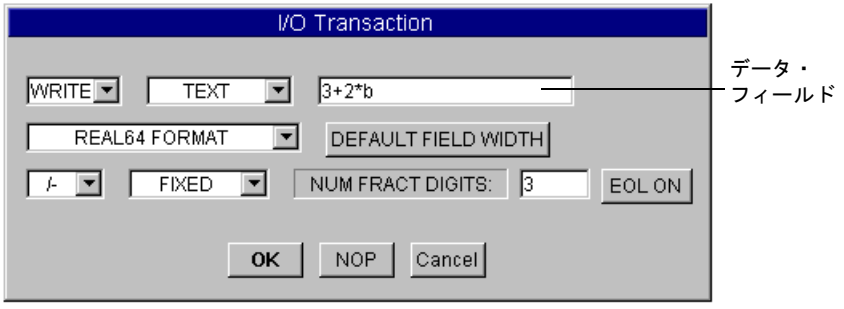


図4-5. データ・フィールドに式を使ったWRITEトランザクション

WRITEトランザクションでは式リスト(変数、定数、演算子)が使用できますが、READでは変数リストだけが使用できます。表4-3に、データ・フィールドの代表的なエントリを示します。

表4-3. データ・フィールドの代表的なエントリ

データ・フィールドのエントリ	意味
X	(READ)データを変数Xに読み取る
A	(WRITE)変数Aの値を出力
X,Y	(READ)データを変数Xに読み取り、データを変数Yに読み取る
A,B	(WRITE)変数Aの値を出力し、変数Bの値を出力
null	(READのみ)指定された変数の値を読み取って廃棄。nullはVEEで定義された特殊変数です。
A,A*1.1	(WRITEのみ)Aの値を出力し、Aに1.1をかけた値を出力
"hello\n"	(WRITE)テキスト・リテラルhelloのあとに改行文字を出力
"FR ",Fr," MHZ"	(WRITE)テキスト・リテラルと数値の組合わせを出力。トランザクションがWRITE TEXT REALでFrにReal値1.234が入っている場合、FR 1.234 MHZと出力されます。

2重引用符で囲んだ文字列としてテキストを入力するすべてのフィールドで、表4-4に示すエスケープ文字が使用できます。

注記

READトランザクションではデータ・フィールドにnull変数が使用できます。null変数にデータを読み込むと、データは廃棄されます。不要なデータを除去するために役立ちます。

表4-4. エスケープ文字

エスケープ文字	ASCIIコード(10進)	意味
\n	10	改行
\t	9	水平タブ
\v	11	垂直タブ
\b	8	バックスペース
\r	13	キャリッジ・リターン
\f	12	改ページ
\"	34	2重引用符
'	39	シングル引用符
\\	92	バックスラッシュ
ddd		3桁の8進数*ddd*に対応するASCII文字

端子の追加

VEEは必要に応じて入出力端子を自動的に追加します。手動で端子を追加するには、オブジェクト・メニューの"Add Terminal"をクリックするか、キーボード・ショートカット**CTRL**+**A**を使います。

WRITEトランザクションは、オブジェクトに対応する出力先にVEEからデータを転送するので、データ入力端子が必要です。WRITEトランザクションはグローバルまたは式("abs(globalA)"など)からデータを出力することもできます。

READトランザクションは、オブジェクトに対応するソースからVEEにデータを転送するので、データ出力端子が必要です。

端子に示される変数名は、図4-6に示すようにトランザクション仕様の変数名と一致する必要があります。

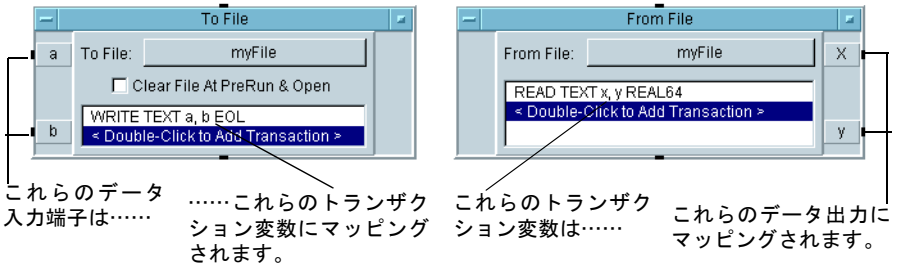


図4-6. 端子と変数の対応

端子の変数名を編集するには、下記のようにします。

1. 端子をダブルクリックして、Terminal Informationダイアログ・ボックスを表示させます。

2. ダイアログ・ボックスのNameフィールドを編集します。

VEEの変数名には大文字小文字の区別がありません。したがって、sはSと同じであり、Signalはsignalと同じです。

トランザクション・データの読取り

データを変数に読み取るには、読み取るデータ・エレメントの数がわかっているか、終了条件が指定されている必要があります。READトランザクションは、指定された数のデータ・エレメントを読み取ろうとするか、EOF指定を探します。これを指定するには、I/O Transactionダイアログ・ボックスの最後のフィールドを使います。

指定された数のデータ・エレメントを読み取るトランザクション

トランザクション・ダイアログ・ボックスの最後のフィールドのデフォルト値はSCALARです。これは、READトランザクションがエレメントを1個だけ読み取ることを指定します。これを変更するには、SCALARフィールドをクリックし、図4-7に示すように可能な選択肢のリストから選択します。

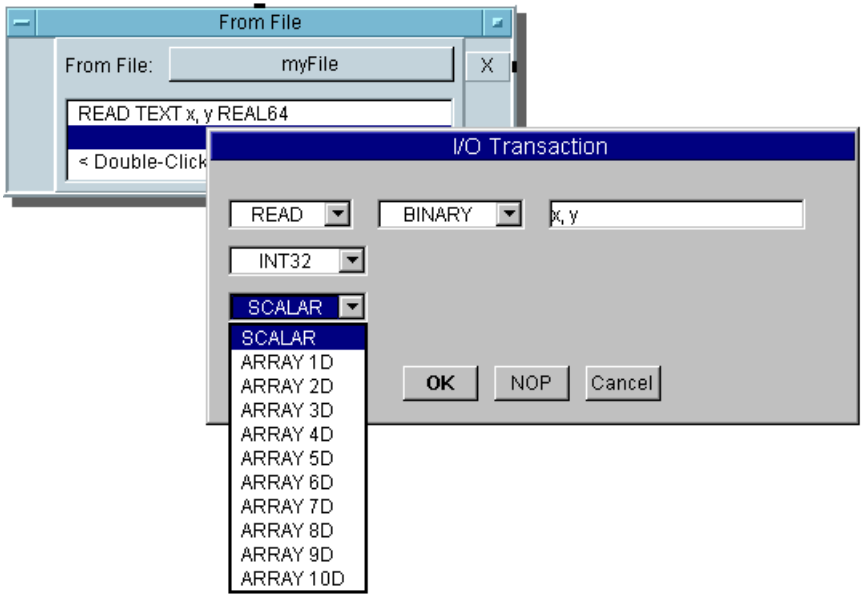


図4-7. 読取り次元をリストから選択

リストの選択肢は、READトランザクションの次元数を示します。例えば、SCALARは0次元、ARRAY 1Dは1次元配列、ARRAY 2Dは2次元配列を示します。

次元を選択すると、指定した各次元に対応する入力フィールドがトランザクション・ダイアログ・ボックスに表示されます。図4-8に示すトランザクション・ダイアログ・ボックスでは、2進整数の3次元配列を`matrix`という名前の変数に読み込むことを指定しています。`SIZE:`のあとの3つのフィールドは、それぞれ対応する次元にある整数の個数を示しています(この例では、すべての次元に2個の要素があります)。

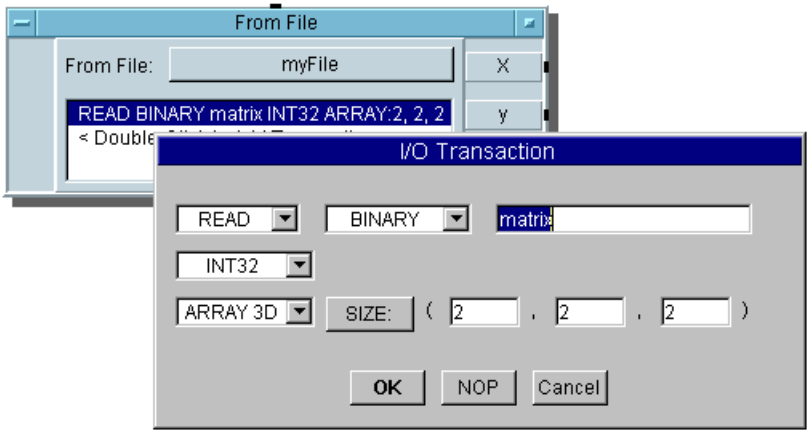


図4-8. 多次元読取りのトランザクション・ダイアログ・ボックス

2次元以上を指定する場合、右端(最も内側)の次元が最初に埋められます。この例では、下記の順序で要素が読取られます。

```
matrix[0,0,0] 最初に読み取る
matrix[0,0,1]
matrix[0,1,0]
matrix[0,1,1]
matrix[1,0,0]
matrix[1,0,1]
matrix[1,1,0]
matrix[1,1,1] 最後に読み取る
```

トランザクション・ダイアログ・ボックスで`OK`ボタンをクリックすると、`ARRAY:`キーワードのあとに各次元のサイズを指定した形で結果のトランザクションが表示されます。例を挙げます。

```
READ BINARY matrix INT32 ARRAY:2,2,2
```

トランザクションがスカラ値を読み取るように構成された場合、次のように表示されます。

```
READ BINARY x INT32
```

SIZE:フィールドに変数名を入力することにより、配列の次元をプログラムで指定することができます。例えば、下記のトランザクションは3次元の行列を読み取ります。

```
READ BINARY matrix INT32 ARRAY:xsize,ysize,zsize
```

この例で、xsize、ysize、zsizeは入力端子名でもよく、同じオブジェクトの前のトランザクションが設定した出力端子の名前でもかまいません。

READ TO ENDトランザクション

一部のVEEオブジェクトは、EOF(End Of File)まで読み取るトランザクションをサポートします。これにより、ファイルの内容を1回のトランザクションで読み取ることができます。この種のトランザクションを**READ TO ENDトランザクション**と呼びます。EOFは標準的なディスク・ファイルではファイルの末尾を表しますが、名前付きパイプやパイプがクローズされたことも表します。

下記のVEEオブジェクトがREAD TO ENDトランザクションをサポートします。

- From File
- From String
- From Stdin (UNIX)
- To/From Named Pipe (UNIX)
- To/From Rocky Mountain Basic (UNIX)
- Execute Program (UNIX)
- To/From DDE (PC)

図4-9に示すのは、2進整数の3次元配列を読み取るFrom Fileオブジェクトで、READ TO ENDに設定されています。

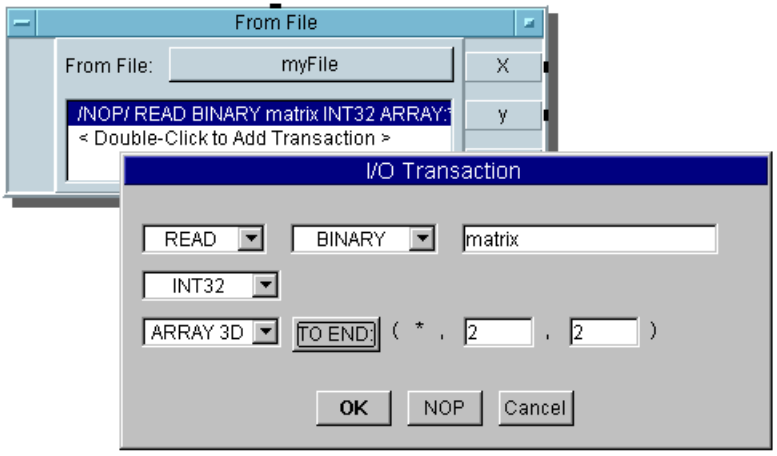


図4-9. 多次元READ TO ENDのトランザクション・ダイアログ・ボックス

READ TO ENDトランザクションはスカラではサポートされません。READ TO ENDを使用するには、1次元以上の配列がトランザクションに設定されている必要があります。READ TO ENDをサポートするVEEオブジェクトでは、トランザクション・ダイアログ・ボックスのSIZE:フィールドがボタンで表示されます。SIZE:フィールドをクリックするとREAD TO ENDが有効になり、フィールドの表示がTO END:に変わります。

1次元配列をREAD TO ENDで読み取る場合、配列の要素数はEOFが見つかるまでわかりません。配列の未知のサイズはトランザクションではアスタリスク(*)で表されます。

多次元配列をREAD TO ENDで読み取る場合、左端(最も外側)以外のすべての次元に対して要素数を指定しておく必要があります。図4-9のトランザクションでは、左端の次元にサイズの代わりに(*)が表示されています。この次元のサイズは、READ TO ENDトランザクションが終了するまでわかりません。

3次元配列は、単に2次元配列をいくつかまとめたものに過ぎません。2次元配列には「行」と「列」の2つの次元があります。2次元配列を(カードのように)重ねると、3番目の次元である「高さ」が生じます。

3次元配列のREAD TO ENDトランザクションでは、「行」と「列」の数は指定されますが、「高さ」はEOFが来るまでわかりません。これは多次元のREAD TO ENDトランザクションすべてに当てはまります。配列にn個の次元がある場合、n-1個の次元に対してはサイズを指定する必要があります。左端の1個の次元だけはサイズが未知でかまいません。

ARRAY 1Dよりも次元の大きいREAD TO ENDトランザクションでは、読み取る要素の総数が既知の次元の積で割り切れる必要があります。例えば、3次元配列のREAD TO ENDトランザクションの例を16個の要素を持つファイルに対して実行すると、「行」と「列」の数がそれぞれ2と指定されているので、2×2の配列を4個読み取ることになります。したがって、読取りが終了したときには未知の次元サイズである「高さ」は4となります。

もし、ファイルに18個の要素があると、2×2配列の1つが、2個の要素しか持たない不完全なものとなります。このファイルに対して、「行」と「列」のサイズを2に指定したREAD TO ENDトランザクションを実行すると、エラーが発生し、データはいっさい読み取られません。一方、「行」の数が1で「列」の数が3ならば、このファイルを読み取ることができます。このファイルに対してREAD TO ENDトランザクションを実行すると、「高さ」は6となります。

ファイル中のデータ要素の総数がわからない場合でも、ARRAY 1DのREAD TO ENDトランザクションは常に使用できます。

注記

READ TO ENDトランザクションは、未知の数の要素を返すシェル・コマンドをExecute Programオブジェクトのプログラムとして使用する場合に便利です。

ノンブロッキング読取り

READトランザクションは、読取りが終わったときに終了します。読取りが終わるまでは、トランザクションは**ブロック**します。ディスク・ファイルを読み取る場合、データが必ずディスクから得られるので、ブロッキング動作が意識されることはありません。一方、名前付きパイプやパイプの場合、他のプロセスからデータが得られるので、READトランザクションがブロックし、結果としてVEEプログラムの実行を止めてしまうことがあります。場合によっては、READトランザクションが無期限に停止することもあります。

READ IOSTATUS DATAREADYトランザクションを使うと、名前付きパイプまたはパイプに対して、READトランザクションで読み取るデータがあるかどうかを調べることができます。

READ IOSTATUS DATAREADYトランザクションは下記のVEEオブジェクトで使用できます。

- To/From Named Pipe (UNIX)
- To/From Socket
- To/From Rocky Mountain Basic (UNIX)
- From StdIn (UNIX)

注記

READ IOSTATUS DATAREADY トランザクションを実行した場合、名前付きパイプが書込みプロセス側でオープンされるまでトランザクションはブロックします。パイプがオープンされると、トランザクションはパイプのステータスを返します。

書込み側プロセスでパイプがクローズされた場合、実効的にパイプにはEOFが書き込まれ、READ IOSTATUS DATAREADYトランザクションは1を返します。これはパイプにEOFが存在することを示します。このあとでREADトランザクションを実行すると、EOFエラーが発生します。EOFエラーをトラップするには、データを読み取るオブジェクトのエラー・ピンを使います。

図4-10は、READ IOSTATUS DATAREADYを使ってStdInパイプのデータを検出するプログラムを示します。

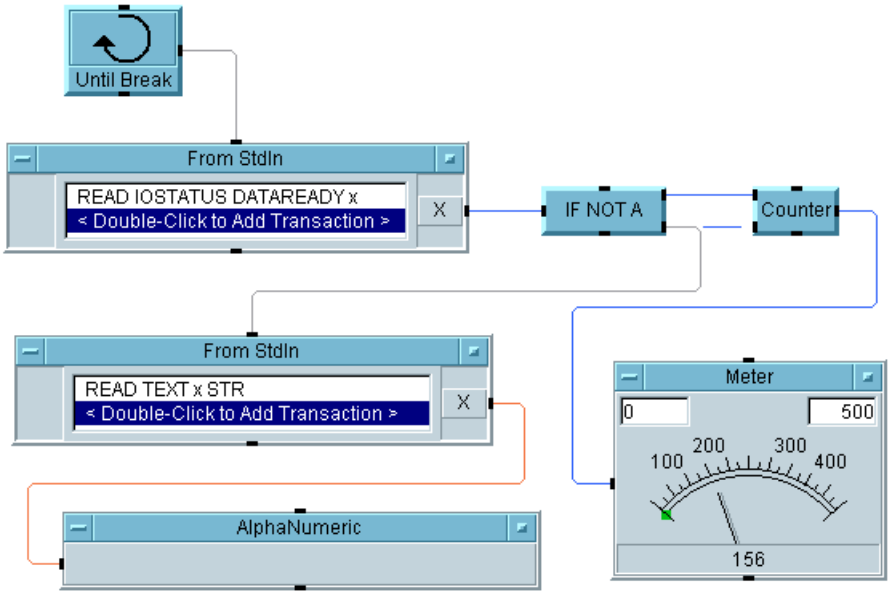


図4-10. READ IOSTATUS DATAREADYを使ったノンブロッキング読取り

このプログラムは、examplesディレクトリのmanual47.veeファイルに保存されています。

図4-10のプログラムでは、From StdInでREAD IOSTATUS DATAREADYトランザクションを使っています。stdinパイプにデータがない場合、トランザクションは0を返します。データが存在する場合、1が返されます。READ IOSTATUS DATAREADYトランザクションの戻り値をIf/Then/Elseでテストします。結果が1なら、2番目のFrom StdInが実行され、VEEスタートアップ・ターミナル・ウィンドウに入力されたデータを読み取ります。

スタートアップ・ターミナル・ウィンドウにデータが入力されていない場合(あるいはReturnが押されていない場合)、スレッドの先頭に戻って実行が継続されます。Until Breakによってスレッドが反復されるので、READ IOSTATUS DATAREADYトランザクションを使ったFrom StdInが繰り返しテストされます。

ファイルから配列を読み取る方法を示すプログラムの全体を見るには、examplesディレクトリにあるファイルmanual27.veeおよびmanual28.veeをオープンして実行してください。

トランザクションの作成に関するヒント

必要なトランザクションを作成するには、多くの場合試行錯誤が最善の方法です。I/Oトランザクションが扱うデータの多くはテキストです(テキストでないものは何らかの形のバイナリ・データです)。TEXT形式で出力されたデータは、人間が理解できるので実験に便利です。TEXTはサイズや速度の点では不利ですが、ほぼいかなる用途にも使用できます。

To Stringオブジェクトを使うことにより、テキストを出力する他のI/Oオブジェクトの出力動作を正確にシミュレートできます。図4-11のプログラムは、このための1つの方法を示します。

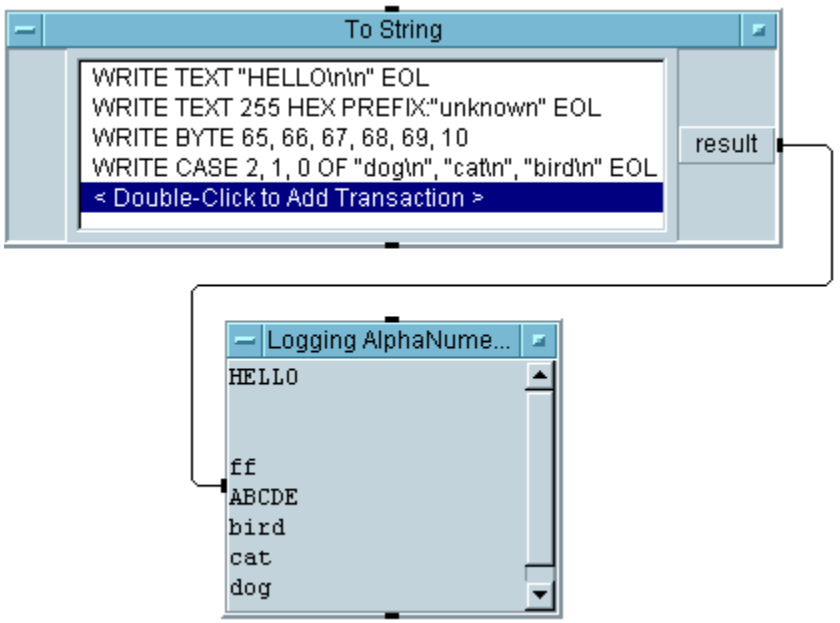


図4-11. To String**の使用例**

トランザクション・ベース・オブジェクトの使用

このセクションでは、トランザクション・ベース・オブジェクトの使用に関する指針について、実行規則とオブジェクト構成を含めて説明します。

実行規則

トランザクションI/Oオブジェクトは、VEEプログラムの一般的な伝搬規則すべてに従います。これに加えて、トランザクション自体に関する規則がいくつかあります。

1. トランザクションの実行はいちばん上のものから始まり、下に向かって順番に進みます。

2. リスト中の各トランザクションの実行が完全に終了してから、次のトランザクションの実行が始まります。1つのオブジェクトの中のトランザクションが並行して実行されることはありません。同様に、1つのソースまたは出力先に同時にアクセスできるトランザクション・オブジェクトは1つだけです。

3. 同じソースまたは出力先にアクセスするトランザクション・ベースの I/O オブジェクトが同じプログラムの別のスレッドまたは同じスレッドに存在してもかまいません。

ファイル関連のオブジェクトに関しては、1つのファイルにつき1つの読取りポインタと1つの書込みポインタが存在します。同じファイルにアクセスするすべてのオブジェクトが同じポインタを共有します。

オブジェクト構成

最も一般的な場合、トランザクションの結果は実際には下記の2つの条件で決まります。

- トランザクションの仕様

- オブジェクト・メニューの`Properties`からアクセスされる設定

`Properties`の設定については、ほとんどの場合デフォルト設定が使用できるため、変更の必要はありません。

データを出力するトランザクションI/Oオブジェクト(Direct I/Oを除く)では、データ・フォーマットの編集のためのタブがPropertiesダイアログ・ボックスに追加されます。このダイアログ・ボックスから、さまざまな設定の表示と編集が可能です。

注記

Direct I/Oオブジェクトのオブジェクト・メニューにはShow Config機能があり、構成設定の表示が可能です(編集はできません)。Direct I/Oオブジェクトの構成を編集するには、I/O⇒Instrument Managerを使います。

トランザクションI/Oオブジェクトのオブジェクト・メニューでPropertiesをクリックすると、図4-12のようなPropertiesダイアログ・ボックスが表示されます。

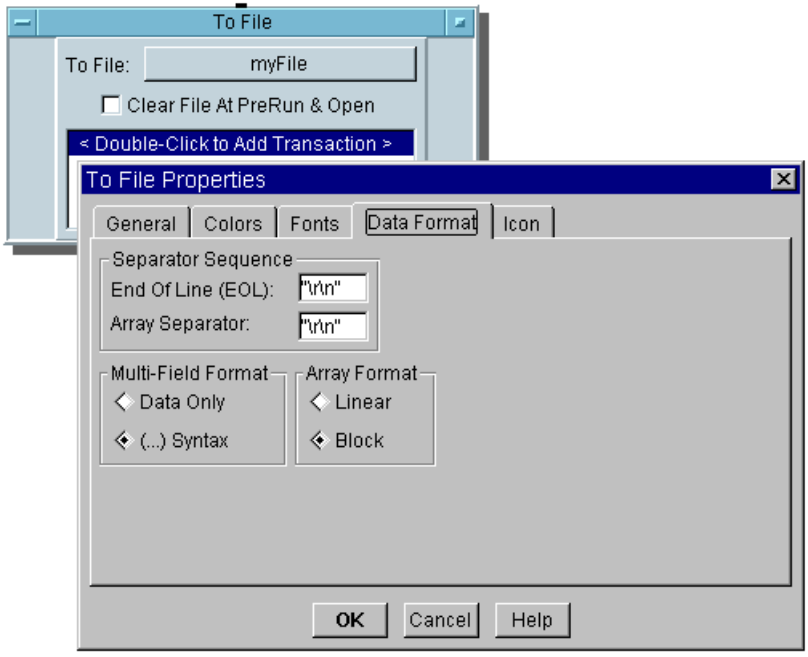


図4-12. Propertiesダイアログ・ボックス

Propertiesダイアログ・ボックスのData Formatタブには、WRITEトランザクションでのデータの出力方法に関する設定が存在します。End Of Line (EOL)は、EOL ONがセットされたすべてのWRITEトランザクションに影響します。これ以外のData Formatフィールドは、WRITE TEXTトランザクションだけに影響します。

以下の部分では、Data Formatタブの各フィールドについて詳しく説明します。

End Of Line (EOL) フィールド

End Of Line (EOL)フィールドは、EOL ONを使用したWRITEトランザクションの最後に送信される文字を指定します。このフィールドのエントリは、0個以上の文字を2重引用符で囲んだものです。2重引用符とは、ASCIIの10進34のことです。EOL内では、任意のASCII文字と、表4-4に示すエスケープ文字が認識されます。

Array Separator フィールド

Array Separatorフィールドは、WRITE TEXTトランザクションで出力される配列の要素を区切る文字列を指定します。このフィールドのエントリは、2重引用符で囲まれている必要があります。2重引用符とは、ASCIIの10進34のことです。任意のASCII文字と、表4-4に示すエスケープ文字が、VEEによってArray Separatorとして認識されます。

Direct I/OオブジェクトのWRITE TEXT STRトランザクションで配列を出力する場合は特別です。この場合、Array Separatorフィールドの値は無視され、配列の要素はラインフィード文字(ASCIIの10進10)によって区切られます。この動作は、大部分の機器の要求に適合しています。

Multi-Field Format フィールド

Multi-Field Formatフィールドは、WRITE TEXTトランザクションでのマルチフィールド・データ型のフォーマット・スタイルを指定します。VEEのマルチフィールド・データ型としては、Coord、Complex、PComplex、Spectrumがあります。他のデータ型や他のフォーマットはこの設定に影響されません。

マルチフィールド・フォーマットとして(...) Syntaxを指定すると、マルチフィールド・アイテムのそれぞれが括弧で囲まれます。Data Onlyを指定すると、括弧がなくなりますが、区切りのカンマは残されます。例えば複素数2+2jは、(...) Syntaxでは(2,2)と書かれ、Data Onlyシンタックスでは2,2と書かれます。

VEEではマルチフィールド・データ型の配列が使用できます。例えば、Complexデータの配列を作成できます。この場合、Multi-Field Formatを(...) Syntaxに設定すると、配列は次のように出力されます。

```
(1,1)array_sep(2,2)array_sep ...
```

ここで、*array_sep*はArray Separatorフィールドに指定された文字です。

Array Formatフィールド

Array Formatフィールドは、多次元配列を出力する方法を決定します。例えば、数学では行列を次のように表記します。

```
1 2 3
4 5 6
7 8 9
```

VEEはこの行列を、Array Formatの設定に応じて2つの方法のどちらかで出力します。以下の2つの例では、End Of Line (EOL)が"\n"(改行文字)に設定され、Array Separatorが" "(スペース)に設定されているものとします。

```
1 2 3    Block Array Format
4 5 6
7 8 9
1 2 3 4 5 6 7 8 9    Linear Array Format
```

どちらの配列フォーマットでも、配列の要素の間はArray Separator文字で区切られます。Block Array Formatでは、配列の各行をEnd Of Line (EOL)文字で区切る操作が加わっています。

より一般的な例(3次元以上の配列)では、Block Array Formatの場合、右端以外の添字が変化するたびにEnd Of Line (EOL)文字が出力されます。

例えば、3次元配列A[x,y,z]を下記のトランザクションでBlock配列フォーマットで出力した場合、

```
WRITE TEXT A
```

xまたはyの値が変化するたびにEnd Of Line (EOL)文字が出力されます。

Aの各次元のサイズが2の場合、要素は下記の順序で出力されます。

```
A[0,0,0]  A[0,0,1]<EOL文字>
A[0,1,0]  A[0,1,1]<EOL文字>
<EOL文字>
A[1,0,0]  A[1,0,1]<EOL文字>
A[1,1,0]  A[1,1,1]<EOL文字>
```

A[0,1,1]が出力されたあとでは、xとyが同時に変化するため、2個の<EOL文字>が出力されます。

正しいトランザクションの選択

このセクションでは、各種のI/Oオブジェクトとそれがサポートするトランザクションの一覧を示します。また、特定の目的に適したオブジェクトとトランザクションを選択する手順を説明します。トランザクションのエンコーディングとフォーマットの詳細については、付録A「I/Oトランザクション・リファレンス」を参照してください。図4-5と図4-6は、VEEで使用できるトランザクション・ベース・オブジェクトとサポートされる動作の一覧です。

表4-5. トランザクション・ベース・オブジェクトの一覧

オブジェクト	説明
To File	ファイルにデータを書き込みます。
From File	ファイルからデータを読み取ります。
To String	VEEコンテナにテキストを書き込みます。
From String	VEEコンテナからテキストを読み取ります。
Execute Program (UNIX)	実行可能ファイルからプロセスを生成して、生成されたプロセスの標準入力に書き込み、標準出力から読み取ります。Execute Program (PC) はトランザクション・ベースではありません。
To Printer	VEEのテキスト・プリンタにテキストを出力します。
To StdOut To StdError From StdIn	VEEの標準出力(PCではファイル)にデータを書き込みます。 VEEの標準エラー出力(PCではファイル)にデータを書き込みます。 VEEの標準入力(PCではファイル)からデータを読み取ります。
Direct I/O	GPIB、VXI、シリアル、GPIOの機器と直接通信します。
MultiInstrument Direct I/O	1つのオブジェクトで複数のGPIB、VXI、シリアル、GPIOの機器と直接通信します。
Interface Operations	GPIBまたはVXIインタフェース上で低レベルのバス・コマンドおよびデータ・バイトを伝送します。

表4-5. トランザクション・ベース・オブジェクトの一覧

オブジェクト	説明
To/From Named Pipe (UNIX) To/From Rocky Mountain Basic (UNIX)	名前付きパイプとの間でデータをやりとりすることにより、プロセス間通信を実現します。 HP-UXの名前付きパイプを使って、Rocky Mountain Basicプロセスとの間でデータをやりとりします。
To/From DDE (PC)	Microsoft Windowsで動作しているプログラムとの間で動的にデータを交換します。
To/From Socket	ネットワークに接続されたコンピュータ・システム同士で、プロセス間通信を使ってデータを交換します。

表4-6. トランザクション・タイプの一覧

動作	説明
EXECUTE	トランザクション・ベース・オブジェクトに対応するファイル、機器、インタフェースを制御する低レベルのコマンドを実行します。この動作は、ファイル・ポインタの調整、バッファのクリア、ファイルやパイプのクローズ、ハードウェア・インタフェースの低レベル制御などに用いられます。
WAIT	指定された時間だけ待ってから次のトランザクションを実行します。GPIB機器、メッセージ・ベースのVXI機器、I-SCPIでサポートされるレジスタ・ベースのVXI機器に対するDirect I/Oの場合、特定のシリアル・ポール応答をWAITで待つこともできます。
READ	対応するオブジェクトからデータを読み取ります。
WRITE	対応するオブジェクトにデータを書き込みます。
SEND	IEEE 488で定義されたバス・メッセージ(コマンドおよびデータ)をGPIBインタフェースに送信します。

正しいオブジェクトとトランザクションの選択

1. I/O動作のソースと出力先、およびデータ伝送のフォーマットを決めます。
2. 目的のソースと出力先をサポートするオブジェクト・タイプを表4-5から求めます。
3. 適切なトランザクションのタイプを表4-6から求めます。
4. トランザクションの残りの仕様(エンコーディング、フォーマットなど)を決めるには、付録A「I/Oトランザクション・リファレンス」を参照してください。

例: オブジェクトとトランザクションの選択

例えば、2列のテキスト・データを記録したファイルを読み取る場合を考えます。それぞれの列にはタイム・スタンプと実数が記録されており、間はホワイト・スペースで区切られています。各行の末尾には改行文字があります。ファイル内容の一部を下に示します。

```
14:18:00        1.001
14:18:30        -2.002
14:19:00        1.0E-03 . . .
```

先に記したオブジェクトとトランザクションの選択手順に基づき、この問題の解決のために下記の手順を実行します。

1. ソースはテキスト・ファイルです。データを構成するのは、24時間制の時-分-秒の形式のタイムスタンプと、10進の科学的記数法で記された符号付き実数です。
2. 表4-5より、ファイルの読取りに使用するオブジェクトは`From File`です。
3. 表4-6より、ファイルからデータを読み取るトランザクションのタイプは`READ`です。
4. 必要なトランザクションは下記の通りです。

```
READ TEXT x TIME
READ TEXT y REAL
```

To StringおよびFrom Stringの使用

フォーマットされたテキストをトランザクションで作成するには、To Stringを使います。テキストはVEEコンテナに書き込まれます。

フォーマットされたテキストをVEEコンテナから読み取るには、From Stringを使います。

To Stringオブジェクトのすべてのトランザクションから生成される文字列が1個だけの場合、出力コンテナはTextスカラです。複数の文字列がTo Stringのトランザクションから生成される場合、出力はTextの1次元配列です。

EOL ONを使用しているWRITEトランザクションは、現在の出力文字列を常に終了します。これにより、次のトランザクションは出力コンテナの次の配列要素に書き込むことになります。

EOL OFFで終わるWRITEトランザクションは出力文字列を終了しないので、次のWRITEトランザクションで出力された文字は現在の文字列の末尾に追加されます。To Stringの最後のトランザクションの場合、EOL設定と無関係に現在の文字列は常に終了されます。

To Stringではほとんどの場合にWRITE TEXTトランザクションを使用するのが適切です。TEXT以外のエンコーディングの詳細については、付録A「I/Oトランザクション・リファレンス」を参照してください。

From Stringは、READ TEXTトランザクションの構成に応じて、Textスカラも配列も読み取ることができます。READ TEXTは、フォーマットに応じて、EOLが見つかったときに読取りを終了するか、EOLを読み出して読取りを続けます。フォーマットの詳細については、付録A「I/Oトランザクション・リファレンス」を参照してください。

注記

READとWRITEの互換性

一般に、データを正しく読み取るには、そのデータがどのように書き込まれたかを知る必要があります。特に、データがバイナリ・フォーマットで、データを直接調べることによりそのフォーマットを知ることができない場合にはこれが重要です。データは書き込まれたのと同じフォーマットで読み取る必要があります。

ファイルとの通信

このセクションでは、ファイルとの通信に関する指針について、ファイル・ポインタの使用とデータのインポートを含めて説明します。

ファイル・ポインタの使用

VEEでは、**1つのファイルに対して**1つずつの読取りポインタと書込みポインタが用いられます。これはファイルにアクセスしているオブジェクトがいくつあっても変わりません。読取りポインタは、次に読み取るデータの位置を示します。同様に、書込みポインタは次のアイテムが書き込まれる位置を示します。表4-7はオブジェクトとソース/出力先ファイルを示します。

表4-7. オブジェクトとソース/出力先

ソースまたは出力先	オブジェクト
データ・ファイル	To File、From File
標準入力	From StdIn
標準出力	To StdOut
標準エラー	To StdErr

これらのポインタの位置は、下記の動作によって変更されます。

- READ、WRITE、EXECUTE動作
- To Fileのオープン・ビューにあるClear File at PreRun & Open設定

同じファイルにアクセスしているすべてのオブジェクトが、同じ読取り/書込みポインタを共有します。別のスレッドや別のコンテキストにオブジェクトがあっても、これは変わりません。

下記の条件のどれかが満たされた場合、ファイルは読取りおよび書込み用にオープンされます。

- そのファイルにアクセスする最初のオブジェクトが、PreRunのあと最初に動作したとき。これが最も普通の場合です。
- ファイル名を指定する任意指定の制御入力端子に新しいデータが届いたとき。これは比較的まれな場合です。

読取りポインタ

From Fileがファイルをオープンした時点では、読取りポインタはファイルの先頭にあります。その後のREADトランザクションは、READ動作に必要なだけファイル・ポインタを進めます。From FileオブジェクトでEXECUTE REWINDトランザクションを使うことにより、任意の時点で読取りポインタをファイルの先頭に戻すことができます。この動作によってファイルのデータは変更されません。

書込みポインタ

書込みポインタの初期位置は、To Fileのオープン・ビューにあるClear File at PreRun & Open設定に依存します。Clear File at PreRun & Openをオンにすると、ファイルをオープンしたときにファイルの内容は消去され、書込みポインタはファイルの先頭に置かれます。この設定をオフにすると、書込みポインタはファイルの末尾に置かれ、書き込んだデータは末尾に付加されます。

EXECUTE REWINDまたはEXECUTE CLEARトランザクションを使うことにより、任意の時点で書込みポインタをファイルの先頭に置くことができます。REWINDの場合、ファイル内の既存のデータは保持されます。ただし、新しいデータを書き込むと、書き込んだ位置の古いデータは上書きされます。CLEARの場合、ファイル内の既存のデータは消去されます。

注記

To DataSetおよびFrom DataSetオブジェクトも、ファイルごとに1つずつの読取り/書込みポインタをTo FileおよびFrom Fileオブジェクトと共有します。ただし、To DataSet/From DataSet動作とTo File/From File動作を同じファイルに対して混用することは推奨されません。

ファイルのクローズ

To Fileが書き込んだデータは、最後のトランザクションの実行が終了し、すべての出力端子がアクティブになったときにオペレーティング・システムに対して書き込まれることが保証されます。

UNIXオペレーティング・システムは、オペレーティング・システムのバッファにあるデータを定期的に(通常15～30秒程度の間隔で)ディスクに書き込みます。このバッファリング動作はオペレーティング・システムの仕様であり、VEEに固有のものではありません。

VEEはPostRun時にすべてのファイルを自動的にクローズします。PostRunが発生するのは、アクティブなスレッドがすべて実行を終了したときです。

プログラムからファイルをクローズするには、To FileおよびFrom Fileの両方でEXECUTE CLOSEトランザクションを実行します。これにより、他のプロセスが作成したファイルを連続的に読み書きすることができます。

プログラムからファイルを削除するには、EXECUTE DELETEトランザクションを使用します。これはテンポラリ・ファイルを削除するのに便利です。

図4-13は、EXECUTE CLOSEを使用した例です。このプログラムは、examplesディレクトリにあるファイルmanual48.veeに保存されています。

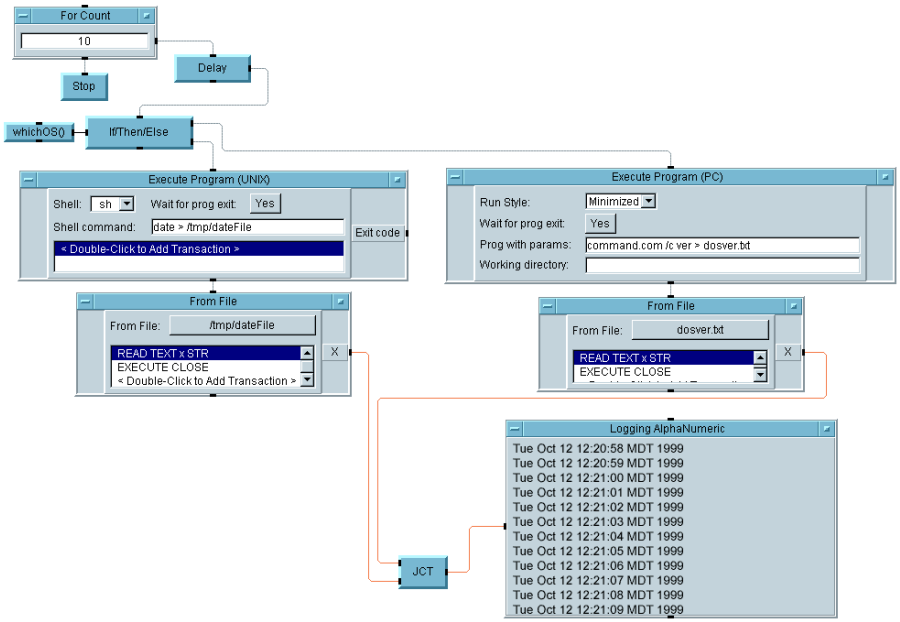


図4-13. EXECUTE CLOSE**トランザクションの使用**

図4-13で、Execute Programはシェル・コマンド(date)を実行し、このコマンドがテンポラリ・ファイル(/tmp/dateFile)を作成して日付と時間を書き込みます。同じスレッドで、From FileがREAD TEXT x STRトランザクションを使ってそのファイルから日付を読み取ります。サブスレッドがFor Countによって複数回実行されるため、EXECUTE CLOSEトランザクションが必要です。

次に実行されるExecute Programがファイルを上書きします。ところが、From Fileはファイルを1回しかオープンしないので、From Fileの2回目の実行では読取りポインタが無効になっています。Execute Programがファイルを作り直しているため、読取りポインタはすでにファイルを指していません。このためエラーが発生します。

データを読み取ったあとで、From FileはEXECUTE CLOSEトランザクションを使ってファイルをクローズする必要があります。EXECUTE CLOSEトランザクションにより、次の実行でFrom Fileはファイルを再オープンするようになります。

図4-13の例で、EXECUTE CLOSEトランザクションをNOPによってコメントアウトすれば、エラーが発生するのを見ることができます。エラーの内容はEnd of file or no data foundとなります。NOPを削除するとプログラムは正常に動作します。

EOFデータ出力

From Fileは、EOF(End-Of-File)という名前の固有のデータ出力端子をサポートします。この端子は、ファイルの末尾からデータを読み取ろうとしたときにアクティブになります。EOF端子は、長さがわからないファイルを読み取るときに便利です。

121ページ「トランザクション・データの読取り」で説明したREAD TO END機能を使っても、長さがわからないファイルを読み取ることができます。ただしこの場合、ファイルの内容がすべて1個のVEEコンテナに入ります。ファイルの内容を1要素ずつ、それぞれ1個のコンテナに読み取りたい場合には、EOF端子を使います。

図4-14は、EOFの代表的な使い方を示します。読み取るファイルには、未知の長さのX-Yデータのリストが入っています。ファイルの内容は下記のようなものです。

```
1.0
5.5
2.1
8
.
.
.
```

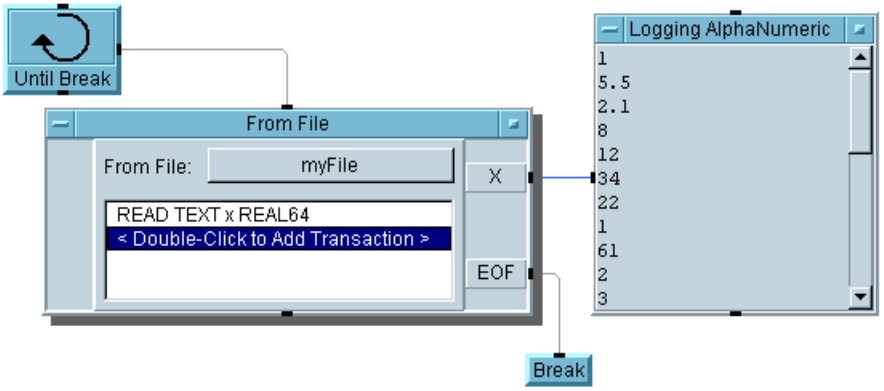


図4-14. EOFを使ったファイル読取りの代表的な例

データのインポート

VEEにはデータの解析と表示のための便利な環境が用意されているので、他のプログラムのデータをVEEにインポートしたい場合があります。他のソフトウェア・アプリケーションのデータをインポートするための一般的な手順は下記の通りです。

1. テキスト・ファイル(ASCIIファイル)にデータを保存します。
2. テキスト・エディタでデータを表示し、データのフォーマットを調べます。
3. `From File`オブジェクトで`READ TEXT`トランザクションを使ってデータ・ファイルを読み取ります。

X-Y値のインポート

よくある問題の1つとして、未知の数の`X/Y`値を記録したテキスト・ファイルを読み取ってプロットする場合があります。図4-15のプログラムは、この問題を解決するものです。

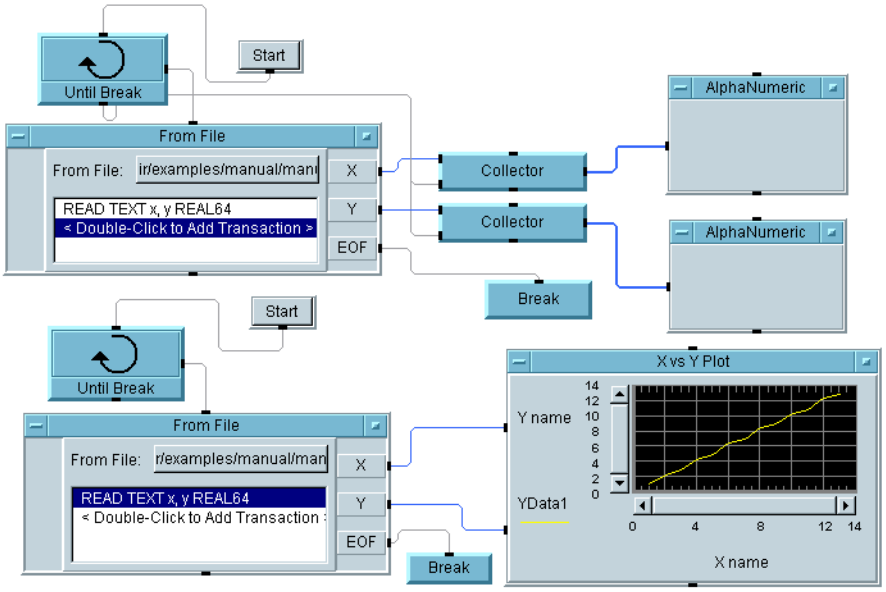


図4-15. XY**値のインポート**

図4-15のプログラムは、examplesディレクトリにあるファイルmanual29.veeに保存されています。

READ TEXT REAL64トランザクションは、符号、小数点、指数など、Y値のさまざまな記数法に対応できます。データ・ファイルの一部を下に示します。

```
.
.
.
8     8.555555
9     9e0
10    1.05e+01
11    +11.
12    12.5
13    1.3E1
```

波形のインポート

他のソフトウェア・アプリケーションは、さまざまな形式のテキスト・ファイルで波形を保存します。ファイルの内容は一般に、波形の属性を記述するいくつかの値

と、Y値の1次元配列から構成されます。このセクションでは、下記の形式で保存された波形をインポートする方法を説明します。

- 固定フォーマットのファイル・ヘッダ。ファイルの先頭の固定位置に波形属性が記述され、そのあとにYデータの1次元配列が記載されます。
- 可変フォーマットのファイル・ヘッダ。ファイルの先頭に可変数の属性が記述され、そのあとにYデータの1次元配列が記載されます。属性の位置は特殊なテキスト・トークンで示されます。

固定フォーマットのヘッダ. 図4-16のプログラムで読み取られるデータ・ファイルの一部を示します。

```
NAME        Noise1
START_TIME  0.0
STOP_TIME   1.0E-03
SAMPLES     32
DATA
                .243545
              .2345776
.
.
.
```

これは固定フォーマットのヘッダなので、NAMEやSAMPLESなどのラベルは無意味です。波形属性は**必ず存在し、同じ位置にあります。**図4-16に波形データ・ファイルを読み取るプログラムを示します。

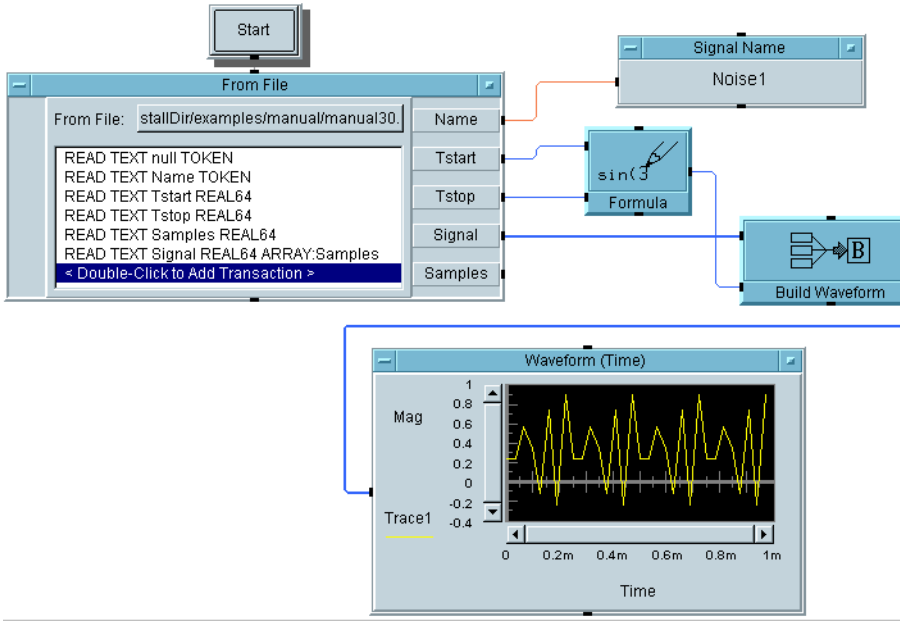


図4-16. 波形ファイルのインポート

図4-16のプログラムは、examplesディレクトリのファイルmanual30.veeに保存されています。

ここではFrom Fileのトランザクションで大半の作業が行われています。各トランザクションの役割を以下に示します。

1. 最初のトランザクションは、NAME ラベルを取り除きます。波形の名前を読み取る前にこれを行っておかないと、NAMEとNoise1がいっしょに1つの文字列として読み取られてしまいます。

2. 2番目のトランザクションは、波形の文字列名を読み取ります。

3. 3～5番目のトランザクションは、指定された数量を読み取ります。ファイル中で数値の前に数値と無関係な文字がある場合、すべて読み捨てられます。

4. 6番目のトランザクションは、前のトランザクションで求められたARRAY SIZEを使ってYデータの1次元配列を読み取ります。このトランザクションで使用するために、Samplesを出力端子にする必要があります。

可変フォーマットのヘッダ. 図4-17で読み取るプログラムの一部を下に示します。

```
First Line Of File
<MARKER1> 1 2 3
<MARKER2> A B C

<DATA>

1    1.1
2    2.2
3    2.9
.
.
.
```

この例では、ファイル内のデータの正確な内容と位置があらかじめ知られていません。このファイルについてわかっているのは、特殊なテキスト・マーカ<DATA>のあとにXY値のリストが存在するということだけです。

例を簡単にするため、図4-17のプログラムは<DATA>に対応するデータだけを探します。実際のアプリケーションでは、このほかにいくつかのマーカを探すことになるでしょう。

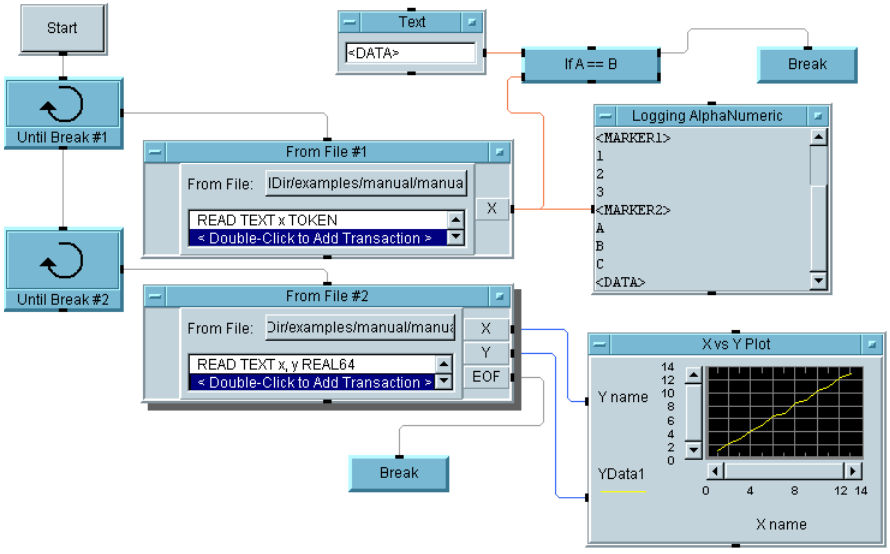


図4-17. 波形ファイルのインポート

図4-17のプログラムは、examplesディレクトリにあるファイルmanual31.veeに保存されています。

From File #1は、トークン(ホワイト・スペースで区切られたワード)を1個ずつ読み取り、<DATA>を探します。<DATA>が見つかったら、From Fileはファイルの終わりまでXYペアを読み取ります。

プログラムとの通信(UNIX)

Rocky Mountain Basic

このセクションでは、UNIXでのプログラムとの通信に関する指針について説明します。下記の内容があります。

- Execute Programの使用(UNIX)
- To/From Named Pipeの使用(UNIX)
- To/From Socketの使用(UNIX)
- Rocky Mountain Basicオブジェクトの使用(HP-UX)

表4-8にプログラムと関連オブジェクトを示します。

表4-8. プログラムと関連オブジェクト(UNIX)

プログラム	オブジェクト
シェル・コマンド	Execute Program (UNIX)
Cプログラム	Execute Program (UNIX) To/From Named Pipe (UNIX) To/From Socket
Rocky Mountain Basic	Initialize Rocky Mountain Basic (UNIX) To/From Rocky Mountain Basic (UNIX)

Execute Programの使用(UNIX)

オペレーティング・システムのコマンド行から通常実行する作業をVEEプログラムから実行できます。このためにはExecute Program (UNIX)オブジェクトを使います。図4-18に、Execute Program (UNIX)オブジェクトを示します。Execute Program (UNIX)を使うと、下記のような任意の実行可能ファイルを実行できます。

- コンパイルされたCプログラム
- シェル・スクリプト
- ls、grepなどのUNIXのシステム・コマンド

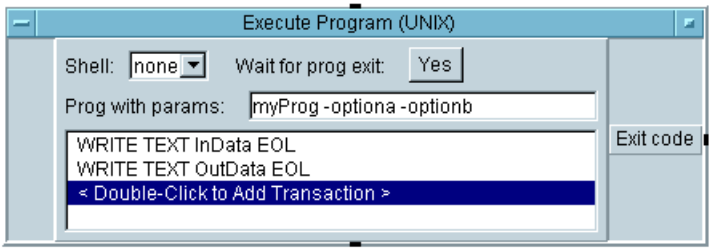


図4-18. `Execute Program (UNIX)` **オブジェクト**

Execute Program (UNIX)のフィールド

以下の部分では、`Execute Program (UNIX)` のオープン・ビューに表示されるフィールドについて説明します。

Shell. `Shell`は、UNIXシェルの名前(`sh`、`csh`、`ksh`など)を指定します。`Shell`フィールドを`none`に指定した場合、`Prog with params`フィールドの最初のトークンが実行可能ファイルの名前であり、その後のトークンがコマンド行パラメータであると仮定されます。実行可能ファイルはVEEの子プロセスとして直接起動されます。他のすべての条件が同じなら、`Shell`を`none`に設定した場合に`Execute Program (UNIX)` の実行は最も高速になります。

`Shell`フィールドにシェルを指定した場合、VEEは指定したシェルに対応するプロセスを生成します。`Prog with params`フィールドの内容が指定したシェルに渡されて解釈されます。一般には、さらに別のプロセスがシェルから生成されます。

Wait for Prog Exit. Wait for prog exitは、Execute Programオブジェクトの動作が終了してデータ出力がアクティブになるタイミングを指定します。Wait for prog exitをYesに設定すると、VEEは下記のように動作します。

1. Execute Program (UNIX)オブジェクトに対応する子プロセスがアクティブであるかどうかをチェックします。アクティブなプロセスがなければ、生成します。
2. Execute Programオブジェクトに指定されたトランザクションをすべて実行します。
3. 子プロセスに対するパイプをすべてクローズし、子プロセスに EOF(End-Of-File)を送信します。
4. 子プロセスが終了するのを待ってから、Execute Program (UNIX)オブジェクトの出力ピンをアクティブにします。Shellフィールドがnoneに設定されていない場合、この条件を満たすにはシェルが終了する必要があります。

Wait for prog exitをNoに設定すると、VEEは下記のように動作します。

1. Execute Program (UNIX)オブジェクトに対応する子プロセスがアクティブであるかどうかをチェックします。アクティブなプロセスがなければ、生成します。
2. Execute Programオブジェクトに指定されたトランザクションをすべて実行します。
3. Execute Programオブジェクトのデータ出力ピンをアクティブにします。子プロセスはアクティブのままであり、対応するパイプはまだ存在します。

他の条件がすべて等しい場合、Wait for prog exitをNoに設定した方がExecute Program (UNIX)の実行は高速になります。

Prog With Params. Prog with paramsは下記のどちらかを設定します。

1. 実行可能ファイルの名前とコマンド行パラメータ(Shellがnoneに設定されている場合)
2. シェルに渡して解釈させるコマンド(Shellがnone以外に設定されている場合)

Prog with paramsフィールドの代表的な設定を下に示します。

シェル・コマンドを実行する場合(Shellをkshに設定):

```
ls -t *.dat | more
```

コンパイルされたCプログラムを実行する場合(Shellをnoneに設定):

```
MyProg -optionA -optionB
```

Prog with paramsフィールドでシェルに依存する機能を使用する場合、目的の結果を得るにはシェルを指定する必要があります。シェル依存の機能のうち一般的なものを下に示します。

- 標準入出力のリダイレクト(<および>)
- ワイルドカードによるファイル名展開(*、?、[a-z])
- パイプ(|)

シェル・コマンドの実行

Execute Program (UNIX)を使って、ls、mkdir、rmなどのシェル・コマンドを実行できます。図4-19に示すのは、ディレクトリに存在するファイルのリストをVEEプログラムから得る方法の例です。

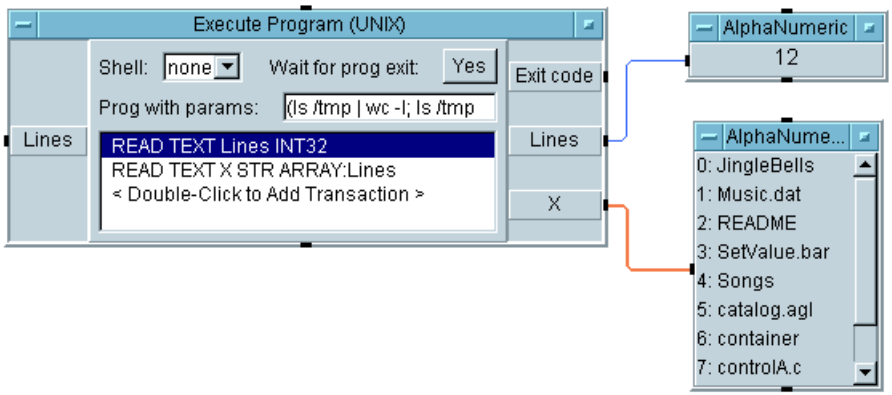


図4-19. Execute Program (UNIX)**でシェル・コマンドを実行する例**

図4-19のプログラムは、examplesディレクトリにあるファイルmanual32.veeに保存されています。

図4-19でExecute Program (UNIX)は、ファイル名の数を求めるために、/tmpディレクトリにあるファイル名を1列にリストしたものを取得し(ls -1)、このリストを

行カウント・プログラム(wc -l)にパイプで渡しています。パイプを使用するため、Prog with paramsフィールドのコマンドをシェルに渡して解釈させる必要があります。Shellフィールドはshに設定します。行の数はREAD TEXTトランザクションで読み取られ、Linesという名前の出力端子に送られます。

2番目のトランザクションは、/tmpディレクトリにあるファイルのリストを読み取ります。2番目のトランザクションは、最初のトランザクションで求められた数だけの行を読み取ります。シェル・コマンドをセミコロンで区切ることにより、2つのコマンドを実行することをシェルに伝えます。

Execute Program (UNIX)では、Wait for prog exitがYesに設定されています。この例ではこの設定はあまり重要ではありません。シェル・コマンドは1回しか実行されないからです。VEEプログラムが実行を続ける間もExecute Program (UNIX)で生成したプロセスをアクティブにしておきたい場合は、Noの設定を使います。

図4-20は、ディレクトリに存在するファイルのリストをVEEプログラムから得るもう1つの方法を示します。

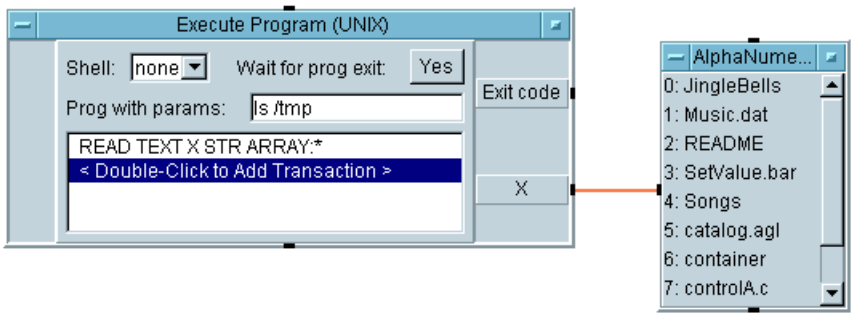


図4-20. Execute Program (UNIX)**でシェル・コマンドを実行し、READ TO ENDを使う例**

このプログラムは、examplesディレクトリにあるファイルmanual50.veeに保存されています。

図4-20のVEEプログラムは、図4-19よりも簡単な方法で/tmpディレクトリの内容を表示しています。

図4-20で、Execute Program (UNIX)のProg with paramsフィールドにはls /tmpという1つのコマンドだけが入っています。図4-19のプログラムと異なり、ディレクトリに存在するファイルの数をあらかじめ知る必要はありません。これは、121ページ「トランザクション・データの読取り」で説明したREAD TO END機能をトランザクションREAD TEXT x STR ARRAY:*で使っているからです。

シェル・コマンドの実行が終わると、ファイルのリストを読み取るためにExecute Program (UNIX)が使っているパイプがシェルによってクローズされます。これによりEOF(End-Of-File)が送られ、トランザクションが終了します。

Cプログラムの実行

図4-21のプログラムは、Cプログラムとデータを共有する方法の1つとして、Cプログラムのstdinとstdoutを使用した例です。この例でCプログラムは、VEEから実数を読み取り、数値に1を足して結果を返します。

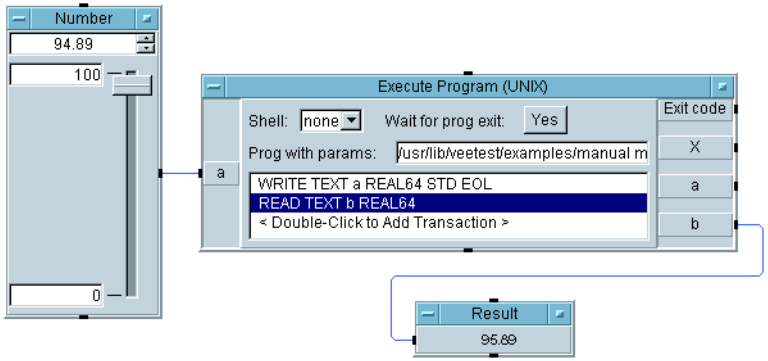


図4-21. Execute Programからプログラムを実行する例

図4-21のプログラムは、examplesディレクトリにあるファイルmanual33.veeに保存されています。

図4-22に示すのは、図4-21のVEEプログラムから呼び出されるCプログラムのリストです。

図4-22のプログラム・リストでは、setbufとfflushを使ってCプログラムのstdoutからデータを強制的に書き出しています。実際には、setbufとfflushのどちらか一方だけで十分です。setbuf(file,NULL)を使うと、**ファイル**に対するすべての出力についてバッファリングがオフになります。fflush(file)を使うと、**ファイル**に対してバッファリングされているデータがすべて書き出されます。

```
#include <stdio.h>
main ()
{
   int c;
   double val;
   setbuf(stdout,NULL);   /* stdoutÇÃÉoÉbÉtÉ@ÉäÉìÉOÇšÉIÉtÇ…Ç²ÇÈ*/

   while (((c=scanf("%lf",&val)) != EOF) && c > 0){
      fprintf(stdout,"%g\n",val+1);
      fflush(stdout);      /* ã¦êßìIÇ…VEEÇ…èoóÕÇšï‘Ç²*/
   }
   exit(0);
}
```

図4-22. Cプログラム・リスト

To/From Named Pipeの使用(UNIX)

To/From Named Pipeは、プロセス間通信を必要とする**上級ユーザ**のためのツールです。UNIXの名前付きパイプの動作はやや複雑なので、初心者にはお勧めできません。名前付きパイプとプロセス間通信について調べるには、本章の先頭にある「参考文献」の注記を参照してください。

すべてのTo/From Named Pipeオブジェクトには、読取りパイプと書込みパイプ用の共通のデフォルト名が入っています。読取りまたは書込みを行うパイプの名前を必ず正しく指定してください。ディスクレス・ワークステーションでVEEを動作させる場合にこれが問題になることがあります。プログラムの名前付きパイプが他のユーザからアクセスされることがないように注意してください。

VEEは必要に応じて自動的にパイプを作成します。VEE環境外部でパイプを作成する必要はありません。

名前付きパイプの使用に関するヒント

- VEEとパイプの反対側のプロセスとの間で、必ず同じタイプのデータ型を使用するようにします。特に、受信側がちょうど必要とするだけのデータを送信側から送り、データが読み取られずにパイプに残ることがないようにします。

- データをVEEに送信するときは、バッファリングなしの出力を使用するか、出力バッファをフラッシュするようにします。このためには、非バッファリングI/Oを使う(write)、バッファリングをオフにする(setbuf)、バッファを明示的にフラッシュする(fflush)という3つの方法があります。

VEEに対する出力バッファリングを制御するためのC関数の例を以下に示します。

`setbuf(out_pipe1,NULL)` **出力バッファリングをオフにします。**

または

`fflush(out_pipe1)` **VEEにデータをフラッシュします。**

または

`write(out_pipe2,data,n)` **バッファリングなしでデータを出力します。**

ここで、*out_pipe1*はファイル・ポインタ、*out_pipe2*はファイル・デスクリプタで、ともにTo/From Named Pipeで指定された読取りパイプを指します。

VEEがパイプにデータを書き込む際には、同様のフラッシュ動作が自動的に行われます。下記の条件のどちらかが満たされたときに、fflushに相当する動作がVEEによって実行されます。

- オブジェクトの最後のトランザクションが実行されたとき

- WRITEトランザクションのあとにWRITEトランザクション以外のものが実行されたとき

To/From Named Pipeは、121ページ「トランザクション・データの読取り」で説明したREAD TO ENDトランザクションをサポートします。また、EXECUTE CLOSE READ PIPEおよびEXECUTE CLOSE WRITE PIPEトランザクションもサポートされます。これらのトランザクションは、VEEと他のプロセスとの間のプロセス間通信でやりとりするデータの量があらかじめ明示的に知られていない場合に使用します。

例えば、VEEと他のプロセスとの間で名前付きパイプによる通信を行うとします。VEEが名前付きパイプに書き込んだデータが、読取りプロセスが期待するよりも少なかった場合、読取りプロセスは名前付きパイプから十分なデータが得られるのを待ちながらハングしてしまいます。

`EXECUTE CLOSE WRITE PIPE`トランザクションを使えば、`EOF`(End-Of-File)が送信されたときに名前付きパイプがクローズされます。これにより、ほとんどの読取り関数呼出し(`read`、`fread`、`fgets`など)が`EOF`で終了するため、読取りプロセスは実行を再開でき、VEEがパイプに書き込んだデータも得ることができます。

反対に、VEEがプロセスを読み取っている場合、READ TO END機能を使った`READ`トランザクションを使えば、書込みプロセスがパイプに対して`close()`を実行し、`EOF`を送信するまで、未知の量のデータを名前付きパイプから読み取ることが可能になります。読取りが無期限にブロックするのを避けるには、`READ IOSTATUS`トランザクションを使用する方法もあります。`READ IOSTATUS`トランザクションの使い方の詳細については、付録A「I/Oトランザクション・リファレンス」を参照してください。

To/From Socketの使用

`To/From Socket`は、システム・インテグレーションのためにプロセス間通信を必要とする**上級ユーザ**のためのツールです。ソケットの動作はやや複雑なので、初心者にはお勧めできません。

ソケットを使うと、プロセス間通信(IPC)を使ってプログラムからLANをファイル・デスクリプタとして扱うことが可能となります。IPCでは、2台の異なるコンピュータ上の2つ以上のプロセスの間で2個のソケットが用いられます。`To/From Named Pipe`オブジェクトで単純なopen()/close()インタフェースが用いられるのに対して、ソケットではアドレスのエクスポートと初期コーラ/レシーバ・ストラテジを使用します。これをコネクション・オリエンテッド・プロトコルと呼びます。

コネクション・オリエンテッド・プロトコル(クライアント・サーバ・アレンジメントとも呼ばれる)では、サーバがソケットを取得し、ポート番号と呼ばれるアドレスをソケットに**バインド**します。ポート番号をバインドしたあと、サーバはブロック状態でコネクション要求が来るのを待ちます。クライアントから**コネクション**を要求するには、ソケットを取得し、2つのエレメントによってサーバを指定します。

2つのエレメントとは、サーバがソケットにバインドしたポート番号と、サーバのホスト名またはIPアドレスです。サーバのホスト名をIPアドレスに解決できない場合、クライアントはIPアドレス自体を使用する必要があります。クライアントのコネク

ション要求をサーバが受け付けると、コネクションが確立され、通常のI/O動作が始まります。図4-23に、To/From Socketオブジェクトの例を示します。

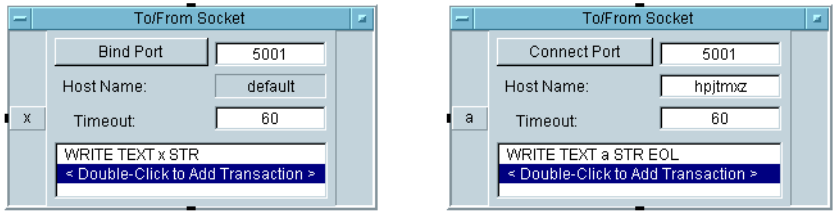


図4-23. To/From Socket**オブジェクト**

To/From Socketのフィールド

To/From Socketオブジェクトには下記の用途のためのフィールドがあります。

- リモート・コンピュータ上のバインドされたソケットと接続する
- VEEが動作しているコンピュータ上でソケットをバインドし、コネクションの発生を待つ

4つのフィールドのうち、下記の3つのフィールドの値は制御ピンとしてオブジェクトに入力する必要があります。

- Connect/Bind Port Mode
- Host name
- Timeout

以下の部分では、To/From Socketのオープン・ビューに表示されるフィールドについて説明します。

Connect/Bind Port Mode. Connect/Bind Port Modeには、モード・ボタンとテキスト・フィールドの2つのフィールドがあります。モード・ボタンは、Bind PortおよびConnect Portの間で切り替わります。テキスト・フィールドにはポート番号を入力します。使用可能なポート番号は1024～65535の整数です。

0～1023の番号は予約されており、使用するとランタイム・エラーが発生します。一般には、トランジェントと呼ばれる5000より上のポート番号を使用します。表4-9に、ソケットのポート番号として使用できる整数の範囲を示します。

表4-9. ソケットのポート番号として使用できる整数の範囲

範囲	予約された用途
0-1023	オペレーティング・システム
1024-5000	商用またはグローバル・アプリケーション[a]
5001-65535	内部用または閉じた分散アプリケーション

a. 通常は登録手続きが含まれています。

Host Name. モードがBind Portに設定されている場合、このフィールドにはVEEが動作しているホスト・コンピュータの名前が表示されます。リモート・コンピュータ上のソケットにポート番号をバインドすることはできないので、このフィールドをリモート・コンピュータのホスト名に変更することはできません。

モードがConnect Portに設定されている場合、このフィールドは編集可能です。接続先のリモート・コンピュータのホスト名またはIPアドレスを入力します。ホスト名はIPアドレスに解決できなければなりません。ネットワークでホスト名テーブルを使用してホスト名をIPアドレスに変換できない場合、15.11.29.103などのアドレスを直接入力する必要があります。

Timeout. Timeoutには、すべてのREADおよびWRITEトランザクションに対するタイムアウト秒数を整数で入力します。このタイムアウト時間は、初期接続時に、To/From Socketオブジェクトが Bind Portモードでコネクションの発生を待っている場合、あるいはConnect Portモードでコネクションが受け入れられるのを待っている場合にも適用されます。リモート・ホストが存在しないかダウンしている場合、この値は無視されます。この場合、VEEインタフェースはコネクションがフェールするまでフリーズします。これには最大1分程度かかります。

トランザクション. To/From Socketオブジェクトは、To/From Named Pipeオブジェクトと同じ通常のI/Oトランザクションを使用します。READおよびWRITEトランザクションは、すべてのデータ型をサポートします。トランザクションの詳細については、付録A「I/Oトランザクション・リファレンス」を参照してください。

データの構造

バイナリ・データはすべて、ネットワーク・バイト順序でLAN上に送出されます。これは上位バイト(MSB)が先、すなわちビッグ・エンディアンの順序です。バイナリ・トランザクションの場合、必要ならREADとWRITEの際にバイトが入れ替えられます。すなわち、VEEが接続する他のプロセスはすべてこの標準に従う必要があります。前の例では、サーバ・プロセスがリトル・エンディアンの順序で、クライアントがビッグ・エンディアンでもかまいません。バイトの交換はVEEが自動的に行います。

オブジェクトの実行

ソケットをポート番号にバインドするように設定されたTo/From Socketオブジェクトは、タイムアウト時間の間コネクションの発生を待ちます。この間、VEEの並行スレッドは実行されません。タイムアウト値を0に設定するとタイムアウトが無効になり、コネクション待ちの時間が実質的に無限になります。タイムアウト違反が起きるとエラーが発生し、VEEの実行が停止します。

コネクションが確立されると、トランザクション・リストのトランザクションが実行されます。トランザクションで指定された量と型のデータが得られるまでREAD動作はブロックし、タイムアウト時間の間待ちます。スレッドのブロックを避けるには、READ IOSTATUSトランザクションを使ってソケットにデータが存在するかどうかを調べます。

コネクションを明示的に終了するには、EXECUTE CLOSEトランザクションを使います。プログラムが実行を終了すると、VEEプログラムで確立されたソケット・コネクションはすべて破棄されます。どちらの方法でコネクションが破棄された場合も、コネクションを再確立するには、サーバとクライアントの両方のオブジェクトがバインド-受け入れおよび接続のプロトコルを繰り返す必要があります。EXECUTE CLOSEは、相互の合意の上での終了の方法として使用すべきであり、ソケットのデータをフラッシュする簡便な方法として使用すべきではありません。

複数のTo/From Socketオブジェクトがソケットを共有することもできます。同一のポート番号をバインドするすべてのオブジェクトは、同一のソケットを共有します。同一のポート番号とホスト名が指定されたオブジェクトは、同じバインドされたソケットに接続しようとし、同じソケットを共有します。コネクション確立のオーバヘッドは、共通の設定を持つオブジェクトの1つが最初に実行されたときに発生します。

To/From Socketオブジェクトの例

図4-24に示すのは、To/From Socketオブジェクトを使って、HP E1413Bを使ったデータ収集のための独立したサーバ・プロセスを実現するVEEプログラムです。このサーバは、クライアントの要求を受けて測定器を初期化し、データを収集してディ

スクに書き込み、シャットダウンして終了します。収集フェーズでは、A/Dの現在値テーブルからデータが読み取られ、クライアントに送られます。

最初に実行されるTo/From Socketオブジェクトは、Until Breakオブジェクトに接続されているもので、hpjtmxzzという名前のホスト・コンピュータ上でポート番号5001にソケットをバインドし、他のプロセスがこのソケットに接続するのを180秒間待ちます。

ここでは、タイムアウトによる停止を避けるためエラー・ピンが使われています。タイムアウトが発生すると、オブジェクトが再実行され、さらに180秒間コネクションを待ちます。コネクションが確立されると、オブジェクトはREADトランザクションでブロックし、クライアントがコマンドを送ってくるのを待ちます。この場合も、READでタイムアウトが発生すると、オブジェクトが再び実行され、READトランザクションでブロックします。

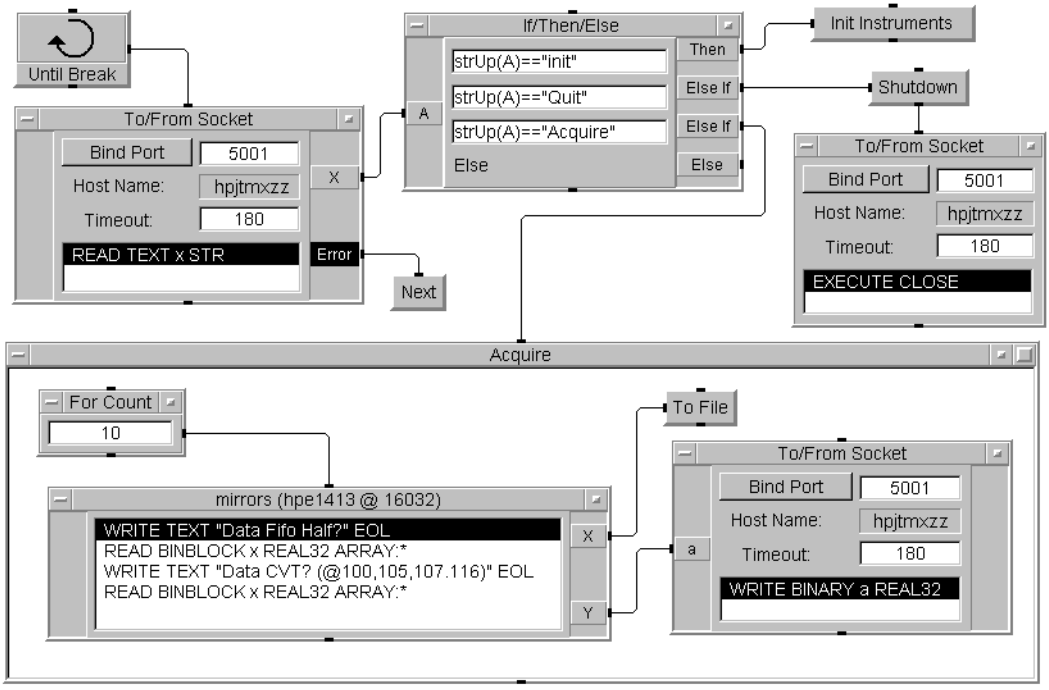


図4-24. サーバ・プロセス用にポートをバインドする To/From Socket

図4-25に示すのは、さきほど説明したサービスのクライアント側です。最初に実行されるTo/From Socketオブジェクトは、試行したコネクションが確立されるのをスリープしながら待ちます。サーバと異なり、タイムアウト・エラーが発生するとプログラムはエラーになり、停止します。最初のオブジェクトは、コマンドInitおよびAcquireを送信したあと、CVTを読み取るループを実行します。

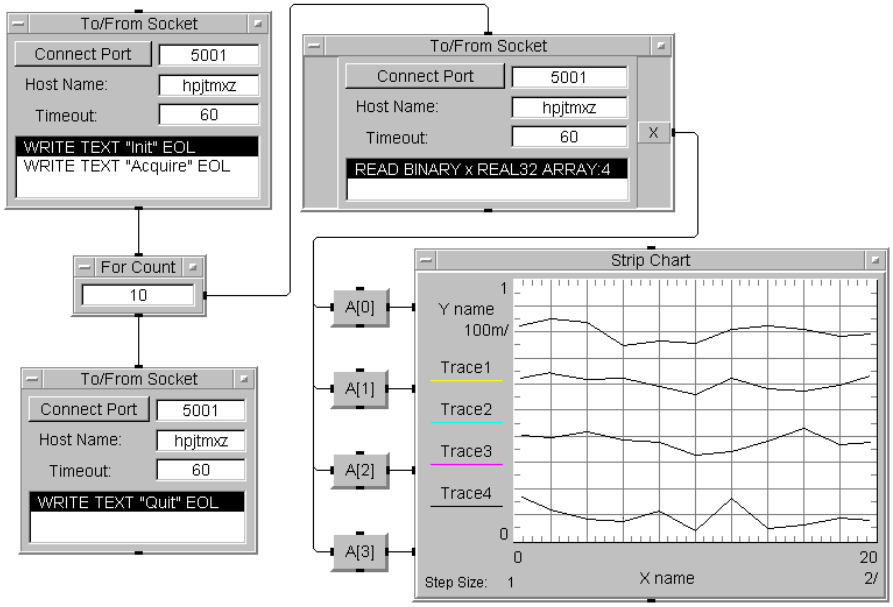


図4-25. クライアント・プロセス用のポートを接続するTo/From Socket

Rocky Mountain Basicオブジェクトの使用(HP-UX)

Initialize Rocky Mountain BasicオブジェクトとTo/From Rocky Mountain Basicオブジェクトは、VEEのすべてのバージョンに用意されています。HP 9000シリーズ700システムで動作するプログラム専用です。

Rocky Mountain Basicオブジェクトは、Rocky Mountain Basicプロセスと通信する必要がある**上級ユーザ**のためのツールです。VEEでパイプを使用するための一般的な説明については、155ページ「To/From Named Pipeの使用(UNIX)」を参照してください。

Initialize Rocky Mountain Basic

`Initialize Rocky Mountain Basic`は、Rocky Mountain Basicのプロセスを生成し、指定されたRocky Mountain Basicプログラムを実行します。

実行するRocky Mountain Basicプログラムのフル・パスとファイル名を`Program`フィールドに入力します。プログラムはSTOREフォーマットとSAVEフォーマットのどちらでもかまいません。

`Initialize Rocky Mountain Basic`では、Rocky Mountain Basicプロセスとの間でデータをやりとりする手段は提供されません。このためには`To/From Rocky Mountain Basic`を使います。

1つのプログラムで複数の`Init Rocky Mountain Basic`オブジェクトを使用することができます。また、1つのスレッドで複数使用することもできます。

Rocky Mountain BasicプロセスをVEEプログラムから直接終了する方法はありません。特に、PostRunはRocky Mountain Basicプロセスを終了しません。PostRunが発生するのは、すべてのスレッドが実行を終了するか、ユーザが`Stop`を押したときです。したがって、Rocky Mountain Basicプロセスを終了するには何らかの方法を用意する必要があります。可能な方法としては下記のものがあります。

- 特定のデータをVEEから受信したときにRocky Mountain BasicプログラムがQUITステートメントを実行するようにする。
- `Execute Program`オブジェクトでシェル・コマンド(`rmbkill`など)を使ってRocky Mountain Basicプロセスを終了させる。

対応するRocky Mountain Basicプロセスがアクティブであるときに`Initialize Rocky Mountain Basic`を`Cut`すると、VEEはRocky Mountain Basicプロセスを自動的に終了します。VEEを終了すると、VEEが起動したすべてのRocky Mountain Basicプロセスが終了されます。

To/From Rocky Mountain Basic

`To/From Rocky Mountain Basic`オブジェクトは、Rocky Mountain BasicプログラムとVEEとの間での名前付きパイプを使った通信をサポートします。

`Read Pipe`フィールドと`Write Pipe`フィールドに、使用するパイプの名前を入力します。Rocky Mountain Basicプログラムで使用しているのと正確に同じパイプ名を指定することと、読取り用パイプと書込み用パイプをRocky Mountain Basicプログラム側から見た逆の設定にしないように注意してください。異なるスレッドの`To/From Rocky Mountain Basic`オブジェクトには異なるパイプを使用してください。

To/From Rocky Mountain Basicを使用した例

スカラ・データの共有. 下記のような作業を実行する場合を例に取ります。

1. Rocky Mountain Basicを起動します。
2. 特定のRocky Mountain Basicプログラムを実行します。
3. Rocky Mountain Basicに1個の数値を送って解析させます。
4. 解析結果を取得します。
5. Rocky Mountain Basicを終了します。

図4-26に、`To/From Rocky Mountain Basic`の代表的な設定を示します。

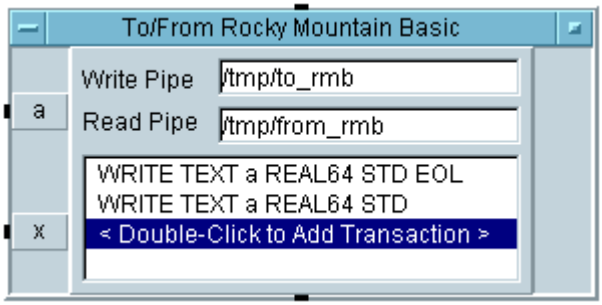


図4-26. `To/From Rocky Mountain Basic`**の設定**

対応するRocky Mountain Basicのプログラムは下記の通りです。

```
100  ASSIGN @From_vee TO "/tmp/to_rmb"
110  ASSIGN @To_vee TO "/tmp/from_rmb"
120  ! ここに解析用のコードを入れる
130  ENTER  @From_vee;Vee_data
140  OUTPUT @To_vee;Rmb_data
150  END
```

この問題を解決するサンプル・プログラムを見るには、サンプル`manual34.vee`をオープンしてください。

配列データの共有. VEEとRocky Mountain Basicとの間でTEXTエンコーディングを使って配列データを共有するには、To/From Rocky Mountain Basicのデフォルトの Array Separatorを変更する必要があります。このためには、To/From Rocky Mountain Basicオブジェクト・メニューでPropertiesをクリックし、Properties ダイアログ・ボックスのData Formatタブをクリックします。Array Separator フィールドを", "(カンマとスペース)に設定します。

VEEとRocky Mountain Basicで同じサイズの配列を使用するように注意してください。

VEEとRocky Mountain Basicが配列要素を読み書きする順序は互換性がありますÅBTo/From Rocky Mountain BasicのREADおよびWRITEトランザクションを使ってVEEとRocky Mountain Basicとの間で配列を共有する場合、VEEとRocky Mountain Basicですべての要素の値が一致します。

VEEとRocky Mountain Basicとの間で配列を共有するサンプル・プログラムを見るには、サンプルmanual35.veeをオープンしてください。

バイナリ・データの共有. 数値データをテキストに変換せずにVEEとRocky Mountain Basicとの間で共有することもできます。このためには、To/From Rocky Mountain Basicトランザクションで BINARYエンコーディングを選択し、Rocky Mountain Basicで名前付きパイプを参照するASSIGNステートメントにFORMAT OFFを指定します。

数値データをバイナリ形式で共有できるのは、下記の2つの場合だけです。

- VEEのBINARY REAL64がRocky Mountain BasicのREALに相当
- VEEのBINARY INT16がRocky Mountain BasicのINTEGERに相当

プログラムとの通信(PC)

このセクションでは、PC上のプログラムとの通信に関する指針について説明します。下記の内容があります。

- Execute Program (PC)の使用
- DDE(Dynamic Data Exchange)の使用

表4-10に、プログラムと関連オブジェクトを示します。

表4-10. プログラムと関連オブジェクト(PC)

プログラム	オブジェクト
MS-DOSコマンド	`Execute Program (PC)`

Windowsアプリケーション[a]	`Execute Program (PC)`
	`To/From DDE (PC)`
	`To/From Socket`

Cプログラム	`Execute Program (PC)`
	`Import Library`
	`Call Function`
	`Formula`

a. Windows 版 VEE では ActiveX オートメーションがサポートされ、他のWindowsアプリケーションの制御に利用できます。この機能の使用法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

Execute Program (PC)の使用

`Execute Program (PC)` オブジェクトを使って、通常はオペレーティング・システムのコマンド行から行う作業を実行できます。図4-27に、`Execute Program (PC)` オブジェクトの例を示します。`Execute Program (PC)` を使えば、下記のような任意の実行可能ファイルを実行できます。

- コンパイルされたCプログラム
- 任意のMS-DOSプログラム(`*.EXE`または`*.COM`ファイル)
- `.BAT`ファイル

■ dirなどのMS-DOSのシステム・コマンド

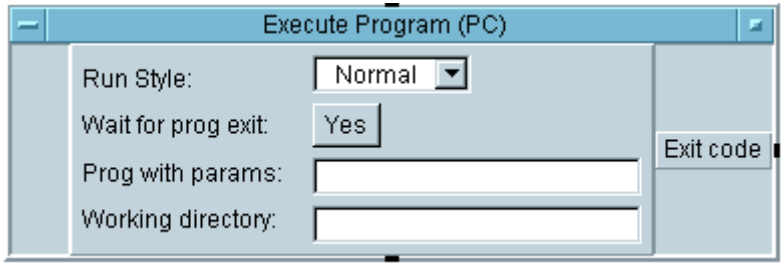


図4-27. Execute Program (PC)**オブジェクト**

Execute Program (PC)のフィールド

以下の部分では、Execute Program (PC)のオープン・ビューに表示されるフィールドについて説明します。

Run Style. 実行するプログラムがウィンドウで実行される場合、Run Styleを使ってウィンドウ・スタイルを指定できます。

■ Normalは、標準のウィンドウでプログラムを実行します。

■ Minimizedは、最小化してアイコンにしたウィンドウでプログラムを実行します。

■ Maximizedは、最大サイズに拡大したウィンドウでプログラムを実行します。

Wait for Prog Exit. Wait for prog exitは、Execute Program (PC)オブジェクトの動作が終了してデータ出力がアクティブになるタイミングを指定します。Wait for prog exitをYesに設定すると、VEEは下記のように動作します。

1. Execute Program (PC)オブジェクトに指定されたコマンドを実行します。
2. プロセスが終了するのを待ってから、Execute Program (PC)オブジェクトの出力ピンをアクティブにします。

Wait for prog exitをNoに設定すると、VEEは下記のように動作します。

1. Execute Program (PC)オブジェクトに指定されたコマンドを実行します。

2. Execute Program (PC)オブジェクトのデータ出力ピンをアクティブにします。

他の条件がすべて等しい場合、Wait for prog exitをNoに設定した方がExecute Program (PC)の実行は高速になります。

Prog With Params . Prog with paramsは下記のどちらかを設定します。

1. 実行可能ファイルの名前とコマンド行パラメータ

2. MS-DOSに渡して解釈させるコマンド

AUTOEXEC.BATファイルに適切なパスが設定されている場合、Prog with paramsフィールドにパスを記述する必要はありません。Prog with paramsフィールドの代表的な設定を以下に示します。

MS-DOSコマンドを実行する場合:

```
COMMAND.COM /C DIR *.DAT
```

コンパイルされたCプログラムを実行する場合:

```
MyProg -optionA -optionB
```

ブラウザでURLをオープンする場合:

```
http://www.agilent.com/find/vee
```

ドキュメントをオープンする場合:

```
D:\path\word.doc
```

Working Directory. Working directoryは、実行するプログラムにとって必要なファイルが存在するディレクトリを示します。例えば、c:\progs\cprog1ディレクトリにあるmakefileを使ってnmakeプログラムを実行する場合、下記のように指定します。

1. Prog with paramsにnmakeと入力します。

2. Working directoryにc:\progs\cprog1と入力します。

DDE(Dynamic Data Exchange)の使用

注記

DDEは古い機能です(まだサポートはされています)。Windows版VEEではActiveXオートメーションがサポートされており、他のWindowsアプリケーションの制御に利用できます。この機能の使用法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。Microsoftアプリケーションの新しいバージョン(Office 2000など)では、DDEはサポートされなくなる可能性があります。Agilentでは、DDEでなくActiveXオートメーションを使用することを強く推奨します。

DDE(Dynamic Data Exchange)は、Windowsアプリケーションの間での通信のためのメッセージ・ベース・プロトコルを定義しています。この通信は、DDEクライアントとDDEサーバとの間で行われます。DDEクライアントは、DDEサーバとの対話を要求します。その後、クライアントはサーバ・アプリケーションに対してデータとサービスを要求します。それに対してサーバは、データを送信したり手順を実行したりします。

DDEをサポートするWindowsアプリケーションは、クライアント、サーバ、またはその両方として動作します。Windows版VEEにはクライアント機能だけが備わっています。VEEのDDE機能はTo/From DDEオブジェクトで実現されています。

Windows版VEEのTo/From DDEオブジェクトは、4つのタイプのトランザクションを使用します。

READ(REQUEST)	DDE転送からデータを読み取ります。
WRITE(POKE)	DDE転送に対してデータを書き込みます。
EXECUTE	通信相手のDDEサーバに対してコマンドを送信します。サーバは受け取ったコマンドを実行します。
WAIT	指定された時間(秒単位)だけ待ちます。

To/From DDEオブジェクトは、その機能の一部としてDDE動作の初期化と終了を行います。初期化/終了機能を明示的に実行する必要はありません。

図4-28に示すように、To/From DDEオブジェクトにはApplication、Topic、Timeoutの3つの主要なフィールドがあります。

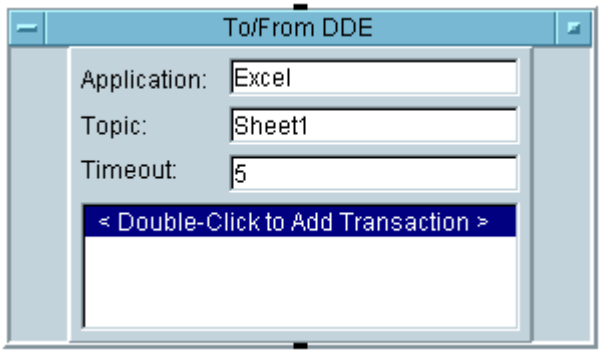


図4-28. `To/From DDE`**オブジェクト**

`Application`フィールドには、通信相手のWindowsアプリケーションのDDEアプリケーション名を入力します。一般に、これは`.EXE`ファイルの名前です。個々のアプリケーションのDDEアプリケーション名を知るには、各アプリケーションのマニュアルを参照してください。

`Topic`フィールドには､データの種類を示すアプリケーション固有の識別子を入力します。例えば、ワード・プロセッサのトピックはドキュメント名になります。

`Timeout`フィールドには、アプリケーションの応答をVEEが待つタイムアウト時間を指定します。デフォルト値は5秒間です。

最後のフィールドには、他のアプリケーションと通信するためのトランザクションを記入します。`READ(REQUEST)`および`WRITE(POKE)`トランザクションの場合、`Item`名も指定する必要があります。`Item`名とは、各データ項目に対するアプリケーション固有の識別子です。例えば、スプレッド・シートのデータ項目としてはセル位置、ワード・プロセッサのデータ項目としてはブックマーク名などが用いられます。

図4-29の`To/From DDE`オブジェクトは、MS Windowsのプログラム・マネージャと通信して、プログラム・グループを作成し、グループにアイテムを追加し、5秒間表示してからプログラム・グループを削除します。

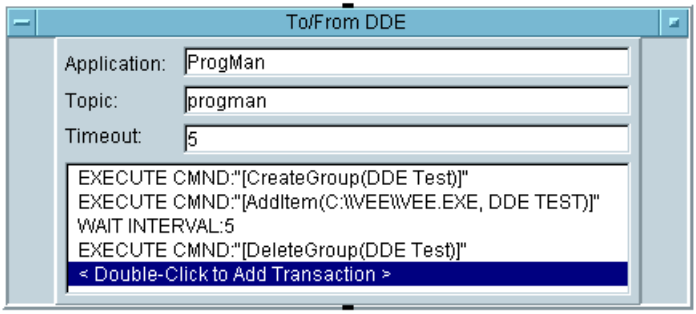


図4-29. `To/From DDE`**の例**

サーバDDEアプリケーションが現在動作していない場合、VEEはアプリケーションを起動しようとします。これが成功するのは、アプリケーションの実行ファイル名が、Applicationフィールドに指定された名前と一致する場合だけです。また、実行ファイルが存在するディレクトリが`PATH`に指定されている必要があります。VEEはアプリケーションの起動を`Timeout`フィールドに指定された時間だけ試みます。

実行ファイルのディレクトリが`PATH`に指定されていない場合、図4-30のように`To/From DDE`オブジェクトの前に`Execute Program (PC)`オブジェクトを使ってアプリケーション・プログラムを起動します。

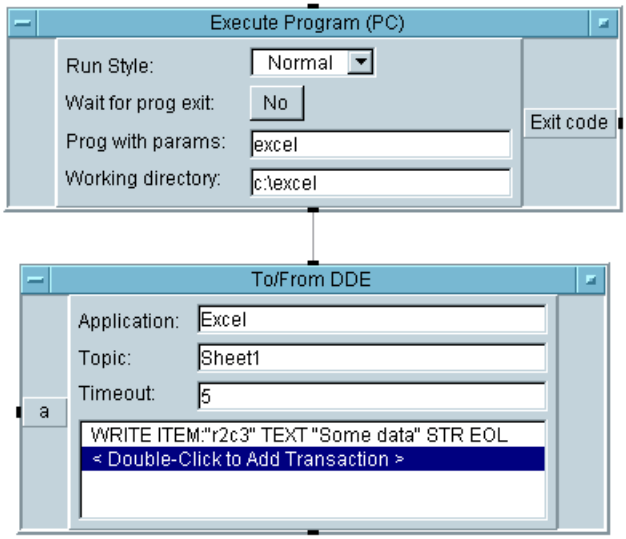


図4-30. To/From DDEの前にExecute Program (PC)を使用

図4-31のプログラムは、`To/From DDE`オブジェクトの入出力端子の使用法を示します。

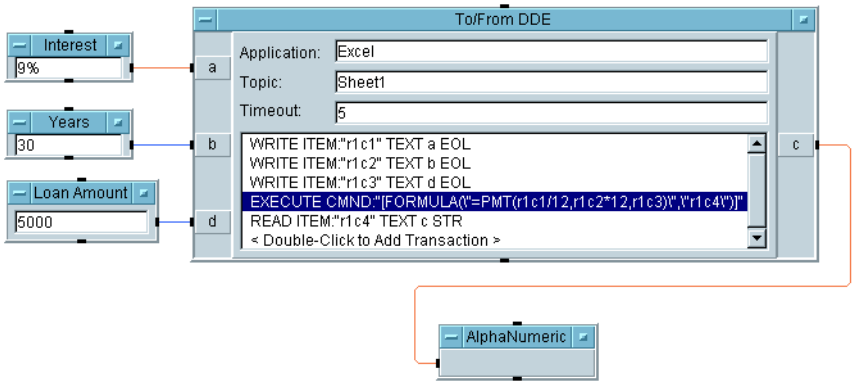


図4-31. I/O端子とTo/From DDE

DDEの例

図4-32から図4-36までは、さまざまなWindowsソフトウェア・アプリケーションとの通信の例です。それぞれの例のNote Padには、例に関する重要な情報が表示されています。

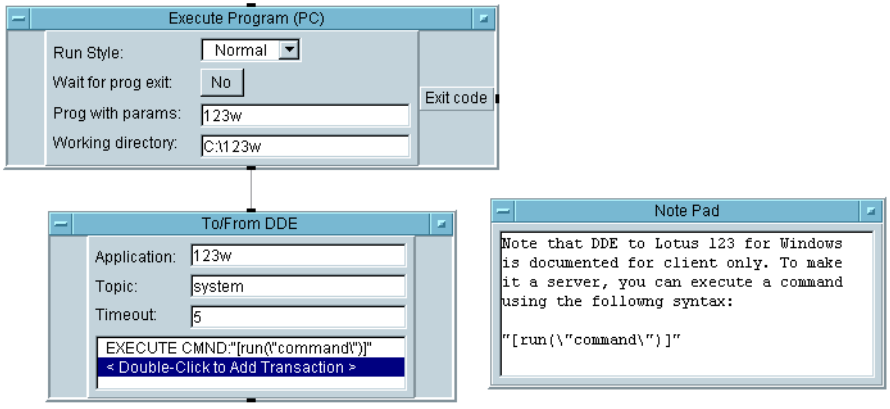


図4-32. Lotus 123とのDDEの例

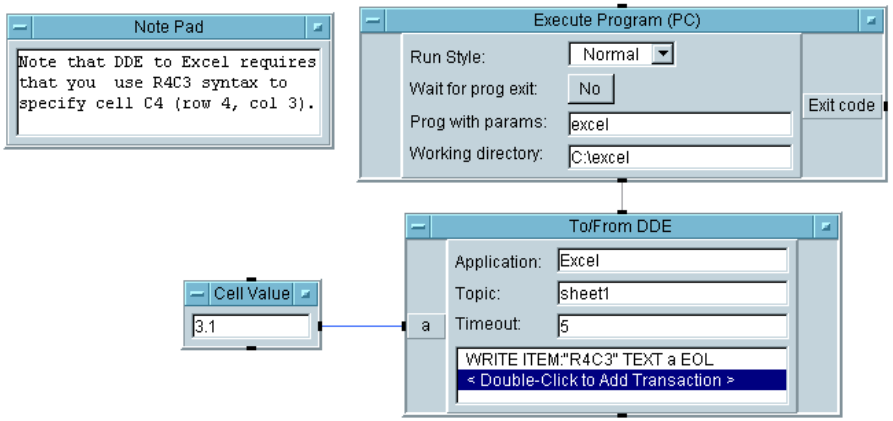


図4-33. ExcelとのDDEの例

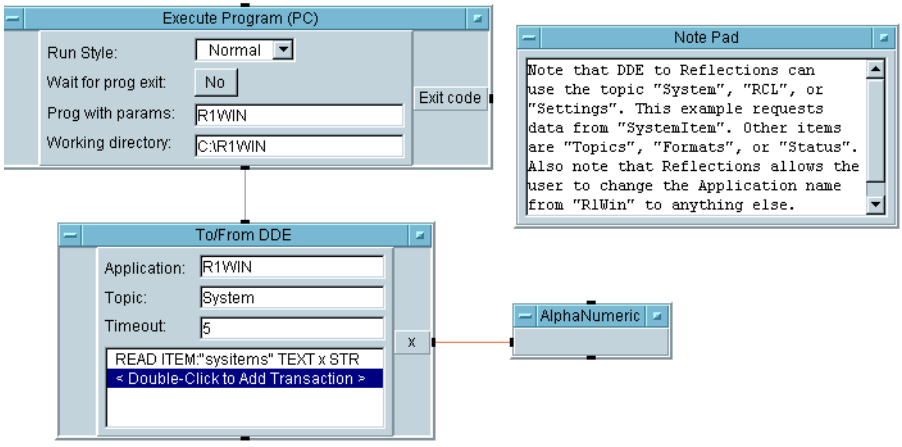


図4-34. ReflectionsとのDDEの例

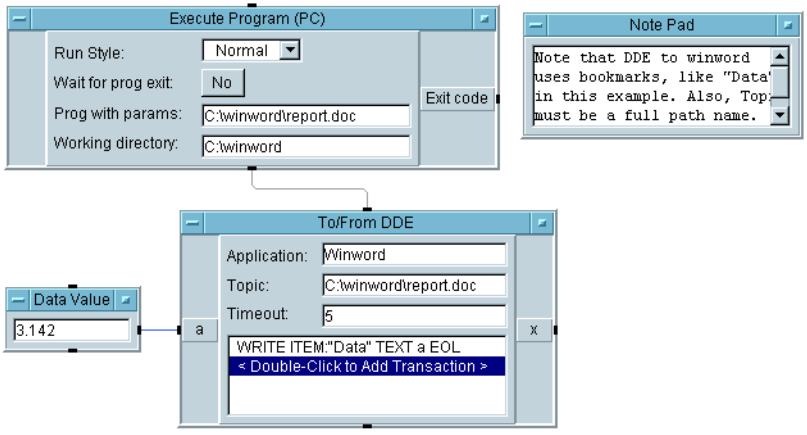


図4-35. Windows版WordとのDDEの例

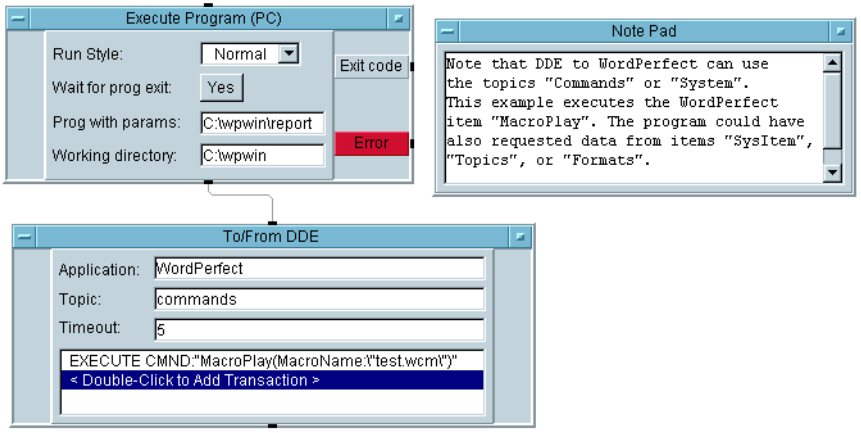


図4-36. WordPerfectとのDDEの例

ダイレクトI/Oおよびインタフェース操作でのトランザクションの使用

I/Oトランザクションを使って機器と通信するには、下記の3つのオブジェクトが役立ちます。

- `Direct I/O`オブジェクトは、GPIB、VXI、シリアル、GPIOの各インタフェースと、LAN接続を経由して機器とデータをやりとりするために使います。
- `MultiInstrument Direct I/O`オブジェクトは、1個のオブジェクトから複数の機器に対してダイレクトI/Oトランザクションを実行するために使います。
- `Interface Operations`オブジェクトは、低レベルのGPIBまたはVXIメッセージ、コマンド、データを送信するために使います。

注記

レジスタ・ベースのVXIデバイスは、I-SCPIドライバによってサポートされる場合に限ってメッセージ・ベースとして扱うことができます。

これらのオブジェクトを使う場合、メッセージを構築してからI/Oトランザクションによって送信します。この章では、`Direct I/O`、`MultiInstrument Direct I/O`、`Interface Operations`の各オブジェクトでI/Oトランザクションを使用するための技法について説明します。

注記

`Direct I/O`、`MultiInstrument Direct I/O`、`Interface Operations`の各オブジェクトを使用するには、機器との通信用にVEEを正しく構成しておく必要があります。詳細については、第3章「機器の構成」を参照してください。

Direct I/Oオブジェクトの使用

Direct I/Oオブジェクトを使うと、機器の組込みコマンドを使って機器を直接制御できます。Direct I/Oで機器を制御する場合、機器ドライバ(ID)やVXI*plug&play*ドライバは不要です。

コマンドの送信

Direct I/Oで機器にコマンドを送信するには、WRITEトランザクションを使います。GPIB機器、メッセージ・ベースのVXI機器、I-SCPIでサポートされるレジスタ・ベースのVXI機器、シリアル機器にコマンドを送信するための最も重要なWRITEトランザクションは下記の通りです。

- WRITE TEXT
- WRITE BINBLOCK
- WRITE STATE

GPIO機器にDirect I/Oでコマンドを送信する場合には、WRITE BINARYおよびWRITE IOCONTROLトランザクションだけが用いられます。

レジスタ・ベースのVXI機器と、メッセージ・ベースのVXI機器の一部にDirect I/Oでコマンドを送信するには、WRITE REGISTERおよびWRITE MEMORYトランザクションが用いられます。I-SCPIドライバでサポートされないレジスタ・ベースのVXI機器との通信には、これらのトランザクションだけが使用できます。

WRITE TEXTトランザクション. Direct I/Oが必要なほとんどの状況では、WRITE TEXTトランザクションだけで機器のセットアップを実行できます。GPIB機器、メッセージ・ベースのVXI機器、シリアル機器のほとんどは、人間が読めるテキスト文字列をプログラミング・コマンドとして使用します。この種のコマンドはWRITE TEXTトランザクションによって簡単に送信できます。

例えば、IEEE 488.2に適合するすべての機器は、*RSTをリセット・コマンドと認識します。これらの機器をリセットするためのトランザクションは下記の通りです。

```
WRITE TEXT "*RST" EOL
```

機器が使用するシンタックスでは、区切り記号が厳密に定義されているのが普通です。例えば、各コマンドのあとやコマンド群の末尾に特定の文字を送信しなければならない場合があります。また、GPIB機器では信号線のEOI(End-Or-Identify)の使用法がそれぞれ異なっています。

これが原因で問題が発生している疑いがある場合、Advanced Instrument Propertiesダイアログ・ボックスの Direct I/O タブにあるEND(EOI) on EOLフィールドとEOL Sequenceフィールドを調べてください。第3章「機器の構成」を

参照してください。各機器に対する正しいコマンド・シンタックスについては、機器のプログラミング・マニュアルを参照してください。

機器が必要とする場合、TEXT以外のWRITEエンコーディングをDirect I/Oで使用することもできます。TEXT以外のエンコーディングで最もよく用いられるのはBINBLOCKとSTATEです。

WRITE BINBLOCKトランザクション. BINBLOCKエンコーディングは、IEEEで定義されたブロック・フォーマットで機器にデータを書き込みます。ブロック・フォーマットは主に、まとまった多量のデータ、例えばオシロスコープやスペクトラム・アナライザのトレース・データなどの転送に用いられます。機器に対してBINBLOCKデータを送信する場合、あらかじめいくつものコマンドを送らなければならないのが普通です。詳しくは機器のプログラミング・マニュアルを参照してください。

BINBLOCKトランザクションを使用する場合、Advanced Instrument Propertiesダイアログ・ボックスの Direct I/Oタブにある Conformanceフィールド(および必要なら Binblock)を正しく設定する必要があります。第3章「機器の構成」を参照してください。

WRITE STATEトランザクション. 一部のGPIB機器やメッセージ・ベースのVXI機器は、ラーン・ストリング機能をサポートしています。これを使うと、機器のすべての設定をアップロードすることができます。あとでWRITE STATEトランザクションを使ってラーン・ストリングをダウンロードすることにより、機器の測定ステートを復元することができます。ラーン・ストリングが特に役立つのは、機器ドライバが存在しない機器に対して測定ステートをダウンロードする場合です。

注記

WRITE STATEトランザクションが使用できるのは、GPIB機器とメッセージ・ベースのVXI機器に対してだけです。

ラーン・ストリングの代表的な使用手順を下に示します。

1. 必要な測定ステートに機器を設定します。これは機器のフロントパネルから行うのが普通です。
2. その機器向けに構成されたDirect I/Oオブジェクトのオブジェクト・メニューでUpload Stateをクリックします。これで、このDirect I/Oオブジェクトのインスタンスに機器ステートが対応付けられます。
3. Direct I/OオブジェクトにWRITE STATEトランザクションを追加します。

WRITE STATEは、通常はDirect I/Oオブジェクトの最初のトランザクションです。WRITE STATEはアップロードされたラーン・ストリングを機器に書き込み、機器のすべての機能を同時に設定します。そのあとでWRITEトランザクションを使って、機器セットアップの必要な個所だけを変更できます。

GPIB機器とメッセージ・ベースのVXI機器に対するUploadおよびWRITE STATEの動作は、Direct I/OタブのConformanceおよびState (Learn String)設定に影響されます。

ConformanceがIEEE 488.2の場合、VEEは自動的にIEEE 488.2の*LRN?の定義に基づいてラーン・ストリングを処理します。ConformanceがIEEE 488の場合、ステートを問い合わせるコマンドをUpload Stringに指定し、ダウンロードの際に文字列の前に送信するコマンドをDownload Stringに指定します。

メッセージ・ベースのVXI機器と、I-SCPIでサポートされるレジスタ・ベースのVXI機器は、IEEE 488.2に適合します。

Direct I/Oオブジェクト・メニューでUpload Stateをクリックすると、下記の3つの結果が生じます。

- ラーン・ストリングが**ただちに**アップロードされます。
- アップロードされたラーン・ストリングは、オブジェクトが存在する間、次のアップロードが行われるまで、Direct I/Oオブジェクトのそのインスタンスに保持されます。**ラーン・ストリングはプログラムといっしょに保存されます。**
- Direct I/Oオブジェクトをコピーした場合、保持されているラーン・ストリングもいっしょにコピーされます。

ラーン・ストリングの例. HP 54100Aディジタイジング・オシロスコープをラーン・ストリングを使ってプログラムする場合を例に取ります。このオシロスコープに関して知っておくべきことを下に記します。

- このオシロスコープはIEEE 488に適合します。IEEE 488.2には適合しません。
- オシロスコープのラーン・ストリングを問い合わせるコマンドはSETUP?です。

- 機器にダウンロードするラーン・ストリングの前には、SETUPコマンドを付ける必要があります。SETUPのPとラーン・ストリングの最初の文字との間にスペースを入れます。

Instrument Manager(第3章「機器の構成」参照)を使って、オシロスコープ向けのDirect I/O構成を正しく指定する必要があります。図4-37にラーン・ストリングに対する設定を示します。

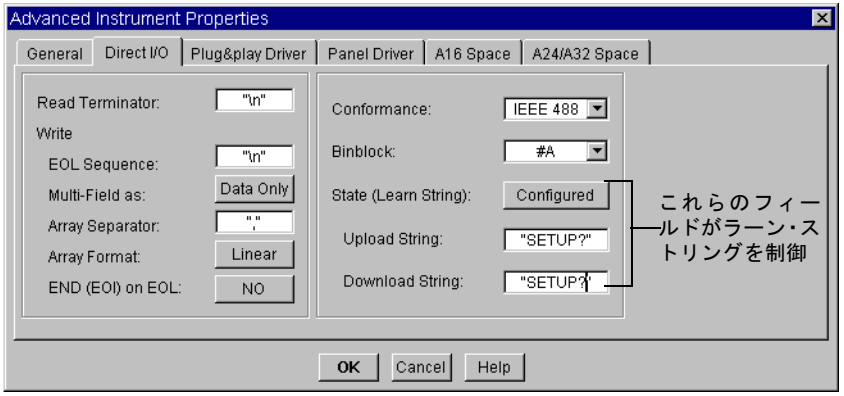


図4-37. ラーン・ストリングに対する設定

オシロスコープからラーン・ストリングをアップロードするには、オシロスコープを制御するDirect I/Oオブジェクトのオブジェクト・メニューでUploadをクリックします。ラーン・ストリングをダウンロードするには、下記のトランザクションを使います。

```
WRITE STATE
```

データの読取り

Direct I/Oで機器からデータを読み取るには、READトランザクションを使います。

注記

機器からのデータはさまざまなフォーマットで返されます。一般には、機器から読み取るデータの**種類**と**量**を知る必要があります。データの種類によって、トランザクションに指定するエンコーディングとフォーマットが決まります。読み取るデータの量によって、トランザクション・ダイアログ・ボックスのSCALARまたはARRAYフィールドの設定が決まります。

GPIB機器、メッセージ・ベースのVXI機器、シリアル機器に対するDirect I/Oで用いられる最も重要なREADトランザクションは下記の通りです。

- READ TEXT
- READ BINBLOCK

GPIO機器からDirect I/Oでデータを読み取る場合には、READ BINARYおよびREAD IOSTATUSトランザクションだけが用いられます。

レジスタ・ベースのVXI機器と、メッセージ・ベースのVXI機器の一部からDirect I/Oでデータを読み取るには、READ REGISTERおよびREAD MEMORYトランザクションが用いられます。I-SCPIドライバでサポートされないレジスタ・ベースのVXI機器との通信には、これらのトランザクションだけが使用できます。

注記

機器からのデータ読取りがうまくいかない場合は、Bus I/O Monitorを使ってデータ・フォーマットを調べてみてください。

READ TEXTトランザクション. 機器に対する問合せの結果として返されるデータとして多いのは、テキスト・フォーマットの1個の数値です。例えば、電圧計は個々の測定値を指数表現の1個の数値(-1.234E+00など)で返します。この電圧計から値を読み取るトランザクションは下記の通りです。

```
"READ TEXT a REAL"
```

一部の機器は、問合せに対して英数字の情報と数値の測定データとの組合わせを返します。一般には、これは問題になりません。READ TEXT REALトランザクションは先頭の英数字を除去し、数値だけを抽出するからです。

注記

機器から数値データを読み取る場合、必要なら機器データのデータ型が自動的に変換されます。変換規則は付録C「機器I/Oのデータ型変換」に記載されています。

MultiInstrument Direct I/Oオブジェクトの使用

MultiInstrument Direct I/Oオブジェクト(I/O⇒Advanced I/O⇒MultiInstrument Direct I/O)を使うと、1個のオブジェクトから複数の機器をダイレクトI/Oトランザクションを使って制御できます。このオブジェクトは標準のトランザクション・オブジェクトであり、VEEがサポートするすべてのインタフェースに対して動作します。

このオブジェクトはDirect I/Oオブジェクトに似ていますが、オブジェクト内のトランザクションがそれぞれ異なる機器を相手にできる点が異なります。Direct I/Oオブジェクトと異なり、MultiInstrument Direct I/Oオブジェクトの制御対象は特定の1台の機器とは限らないため、機器名、アドレス、ライブ・モード条件はタイトルに表示されません。

MultiInstrument Direct I/Oを使うことにより、プログラムに存在する特定機器専用のDirect I/Oオブジェクトの数を減らすことができ、アイコンからアイコンへの解釈時間を節約できます。GPIBに比べて機器制御が高速なVXIインタフェースでは、これによる性能の向上が特に顕著です。

図4-38に示すのは、4台の機器と通信するように構成されたMultiInstrument Direct I/Oオブジェクトとそのトランザクション・ダイアログ・ボックスです。

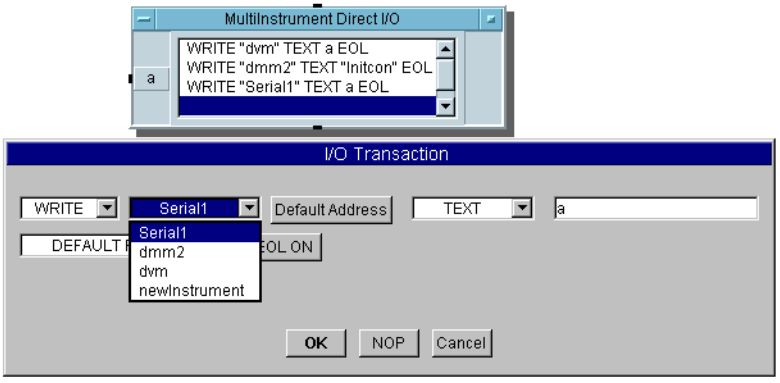


図4-38. MultiInstrument Direct I/Oによる複数機器の制御

トランザクション・ダイアログ・ボックス

I/O Transactionダイアログ・ボックスはDirect I/Oのものと似ていますが、フィールドが2つ増えています。共通のフィールドの働きは同じです。以下の部分では、追加されている2つのフィールドについて説明します。

Instrumentフィールド. Instrumentフィールドには、現在構成されている機器のどれかの名前が表示されます。下向き矢印をクリックすると、構成済みの機器のリストが表示されます。トランザクションごとに異なる機器を選択できます。

Addressフィールド. Addressフィールドは、Instrumentフィールドに表示されたデバイスのアドレスが示されます。Addressフィールドには、Default AddressとAddressの2つのモードがあります。

Default Addressでは、機器の最初の構成時に入力されたアドレスが用いられます。Address:の場合は、テキスト・ボックスに異なるアドレスを入力できます。

アドレスとしては、数値、変数名、式のいずれかが指定できます。エントリは有効なアドレスに評価される必要があります。Address:に入力された値は、オブジェクトが実行されたときにデバイスのアドレスを変更します。これはアドレス制御ピンの動作と同じです。図4-39に、I/O TransactionダイアログボックスでAddress:を使用した例を示します。

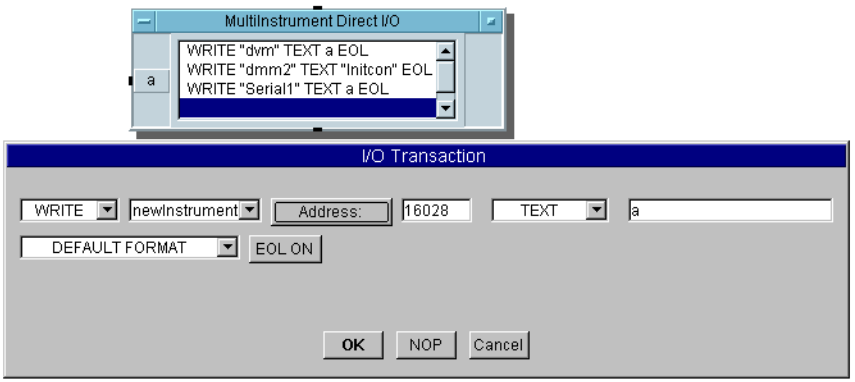


図4-39. 機器アドレスを変数で入力

トランザクションの編集

I/O Transactionダイアログ・ボックスでトランザクションを編集する場合、機器のタイプによって使用可能なトランザクションだけが指定できます。例えば、Instrumentフィールドに表示されている名前がVXIバックプレーン経由で制御されるVXIデバイスの場合、REGISTERまたはMEMORYアクセス・トランザクションを指定できます。

I/O Transactionダイアログ・ボックスに特定のタイプのトランザクションが設定されているときにInstrumentフィールドの名前を変更する場合、変更後の機器に対してもトランザクションが有効でなければなりません。トランザクションが無効になる場合、I/O Transactionダイアログ・ボックスの内容はその機器タイプに対する最後の有効なトランザクションに変更されます。Instrumentフィールドの名前を

VXI機器以外に変更した場合、REGISTERアクセス・トランザクションは無効になります。

オブジェクト・メニュー

MultiInstrument Direct I/Oのオブジェクト・メニューは、Direct I/Oオブジェクトのものに似ています。MultiInstrument Direct I/OメニューにはShow Config...とUpload Stateが存在しません。これらは特定の機器構成のためのものだからです。機器構成を表示したり、物理機器の設定をアップロードしたりするには、Direct I/Oオブジェクトを使用します。

トランザクションに関わるデバイスのライブ・モード・インジケータは存在しません。機器のライブ・モードを制御するには、I/O⇒Instrument Manager...をクリックし、機器構成を選択して編集します。

Interface Operationsオブジェクトの使用

Interface Operationsオブジェクト(I/O⇒Advanced I/O⇒Interface Operations)は、GPIB、VXI、シリアルの各機器を低レベル・コマンドを使って制御するためのものです。Interface Operationsでは、低レベル制御のための2つのタイプのトランザクション(EXECUTEとSEND)がサポートされます。

EXECUTEトランザクション

EXECUTEトランザクションは下記の形式を取ります。

```
EXECUTE Command
```

ここで、*Command*は表4-11に記されたバス・コマンドの1つです。表4-11に記されたコマンドの名前はDirect I/OのEXECUTEコマンドと同じですが、下記の重要な違いがあります。

- Direct I/OのEXECUTEコマンドは、1台の機器に対してコマンドを送信します。
- Interface OperationsのEXECUTEコマンドは、複数の機器に影響することがあります。GPIBでは、対象となる機器はリスン指定されている必要があります。

表4-11. EXECUTEコマンド(Interface Operations)一覧

コマンド	説明
ABORT	IFC(Interface Clear)ラインをアサートすることにより、GPIBインタフェースをクリアします。VXIインタフェースのクリアとリセットにはCLEARを使います。
CLEAR	DCL(Device Clear)を送信してすべてのGPIBデバイスをクリアします。VXIでは、インタフェースをリセットし、リソース・マネージャを実行します。
TRIGGER	GPIBでは、リスン指定されたすべてのデバイスにGET(Group Execute Trigger)を送ってトリガします。VXIでは、TTL、ECL、または外部トリガをトリガします。
REMOTE	GPIBでは、REN(Remote Enable)ラインをアサートします。VXIには対応するものが存在しません。
LOCAL	GPIBでは、REN(Remote Enable)ラインをリリースします。VXIには対応するものが存在しません。
LOCAL LOCKOUT	GPIBでは、LLO(Local Lockout)メッセージを送信します。LLOが送信された時点でリモート・モードにあるデバイスは、フロントパネル動作をロックします。VXIには対応するものが存在しません。
LOCK INTERFACE	リソースが共有されるマルチプロセス・システムで、1つのプロセスがクリティカル・セクションを実行する間リソースをロックし、他のプロセスが使用できないようにします。
UNLOCK INTERFACE	マルチプロセス・システムで1つのプロセスが共有リソースをロックしている場合、リソースのロックを解除して他のプロセスがアクセスできるようにします。
PASS CONTROL	指定したアドレスのGPIBデバイスがアクティブ・コントローラになれる場合、そのデバイスに制御を渡します。VXIには対応するものが存在しません。

SENDトランザクション

SENDトランザクションは下記の形式を取ります。

```
SEND BusCmd
```

ここで、*BusCmd*は表4-12に示すバス・コマンドのどれかです。これらのメッセージは、IEEE 488.1で詳細に定義されています。*BusCmd*はGPIB専用です。VXIには対応するものが存在しません。

表4-12. SENDバス・コマンド

コマンド	説明
COMMAND	ATNを真にし、指定されたデータ・バイト群を送信します。ATNが真の場合、データがバス・コマンドであることを示します。
DATA	ATNを偽にし、指定されたデータ・バイト群を送信します。ATNが偽の場合、データがデバイス依存の情報であることを示します。
TALK	指定された1次バス・アドレス(0～30)のデバイスをトーク指定します。
LISTEN	指定された1次バス・アドレス(0～30)のデバイスをリスン指定します。
SECONDARY	TALKまたはLISTENコマンドのあとで2次バス・アドレスを指定します。2次アドレスはカード・ケージ機器で主に用いられ、カード・ケージが1次アドレスで、各プラグイン・モジュールが2次アドレスに存在します。
UNLISTEN	すべてのデバイスにリスンを停止させます。UNLを送信します。
UNTALK	すべてのデバイスにトークを停止させます。UNTを送信します。
MY LISTEN ADDR	VEEが動作しているコンピュータをリスン指定します。MLAを送信します。
MY TALK ADDR	VEEが動作しているコンピュータをトーク指定します。MTAを送信します。
MESSAGE	マルチライン・バス・メッセージを送信します。詳細についてはIEEE 488.1を参照してください。VEEがサポートするマルチライン・メッセージは下記の通りです。 DCL Device Clear SDC Selected Device Clear GET Group Execute Trigger GTL Go To Local LLO Local Lockout SPE Serial Poll Enable SPD Serial Poll Disable TCT Take Control

5

I/Oに関する高度なトピックス

I/Oに関する高度なトピックス

この章では、I/Oに関する下記の高度な内容を扱います。

- I/O構成の技法
- I/O制御の技法
- 論理ユニットとI/Oアドレッシング

I/O構成の技法

このセクションには、VEEによる機器構成に関する情報を記します。Agilentが製造している機器は、従来HPが製造していたものです。一般に、機器のモデル番号は同じですが、HPの代わりにAgilentが前につきます。多くのVEEユーザがHPブランドの機器を使用しているため、本書では混乱を避けるためにこの記法を使っているところがあります。

I/O構成ファイル

プログラムのI/O構成は、プログラム・ファイルに埋め込むことも(推奨される方法)、別のファイルに記録することもできます。別のファイルに記録する場合、VEE.IO(UNIXではvee.io)というファイルが用いられます。このファイルはPCでは下記のパスに置かれます。

```
%userprofile%\Local Settings\Application Data\Agilent\VEE Pro
```

UNIXシステムでは$HOMEディレクトリに置かれます。

埋込み構成を持たない新しいプログラムの中で機器を構成すると、新しい設定がメモリ(作業セッションの終わりまで)とVEE.IOまたは.veeioファイルに保存されます。

I/O構成がプログラムとともに保存されると、Instrument ManagerのSaveボタンが無効になります。構成を保持するには、OKをクリックしてプログラムを保存します。これにより、更新された構成がプログラムとともに保存されます。

機器制御オブジェクトが含まれるプログラムをオープンするためには、Nameが一致するデバイスがI/O 構成に含まれる必要があります。この説明で、Name とはInstrument Propertiesダイアログ・ボックスのフィールドの値であり、オブジェクトのタイトル・バーに表示されるテキストのことではありません。

オブジェクトがPanel DriverまたはComponent Driverの場合、ID Filenameも構成に一致する必要があります。NameとID Filename以外の設定はプログラムを**オープン**できるかどうかには無関係ですが、プログラムの**動作**には影響する可能性があります。

構成ファイルの変更 一般に、VEE.IOまたは.veeioファイルはVEEが管理します。ただし、必要ならVEE環境の外でこのファイルを消去、更新、コピーすることも可能です。

他の人が作成した機器制御プログラムを実行したい場合、I/O構成がプログラムに埋め込まれていなければ、プログラムが使用するI/O構成を入手する必要があります。このためには3つの方法があります。

1. Instrument Managerと構成ダイアログ・ボックスを使って、すべての機器を手動で構成に追加します。

2. そのプログラム用のVEE.IOまたは.veeioファイルを、このセクションの初めに記載されたパスにあるAgilentディレクトリ(PCの場合)または$HOMEディレクトリ(UNIXの場合)にコピーします。

   ファイルをコピーする場合、あとで必要になった場合に備えて、元のVEE.IOファイルを別の名前(VEEIO.OLDなど)でコピーしておきます。UNIXシステムの場合、$HOMEディレクトリに置く.veeioファイルにはVEEから書込みが可能なようにパーミッションを設定しておきます。

3. プログラムに構成を埋め込んで保存することができます。Save Asオプションを使用し、"Save I/O configuration with program"オプションをチェックします。

注記

サンプル・プログラムにはI/O構成が埋め込まれています。外部I/O構成ファイルは使用されません。

プログラムによるI/O構成

プログラムから機器構成を変更することができます。I/O構成をプログラミングするための推奨される方法は、Function and Object Browserのプログラム機器構成を使うことです。図5-1に、ブラウザ・ウィンドウのTypeウィンドウでInstrumentsを選択したところを示します。Instrumentsを選択すると、ウィンドウ下部のCreate Set Formulaボタンがアクティブになります。

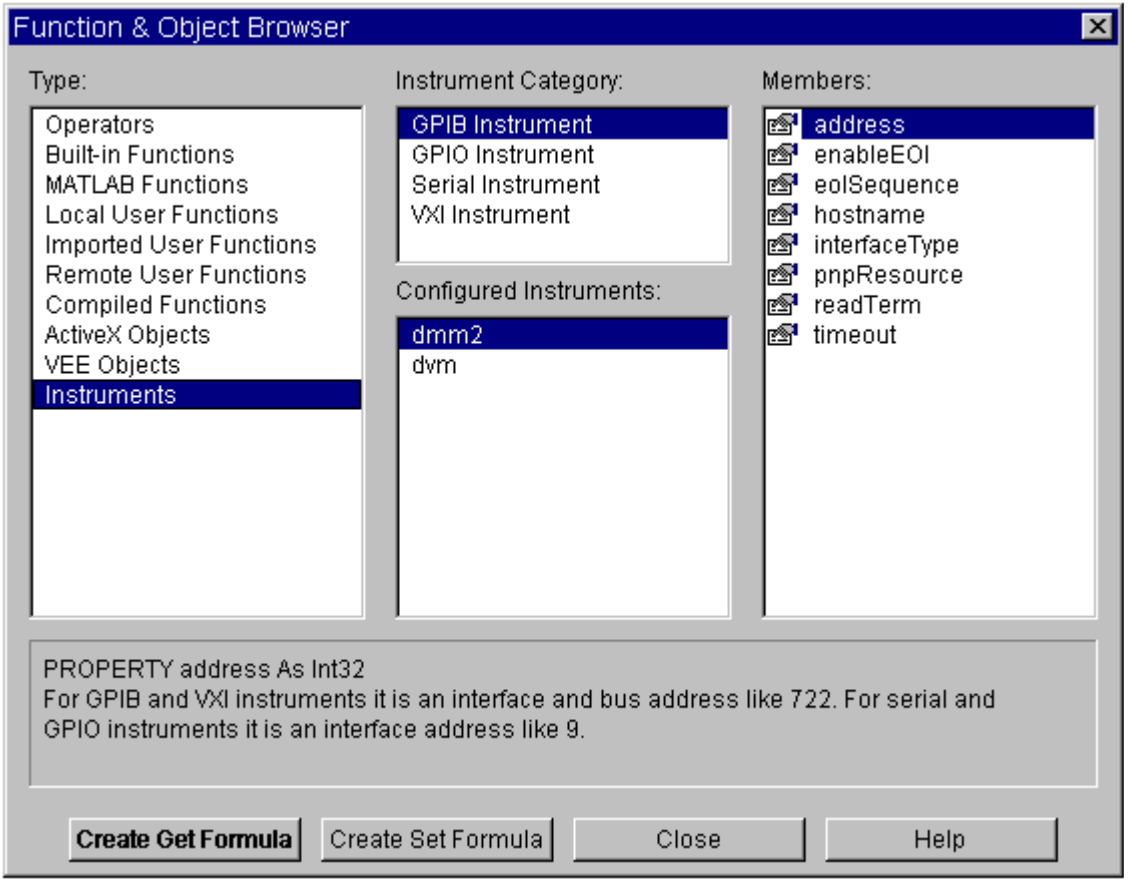


図5-1. Function and Object Browser

Create Set Formulaをクリックすると、図5-2に示すFormula Objectダイアログ・ボックスが表示されます。

図5-2. Create Set Formulaダイアログ・ボックス

VEEの以前のバージョンでは、制御ピンからプログラム構成が可能でした。この制御ピンは現在では推奨されませんが、まだサポートされています。`Panel Driver`、`Component Driver`、`Direct I/O`の各オブジェクトでは、制御ピンを使ってデバイス・アドレスとタイムアウトの値を変更できます。タイムアウト値を設定する制御ピンは、`Interface Operations`、`Instrument Event`、`Interface Event`の各オブジェクトにもあります。

制御ピンの1つに新しいタイムアウト値またはアドレスが送られると、そのデバイスに対してグローバルに値が変更されます。すなわち、同じデバイスと通信している**すべての**機器制御オブジェクトが新しいタイムアウトまたはアドレスの値を使用します。新しい値は、Instrument Propertiesダイアログ・ボックスに入力されてVEE構成ファイルに書き込まれている値とは異なっていてもかまいません。ただし、この新しい値がVEE構成ファイルに書き込まれることはありません。

図5-3の例は、`Address`制御ピンを持つ`Direct I/O`オブジェクトを示します。HP E1413Bは、タイトル・バーに示されているように当初はアドレス16032に設定されています。制御ピンへの入力は16040(新しいアドレス)です。制御ピンに新しいアドレスが送られると、HP E1413Bと通信している他のすべてのオブジェクトで16040が用いられるようになります。

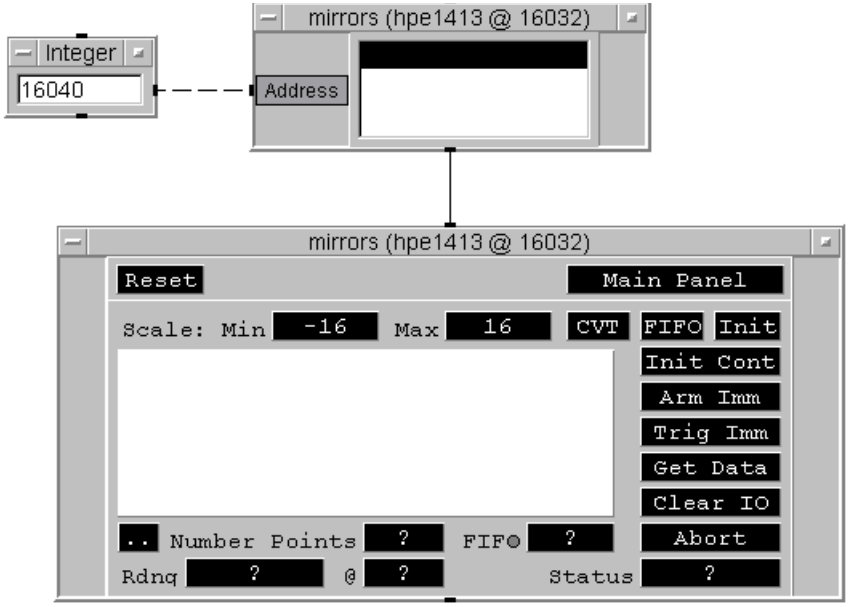


図5-3. プログラムによるデバイスI/Oの構成変更

LANゲートウェイ

VEEでは、LANゲートウェイにアクセスして機器を制御することができます。LANゲートウェイとは、VXI、GPIB、GPIO、シリアルの各インタフェースとそれにつながれた機器にリモート・プロセスからアクセスできるようにするコントローラのことです。

この配置はクライアント・サーバ・モデルで最もよく表現できます。VEEプロセスはクライアントとして、リモート・コンピュータ上のLANゲートウェイ(サーバ)にアクセスします。サーバには**デーモン**と呼ばれる専用のプロセスがあり、これはサーバ上で動作しているSICLプロセスの一部です。デーモンはVEEクライアントと通信し、インタフェースとそのデバイスに対するアクセスを可能にします。

クライアント・プロセスはSICLを呼び出すことにより、SICLがサポートするインタフェース上のデバイスを制御することができます。インタフェースは通常、SICLプロセスが動作しているLANゲートウェイ上に構成されています。LANゲートウェイを使うことにより、リモート・コンピュータ上のインタフェースにアクセスできるようになります。

クライアント側から見ると、インタフェースとそのデバイスが物理的にリモート・コンピュータにつながれていることはわかりません。

構成

LANゲートウェイを使用できるようにVEEとLANハードウェアを構成する必要があります。

VEEの構成. デバイス構成の際に、ゲートウェイにアクセスできるようにVEEを構成します。第3章「機器の構成」を参照してください。図5-4に`Instrument Properties`ダイアログ・ボックスを示します。`Gateway`フィールドはデフォルト設定(`This host`)を示しています。

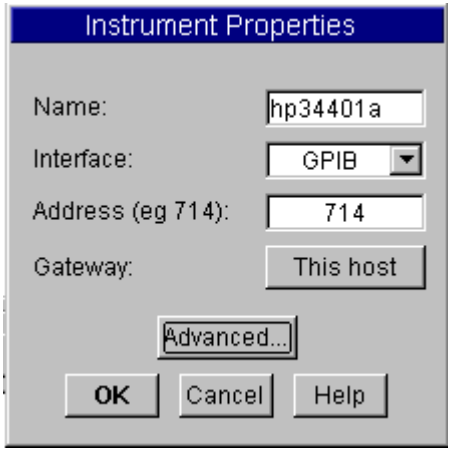


図5-4. ゲートウェイ構成

ゲートウェイ名を選択するには、`Gateway`フィールドをクリックします。構成済みのすべてのゲートウェイを示すリスト・ボックスが表示されます。`This host`は常に、VEEが動作しているコンピュータを指します。

他にゲートウェイの選択肢がない場合、ゲートウェイ名を入力することもできます。入力した名前は、シンボリック・ホスト名テーブルまたはネーム・サーバによってIPアドレスに解決できる必要があります。`55.55.55.555`のようにドット記法でIPアドレスを入力することもできます。

ゲートウェイを選択することを除いて、構成プロセスは同じです。`Panel Driver`オブジェクトと`Direct I/O`オブジェクトは前に説明した方法で構成されます。図5-5に、さまざまなI/Oデバイスをリモート・コンピュータ上のインタフェースとデバイス向けに構成したものを示します。

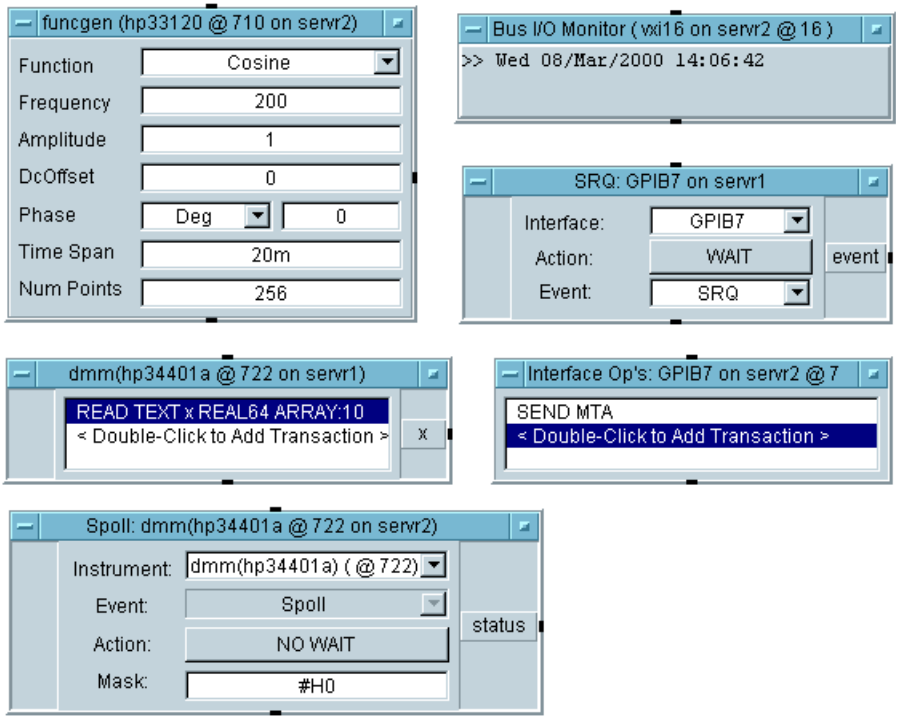


図5-5. リモート・マシン上で構成されたデバイスの例

LANハードウェアの構成. SICL LANゲートウェイのサポートは、VEEが動作しているマシンの構成、ゲートウェイ・デーモンが動作しているマシンの構成、LAN全体の構成に依存します。システム管理者に相談して、名前とIPアドレスが解決できるようにLANを構成してください。

ゲートウェイ・デーモンが動作しているマシンについては、デーモンのインストール手順によってネットワーク関係のローカル・ファイルが正しく構成されている必要があります。HP E2050A LAN/GPIBゲートウェイの場合、必要な内部構成はすべてあらかじめ済んでいます。

HP-UXオペレーティング・システムを使用しているシステムでは、クライアント・マシンに特別なネットワーク構成ファイルは必要ありません。ただし、SICL構成ファイル`hwconfig.cf`に下記の行が必要です。

```
#
# LAN Configuration
#     <lu> <symname> ilan <not used> <not used> <sicl_infinity>
      <lan_timeout_delta> 30 lan ilan 0 0 120 25
```

このエントリには、SICLの通常の論理ユニット/シンボル名が記載されています。インタフェース・タイプはilanです。sicl_infinityとlan_timeout_deltaの2つのエントリは特別なタイムアウト値であり、次のセクションで詳しく説明します。

サーバ・マシンでは、/etc/rpcと/etc/inetd.confの2つのファイルにエントリを追加する必要があります。

/etc/rpcには下記の行を追加します。

```
   siclland      395180
```

/etc/inetd.confには下記の行を追加します。

```
rpc stream tcp nowait root /opt/sicl/bin/siclland 395180 1 siclland -e -l
/tmp/siclland.log
```

サーバ・マシンで、inetデーモンにinetd.confファイルを再読込みさせる必要があります。このためには、sys-admin(root)権限で下記のコマンドを実行します。

```
   /etc/inetd -c
```

LANリソースの検索がローカル・ファイルでなくNIS(Network Information Services、Yellow Pages参照)で管理される場合、データベース・マシン上で上記のファイルを変更して、データベースを再コンパイルする必要があります。

実行の動作

理想的には、ゲートウェイを通じたI/O動作は、インタフェースとデバイスがクライアント・コンピュータに直接接続されている場合と同じであるべきです。しかし、実際にはLAN構成によって応答時間が異なります。接続されたホストの数、LAN間ゲートウェイ、現在の負荷などが応答時間に影響します。場合によっては、ケーブルが外れたりLAN上のコンピュータがダウンしたりしたために接続がとぎれることもあります。このような事態を考慮してタイムアウト時間を決める必要があります。

クライアント・アプリケーション(VEE)からのI/O要求を受け取ると、サーバはInstrument Propertiesダイアログ・ボックスに入力されたタイムアウト値を使用します。これをSICLタイムアウトと呼びます。指定した時間内にサーバの動作が終了しなかった場合、サーバはクライアントに対してタイムアウトが発生したという応答を送り、通常のVEEタイムアウト・エラーが発生します。

サーバに対してI/O要求を送ったクライアントは、タイマを始動し、サーバからの応答を待ちます。サーバが時間内に応答しなかった場合、タイムアウトが発生し、VEEタイムアウト・エラーが生じます。これをLANタイムアウトと呼びます。

クライアントのタイムアウト時間がサーバのタイムアウト時間と異なるのは、サーバのI/Oトランザクション時間とLAN上の伝送時間が通常は異なるからです。サーバがI/Oトランザクションを5秒間(VEEのデフォルトのタイムアウト時間)で終了したとしても、クライアントまでのLAN上の実際の伝送時間は5秒より長くかかる可能性があります。

2つのタイムアウト値は別々の値であり、SICL構成ファイルの別のエントリで設定されます。

`sicl_infinity`	`Advanced Instrument Properties`ダイアログ・ボックスに入力されたユーザ定義タイムアウト(SICLタイムアウト)が無限大(0)の場合にサーバによって用いられます。サーバでは無限大のタイムアウト時間は使用できません。この値は、サーバ内部でトランザクションが終了するのを待つ秒数を指定します。
`lan_timeout_delta`	クライアントのタイムアウト時間(LANタイムアウト)を決めるためにサーバのタイムアウト値に加算される値。計算結果のLANタイムアウトは、I/Oデバイスが必要とする時間だけ増加しており、減少することはありません。これにより、SICLタイムアウトが変更されたときにいちいちLANタイムアウトを再調整する手間が省けます。

クリティカル・セクションの保護

マルチプロセスのテスト・システムでは、プロセス間でリソースが共有されるため、クリティカル・セクションを保護するためのロック機構が必要です。クリティカル・セクションが必要になるのは、共有された機器リソースに対する排他的アクセスをどれかのプロセスが必要とする場合です。

クリティカル・セクションの間に他のプロセスが機器にアクセスするのを避けるため、最初のプロセスは機器をロックします。作業が終了するまでの間ロックは有効になります。この間、他のプロセスは機器に対するいっさいの操作ができず、自分が使用するために機器をロックすることもできません。

下記のEXECUTEトランザクションはクリティカル・セクションを保護するためのもので、Direct I/O、MultiInstrument Direct I/O、Interface Operationsの各トランザクション・オブジェクトで使用できます。トランザクションのシンタックスは、使用するインタフェースとトランザクション・オブジェクトによって異なります。GPIB、シリアル、GPIOの場合、インタフェース全体がロックされます。VXIの場合、個々のデバイスがロックされます。

Direct I/Oオブジェクトからの直接バックプレーン・アクセスによってVXIデバイスをロックするには、下記のトランザクションを使います。

```
EXECUTE LOCK DEVICE
EXECUTE UNLOCK DEVICE
```

MultiInstrument Direct I/Oオブジェクトの場合、下記のトランザクションを使います。

```
EXECUTE vxiScope LOCK DEVICE
EXECUTE vxiScope UNLOCK DEVICE
```

ここで、vxiScopeはHP E1428BなどのVXIオシロスコープに対して構成された名前です。

GPIB、シリアル、GPIOの各インタフェースをInterface Operationsオブジェクトでロックするには、下記のトランザクションを使います。

```
EXECUTE LOCK INTERFACE
EXECUTE UNLOCK INTERFACE
```

サポートされるプラットフォーム

表5-1. EXECUTE LOCK/UNLOCKのサポート

プラットフォーム	サポートされるI/Oインタフェース
Windows 95 (PC、HP 6232、HP 6233、EPC7/8)	GPIB[a] シリアル VXI (VXLink付きPC、または組込み)[b]
Windows NT (PC、HP 6232、HP 6233、EPC7/8)	GPIB[a] シリアル VXI (VXLink付きPC、または組込み)[b]
HP-UX (HP 9000シリーズ700またはV/743)	GPIB シリアル GPIO VXI (MXI付きS700、VXLink、組込み)[b]

a. National Instruments製GPIBインタフェースはLOCKをサポートしません。

b. VXIデバイスのレジスタ/メモリ・アクセス(READ/WRITE REGISTER/MEMORYトランザクション)はロックできません。他のプロセスでのロックのためにVEEのプロセス空間にメモリがマッピングされている場合、直接メモリ・アクセスを試みるトランザクションの最初の実行だけが阻止されます。それ以後は、複数のプロセスが同時に1つのメモリ位置にアクセスするのを防ぐ方法はありません。これは共有メモリだからです。

実行の動作

どれかの種類のEXECUTE LOCKトランザクションが実行されると、トランザクションはデバイスまたはインタフェースをロックしようとします。他のプロセスがそれより前にロックをかけていない場合、トランザクションは終了し、ロックの取得は成功します。すでにロックが存在する場合、そのデバイスまたはインタフェースに対して設定されたタイムアウト時間の間トランザクションはブロックします。

タイムアウト時間の間に他のプロセスがロックを解放した場合、トランザクションは終了し、ロックを取得します。タイムアウト時間が経過した場合、エラーが発生し、エラー・メッセージ・ボックスが表示されます。このエラーはトランザクション・オブジェクトのエラー・ピンでキャプチャできます。

ロックが取得されると、`Direct I/O`、`MultiInstrument Direct I/O`、`Panel Driver`、`Component Driver`、`Interface Operations`によるそれ以降のI/Oは、そのデバイスまたはインタフェースに対して他のプロセスが通信しようとしても保護されます。

クリティカル・セクションが終了したら、対応するEXECUTE UNLOCKトランザクションを実行します。

ロックは他のプロセスからのみクリティカル・セクションを保護します。1つのプロセスからは、EXECUTE LOCKトランザクションを2回実行することによりネストしたロックを作成できます。他のプロセスがすでにロックをかけていない限り、どちらのトランザクションも成功します。

この場合、プロセスはEXECUTE UNLOCKトランザクションを2回実行する必要があります。1回しか実行しないと、デバイスまたはインタフェースはロックされたままになります。ロックがない状態でトランザクションがアンロックを実行しようとすると、ランタイム・エラーが発生します。

ロックが存在するのはVEEプログラムの実行中だけです。VEEプログラムの実行が終了すると、デバイスおよびインタフェースからすべてのロックが除去されます。これにより、正常終了、ランタイム・エラー、Stopボタンなどでプログラムが実行を終了し、EXECUTE UNLOCKトランザクションが実行されなかった場合でも、デバイスやインタフェースがロックされたままになるのを防ぐことができます。

例: EXECUTE LOCK/UNLOCKトランザクション - GPIB

図5-6のサンプル・プログラムは、GPIB向けに構成されたInterface Operationsオブジェクトで EXECUTE LOCK/UNLOCK INTERFACEトランザクションを使っています(この例はシリアル・インタフェースでも同じです)。ロック・トランザクションとアンロック・トランザクションに挟まれて、論理ユニット7のGPIBインタフェース上のデバイスに対するI/Oを実行するUserObjectがあります。このプログラムは、ロックを3回取得しようとします。3回目にロックの取得が失敗した場合、ユーザ定義エラーが発生します。

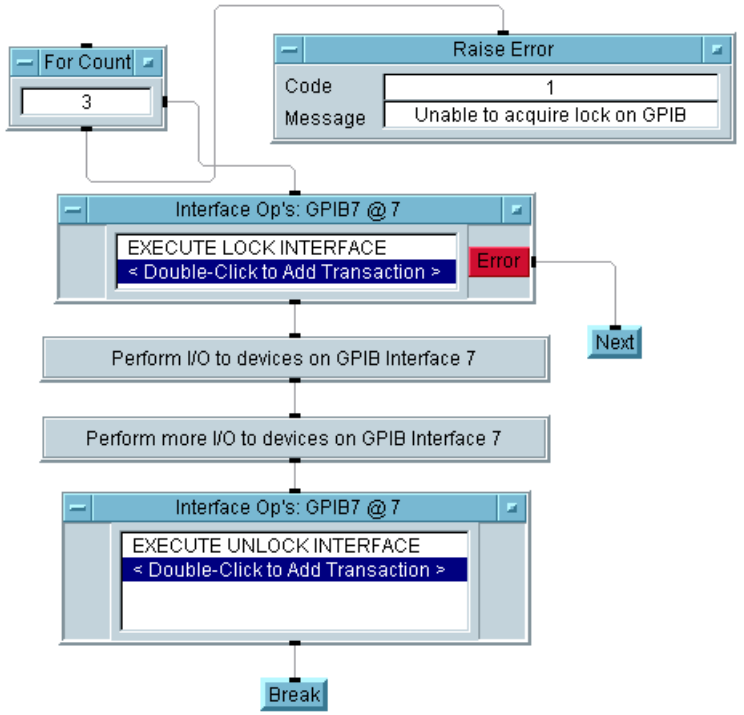


図5-6. EXECUTE LOCK/UNLOCKトランザクション - GPIB

各回の試行で、EXECUTE LOCK INTERFACEトランザクションは設定されたタイムアウト時間の間ロックの取得を待ちます。タイムアウト時間は、Interface Operationオブジェクトの Propertiesダイアログ・ボックスで設定できます。最初のトランザクション・オブジェクトのエラー・ピンがNextオブジェクトに接続されているため、スレッドが再実行されて次の試行が行われます。最後のトランザクション・オブジェクトのあとのBreakオブジェクトにより、スレッドが誤ってもう一度実行されることが防がれます。

例: EXECUTE LOCK/UNLOCKトランザクション - VXI

図5-7のサンプル・プログラムは、MultiInstrument Direct I/Oオブジェクトで EXECUTE LOCK/UNLOCK DEVICEトランザクションを使ったものです。MultiInstrument Direct I/Oの代わりにDirect I/Oを使うこともできますが、この場合全部のデバイスに対して1個のオブジェクトでなく、各デバイスに対して1個ずつのオブジェクトが必要になります。このプログラムは図5-6のプログラムとよく似ています。For Countオブジェクトにより、スレッドは3つの異なるデバイスに対してロックを取得しようとします。UserObjectでのI/O動作が終了したあと、一連のアンロックが実行されます。

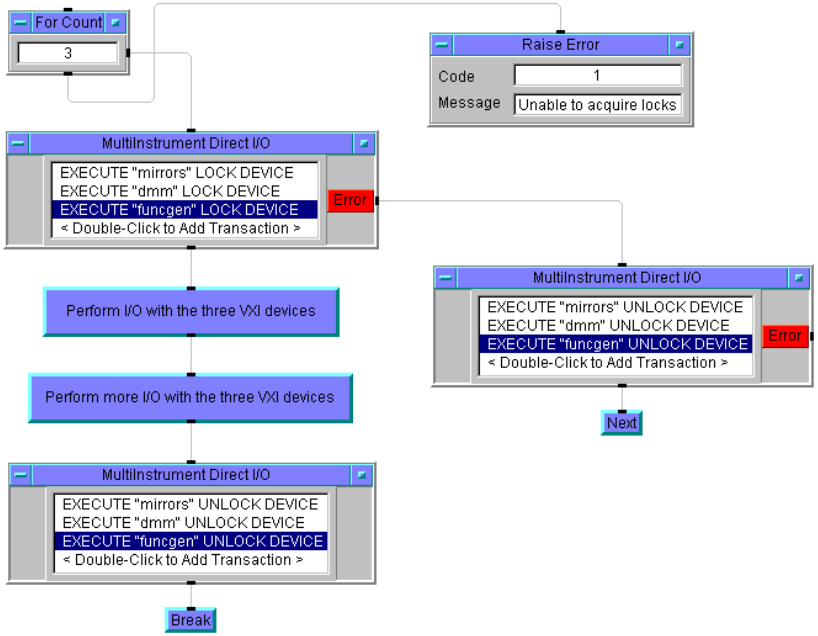


図5-7. EXECUTE LOCK/UNLOCKトランザクション - VXI

各トランザクションは、各デバイスに対して設定されたタイムアウト時間の間、それぞれのロックを取得しようとします。3つのトランザクションのどれかがタイムアウトした場合、エラーが発生し、エラー・ピンでトラップされます。ロックが成功したあとで別の試行がタイムアウト・エラーになった場合、エラー・ピンがエラーをトラップします。

ロック・トランザクションをプログラムが再実行する前に、取得済みのロックをすべてアンロックする必要があります。このために、MultiInstrument Direct I/Oオブジェクトがエラー・ピンに接続されています。重要なこととして、このオブジェクトは最初のオブジェクトがロックを取得したのと**同じ順序で**各デバイスをアンロックしようとします。

ロック・トランザクションより前にアンロック・トランザクションが実行されるとエラーが発生するため、アンロック・トランザクションのオブジェクトにもエラー・ピンが置かれています。最初のオブジェクトでトランザクションがロックの取得に失敗していると、次のオブジェクトの対応するアンロック・トランザクションも失敗します。

I/O制御の技法

このセクションでは、機器I/O制御に関するその他の技法について説明します。

ポーリング

VEEでは、IEEE 488.1で定義されたすべてのシリアル・ポール動作がサポートされます。GPIB機器とVXIメッセージ・ベース機器は、すべてシリアル・ポール動作をサポートします。VXIメッセージ・ベース機器は定義上IEEE 488.2に適合しています。VXIレジスタ・ベース機器は、I-SCPIドライバが用意されていればIEEE 488.2に適合しています。パラレル・ポール動作はVEEではサポートされません。

機器からシリアル・ポール応答を得るには、下記の2つの方法があります。

オブジェクト	シリアル・ポール動作
Instrument Event	Instrument Eventオブジェクトは、指定された機器を1回ポーリングし、スカラ整数を出力します。この値は、NO WAITオプションを使ったシリアル・ポール応答です。Instrument Eventオブジェクトを使って、シリアル・ポール応答バイトに特定のビット・パターンが現れるのを待つこともできます。このためには、ビット・マスクと、ALL CLEARおよびANY SETオプションを指定します。
Direct I/O	GPIB機器向けのDirect I/Oオブジェクトは、WAIT SPOLLトランザクションをサポートします。このトランザクションは、シリアル・ポール応答バイトが特定のビット・パターンにマッチするまで機器を繰り返しポーリングします。ビット・パターンの指定には、ビット・マスクと、ALL CLEARまたはANY SETオプションを使用します。Direct I/Oの詳細については、第4章「トランザクションI/Oの使用」を参照してください。

Instrument EventオブジェクトでSpollを設定した場合、特別な実行プロパティが使用できます。これについては次の「サービス・リクエスト」のセクションで説明します。この動作を使えば、並行する他のスレッドの実行を止めずに、マスク値とALL CLEARまたはANY SETオプションを使って特定のビット・パターンを待つことができます。

NO WAITはただちに実行され、GPIBまたはメッセージ・ベースVXI機器のステータス・バイトを返します。どちらのオブジェクトも、オブジェクト・メニュー(Add Terminal)にTimeout制御入力が用意されているので、タイムアウト時間をプログラムから設定できます。図5-8に例を示します。

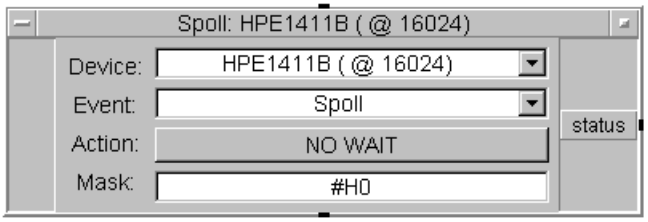


図5-8. シリアル・ポール向けに構成されたInstrument Event

サービス・リクエスト

VXI機器のサービス・リクエスト(SRQメッセージ)を検出するには、Instrument Eventオブジェクト(I/O ⇒ Advanced I/O ⇒ Instrument Event)を使います。GPIB機器またはRS-232のサービス・リクエストを検出するには、Interface Eventオブジェクト(I/O ⇒ Advanced I/O ⇒ Interface Event)を使います。

Instrument EventオブジェクトとInterface Eventオブジェクトには、割込み型の実行のための特別な動作が用意されています。この動作を見るには、Debug ⇒ Show Execution Flowをオンにしてプログラムを実行してみてください。

例えば、Interface Eventはプログラム内で下記のように動作します。

1. Interface Eventオブジェクト(GPIB向けに構成され、WAITオプションが指定されているもの)が動作する前は、実行は通常通りに行われ、各スレッドが同じ優先度で実行時間を分け合います。

2. Interface Eventオブジェクトが動作すると、Interface Eventのデータ出力に接続されたスレッドの実行はInterface Eventオブジェクトで停止します。それ以外のスレッドの**実行は継続します。**

3. 指定されたインタフェースでSRQが検出されると、`Interface Event`のデータ出力がアクティブになります。この時点で、**他のすべてのスレッドの実行は、**`Interface Event`**のデータ出力に接続されたスレッドの実行が終了するまでブロックされます。**

例: サービス・リクエスト. 図5-9のプログラムは、サービス・リクエストを処理する1つの方法を示します。この例では、サービス・リクエストは`dvm`と`scope`のどちらかから発生します。このプログラムは、examplesディレクトリにあるファイル`manual16.vee`に保存されています。

注記

図5-9のプログラムは、指定した機器が接続され、構成され、電源が投入されていなければ動作しません。ただし、これをプログラミング技法の例として独自のプログラムを作成したり、このプログラムを変更して実際の機器との通信に利用したりすることはできます。

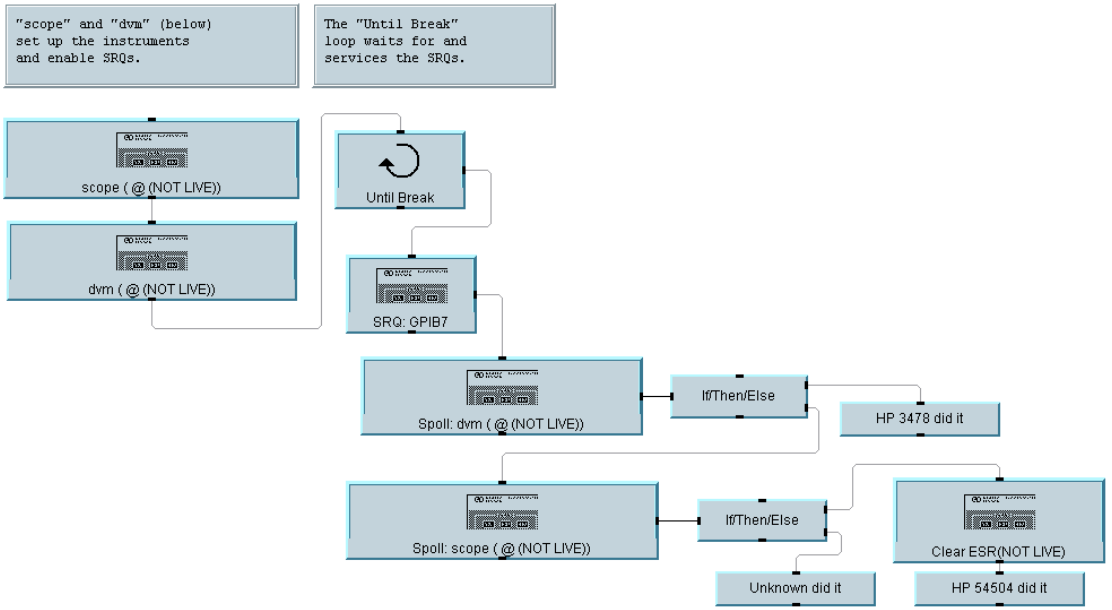


図5-9. サービス・リクエストの処理

このプログラムは、サービス・リクエストの発生元を知るために、`Instrument Event`を使って各機器のステータス・バイトを取得します。それぞれのステータス・バイトを`If/Then/Else`と`bit(x,n)`関数でテストし、ビット6が真であるかどうかを

調べます。ビット6がセットされている場合、対応する機器がサービス・リクエストを発生しています。

このあとのサービス・リクエストを処理するため、Until Breakオブジェクトが自動的にスレッド全体を再実行します。Instrument EventオブジェクトはNO WAITに設定されているため、ステータス・バイトはマスク値を使わずに返されます。マスク値64が用いられ、Instrument EventオブジェクトがANY SETに設定されていれば、If/Then/Elseとbit(x,n)関数を使う必要はありません。

サービス・リクエストをクリアして再びイネーブルする方法は、機器ごとに異なります。図5-9で、dvmはシリアル・ポールだけでSRQ機能をクリアして再びイネーブルできます。これに対して、scopeでは発生元のイベント・レジスタをクリアするという余分の手順が必要です。

Instrument Eventオブジェクトを使って、メッセージ・ベースVXI機器からのサービス・リクエストを検出することができます。VXIコントローラのシグナル・レジスタにrequest trueイベント(RT)を機器が書き込むと、request for serviceと評価され、機器は*Read STB* ワード・シリアル・プロトコル・コマンドを受信します。

メッセージ・ベース機器は自分のステータス・バイトをコントローラに返し、VXIコントローラのシグナル・レジスタにrequest falseイベント(RF)を書き込みます。返されたステータス・バイトに対して、指定されたマスク値とANY SETまたはALL CLEARオプションを適用することで、どのビット(ビット6以外)がセットされているかがわかります。このようにして、1個のオブジェクト(Instrument Event)を使って、メッセージ・ベースVXIデバイスからのサービス・リクエストを検出し、リクエスト発生の理由を調べることができます。

どちらのオブジェクトも、オブジェクト・メニュー(Add Terminal)にTimeout制御入力が用意されているので、タイムアウト時間をプログラムから設定できます。詳細については、**VEEオンライン・ヘルプ**のInstrument EventとInterface Eventに関するリファレンス・セクションを参照してください。

バス動作のモニタ

Bus I/O Monitorオブジェクト(I/O ⇒ Bus I/O Monitor)を使うことにより、VEEと他のトーカおよびリスナとの間で伝送されるすべてのバス・メッセージを記録することができます。Bus I/O Monitorは、VEEとの間でやりとりされるバス・メッセージだけを記録します。

Bus I/O Monitorを使えば、サポートされるすべてのインタフェース(GPIB、VXI、シリアル、GPIO)をモニタできます。Bus I/O Monitorオブジェクトの1つのインスタンスが1つのハードウェア・インタフェースをモニタします。

図5-10に示すのは、GPIBアドレス717の機器に*RSTを送信するバス・メッセージです。

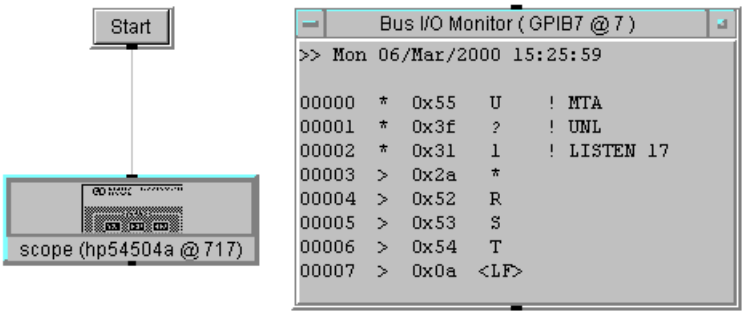


図5-10. Bus I/O Monitor

Bus I/O Monitorの表示エリアには5つのカラムがあります。

- カラム1 - 行番号
- カラム2 - バス・コマンド(*)、出力データ(>)、入力データ(<)の別
- カラム3 - 伝送バイトの16進表現
- カラム4 - 伝送バイトの7ビットASCII文字表現
- カラム5 - バス・コマンド・ニーモニック(バス・コマンドのみ、データでは空白)

Bus I/O Monitorの動作は、オープン・ビュー・オブジェクトよりもアイコンの状態のほうがはるかに高速です。

低レベル・バス制御

低レベル・バス・メッセージを送信するには、図5-11に示す2つの方法があります。

オブジェクト	バス・メッセージ機能
`Interface Operations`	このオブジェクトを使うと、任意のGPIBデバイスに任意のバス・メッセージを送信したり、VXIインタフェースをリセットしたり、VXIバックプレーンの各種トリガ・ラインをトリガしたりすることができます。
`Direct I/O`	GPIB機器、メッセージ・ベースVXI機器、I-SCPIでサポートされるレジスタ・ベースVXI機器向けの`Direct I/O`オブジェクトを使うと、`EXECUTE`トランザクションで`CLEAR`、`LOCAL`、`REMOTE`、`TRIGGER`の各コマンドを送信できます。

`Interface Operations`と`Direct I/O`に関する詳細については、第4章「トランザクションI/Oの使用」を参照してください。

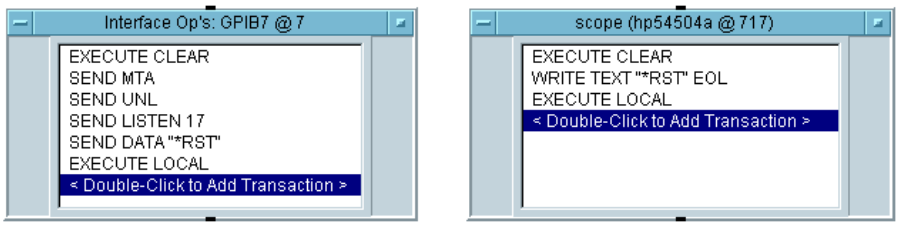


図5-11. 低レベルGPIB制御の2つの方法

機器へのダウンロード

一部の機器に対しては、マクロ、測定ルーチン、測定プログラム全体などのダウンロードが可能です。例えば、HP Instrument BASICをサポートする一部のHP機器では、機器内部で動作するHP Instrument BASICプログラムを機器に対して書き込むことができます。VEEを使って測定ルーチンを機器にダウンロードする方法の1つを下に示します。

1. viなどのテキスト・エディタを使って測定ルーチンを作成、編集します。通常のテキスト・ファイルとして測定ルーチンを保存します。

2. From Fileでファイルを読み取ります。

3. Direct I/Oを使って、ファイルの内容を機器に書き込みます。

このセクションでは、この方法によるダウンロードの例を紹介します。Direct I/Oの詳細については、第4章「トランザクションI/Oの使用」を参照してください。

図5-12は、HP 3852Aに測定サブプログラムをダウンロードするプログラムを示します。この例は、ビープ音を2回発してメッセージを表示するサブプログラムBEEP2をダウンロードします。このプログラムは、examplesディレクトリにあるファイルmanual17.veeに保存されています。

注記

図5-12のプログラムは、指定した機器が接続され、構成され、電源が投入されていなければ動作しません。ただし、これをプログラミング技法の例として独自のプログラムを作成したり、このプログラムを変更して実際の機器との通信に利用したりすることはできます。manual17.veeプログラムには埋込み構成が存在します。

下に示すのは、ダウンロードされるファイルmanual17.datの内容です。manual17.datファイルはサンプル・ディレクトリに存在します。

```
DISP MSG "LOADING BEEP2"
WAIT 1

SUB BEEP2
DISP "BEEP2 CALLED"
BEEP
WAIT .5
BEEP
SUBEND

DISP MSG "BEEP2 LOADED"
```

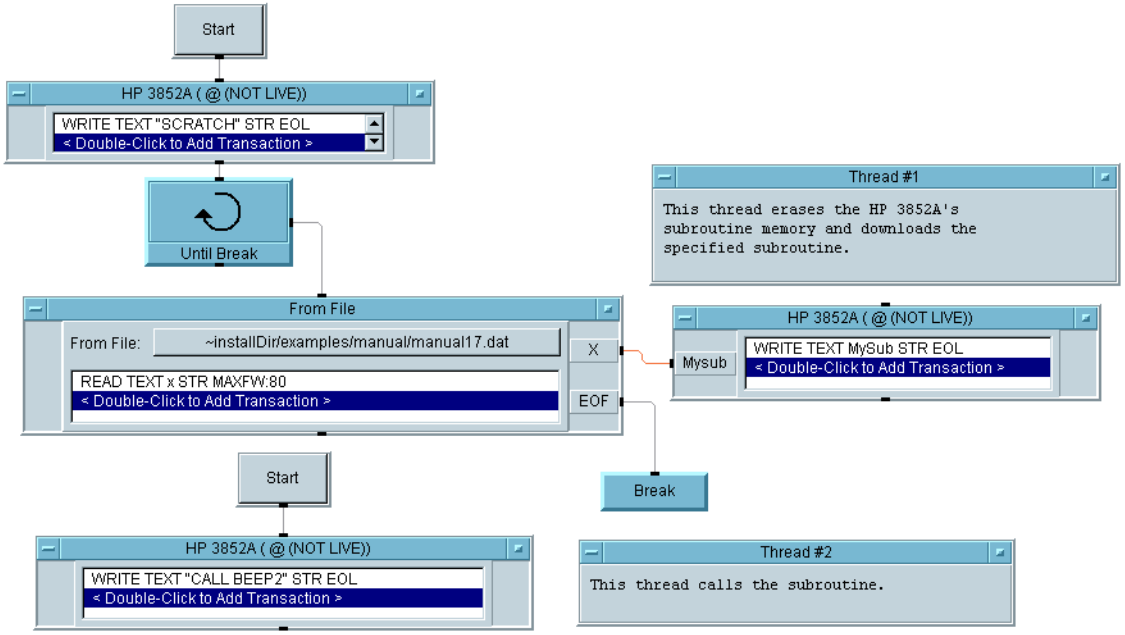


図5-12. 例: 機器へのダウンロード

論理ユニットとI/Oアドレッシング

I/Oデバイスにアクセスするには、正しいアドレスを調べ、Instrument Managerを使って(第3章「機器の構成」を参照)、Instrument Propertiesダイアログ・ボックスのAddressフィールドに入力する必要があります。

このセクションでは、VEEのI/Oアドレッシング方式のうち、インタフェース論理ユニットと機器アドレスについて説明します。このアドレッシング方式はDirect I/O、Panel Driver、Component DriverのI/O動作をサポートするためのもので、VXI*plug&play*のI/O動作には用いられません。VEEにおけるVXI*plug&play*アドレッシングについては、79ページ「VXIplug&playドライバの構成」を参照してください。

注記

VEEでは、GPIB、RS-232シリアル、GPIOの各インタフェースがサポートされます。また、サポートされるGPIBインタフェースにHP E1406コマンド・モジュールを接続することにより、VXIデバイスにアクセスできます。

また、組込みVXIコントローラを使ったVXIバックプレーンへの直接アクセスもサポートされます。PCではE1383AおよびE1483A VXLinkインタフェース、HP 9000シリーズ700コンピュータではHP E1489C EISA/ISA-MXIbusインタフェースが使用できます。

VEEのアドレッシング方式では、論理ユニットを使用します。論理ユニットのセットアップは、VEEに付属するI/Oライブラリのインストールおよび構成時に、I/O Configユーティリティ・プログラムを使って行います。HP I/Oライブラリのインストールおよび構成方法と、I/O Configを使った論理ユニットのセットアップ方法については、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。インタフェースに対する論理ユニットの推奨される設定方法を表5-2に示します。

VEEで推奨されるI/O論理ユニット

VEEで推奨されるインタフェース論理ユニットを下に示します。I/Oライブラリのインストールおよび構成方法と、I/O Configユーティリティ・プログラムを使った論理ユニットのセットアップ方法については、*Installing the Agilent I/O Libraries (VEE for*

*Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。

表5-2. 推奨されるI/O論理ユニット

論理ユニット	PC(Windows 95、NT)	シリーズ700(HP-UX)
1	GPIB (82340または82341)	GPIB (E2070またはE2071)
2	GPIB (82340または82341)	GPIB (E2070またはE2071)
3	GPIB (82340または82341)	GPIB (E2070またはE2071)
4	GPIB (82340または82341)	GPIB (E2070またはE2071)
5	GPIB (82340または82341)	GPIB (E2070またはE2071)
6	GPIB (82340または82341)	GPIB (E2070またはE2071)
7	GPIB (82340または82341)	GPIB (E2070またはE2071)
8	GPIB (82340または82341)	GPIB (E2070またはE2071)
9	COM1シリアル・ポート	COM1シリアル・ポート
10	COM2シリアル・ポート	COM2シリアル・ポート
11	COM3シリアル・ポート	COM3シリアル・ポート
12	COM4シリアル・ポート	COM4シリアル・ポート
13	GPIO (HP E2075)	GPIO (HP E2075)
14	GPIB0 (National GPIBカード)	未使用
15	GPIB1 (National GPIBカード)	未使用
16	VXI (組込み、またはVXLinkを使ったPC)	VXI (組込み、またはEISA/ISA-MXIbusを使ったS700)
17	GPIB2 (National GPIBカード)	未使用
18	GPIB3 (National GPIBカード)	未使用

注記

最初のGPIBカードに対して推奨されるデフォルト論理ユニットは7です。それぞれのカードに固有の論理ユニットが必要です。

PC上のWindows 95/98では、82335 GPIBカードもサポートされます(Windows NTでは未サポート)。ただし、82335 GPIBカードで推奨される論理ユニットは3～7だけであり、論理ユニットの設定にはカード上のスイッチを使います(デフォルト設定は7)。また、218ページ「82335カード用のアドレス空間の除外(Windows 95/98のみ)」で説明するように、82335用のアドレス空間を除外する必要があります。

PC上のNational GPIBカードに対しては、論理ユニット14、15、17、18だけがサポートされます。HP 9000シリーズ700コンピュータ上ではこれらのGPIBカードはサポートされません。

I/Oアドレッシング

以下の部分では、各種のデバイスに対するアドレッシング方式について説明します。

シリアル・ポートのアドレッシング

シリアル・ポートは、`I/O Config`で割り当てる論理ユニットを使ってサポートされます。通常は、COM1シリアル・ポートに論理ユニット9が割り当てられます(表5-2参照)。この場合、COM1に接続されたデバイスのアドレスとして`9`を使用します。

GPIOデバイスのアドレッシング

GPIOデバイスは、`I/O Config`でGPIOインタフェースに割り当てる論理ユニットを使ってサポートされます。通常は、論理ユニット13がGPIOに用いられます。この場合、GPIOデバイスのアドレスとして`13`を使用します。

GPIBインタフェースおよびデバイスのアドレッシング

GPIBデバイスは、下記の方式でアドレッシングされます。

SPA[SA]

ここで:

S	GPIBインタフェースの論理ユニット
PA	GPIBデバイスの1次アドレス(有効範囲00～30)
SA	オプションの2次アドレス(有効範囲00～31)

例を2つ挙げます。

- 論理ユニット7、1次アドレス01のGPIBデバイスの場合、`Instrument Properties`ダイアログ・ボックスの`Address`フィールドに`701`と入力します。
- 論理ユニット14、1次アドレス09、2次アドレス02のGPIBデバイスの場合、`Instrument Properties`ダイアログ・ボックスの`Address`フィールドに`140902`と入力します。

GPIB論理ユニット. GPIBインタフェースは、`I/O Config`で割り当てる論理ユニットを使ってサポートされます。GPIBインタフェースの推奨される論理ユニットは表5-2に示されています。推奨される論理ユニット(1～8)がI/OライブラリによってGPIBインタフェースに割り当てられている場合、理論的にはVEEは最大8枚のGPIBカードにアクセスできます。この中にはサポートされる複数の種類のカードを混在させることができます。

- E2070またはHP E2071(シリーズ700コンピュータ用)の場合、論理ユニットはソフトウェアによって割り当てられます。論理ユニットは7、8、1、2、3、4、5、6という順序で割り当てられます。ただし、それぞれのカードに固有のベース・アドレスを設定する必要があります(ベース・アドレスの設定方法については、オーナーズ・マニュアルを参照してください)。
- 82340または82341(PC用)の場合、論理ユニットはソフトウェアによって割り当てられます。論理ユニットは7、8、1、2、3、4、5、6という順序で割り当てられます。ただし、それぞれのカードに固有のベース・アドレスを設定する必要があります(ベース・アドレスの設定方法については、オーナーズ・マニュアルを参照してください)。

- 82335(PC用、Windows 95/98のみ)の場合、論理ユニットはカード上のスイッチ設定によって決まります(デフォルトは7)。82335カードを複数インストールする場合、それぞれのカードに3～7の範囲の固有の論理ユニットを設定する必要があります(手順についてはオーナーズ・マニュアルを参照してください)。また、各カード用のアドレス空間を除外する必要があります。218ページ「82335カード用のアドレス空間の除外(Windows 95/98のみ)」を参照してください。

GPIB論理ユニット(PCのみ). National Instruments GPIBドライバでは、GPIB0、GPIB1、GPIB2、GPIB3という識別子を持つ最大4枚のGPIBカードを構成できます。これらのGPIBカードにアクセスするには、論理ユニット14、15、17、18を`I/O Config`を使ってGPIBカードに割り当てる必要があります(表5-2参照)。VEEでは、これらのGPIBカードに対して上記以外の論理ユニットはサポートされません。これを除けば、アドレッシング方法は他のGPIBカードと同じです。

GPIB上のVXIデバイスのアドレッシング

HP-IBコマンド・モジュールを使ってGPIB経由でVXIデバイスにアクセスするには、2次アドレスを使用します。VXIカード・ケージ内のHP E1406コマンド・モジュールを使用する場合、1次アドレスはコマンド・モジュール上のスイッチで設定され(デフォルトは09)、2次アドレスは個々のVXIデバイスの論理アドレスを8で割ったものになります。

例えば、Cサイズ・メインフレーム内にHP E1406Aコマンド・モジュール(アドレス09)があり、HP E1406Aが論理ユニット7のGPIBインタフェースに接続されている場合を考えます。VXIスロット内にHP E1326Bマルチメータがあり、論理アドレスが24に設定されている場合、マルチメータのアドレスは`70903`となります。

GPIBコマンド・モジュール経由で接続されたVXIデバイスの正しいアドレスを求めるために、2つの機器ドライバが用意されています。

- CサイズVXIメインフレーム内のHP E1405またはHP E1406 GPIBコマンド・モジュールに接続されたVXIデバイスのアドレスを知るには、`hpe140x.cid`ドライバを使います。

- BサイズVXIメインフレーム内のHP E1306 GPIBコマンド・モジュールに接続されたVXIデバイスのアドレスを知るには、`hpe1300a.cid`ドライバを使います(このドライバは、コマンド・モジュールを内蔵したHP E1300およびHP E1301 BサイズVXIメインフレームもサポートします)。

これらのドライバを使用するには、Instrument Managerを使ってドライバ用の機器パネルを追加します。手順については第3章「機器の構成」を参照してください。

注記

VXIスイッチ・ボックス内のモジュールの場合を除いて、VXIデバイスにはサブ・アドレス値を指定しないでください。詳細については次のセクションを参照してください。

アドレス/サブ・アドレス値の設定

ほとんどのGPIBおよびVXI機器は、サブ・アドレスを使用しません。VXIスイッチ・ボックスを使用する場合、またはサブアドレスを使用するメインフレーム機器(HP 3235Aスイッチ/テスト・ユニット、HP 3488Aスイッチ/コントロール・ユニットなど)にアクセスする場合を除いて、サブ・アドレス値は入力しないでください。

注記

サブ・アドレス値が用いられるのは、サブ・アドレスをサポートするデバイスに対するHP機器ドライバを使用する場合だけです。Direct I/Oを使用する場合はサブ・アドレスを使用しないでください。

例を2つ示します。

- HP 3235Aスイッチ/テスト・ユニット内のモジュールにアクセスする場合、Instrument Managerを使って(第3章「機器の構成」参照)、Instrument Propertiesダイアログ・ボックスの Addressフィールドに HP 3235A自体のGPIBアドレス(701など)を入力します。

  個々のモジュールのサブ・アドレスを、Advanced Instrument Propertiesダイアログ・ボックス(Panel Driverタブ)のSub Addressフィールドに入力します。Sub Addressフィールドのエントリについては、HP 3235A機器ドライバのオンライン・ヘルプ(Help ⇒ Instruments)を参照してください。

- VXIスイッチ・ボックス内のモジュールにアクセスするには、スイッチ・ボックスのGPIBアドレス(70902など)をAddressフィールドに入力し、個々のモジュールのサブ・アドレスをSub Addressフィールドに入力します。Sub Addressフィールドのエントリについては、VXIスイッチ・ボックスの機器ドライバのオンライン・ヘルプを参照してください。

VXIバックプレーンの直接アドレッシング

VEEでは、下記のシステムでVXIバックプレーンを直接アドレッシングできます。

- HP 623x VXI Pentiumコントローラ
- EPC-7またはEPC-8 VXIコントローラ(EPConnectソフトウェアがインストールされている場合)
- HP E1383AまたはHP E1483A VXLink(ISA-VXI)インタフェースによってVXIカード・ケージに接続されたPC(EPConnectソフトウェアがインストールされている場合)
- HP V743 VXI組込みコントローラ
- HP E1489C EISA/ISA-MXIbusインタフェースによってVXIメインフレームに接続されたHP 9000シリーズ700コンピュータ

推奨される論理ユニットが`I/O Config`によって設定されている場合(表5-2参照)、VEEは論理ユニット16によってVXIバックプレーンにアクセスします。VXIデバイスのアドレスは、論理ユニット(16)にVXIデバイスの論理アドレスを付加したものです。

例えば、HP EPC-7 VXIコントローラとHP 1411Bディジタル・マルチメータがVXIメインフレームにインストールされているとします。HP 1411Bの論理アドレスが24に設定されていたとすると、VXIアドレスは`16024`になります。GPIB経由でVXIデバイスにアクセスする場合と異なり、論理アドレスを8で割る必要はありません。

82335カード用のアドレス空間の除外(Windows 95/98のみ)

82335カードではメモリ・マップ方式のI/Oアドレッシングが用いられるため、GPIBインタフェースが必要とするアドレス空間がメモリ・マネージャ・プログラムによって使われないように除外しておく必要があります。

注記

82340および82341カードと、National Instruments GPIBカードはメモリ・マップ方式のI/Oアドレッシングを使用しないため、これらのカードにはこのセクションの内容は当てはまりません。また、組込みコントローラの内蔵GPIBインタフェースにもこのセクションの内容は当てはまりません。
82335カードはWindows 95/98のみでサポートされ、Windows NTではサポートされません。

アドレス空間を除外する手順:

1. 82335カードをインストールします。カードは出荷時に論理ユニット7に初期設定されています。通常は論理ユニット7を使いますが、複数の82335カードを使用する場合、それぞれのカードに3～7の範囲の異なる論理ユニットを設定する必要があります。

2. SYSTEM.INI ファイル(C:\Windows ディレクトリに存在)の [386Enh] セクションに、論理ユニットに合わせて適切な行を追加します。

論理ユニット:	SYSTEM.INIに追加する行:
3	EMMEXCLUDE=0CC00-0CFFF
4	EMMEXCLUDE=0D000-0D3FF
5	EMMEXCLUDE=0D400-0D7FF
6	EMMEXCLUDE=0D800-0DBFF
7 (デフォルト)	EMMEXCLUDE=0DC00-0DFFF

3. CONFIG.SYS ファイル (ルート・ディレクトリに存在) にメモリ・マネージャの DEVICE行(DEVICE=EMM386.EXEなど)が存在する場合、このファイルも修正する必要があります。下の表に従って、アドレス空間を除外するパラメータ(論理ユニット7の場合はX=DC00-DFFF)を追加します。

論理ユニット:	CONFIG.SYSの修正内容:
3	DEVICE=EMM386.EXE X=CC00-CFFF
4	DEVICE=EMM386.EXE X=D000-D3FF
5	DEVICE=EMM386.EXE X=D400-D7FF
6	DEVICE=EMM386.EXE X=D800-DBFF
7 (デフォルト)	DEVICE=EMM386.EXE X=DC00-DFFF

注記

複数の82335カードをインストールしている場合、それぞれのカードに対応するアドレス空間を除外する必要があります。例えば、2枚のカードが(論理ユニット3と7に)インストールされている場合、SYSTEM.INIの[386Enh]セクションに下記の行を追加します。

```
EMMEXCLUDE=0CC00-0CFFF
EMMEXCLUDE=0DC00-0DFFF
```

また、CONFIG.SYSファイルにEMM386.EXEのDEVICE行が存在する場合、下記のようにパラメータを追加します。

```
DEVICE=EMM386.EXE X=CC00-CFFF X=DC00-DFFF
```

4. コンピュータをリブートし(Start ⇒ Shut Down)、Windowsを再起動します。

6

パネル・ドライバおよびコンポーネント・ドライバ・オブジェクトの使用

パネル・ドライバおよびコンポーネント・ドライバ・オブジェクトの使用

この章では、VEEのPanel Driverオブジェクトと Component Driverオブジェクトの使用法を説明します。

Panel DriverオブジェクトとComponent Driverオブジェクトの理解

このセクションでは、Panel Driverオブジェクトと Component Driverオブジェクトを効率的に使うために役立つ背景情報と詳細情報について説明します。

パネル・ドライバの内部情報

VEEでPanel Driverオブジェクトと Component Driverオブジェクトを使う場合、適切なパネル・ドライバ("ID")が存在し、そのドライバに合わせて機器が構成されている必要があります。これらの機器ドライバは、「VEEドライバ」または「機器ドライバ」と呼ばれることがあります。パネル・ドライバ・ファイル(.cidファイル)が存在し、Panel Driverオブジェクトと Component Driverオブジェクトを使用するように構成されている必要があります。ただし、これらのファイルはDirect I/Oや VXI*plug&play*の動作には用いられません。

パネル・ドライバ・ファイル

パネル・ドライバは、特定のテスト機器に固有の性質を記述します。Panel Driverオブジェクトまたは Component Driverオブジェクトを使って機器を制御するには、ドライバ・ファイルが必要です。

パネル・ドライバ・ファイル(.cidファイル)は、VEEのインストール時に選択によってシステム・ディスクにコピーされます。各ドライバ・ファイルには2種類の基本的な情報が含まれます。

- 機器の機能と、機能の設定および問合せのためのコマンドの記述
- Panel Driverオブジェクトのオープン・ビューに表示されるグラフィカル操作パネルの外観と操作の記述

コンポーネント

Panel Driverオブジェクトと Component Driverオブジェクトは、各機器の機能を内部的に**コンポーネント**として表現しています。コンポーネント名は、プログラミング言語の変数名に相当します。コンポーネントは機器の機能設定や測定値を保持するために用いられます。

図6-1に、HP 3478A電圧計のコンポーネントの一部を示します。

コンポーネント名	機器機能
ARANGE	オートレンジ機能のオン/オフ
FUNCTION	測定機能が電圧、電流、抵抗のうちどれか
TRIGGER	トリガ・ソースが内部、外部、高速、シングルのうちどれか
READING	最新の測定値

コンポーネントへのアクセスは、対話的にも、プログラムからでも実行できます。対話的にコンポーネントにアクセスするには、Panel Driverのオープン・ビューにあるラベル付きのボタンまたは表示をクリックします。グラフィカル・プログラムからコンポーネントにアクセスするには、入力端子または出力端子としてコンポーネントを追加します。図6-1に例を示します。コンポーネントの詳細な使用手順については、229ページ「技法の紹介」と230ページ「プログラムでのComponent Driverオブジェクトの使用」を参照してください。

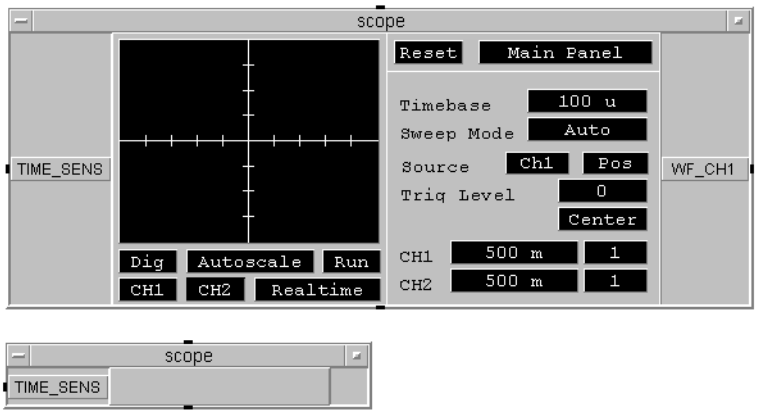


図6-1. ドライバ・コンポーネントへのアクセス

ステート

機器の**ステート**とは、あるドライバの全コンポーネントの値の特定の組合わせをいいます。電圧計で交流電圧を測定する場合、ドライバの全コンポーネントを特定の値の組合わせに設定します。直流電圧を測定する場合は別の値の組合わせを使います。これらは電圧計の2つのステートです。図6-2に電圧計の2つのステートを示します。

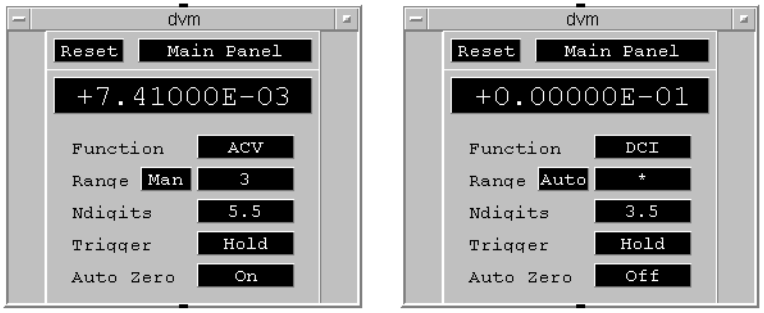


図6-2. 電圧計の2つのステート

VEEでは、Panelドライバの各インスタンスが異なる測定ステートを表します(Panel Driverオブジェクトは「ステート・ドライバ」と呼ばれることもあります)。同じ物理機器を異なる測定ステートにプログラムするために、1つのプログラムで複数のPanel Driverを使用することもできます。

同じ機器名(Name)で複数のPanel Driverオブジェクトを作成すると、それらはすべて同じ物理機器と通信します。

パネル・ドライバによるI/Oの仕組み

Panel Driverオブジェクトまたは Component Driverオブジェクトをプログラムに配置すると、VEEはステート・レコードをメモリ上に作成します。このステート・レコードは、特定の機器名(Name)に対応します。Nameの詳細については、227ページ「名前の重要性」で説明します。

同じNameを参照するすべてのドライバ・ベース・オブジェクトは、1つのステート・レコードを共有します。ステート・レコードは、機器の各コンポーネントの**現在**値を反映します。Panel Driverオブジェクトまたは Component Driverオブジェクトを使ってコンポーネントに書き込むと、物理機器とステート・レコードの両方が更新されます。

Direct I/Oを使って機器に書き込んだ場合、ステート・レコードが物理機器の実際のステートに一致しなくなるため、ステート・レコードは無効と記録されます。しかし、そのあとPanel Driverオブジェクトまたは Component Driverオブジェクトを使うと、機器のステートがリコールされ、物理機器とステート・レコードが再び一致するようになります。

Panel DriverオブジェクトとComponent Driverオブジェクトの動作には、重要な違いがいくつかあります。

パネル・ドライバの動作

Panel Driverが動作するときには、グラフィカル操作パネルで定義されたステートに物理機器のステートを一致させるために必要なコマンドだけが送信されます。

必要な場合、Panel Driverはリセット・コマンドを送り、物理機器のすべての設定をアップデートします。この動作は、第3章「機器の構成」で説明したIncremental Modeの設定に依存します。

Incremental ModeをONに設定した場合、現在のステート・レコードと、Panel Driverに定義された目的のステートとが比較され、どのコンポーネントを変更するかが決められます。必要なコンポーネントを更新するためのコマンドだけが送信されます。

Incremental ModeをOFFに設定した場合、または現在のステート・レコードが無効と記録されている場合、物理機器のステートと目的のステートとの同期を保証するため、すべてのコンポーネントを更新するコマンドが明示的に送られます。

Panel Driverが動作するのは、シーケンス入力ピンがアクティブになるか、オープン・ビューに表示されている操作パネル・ボタンのどれかをクリックした場合です。

コンポーネント・ドライバの動作

Component Driverが動作するときには、入力端子になっているコンポーネントだけに書き込み、出力端子になっているコンポーネントだけを読み取ります。このため、Component Driverオブジェクトの動作はPanel Driverオブジェクトよりも一般に高速です。Panel Driverの場合、特定のステートを実現するために**多数の**コンポーネントに書き込む場合がありますが、コンポーネント・ドライバは指定されたコンポーネントにしか書き込みません。

コンポーネントの読み書きは、表示された端子の上から下へという順序で行われます。この動作順序が重要なのは、1つのコンポーネントの値に基づいて別のコンポーネントの値を変更する場合です。この相互作用のことを**カップリング**と呼びます。コンポーネント・ドライバの場合、これは手動で管理する必要があります。

複数のドライバ・オブジェクト

下記のような複数のオブジェクトを使用する場合、いくつか注意しなければならないことがあります。

- 同じ機器名(Name)を使用するもの
- 同じ機器アドレスを使用するもの
- 同じドライバ・ファイルを使用するもの

名前の重要性. Instrument Propertiesダイアログ・ボックスのNameフィールドは、各機器オブジェクトを物理機器のアドレスやその他の構成情報に論理的にマッピングする役割を果たします。機器オブジェクトのNameを調べるには、オブジェクト・メニューでShow Configをクリックします。オブジェクト・タイトルのテキストは、Nameと一致しない場合があります。

一般に、特定の物理機器を参照するには1つのNameだけを使用するようにします。同じ機器アドレスを複数のNameで参照すると、Panel Driverオブジェクトを使ったプログラムで予期しない結果が発生するおそれがあります。VEE内部の機器ステート・レコードはNameによって管理されています。Nameが異なる2個のPanel Driverオブジェクトが同じアドレスに書き込むと、それぞれのステート・レコードが無効になってしまいます。

Direct I/Oを使う場合には、同じ物理機器に対して複数のNameを使用しなければならない場合もあります。これが必要になるのは、Advanced Instrument Propertiesダイアログ・ボックスのDirect I/Oタブの設定を、それぞれのダイレクトI/O動作ごとに切り換えなければならない場合です。

例えば、オシロスコープに対して、あるコマンドはデータの最後の文字でEOIをアサートし、あるコマンドはEOIなしで送りたいとします。この場合、Nameが Scope_EOIの機器と、NameがScopeの機器の2つを用意します。ScopeとScope_EOIはAddress設定を同じにし、END on EOLの設定を変えます。

設定されたNameは、機器オブジェクトをメニューから選択したときにデフォルトのタイトルとして表示されます。タイトルを編集しても、Nameとの関係には**一切影響しません。**

機器を含むプログラムを保存したりオープンしたりする場合にも、Nameが重要になります。プログラムを保存する場合、プログラム内の各機器オブジェクトのNameが保存されます。プログラムをオープンしたときに、ロードする各機器のNameに対応するI/O構成が参照されます。

例えば、My_Scopeという名前のDirect I/Oオブジェクトを含むプログラムを保存した場合、現在のI/O構成にMy_Scopeという名前の機器が存在しなければなりません。問題のオブジェクトがPanel DriverまたはComponent Driverの場合、現在のI/O構成のID Filename(ドライバ・ファイル)が、保存されたプログラムで用いられているものと一致する必要があります。

Namesはスペースを含めて**正確に**一致する必要がありますが、大文字小文字は区別されません。

ドライバ・ファイルの再使用. 名前が異なる複数のオブジェクトが同じドライバ・ファイルを使用することは可能であり、実際によくあることです。例えば、Supply1、Supply2、Supply3という名前の3つのプログラマブル電源が3つの異なるアドレスにあり、それらすべてがhp665x.cidドライバを使用する場合を考えます。Nameが異なっているので、それぞれの名前ごとに異なるステート・レコードが管理されます。したがって、Supply1のPanel DriverはSupply2やSupply3に対して何の影響も与えません。

技法の紹介

このセクションでは、Panel Driverオブジェクトと Component Driverオブジェクトを対話的に、またはプログラムで使用する場合のいくつかの技法について説明します。

Panel Driverオブジェクトの対話的使用

Panel Driverオブジェクトのオープン・ビューにはグラフィカル操作パネルがあり、測定ステートを対話的に構築できます。対応する物理機器をコンピュータに接続してLive Modeをオンにすると、測定ステートを構築しながら物理機器を対話的に制御することができます。

個々の設定を変更するには、グラフィカル操作パネルの対応するフィールドをクリックして、表示されるダイアログ・ボックスに入力します。測定を実行して結果を表示するには、数値またはXY表示の表示領域をクリックします。XY表示が更新されるには数秒間かかる場合があります。

プログラムによるPanel Driverオブジェクトの使用

プログラムにPanel Driverオブジェクトを追加する手順:

1. I/O ⇒ Instrument Manager...をクリックします。Instrument Managerダイアログ・ボックスが表示されます。

2. 目的の機器をクリックして強調表示させ、Panel Driverボタンをクリックします。

注記

機器に対してパネル・ドライバ・ファイルが構成されていない場合、Panel Driverボタンは非アクティブ(淡色表示)になります。構成手順については第3章「機器の構成」」を参照してください。

3. オブジェクトの輪郭が表示されたら、ワーク・エリアにカーソルを動かして1回クリックし、オブジェクトを配置します。

Panel Driverオブジェクトをプログラムで使用する場合、入出力端子を使ってコンポーネントの値を設定する方法があります。入出力端子は、実際にはそれぞれドライバのコンポーネントに対応します。端子を追加するには下記の2つの方法があります。

- Panel Driverオブジェクト・メニューからAdd Terminal ⇒ Data InputまたはAdd Terminal ⇒ Data Outputを選択します。端子としてまだ使われていない有効なドライバ・コンポーネントをすべて記載したリスト・ボックスが表示されます。端子として追加するコンポーネントをリスト内でダブルクリックします。

- Panel Driverオブジェクト・メニューから、Add Terminal by Component ⇒ Select Input ComponentまたはAdd Terminal by Component ⇒ Select Output Componentを選択します。その後、グラフィカル操作パネルのフィールドまたは表示エリアをクリックすると、対応するコンポーネントが端子として追加されます。

一般に、最初の方法のほうが、使用するコンポーネントの名前を推測しなくて済むので便利です。ただし、コンポーネントによってはグラフィカル操作パネルのどこにも表示されていないものがあります。この場合は2番目の方法でないとアクセスできません。

プログラムでのComponent Driverオブジェクトの使用

Component Driverオブジェクトをプログラムに追加する手順:

1. I/O ⇒ Instrument Manager...をクリックします。構成済みの機器のリストが表示されます。

2. 目的の機器をクリックして強調表示させ、Component Driverボタンをクリックします。

注記

機器に対してパネル・ドライバ・ファイルが構成されていない場合、Component Driverボタンは非アクティブ(淡色表示)になります。構成手順については第3章「機器の構成」を参照してください。

3. オブジェクトの輪郭が表示されたら、ワーク・エリアにポインタを動かして1回クリックし、オブジェクトを配置します。

コンポーネント・ドライバ・オブジェクトは主に、少数のコンポーネントを変更しながら機器制御オブジェクトを繰り返し実行する場合に用いられます。このような場合にPanel Driverオブジェクトよりも Component Driverオブジェクトが適しているのは、指定したコンポーネントしか読み書きしないため、高速に動作するからです。

図6-3にこのような状況を示します。このプログラムは、フィルタの周波数応答を測定するため、fgenから供給される入力周波数を掃引しながらdvmで応答を測定します。For Log Rangeに接続されたサブスレッドは繰り返し実行されるため、実行速度を改善するためにコンポーネント・ドライバが用いられます。なお、fgenとdvmの初期設定にはPanel Driverオブジェクトを使うのが便利です。

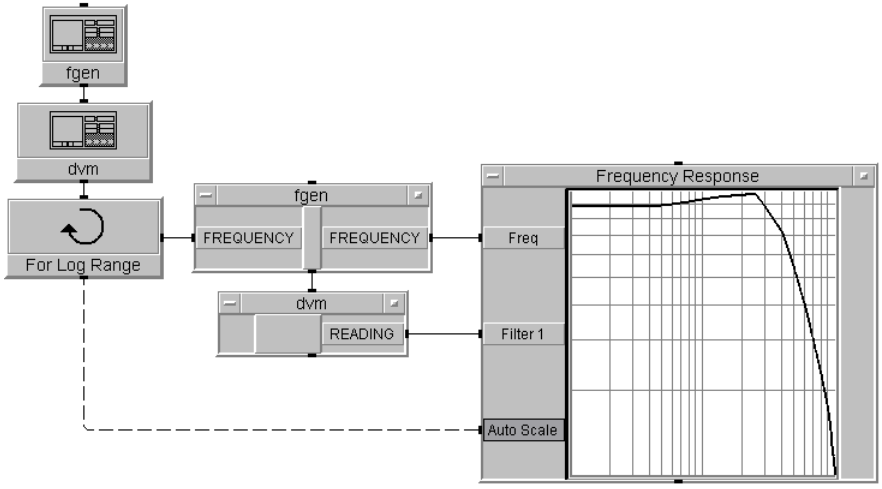


図6-3. Panel DriverとComponent Driverの使用

図6-3のプログラムは、サンプル・ディレクトリにあるファイルmanual15.veeに保存されています。

パネル・ドライバに関するヘルプの表示

パネル・ドライバに関するヘルプを見るには、Panel Driverオブジェクトまたは Component Driverオブジェクトのオブジェクト・メニューでHelpを選択します。表示されるダイアログ・ボックスから目的のヘルプ・トピックをオープンします。

7

VXI*plug&play*ドライバの使用

VXI*plug&play*ドライバの使用

VXI*plug&play*ドライバを使って機器と通信するには、適切なVXI*plug&play*ドライバ・ファイルと、VISA I/Oライブラリをインストールする必要があります。VISAのインストール方法については、46ページ「VXIplug&playについて」を参照してください。また、79ページ「VXIplug&playドライバの構成」で説明した手順で、機器に対してVEEを構成する必要があります。

VEEでVXI*plug&play*ドライバと通信するには、次のセクションで説明する`To/From VXIplug&play`オブジェクトを主に使用します。VEEの`Call`オブジェクトからVXI*plug&play*関数を呼び出すこともできます(250ページ「Callオブジェクトからの VXIplug&play関数の使用」を参照)。この2番目の方法は、VEEバージョン3.1との互換性のために残されています。

注記

プログラムの互換性:

VEEの以前のバージョンでも、VXI*plug&play*ドライバはサポートされていました。VEEバージョン3.2で`To/From VXIplug&play`オブジェクトが提供されました。このオブジェクトを使用しているVEE 3.2のプログラムは、VEEの後のバージョンと互換性があります。

VEEバージョン3.1では、VXI*plug&play*ドライバへの直接の`Call`アクセスだけが提供されていました。VEEバージョン3.1で`Call`オブジェクトを使ってVXI*plug&play*機器を制御していた場合、ドライバの32ビット・バージョンを使用するためにいくつかの変更を行えば、VEEの後のバージョンでもプログラムは動作します。

VISAのWindows 95/98バージョンと、VXI*plug&play*ドライバの32ビット・バージョンをインストールする必要があります。また必要なら、VXI*plug&play*ドライバの新しい位置を参照するように`Import`オブジェクトを変更します。`Call`オブジェクトを使ってVXI*plug&play*ドライバにアクセスする方法の詳細については、250ページ「Callオブジェクトからの VXIplug&play関数の使用」を参照してください。

To/From VXIplug&playオブジェクトの使用

VEEの機器構成にVXI*plug&play*機器を追加したら、プログラムからVXI*plug&play*ドライバが使用できるようになります。機器にアクセスするには、ドライバに含まれる関数を使います。`To/From VXIplug&play`オブジェクトからVXI*plug&play*関数パネルにアクセスできます。

`To/From VXIplug&play`を配置する手順:

1. `I/O` ⇒ `Instrument Manager`を選択します。`Instrument Manager`が表示され、現在構成済みのVXI*plug&play*機器(および構成済みの他の機器)がすべて表示されます。

2. 通信相手の機器を選択し、`Plug&play Driver`ボタンをクリックします。オブジェクトの輪郭が表示されます。

3. `To/From VXIplug&play` オブジェクトの輪郭をワーク・エリアの必要な位置に移動して、マウス・ボタンをクリックします。図7-1のようにオブジェクトが表示されます。

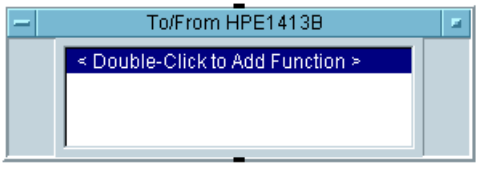


図7-1. To/From VXIplug&playオブジェクト

関数の選択

VXI*plug&play*関数は`To/From VXIplug&play`オブジェクトから選択します。

1. 空のトランザクションをダブルクリックするか、オブジェクト・メニューから`Add Trans`または`Insert Trans`を選択します。`Select a Function Panel`ダイアログ・ボックスが表示されます。ここには、測定や構成などの論理的カテゴリに分類されて関数パネルが表示されます(図7-2)。カテゴリはドライバごとに異なります。

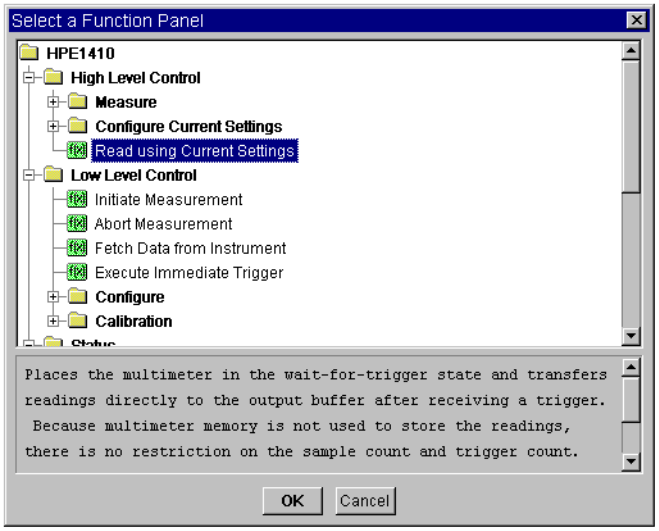


図7-2. Select a Function Panelダイアログ・ボックス

- ❑ `[+]`アイコンをクリックすると、関数パネルの階層構造が表示されます。
- ❑ `[-]` アイコンをクリックすると、階層構造内部の関数パネルが表示されなくなります。

- ❑ `[f(x)]` アイコンを選択すると、関数パネルが選択されます。ダイアログ・ボックス下部にその関数パネルに関する簡単な説明が表示されます。

ツリーの枝を完全に展開するには、展開するアイテムを選択して*キーを押します。

一般には、VXI*plug&play*バージョン3.x仕様に適合し、VEEで使用できる関数パネルだけが表示されます。

注記

VEEは適切なタイミングで自動的に`init()`を呼び出します。ただし、その他の初期化関数(`init_all()`、`init_next()`、`init_first()`など)がリストに表示される場合もあります。これらの関数はVXI*plug&play*仕様に定義されておらず、VEEではサポートされません。

これらの関数は選択しないでください。これらを使用しなければならない場合、プログラムの作成方法を変え、`Call`オブジェクトからVXI*plug&play*ドライバを呼び出すようにしてください。詳細については、250ページ「Callオブジェクトからの VXIplug&play関数の使用」を参照してください。

PREFIX`_init()`または*PREFIX*`_close()`に対するエントリはありません。これらの関数はVEEから自動的に実行されます。

2. `Select a Function Panel`ダイアログ・ボックスで`OK`をクリックします。
3. `Edit a Function Panel`というタブ付きのダイアログ・ボックスが表示されます。ここでは関数パネルのパラメータを指定します。

関数パネルのパラメータの編集

Edit a Function Panelダイアログ・ボックスを使うと、選択したVXI*plug&play*ドライバ関数に渡すコントロールや変数を設定できます。ダイアログ・ボックスにはPanelとConfigurationの2つのタブがあります。

Panelタブ. Panelタブ(図7-3)は、関数に渡す定数(コントロール)値を指定するために使います。

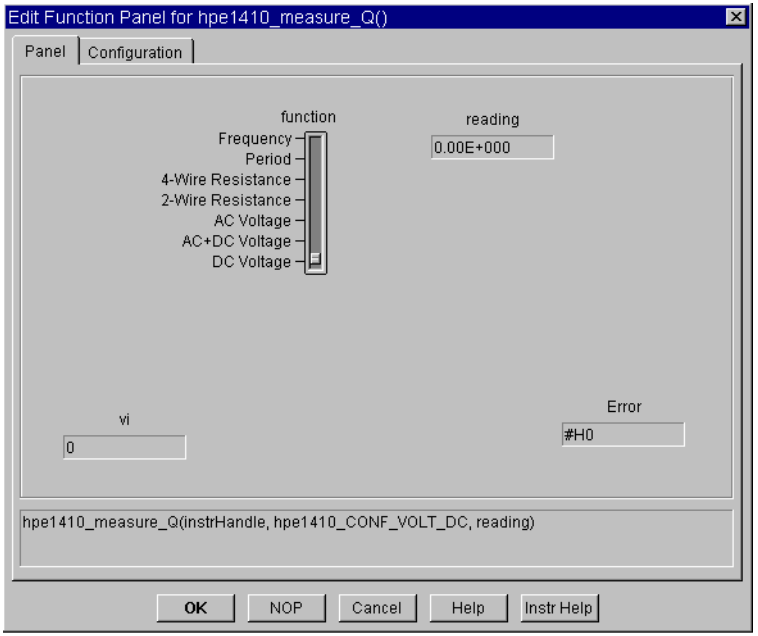


図7-3. Edit Function Panelダイアログ・ボックスのPanelタブ

- コントロール - このダイアログ・ボックスの上部には、定数パラメータを指定するためのコントロールが用意されています。コントロールの名前は、関数パネル・ファイルで指定されているラベルです。

- vi - 機器固有の「バーチャル機器」ハンドル(「セッション・ハンドル」とも呼ぶ)を表示します。ドライバのバージョンによってフィールドの名前は変わる場合がありますが、位置は常に関数パネルの左下隅です。

- Error - この関数パネルの実行時にエラーが発生した場合、0でない値を表示します。ドライバのバージョンによってフィールドの名前は変わる場合がありますが、位置は常に関数パネルの右下隅です。

- 関数呼出し - ダイアログ・ボックスの下部には、オブジェクトが実行されたときにドライバに送られるC関数とパラメータが表示されます。このコマンド文字列は、オブジェクトのオープン・ビューにトランザクションとして表示されます。

VXIplug&play関数パネルに関するヘルプの表示. Edit Function Panelダイアログ・ボックスのPanelタブの背景で右マウス・ボタンをクリックすると、関数パネルに関するヘルプが表示されます。関数の説明を記載したダイアログ・ボックスが表示されます。

コントロール(ラベル以外)の上で右マウス・ボタンをクリックすると、パラメータに関する説明が表示されます。

VXI*plug&play*ドライバの詳細なヘルプを見るには、To/From VXIplug&playオブジェクトのオブジェクト・メニューでInstrument Helpを選択します。

Configurationタブ. Configurationタブ(図7-4)は、関数に渡す変数を指定するために使います。これにより、パラメータ値をプログラムで設定することができます。

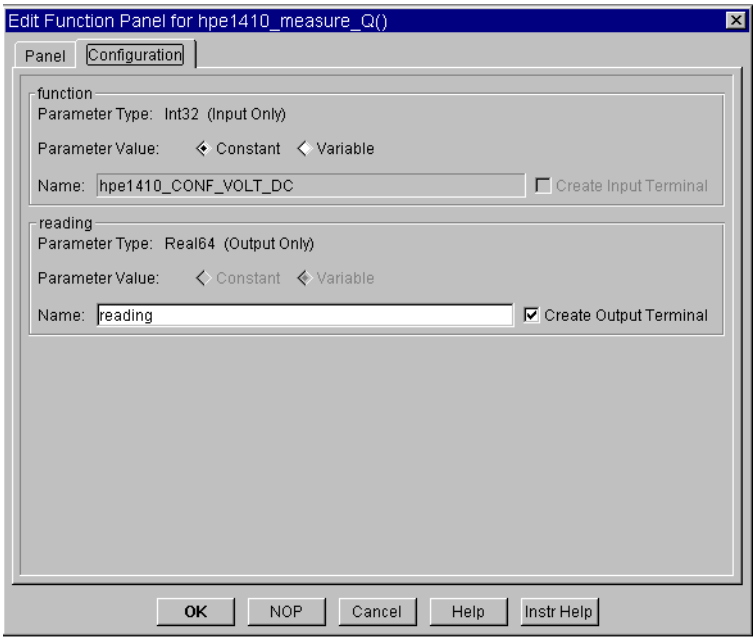


図7-4. Edit Function Panelダイアログ・ボックスのParameterタブ

パラメータ値はグループ単位で表示されます。各グループの名前は、VXI*plug&play*関数パネルで指定されたパラメータのラベル名です。図7-4では、functionとreadingがラベルです。各グループには下記のような情報が含まれます。

- Parameter Type - パラメータのデータ型と、パラメータが入力専用、入出力、出力専用のどれなのかを示します。
- Parameter Value - Constantを選択すると、このパラメータはPanelタブで設定される定数値として渡されます。Variableを選択すると、このパラメータは変数として渡され、パラメータの値をプログラムで変更できるようになります。一部のフィールド(測定値の出力など)は常に変数です。

- Name - Parameter TypeをVariableに設定すると、このフィールドが編集可能になります。デフォルトでは、変数名はラベル名と同じに設定されます(VEEの有効な変数名にするために多少の変更が加えられる場合があります)。これはVEEの任意の有効な変数名に変更できます。変数が入力変数の場合、式、関数呼出し、グローバル変数をこの編集フィールドに入力することもできます。

- Create Terminal - Parameter TypeをVariableに設定すると、このフィールドが編集可能になります。チェック・ボックスがチェックされており、Nameが端子名として存在しない場合、OKを押した時点で、ダイアログ・ボックスの指定に応じて入力端子、出力端子、入出力端子のどれかが(指定されたNameを名前として)作成されます。作成された端子を削除するには、オブジェクト・メニューからDelete Terminalを選択します。

  Nameを変更してCreate Terminalをチェックすると、新しい端子が追加されます。

  Nameが端子名として無効な場合、Create Terminalは淡色表示になります。

- Auto-Allocate Input - これは、入出力パラメータに対してConstantでなくVariableを設定した場合に表示されます。次のセクションで詳しく説明します。

NOPボタンを押すと、ダイアログ・ボックスの最新の設定が保存され、このトランザクションが"NOP"(No Operation)になります。これは、テキスト・ベースのコンピュータ・プログラムでコード行をコメント・アウトするのと同じ意味です。

To/From VXIplug&playオブジェクトに関するヘルプを表示するには、Helpボタンを押します。編集が終了したらOKボタンを押します。

ドライバ開発者によって作成された機器固有のヘルプを表示するには、Instr Helpボタンを押します。

Auto-Allocate機能(配列と文字列の受け渡し). 一部のVXI*plug&play*関数は、配列やテキスト文字列の形でデータを返します。VXI*plug&play*仕様では、VXI*plug&play*関数でメモリを割り当てて返すことができないので、アプリケーション(VEE)が配列や文字列のためのメモリを割り当てるように決められています。VEEがメモリを割り当て、そのメモリに対して関数が書き込みます。

`Auto-Allocate`機能を使うと、割り当てるメモリ量をVEEに指示するのが簡単になります。VEEは、正しいデータ型と形状を要求されたサイズで割り当てます。

関数へのパラメータが配列またはテキスト文字列を必要とする変数の場合、`Parameters`タブに`Auto-Allocate input`というフィールドが追加されます。例えば、図7-5 のダイアログ・ボックスでは、`readings` に配列を入力できます。`Parameters`タブで`Auto-Allocate input`が選択されています。

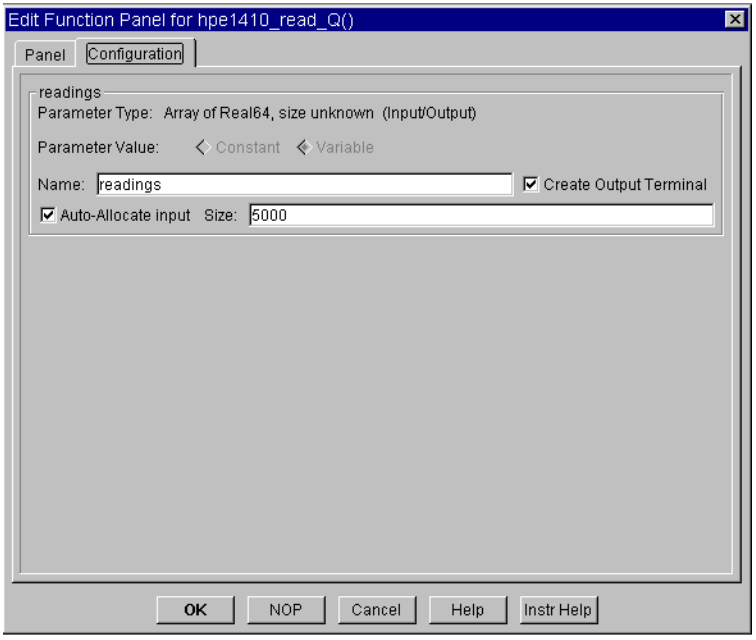


図7-5. Auto-Allocate Input機能の選択

`Auto-Allocate input`を選択すると、`Size`フィールドがアクティブになります。デフォルトのサイズは`5000`ですが、入力データを割り当てる任意のサイズを指定できます。渡す配列または文字列がどれだけの大きさになるのかを知る必要があります。

このパラメータに対しては入力端子は作成されず、パラメータのためのメモリがVEEによって自動的に割り当てられます。

配列の場合、Sizeは配列の要素数を表します。テキスト文字列の場合、Sizeは文字数(バイト数)を表します。関数が必要とする配列または文字列のサイズについて知るには、Instrument Helpを参照するか、Panelの背景またはパラメータ上で右マウス・ボタンをクリックします。

注記

Auto-Allocate input機能を使う場合、関数に対してデータ入力端子は作成されません。データ入力端子がすでに存在する場合、To/From VXIplug&playオブジェクトから削除する必要があります。

Auto-Allocate inputを選択(チェック)しないと、関数に対してデフォルトで入力端子と出力端子が作成されます。正しい型、形状、メモリ量を割り当てるオブジェクトを作成して、入力端子に接続する必要があります。入力に必要なメモリを手動で割り当てる方法については、246ページ「パラメータの受渡し」を参照してください。

関数が返すすべての値を格納できる量のメモリを必ず割り当ててください。メモリが不足していると、メモリが上書きされ、一般保護違反やセグメンテーション違反が発生します。VXI*plug&play*のDLLはVEEに直接リンクされるので、これが発生するとVEEがクラッシュして終了するおそれがあります。

VXIplug&playドライバに関するヘルプの表示

To/From VXIplug&playオブジェクトのオブジェクト・メニューからInstrument Helpを選択すると、機器メーカから提供されたヘルプ・ファイルが表示されます。このヘルプ・トピックには、パラメータに必要なデータ型など、VXI*plug&play*ドライバを使用するための情報が記載されています。

各関数に関するヘルプを見る方法については、239ページ「VXIplug&play関数パネルに関するヘルプの表示」を参照してください。

VEEプログラムの実行

To/From VXIplug&playオブジェクト内のトランザクションは、上から下へという順番で実行されます。このセクションでは、To/From VXIplug&playオブジェクトの実行時の動作について説明します。

ドライバの初期化とクローズ

プログラムをロードまたは作成したあと最初に実行する際には、To/From VXIplug&playオブジェクトが制御する機器が初期化されるため、遅延が生じます。この初期化では、機器が既知の初期状態に設定されます。以後のプログラム実行では初期化は行われず、普通にプログラムが実行されます。

プログラムから制御される機器は、1つのVEEセッションで1回必ず初期化される必要があります。VXI*plug&play*リソース・マネージャが「機器検索」を行って、指定のアドレスに機器が接続されているかどうかを確認し、機器を既知の状態に設定します。このためにかかる時間は不定であり、機器1台あたり10秒程度かかることもあります。この遅延は、各機器に対するTo/From VXIplug&playオブジェクトが初めて実行されるときに発生します。

初期化は1回のVEEセッションで1回しか実行されないので、機器を既知の状態に設定する関数(クリア、リセットなど)をプログラムが動作するたびに実行する必要があります。別のプログラムをロードするか、VEEを終了すると、VXI*plug&play*ドライバは自動的にクローズされます。

初期化に関する高度な情報

このセクションでは、VXI*plug&play*の初期化に関するVEEの実装の詳細について説明します。このセクションの内容を理解することは、VXI*plug&play*ドライバを使用するVEEプログラムの作成に必須ではありません。

すべてのVXI*plug&play*ドライバには、*PREFIX*_init()関数と*PREFIX*_close()関数がなければならないと決められています。これらの関数はVEEから自動的に呼び出されます。

init()関数の目的は、機器を既知の状態にし、「セッション・ハンドル」を取得することです。構成時にVEE Nameで指定された機器は、プログラムで最初に実行されたときに固有のセッション・ハンドルを割り当てられます。このセッション・ハンドルはプログラム全体を通してその機器を一意に識別するために用いられます。

同じ機器(同じVEE Nameを持つもの)と通信するすべてのTo/From VXIplug&playオブジェクトは、同じセッション・ハンドルで識別されます。セッション・ハンドルは、関数パネルのPanelタブの左下隅にあるviフィールドに表示されます。複数

のTo/From VXIplug&playオブジェクトの間でのセッション・ハンドルの受渡しは、VEEが自動的に管理します。

init()関数は通常時間がかかるので、必要なときだけ呼び出されます。最初のTo/From VXIplug&playオブジェクトがプログラムで実行されたときに、適切なinit()関数が呼び出されます。init()が呼び出されたときには、ドライバの構成に応じて、識別情報の問合せやリセットが実行されることもあります。

close()関数の目的は、セッション・ハンドルをクローズし(セッション・ハンドルの数には限りがあるため)、機器をオフラインにし、機器に関連づけられたデータをクリアし、必要なら機器固有の動作を実行することです。VEEはclose()関数を下記の場合に呼び出します。

- New、Open、Exitのどれかが選択されたとき
- 1つのVEE Name(dvmなど)に対するすべてのTo/From VXIplug&playオブジェクトが削除されたとき
- VXIplug&play Instrument Propertiesダイアログ・ボックスでAddressまたはinit()パラメータの値が変更されたとき。この場合、close()が呼ばれたあと、新しい値でinit()が呼び出されます。

エラーと注意のチェック

トランザクションが実行されるたびに、関数はVEEにステータス値を返し、VEEは自動的にこれをチェックします。関数の成功を示す値の場合は、次のトランザクションが実行されます。

エラー・チェック. 返されたステータス値がエラーを示していた場合、VEEはプログラムを停止し、エラーを報告します。エラーをトラップするためのエラー出力ピンが存在する場合、プログラムは停止せず、エラー出力ピンが伝搬します。エラー・メッセージの詳細を知るにはerrorInfo()オブジェクトを使います。

VEEは自動的に*PREFIX*_error_message()関数を呼び出し、メーカが用意したエラー情報をVXI*plug&play*ドライバから取得します。この情報はVEEのエラー・メッセージまたはerrorInfo()の結果として用いられます。

注記

エラーが発生すると、機器は未知の状態になります。プログラムの先頭で固有のリセットまたはクリア関数を呼び出していないと、次にプログラムを実行したときには機器がどんな状態にあるかわからなくなります。

注意チェック. 返されたステータス値が注意の場合、プログラムは休止し、注意ダイアログ・ボックスが表示されます。注意ダイアログ・ボックスには機器メーカからの情報が表示され、プログラムの実行を停止するか継続するかをユーザが選択できます。

注意メッセージはプログラムからトラップすることができません。しかし、どのような注意メッセージがドライバから一般的に生成されるかを知っていれば、VEEプログラムで処理することは可能です。例えば、機器からデータを読み取る準備ができていないという注意メッセージが生成される場合、Delayオブジェクトを使うか、To/From VXIplug&playオブジェクトをループの中に入れて再読取りを行う方法があります。

既知の注意条件をVEEプログラムで処理する場合、注意メッセージ・ダイアログ・ボックスを抑制することができます。このためには、To/From VXIplug&playオブジェクトのPropertiesダイアログ・ボックスで、Ignore Cautions Returnedのチェック・ボックスを選択します。

注記

一般に、注意メッセージを無視する(Ignore Cautions Returnedチェック・ボックスをチェックする)のは不要であり、プログラムで注意条件を処理する方法が確実にわかっていない限り避けるべきです。

パラメータの受渡し

VXI*plug&play*仕様では、データを要求する前に呼出し側がメモリを割り当てて渡さなければならないと決められています。一部のVXI*plug&play*関数は、読み取るデータを配列に格納します。これらのVXI*plug&play*関数のほとんどには、渡された配列のサイズを指定するパラメータがあり、配列の大きさが十分でない場合にはエラーが発生します。この場合、任意サイズの配列を割り当てて、関数にその大きさを知らせます。すると関数は、指定されたサイズまで配列にデータを書き込みます。

注意

注意他のVXI*plug&play*関数の中には、渡された配列が読み取るデータに十分であると仮定し、サイズを考慮せずに書き込むものがあります。これは特にテキスト文字列を扱うものに見られます。メモリが不足していると、メモリが上書きされ、一般保護違反やセグメンテーション違反が発生します。VXI*plug&play*のDLLはVEEに直

接リンクされるので、これが発生するとVEEがクラッシュして終了するおそれがあります。

注記

配列または文字列データ入力に対してメモリを割り当てる最も簡単な方法は、Auto-Allocate機能を使うことです。239ページ「VXIplug&play関数パネルに関するヘルプの表示」を参照してください。割り当てるサイズを決める必要はありますが、サイズを指定すればメモリは自動的に割り当てられます。

データに必要なメモリ量を調べるには、ドライバのヘルプ・ファイルを参照してください。To/From VXIplug&playオブジェクトのオブジェクト・メニューから、Instrument Helpを選択します。ヘルプ・ファイルを見れば、配列に必要な大きさがわかります。

Auto-Allocateを使わない場合、メモリを割り当てるオブジェクトを作成し、To/From VXIplug&playオブジェクトのデータ入力端子に接続します。

- 配列入力の場合、適切な型のAlloc Arrayオブジェクトを使い、サイズを正しく設定します。
- 文字列入力の場合、Formulaオブジェクトを使います。Formulaオブジェクトのデータ入力端子を削除し、256*"a"のような式を入力します。これにより、256文字のa(+ヌル・バイト)からなる文字列が作成されます。大部分のVXI*plug&play*関数はテキスト・パラメータに256文字より多くは書き込みませんが、念のためテキスト入力を必要とする関数パネルのヘルプをチェックしてください。

サンプル・プログラム

図7-6に示すのは、`To/From VXIplug&play`オブジェクトを使ってHP E1410A VXIマルチメータと通信する簡単なプログラムです。

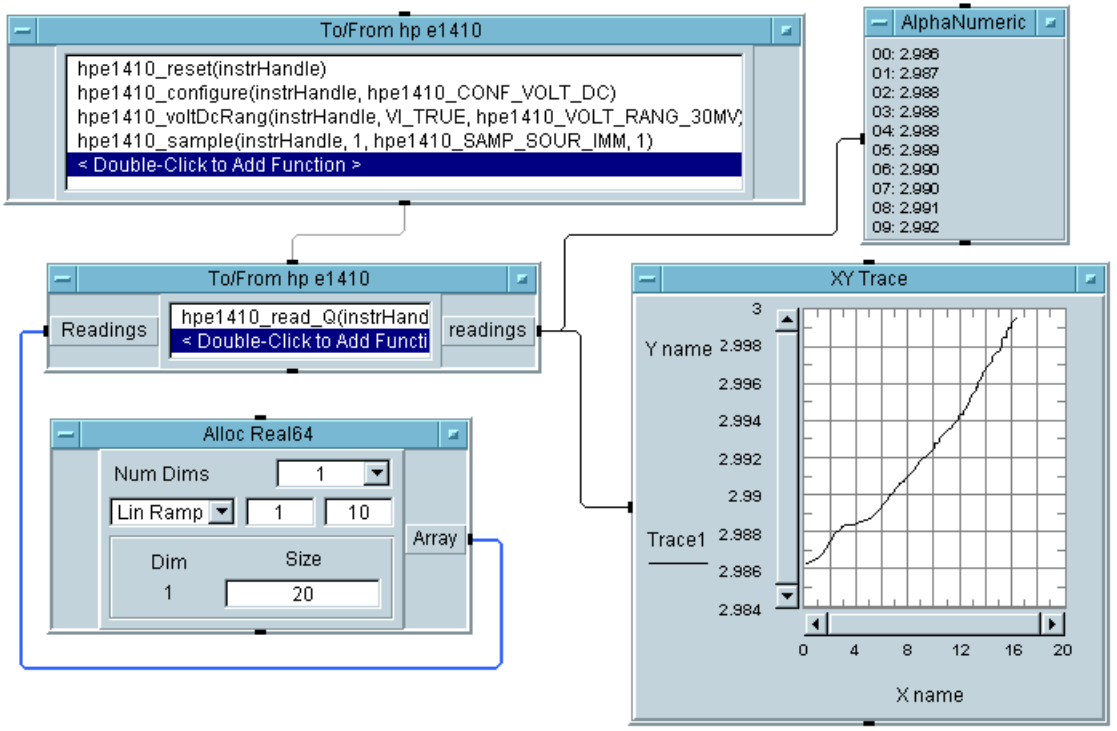


図7-6. To/From VXIplug&play Objectsを使ったプログラム

VXIplug&playに関する制限

VEEでVXI*plug&play*ドライバを使用する場合、下記のような制限があります。

- `Bus I/O Monitor`オブジェクトはVEE自体からのI/Oだけを表示し、VXI*plug&play*ドライバからのI/Oは表示しません。VXI*plug&play*ドライバはVEEにリンクされるCプログラムです。必要ならハードウェア・バス・モニタを使用することをお勧めします。

- VXI*plug&play*仕様で必須とされていないオプション機能のいくつか(コールバックなど)はVEEでサポートされません。

- `I/O ⇒ Advanced I/O`オブジェクト(`Interface Operations`、`Instrument Event (SPOLL)`、`Interface Event`など)はすべてVXI*plug&play*ではサポートされません。

- VXI*plug&play*では、ライブ・モード/非ライブ・モードの概念はサポートされません。プログラムを実行する際には、プログラムで使用するすべての機器がコンピュータに接続されている必要があります。ただし、プログラムをオープンする際には、プログラムで使用する機器が接続されていなくてもかまいません。また、プログラムを作成する際にも機器が接続されている必要はありません。VXI*plug&play*ドライバがインストールされていれば、`To/From VXIplug&play`オブジェクトを使用して関数呼出しを指定できます。

- VXI*plug&play*ドライバと、VEEのその他の機器制御方法(`Direct I/O`、`Panel Driver`、`Component Driver`などのオブジェクト)を、同じプログラムで同じ機器と通信するために併用することはできません。ただし、1つのプログラムで、ある機器との通信にVXI*plug&play*ドライバを使い、別の機器との通信に他の機器制御方法を使うのはかまいません。

注記

VXI*plug&play*仕様は、現在もアップデートと拡張が続けられています。VEEの新しいリビジョンが出る前に、VXI*plug&play*コンソーシアムによって新しい機能が仕様に採用される可能性があります。VXI*plug&play*仕様ではリビジョン情報をドライバ・ライブラリに記録することが決められていないため、VEEからはドライバの互換性をチェックすることができません。したがって、VEEが現在サポートしているVXI*plug&play*仕様にドライバが適合するかどうかを機器メーカの情報によって確認していただく必要があります。

CallオブジェクトからのVXIplug&play関数の使用

下記の場合には、VEEのCallオブジェクトからVXI*plug&play*を使用します。

■ 既存のプログラムとの互換性が必要な場合

VEEバージョン3.1で作成された既存のプログラムでVXI*plug&play*を使用している場合、Callオブジェクトを使えば最小限の変更でプログラムを使い続けることができます。ただし、長期的にそのプログラムを使用する予定がある場合は、To/From VXIplug&playオブジェクトで標準の関数パネル・アクセスを使って書き直した方がいいかもしれません。詳しくは、235ページ「To/From VXIplug&playオブジェクトの使用」を参照してください。

■ 古いドライバにアクセスする場合

HP 以外のVXI*plug&play* ドライバの古いバージョン(1995 年以前) には、VXI*plug&play*仕様の旧バージョンに従って作成されているものがあります。VEEのCallオブジェクトを使えば、これらのドライバにアクセスできます。

上記の理由がある場合以外は、235ページ「To/From VXIplug&playオブジェクトの使用」に記された方法でVXI*plug&play*ドライバを使用するようにしてください。

ダイナミック・リンク・ライブラリまたは共有ライブラリのVEEでの使用

このセクションでは、必要なファイルをインストールしたあとでVXI*plug&play*ドライバをVEEにロードする手順について説明します。

VXI*plug&play*ドライバをVEEプログラムで使用するには、下記の手順を実行します。

1. ライブラリをインポートします。
2. ライブラリを使用するルーチンを実行します。
3. プログラムが終了したらライブラリを消去します。

この手順で使用するVEEオブジェクトは、Import Library、Call、Delete Libraryの3種類です。

ライブラリのインポート

Callオブジェクト(またはFormulaオブジェクト)を使ってドライバを実行するには、Import Libraryオブジェクトを使ってVEE環境に関数をインポートする必要があります。

Import LibraryオブジェクトのLibrary Typeで、Compiled Functionを選択します。Definition Fileボタンを使って*PREFIX*.hのパスと名前を入力します。ファイルの位置については、47ページの表2-2「WIN95およびWINNTフレームワークのドライバ・ファイルの位置」と、48ページの表2-3「HP-UXフレームワークのドライバ・ファイルの位置」を参照してください。

次に、File Nameボタンを使って*PREFIX*_32.DLL(HP-UXでは*PREFIX*.sl)のパスと名前を選択します。Library Nameボタンは、関数のセットに論理名を割り当てます。この名前は*PREFIX*(ここで*PREFIX*はHP E1410などの機器の名前を指す)とすることを推奨します。

Callオブジェクトからドライバを使用するには、Callオブジェクトを構成する必要があります。このための最も簡単な方法は、Import Libraryオブジェクト・メニューからLoad Libを選択して、ドライバ・ファイルをVEE環境にロードすることです。DeviceメニューからCallオブジェクトを呼び出します。次に、Callオブジェクト・メニューでSelect Functionを選択します。ヘッダ・ファイルに含まれる関数のうち、ドライバ・ファイルからエクスポートされているものすべてのリストを記載したダイアログ・ボックスが表示されます。

VEEからのVXIplug&playドライバの呼出し

VXI*plug&play*ドライバを呼び出すには、Callオブジェクトを使います。

呼出しシーケンス. VXI*plug&play*ドライバの呼出しシーケンスは非常に重要です。下記のシーケンスを使用します。

1. 初期化関数を呼び出します(この関数はセッション・ハンドルを返します)。
2. 初期化関数から返されたハンドルを使って、ドライバへの呼出しを実行します。
3. クローズ関数を呼び出します。

初期化関数. 初期化関数*PREFIX*_initには、3本の入力ピンと2本の出力ピンがあります。

3つの入力パラメータは下記の通りです。

■ 機器アドレス

VXI*plug&play*のアドレッシングについては、79ページ「VXIplug&playドライバの構成」を参照してください。

■ 識別情報確認フラグ

確認フラグが1の場合、初期化関数は機器の識別情報をチェックします。このためには、"*IDN?"クエリまたは機器メーカが指定する他の方法を使って、メーカIDとモデル番号をチェックします。チェックをしない場合はフラグを0にします。

■ リセット・フラグ

リセット・フラグが1の場合、初期化関数は機器を既定義の状態に設定します。フラグが0の場合、リセットは行われません。

2つの出力パラメータは下記の通りです。

■ 戻り値

VXI*plug&play*では、実行した動作のステータスがVXI*plug&play*ドライバからの戻り値として返されると決められています。返された整数を意味のあるメッセージに変換するには、別のCallオブジェクトから*PREFIX*_error_queryを呼び出します。戻り値が0の場合、init()呼出しは成功しています。

■ VXI*plug&play*関数で使用するハンドル

初期化関数の戻り値が0の場合、出力パラメータに機器ハンドルが格納されています。機器ハンドルとは、関数呼出しとこの初期化とを対応づける単なる数値です。大部分のVXI*plug&play*関数には、入力パラメータとしてこのハンドルを渡す必要があります。

初期化を行うたびに、固有のハンドルが出力パラメータviに返されます。パラメータの名前はsession handleなど別の名前でもかまいませんが、必ずinit()関数から返される最後のパラメータになります。close()関数が呼び出されると、ハンドルはシステムに返却されます。

VXIplug&play関数の呼出し. Callオブジェクトを使ってその他の関数を呼び出すこともできます。関数を呼び出すたびに、*PREFIX*_init関数が返したハンドルをCallオブジェクトのinstrID入力ピンに与える必要があります。

その他の一般的なVXIplug&play関数の使用. *PREFIX*_init関数と*PREFIX*_close関数の他に、VXI*plug&play*ドライバには他の一般的なドライバ関数が実装されているのが普通です。一般的な関数としては、*PREFIX*_reset、*PREFIX*_self_test、*PREFIX*_revision_query、*PREFIX*_error_query、*PREFIX*_error_messageなどがあります。

配列パラメータの使用. VXI*plug&play*仕様では、配列または文字列パラメータのためのスペースは呼出し側が割り当てると決められています。すなわち、VEEで配列を割り当ててからVXI*plug&play*関数へのパラメータとして渡す必要があります。図7-8を参照してください。

Close関数の使用. クローズ関数*PREFIX*_closeには1個の入力パラメータがあり、出力パラメータはありません。入力パラメータは*PREFIX*_initから返されたハンドルです。*PREFIX*_closeを実行すると、機器がオフラインになり、機器ハンドルに関連づけられたデータがクリアされます。このほかに、機器のクローズに関連するドライバ固有の動作が実行される場合もあります。クローズしたハンドルは機器関数で使用できなくなります。新しいハンドルを得るにはもう一度*PREFIX*_initルーチンを呼び出す必要があります。

ライブラリの削除

VXI*plug&play*ドライバの使用が終わったら、ロードしたドライバのそれぞれに対してDelete Libraryオブジェクトを呼び出す必要があります。ライブラリをアンロードしたら、そのライブラリを使用する関数を呼び出すにはImport Libraryオブジェクトを使ってもう一度ライブラリをロードする必要があります。

簡単な例

図7-7は、VEEでVXI*plug&play*ドライバを使用した簡単なサンプル・プログラムです。このプログラムは、ライブラリをインポートし、デバイスを初期化し、デバイスをクローズし、ライブラリを削除します(各プログラム・スレッドは`Start`ボタンで独立に起動されます)。

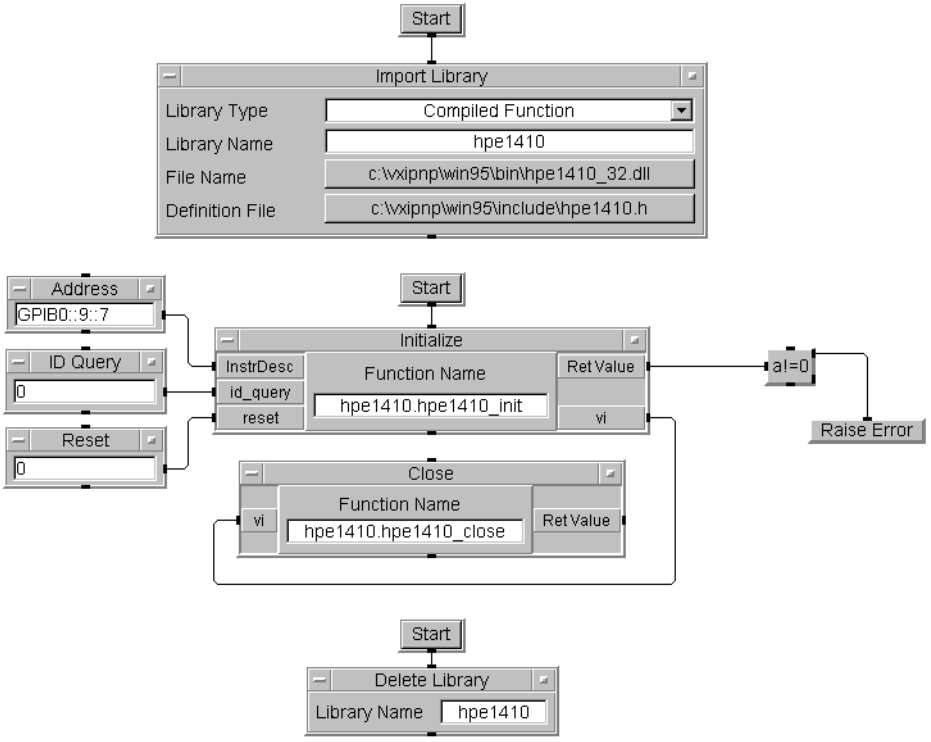


図7-7. 簡単な例: VXI*plug&play*ドライバの使用

より複雑な例

図7-8はVXI*plug&play*ドライバを使ったもっと複雑な例で、配列を割り当てて出力パラメータとして使います。

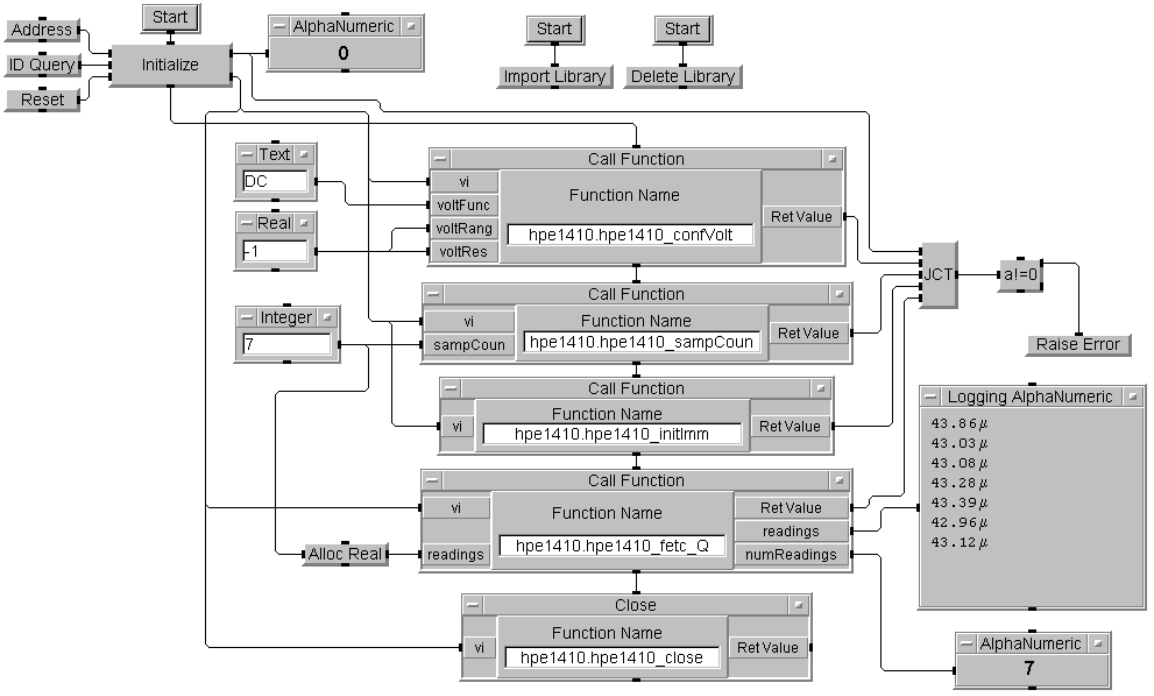


図7-8. より複雑な例: VXI*plug&play*ドライバの使用

有用なヒント

ハンドルの管理. *PREFIX*_initから返されたハンドルは、以後のドライバ関数で使用する必要があります。このためには2つの方法があります。

- ピンを接続

  *PREFIX*_initルーチンのデータ出力ピンを各関数のviデータ入力ピンに接続することにより、ハンドルの値を渡します。

- グローバルにハンドルを管理

  ハンドルをグローバル変数に格納します。*PREFIX*_initルーチンが返したハンドルをSet Globalオブジェクトに接続します。このハンドルを使用する関数はGet Globalオブジェクトでハンドルを取得します。

制御フロー . ドライバは一定のシーケンス(初期化、関数呼出し、クローズ)で動作を実行する必要があります。ハンドルを使用するすべての関数に対してハンドルが有効になるようにVEEプログラムを作成する必要があります。

8

データ伝搬

データ伝搬

VEEプログラムを作成するのに、テキスト・プログラミング言語の技法をそのまま使い、プログラム・コードをビジュアルに再現することもできます。しかし、オブジェクトの間のデータ伝搬に基づき、VEEオブジェクトを使ってプログラムを作成した方が効率的です。この章では、VEEのデータ伝搬について説明します。下記の内容があります。

- 伝搬の理解
- UserObject内部の伝搬
- プログラム・フローの制御
- 伝搬に関するトラブルの扱い

伝搬の理解

伝搬とは、VEEプログラム内の実行の一般的な流れです。VEEオブジェクトの動作順序は、伝搬の指針によって定義されます。一般に、伝搬は**データ・フロー** (VEEプログラム内のオブジェクトからオブジェクトへのデータの流れ)によって決定されます。

オブジェクトの動作方法

VEEオブジェクトが動作する際には、入力ピンからデータを受け取り、そのデータを処理し、結果データを出力ピンに返します。すべてのデータ・ピンがデータによってアクティブにならない限り、VEEオブジェクトは動作しません(1つだけ例外があります。JCTオブジェクトは、どれかのデータ入力ピンがデータによってアクティブになれば動作します)。

図8-1のプログラムで、a+bオブジェクトは両方のデータ入力ピンにデータがない限り動作しません。2個のReal定数オブジェクトが先に動作する必要があります(動作順序は問いません)。

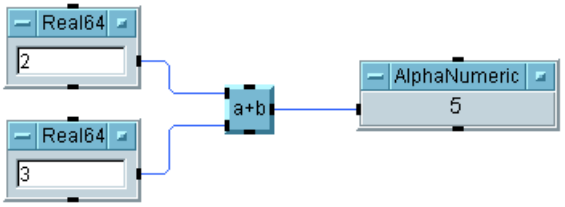


図8-1. a+bオブジェクトは両方の入力にデータが存在すると伝搬する

a+bオブジェクトが動作すると、データが加算され、オブジェクトの出力ピンが結果データによってアクティブになります。AlphaNumericオブジェクトはデータ入力ピンにデータが来るまで動作しないので、このオブジェクトは最後に動作して結果を表示します。

これからわかるように、上記のプログラムのオブジェクトの動作順序はデータ・フローによって決まります。すなわち、データ・フローが伝搬順序を決定します。

シーケンス・ピンを使って、必要なときまでオブジェクトが動作しないように制御することもできます。これは、有効なデータが得られるまでオブジェクトを動作させたくない場合に便利です。図8-2のプログラムでは、前の例にConfirm OKオブジェクトが追加されています。

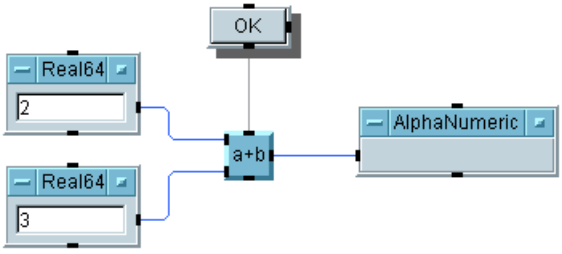


図8-2. シーケンス入力ピンを使った伝搬の制御

Confirm OKオブジェクトのシーケンス出力ピンは、a+bオブジェクトのシーケンス入力ピンに接続されています。シーケンス入力ピンは接続しなくてもかまいません。シーケンス入力ピンを接続しないと、オブジェクトはこれを無視します。一方、シーケンス入力ピンを接続すると、オブジェクトはこのピンがアクティブになるまで動作しません。上の例では、ユーザがOKボタンを押す(クリックする)までa+bオブジェクトは動作しません。OKを押すと、データ入力ピンにデータが入り、オブジェクトが実行されます。

XEQピンは、これと反対の影響をオブジェクトの動作に対して与えます。XEQピンがアクティブになると、そのときデータ入力ピンにあるデータを使ってオブジェクトはただちに伝搬します。XEQピンとシーケンス入力ピンを1つのオブジェクトで同時に使用する場合はこの点に注意が必要です。XEQピンは必ず接続する必要があり、XEQピンがアクティブになるまでオブジェクトは伝搬しません。図8-3に例を示します。

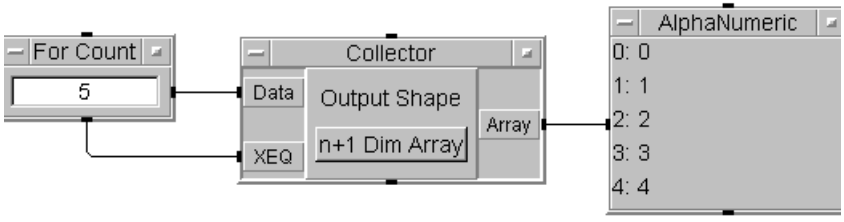


図8-3. XEQピンによる伝搬の制御

For Countオブジェクトは5回繰り返され、CollectorのData入力端子にデータを出力します。Collectorは5個の値を配列に収集し、For Countオブジェクトのシーケンス出力ピンによってXEQ端子がアクティブにされたときにこの配列を伝搬します。

注記

オブジェクト・メニューからPropertiesを選択することで、Show Terminalsをオンにすることができます。Show Terminalsがオンの状態では、データ入出力ピンが「端子」となり、名前が表示されます。

基本的な伝搬順序

VEE 4以降のモードで実行されるVEEプログラムのオブジェクトは、伝搬規則に基づいて、Runを押したときに下記の基本的順序で動作します。

1. データ入力ピンもシーケンス入力ピンも接続されていないオブジェクトが最初に動作します。

2. 他のオブジェクトは、データ・フローによって決まる順序で動作します。言い換えれば、データ入力ピンを持つオブジェクトは、すべてのデータ入力にデータが存在する場合に限って動作します。ただし、259ページ「オブジェクトの動作方法」で説明したように、JCT、XEQ、シーケンス・ピンに関する例外があります。

3. 伝搬の順序は、シーケンス・ピンを接続することによって変更できます。

次のセクション「ピンと伝搬」では、各種ピンの役割について説明します。

ピンと伝搬

このトピックでは、ピンのすべての種類と、伝搬に対するその影響について説明します。オブジェクトのオープン・ビューにはピン・ラベルが表示され、Propertiesダイアログ・ボックスのShow Terminalsがオンの場合は端子情報を見ることができます(図8-4参照)。オブジェクトによっては、ここで説明するピンの一部しか持たないものもあります。

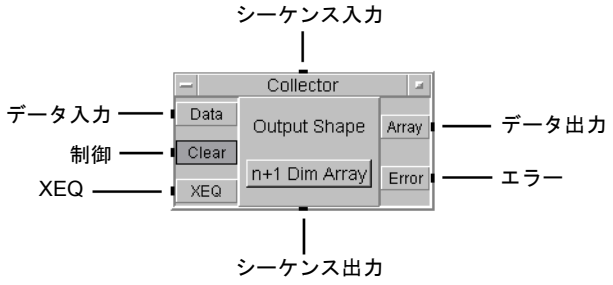


図8-4. オブジェクトで使用できるピン

■ データ・ピンは、データ・コンテナを入力または出力します。

- ❑ オブジェクトは、すべてのデータ入力ピンがアクティブになるまで動作しません(ただしJCTオブジェクトは例外で、どれかのデータ入力ピンがアクティブになれば動作します)。
- ❑ オブジェクトが動作すると、そのデータ出力ピンが伝搬します(エラー条件が発生した場合を除く)。

オブジェクトによっては、データ出力ピンの一部しか伝搬しないものがあります。この動作は混乱を招くことがあります。このようなオブジェクトとしては、If/Then/Else、DeMultiplexer、Comparator、およびすべてのData ⇒ Dialog Boxオブジェクトがあります。詳細については、284ページ「伝搬に関するトラブルの扱い」を参照してください。

■ 制御ピン(任意指定)は、オブジェクトのステートに影響を与える入力で、伝搬には影響しません。一般的な制御ピンとしては、Clear、Reset、Default Valueがあります。他のオブジェクトから制御ピンへの出力は、伝搬に影響しないことを示すため破線で接続されます。
制御ピンが伝搬に影響しないために、プログラムが正しく動作しなくなる場合があります。制御ピンの詳細については、284ページ「伝搬に関するトラブルの扱い」を参照してください。

■ シーケンス・ピンは、実行の順序を指定するためだけに用いられます。プログラムの伝搬におけるあいまいさを解消するために使います。シーケンス・ピンは必

要でない場合が多く、濫用されるおそれがあるので、なるべくシーケンス・ピンを使わずにデータ・フローを明確にするよう心がける必要があります。

- ❑ オブジェクトは、すべてのデータ入力ピンとすべてのシーケンス入力ピン(接続されている場合)がアクティブになるまで動作しません。

  シーケンス入力ピンはデータ・コンテナが存在するときにアクティブになりますが、コンテナ内のデータは無視されます。

- ❑ シーケンス出力ピンが伝搬するのは、すべてのデータ出力ピンがアクティブになり、データ・フローが可能な限り遠くまで伝搬したあとです。

  シーケンス出力ピンがアクティブになると、空の(nil)コンテナが伝搬されます。

■ エラー・ピン(任意指定)。エラー・ピンを追加することにより、オブジェクトが生成するエラー条件をトラップできます。エラー条件が発生すると、対応するエラー番号がエラー・ピンから伝搬されます。

エラーが発生した場合、エラー・ピンとシーケンス出力ピン(接続されている場合)が伝搬します。**データ出力ピンはエラー発生と同時に伝搬を停止します。**一部のデータ出力ピンはエラー条件の発生前に伝搬するため、混乱の元になる可能性があります。

■ XEQピンは、(データ・ピンがアクティブになっていなくても)オブジェクトがただちに動作するようにします。XEQピンはCollectorオブジェクトとSample & Holdオブジェクトだけで用いられ、ただちにオブジェクトを実行してデータを伝搬させる役割を果たします。

XEQピンはデータ・コンテナが存在するときにアクティブになりますが、コンテナ内のデータは無視されます。

注記

データ入力ピンやXEQ ピンを接続しないでおくと、プログラムを実行したときにエラーが発生します。

データ出力ピン、制御ピン、エラー・ピンは接続しなくてもかまいません。シーケンス・ピンは、プログラムの伝搬におけるあいまいさを解消するのに必要な場合を除いては接続しないでおきます。

詳細については、284ページ「伝搬に関するトラブルの扱い」を参照してください。

スレッドとサブスレッドの伝搬

非常に単純なVEEプログラムは、通常1つのスレッドからなります。より複雑なプログラムでは、複数のスレッドやサブスレッドが存在し、プログラムの伝搬に影響を与えます。

- スレッド - データ・ラインおよびシーケンス・ライン(実線)で接続されたオブジェクト群はスレッドを構成します。制御ライン(破線)だけで接続されたオブジェクト群はスレッドと見なされません。プログラムには複数のスレッドが存在できます。例えば、図8-5のプログラムには2つの並列スレッドが存在します。スレッド同士はデータ・ラインやシーケンス・ラインで接続されていないため、互いに独立しています。

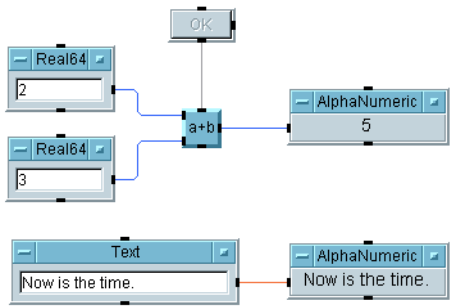


図8-5. 2つの並列スレッドを持つプログラム

- サブスレッド - スレッドの枝の1つをサブスレッドと呼びます。2つのサブスレッドが同じオブジェクトの同じデータ出力ピンから始まり、その間にシーケンス・ラインもデータ・ラインもない場合、これらは並列サブスレッドです。図8-6のプログラムでは、`Real64`定数オブジェクトのデータ出力ピンから2つの並列サブスレッドが分岐しています。

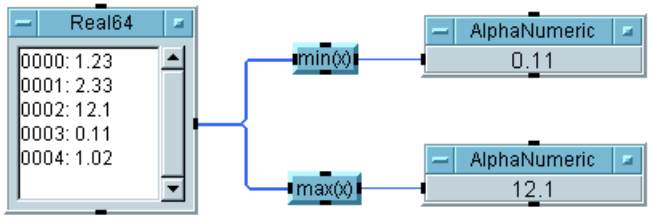


図8-6. 2つの並列サブスレッドを持つプログラム

並列スレッドおよびサブスレッドは、互いにランダムな順序で動作します。1つのスレッドの1つまたは複数のオブジェクト(または全部のオブジェクト)が動作したあとで、別のスレッドの1つまたは複数のオブジェクトが動作します。ただし、これには下記の2つの例外があります。

- スレッドにInterface EventまたはInstrument Eventオブジェクトが存在する場合、イベントがトラップされるとこのオブジェクトが実行を独占します。例えば、Interface EventがGPIBのSRQメッセージを検出した場合、このスレッドが終了するまで他のスレッドは実行されません。イベントの処理を優先するために他のスレッドは一時停止されます。詳細については、**VEEオンライン・ヘルプ**のInterface EventおよびInstrument Eventを参照してください。
- スレッドにStartオブジェクトが存在し、ユーザがStartボタンを押すことによりスレッドを始動した場合、そのスレッドが終了するまで他のスレッドを始動することはできません。VEE 4以降のモードでは、Startオブジェクトは推奨されません。

伝搬のまとめ

以下に示すのは、VEE 4以降の実行モードでプログラムが実行されたときに用いられる伝搬規則のまとめです。

- データは左から右の順序でオブジェクトを通過します。シーケンスは上から下の順序で流れます。
- データ・ピンとXEQ入力ピンはすべて接続されている必要があります。
- データ入力ピンもシーケンス入力ピンも持たないオブジェクトは最初に動作します。

- オブジェクトが動作するには、すべてのデータ入力ピンがアクティブになる必要があります(JCTオブジェクトを除く)。
- シーケンス入力ピンが接続されている場合、このピンがアクティブになるまでオブジェクトは動作しません。
- オブジェクトは、繰返しオブジェクト(For Countなど)に接続されているか、XEQピンで強制的に実行される場合を除いて、1回しか動作しません。
- 制御ピンはただちに実行され、オブジェクトの動作や伝搬を引き起こしません。285ページ「制御ピン・エラーの捕捉」を参照してください。
- オブジェクトからErrorピンでエラーが生成された場合、データ出力ピンは伝搬せず、Errorピンが伝搬します。ただし、シーケンス出力ピンはアクティブになります(Errorピンがない場合、エラー・メッセージが表示されます)。
- 並列サブスレッドは任意の順序で動作します。
- 複数のスレッドは任意の順序で動作します。

UserObject内部の伝搬

UserObjectは、特定の作業を実行するオブジェクト群を1個のカスタム・オブジェクトにカプセル化するための手段です。カプセル化により下記のことが可能になります。

- VEEプログラムの作成にモジュール型のデザイン技法が使用できます。これにより、複雑な問題を組織的な手法で解決できます。UserObjectを使うことで、トップダウン・デザイン技法による、柔軟で保守の容易なプログラムを作成できます。
- ユーザ定義オブジェクトをライブラリに保存し、あとで再利用できます。UserObjectを作成して保存したあと、他のプログラムにマージして使用できます。

UserObjectの機能

メイン・ウィンドウにUserObjectを追加すると、アイコン・ビューで表示され、その状態でプログラムの一部となります。アイコンをダブルクリックすると、UserObjectの編集ウィンドウがポップアップし、ワーク・エリアが表示されます。このワーク・エリアにオブジェクトを追加して接続することにより、固有のプログラム・セグメントを作成できます。端子エリアには、UserObjectがプログラムの他の部分と通信するためのデータ端子と制御端子が置かれます。図8-7に示すのは、UserObject1という名前のUserObjectのアイコン・ビューと編集ウィンドウです。

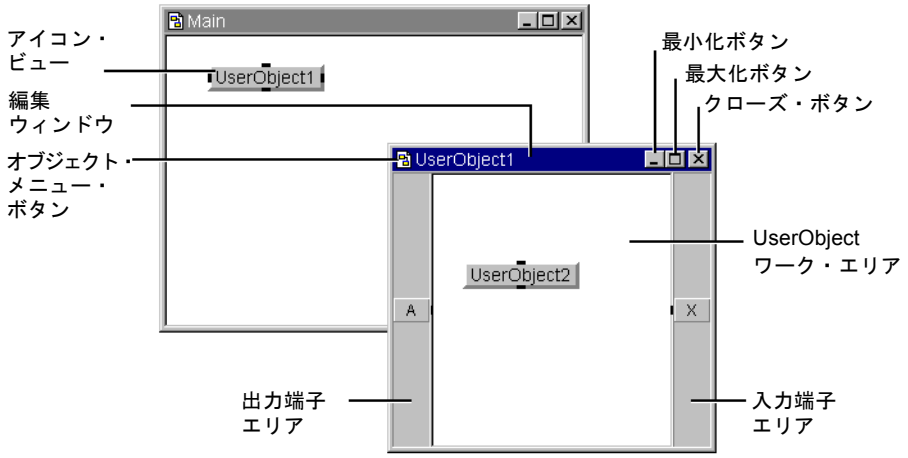


図8-7. UserObjectの各部分

コンテキストとUserObject

メイン・ウィンドウとUserObjectは、VEEプログラム内で異なる**コンテキスト**に属します。これは、CやBASICのサブプログラムが別のコンテキストに属するのと同じです。図8-7に示すように、VEEプログラム内にUserObjectをネストすると、新しいコンテキストが生成されます。図8-7では、3つのコンテキストがあります。各コンテキストにはさらに別のオブジェクトを追加できます。

1. メイン・ウィンドウは1つのコンテキストであり、`UserObject1`はこれに含まれます。

2. `UserObject1`は1つのコンテキストであり、`UserObject2`はこれに含まれます。

3. `UserObject2`は1つのコンテキストであり、他のオブジェクトを含むことができます。

伝搬とUserObject

UserObjectを含むプログラムでの伝搬は、UserObjectが独立のコンテキストであるという事実に影響されます。UserObjectの伝搬規則は下記の通りです。

注記

UserObjectの伝搬規則はUserFunctionにも適用されます。UserFunctionの詳細については、第12章「ユーザ定義関数/ライブラリ」を参照してください。

- UserObject 内部のオブジェクトが動作するには、UserObject のすべてのデータ入力端子が(シーケンス入力端子が接続されていればそれも)アクティブになる必要があります。

- UserObject のデータ入力端子が (シーケンス入力端子が接続されていればそれも)アクティブになると、UserObjectが動作します。UserObject内部のオブジェクトは、伝搬規則に従って動作します。

- VEEバージョン4.0より前に作成されたプログラムのUserObjectには、任意指定のXEQ端子が存在する場合があります。この端子がアクティブになると、UserObject はただちに内部のオブジェクトの動作を開始します。その際、UserObjectのアクティブでない入力端子に関しては、その時点での「古い」値が用いられます。

  ほとんどの場合、UserObjectでXEQ端子を使用する必要はありません。VEE 4.0 以降のバージョンでは、UserObjectでXEQ端子は使用できません。UserObject にXEQ端子が存在する既存のプログラムは、VEE 4以上の実行モードではコンパイルできません。

- UserObject のデータ出力端子は、UserObject 内部のすべてのオブジェクトが実行を終了したあとで伝搬します(エラーまたはExit UserObjectによってUserObjectが途中で終了した場合を除く)。UserObject内部からアクティブにされた出力端子からだけ、UserObject外部のオブジェクトにデータが渡されます。アクティブにされたデータ出力端子は、それぞれ1個のデータ・コンテナだけを伝搬します。

- VEEバージョン4.0より前に作成されたプログラム(VEE 3実行モードで動作するもの)では、UserObject内部のオブジェクトは異なるサブスレッドに存在する外部オブジェクトと時分割実行されます。これをタイムスライス実行といいます。UserObject外部のオブジェクトの動作はUserObjectによってブロックされません。VEE 4以上の実行モードで動作するプログラムでは、別々のスレッドから起動された場合に限ってUserObjectはタイムスライス実行されます。

基本的な伝搬規則については、265ページ「伝搬のまとめ」を参照してください。

注記

UserObject内部にStartオブジェクトが存在する場合、Startを押すとStartオブジェクトと同じスレッドに接続されたオブジェクトだけが実行されます。UserObjectの入力端子からデータは読み取られず、UserObjectの出力端子は伝搬しません。したがって、UserObject外部への伝搬は生じません。

UserObjectからのデータ出力

UserObject内部のオブジェクトが伝搬を終了すると、UserObjectの各データ出力端子がそれぞれ1個の**データ・コンテナ**(端子が最後に受け取ったもの)をUserObject外部のコンテキストに伝搬します。これを考慮しないと、プログラムで予期しない結果が生じるおそれがあります。図8-8の例はこの状況を示します。

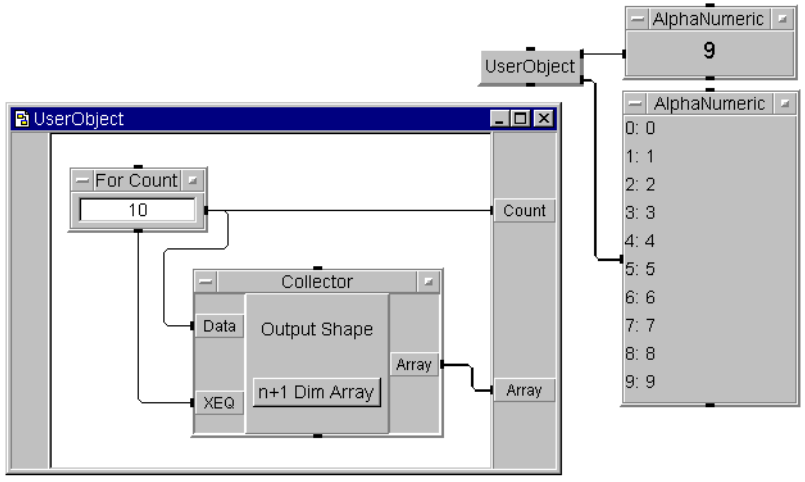


図8-8. UserObjectからのデータ伝搬

For Countオブジェクトは10個のデータ・コンテナ(番号0～9)をCount出力端子に送りますが、UserObjectから伝搬するのは1個のデータ・コンテナ(最後の番号)だけです。一方、Collectorオブジェクトを使えばFor Countオブジェクトからのデータ

を配列に収集することができます。Array出力端子も1個のデータ・コンテナだけを伝搬しますが、このコンテナは10個の値(0～9)を持つ1次元配列です。

プログラム・フローの制御

VEEの伝搬規則は論理的ですが、プログラムがどのように伝搬するかがわかりにくい場合もあります。このセクションの例は、実際のプログラムを作る際に、伝搬の概念を理解し、応用するために役立つものです。最初に、VEEのプログラミング・スタイルに関する規則をいくつか挙げます。

- 階層的に伝搬する明確なプログラム・フローを使ってプログラムを作成します。フローを簡単に視覚化できれば、問題は通常生じません。

- オブジェクト間の実行順序が重要だがあいまいな場合、シーケンス入力ピンとシーケンス出力ピンを接続します。これらが必要な場合はそれほど多くありませんが、プログラムで要求される実行順序を保証するために必要になることもあります。

- フィードバック・ループを使った繰返しは避けます。このような構文は予期しない結果につながります。データが入ったコンテナをスレッドの先頭に返すためには、ループが用意されています。フィードバックが必要な場合、フィードバック・ループ内にJCT(ジャンクション)オブジェクトが必要です。

- ループするオブジェクトから生じる並列スレッドは避けます。どのスレッドが実行されるかがわかりにくくなるからです。

- GateオブジェクトとSample & Holdオブジェクトの使用は避けます。これらのオブジェクトは、伝搬規則に関する知識不足を補うために用いられる場合が多いのです。よいプログラミング・スタイルを使えば、これらのオブジェクトは使わずに済みます。

基本的なプログラム制御

オブジェクトの基本的な組合わせによってプログラム・フローがどのように制御されるかを理解することは重要です。図8-9のプログラムは、一般的なプログラム制御であるループに使用する単純なカウントを生成する方法を示します。プログラムが動作すると、For Countオブジェクトは0から9までカウントします。

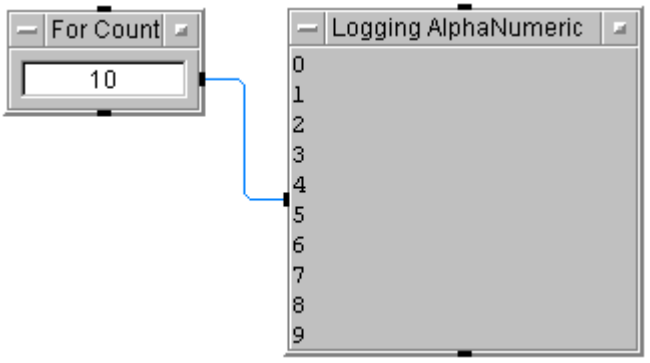


図8-9. 単純なループ・カウンタ

For Countオブジェクトをネストすることで、ネストしたループを作成できます。図8-10のプログラムで、内側のループのFor Countは、外側のループのFor Countからシーケンス入力ピンにカウントが送られるたびに、0から9までカウントします。外側のFor Countは、内側のFor Countがループをすべて終了してから次の出力カウントを送ります。

外側のFor Countが最後のカウントを送ると、そのシーケンス出力ピンからパルスが出力され、Beepオブジェクトをアクティブにします。これはこの種のループ・オブジェクトの重要な特徴です。シーケンス出力パルスが生成されるのは、ループ・オブジェクトが駆動しているスレッドの実行が終了してからです。

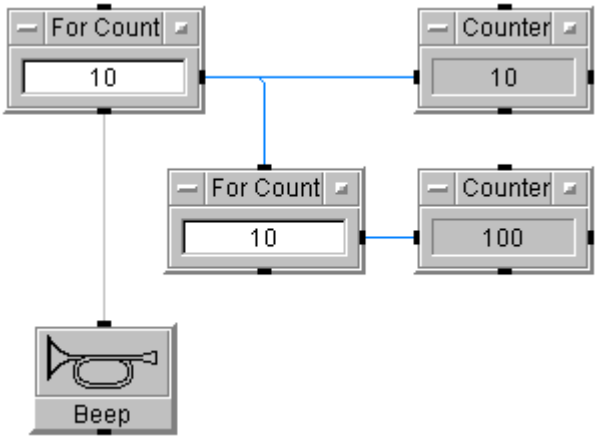


図8-10. 単純なネストしたループ・カウンタ

連続ループ

連続ループを生成するには、図8-11のプログラムに示すUntil Breakオブジェクトを使います。Delayオブジェクトにより、プログラムは1秒に1回アップデートするように制御されます。もっとよい方法は、Until Breakの代わりにOn Cycleを使うことです。このオブジェクトは、任意の遅延設定でコンテナを生成してnow()オブジェクトを駆動します。AlphaNumericの表示フォーマットは、このオブジェクトのPropertiesダイアログ・ボックスのNumberタブで設定できます。

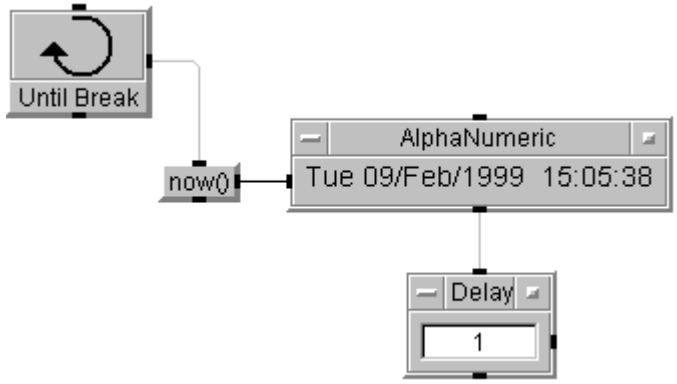


図8-11. 単純な連続ループ

連続ループは、特定の条件が満たされるまでプログラムの動作を繰り返すときに有効です。任意の時点でこのループを終了できるようにするには、図8-12のプログラムのようにOK、Break、Nextの各オブジェクトを追加します。

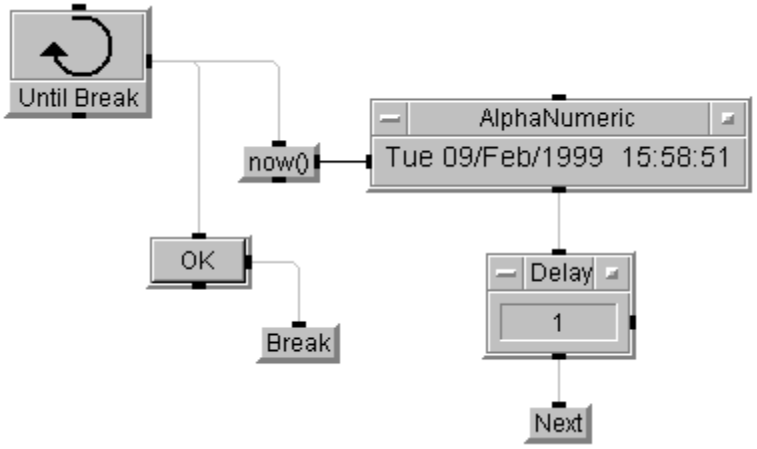


図8-12. 連続ループを停止する方法

この例では、制御構文のもう1つの機能を2つの並列スレッドで示しています。このプログラムは、ユーザがOKを押してBreakを実行させるまで、時刻を連続的に更新するように作られています。もしNextオブジェクトがないと、Until Breakはコンテナを生成し、下流のすべてのオブジェクトが実行されるまで待ちます。時刻が更新されたあと、OKが押されるまでプログラムは待ち状態になります。Nextオブジェクトがあるために、Until BreakはOKが押されるまで連続的にコンテナを出力します。

この例では、Breakオブジェクトの代わりにStopオブジェクトを使っても同じように動作します。

連続ループをもっと直接的に制御するには、Toggleオブジェクトを使います。図8-13のプログラムは、Toggle(Buttonフォーマット)を使ってループを中断する方法を示します。

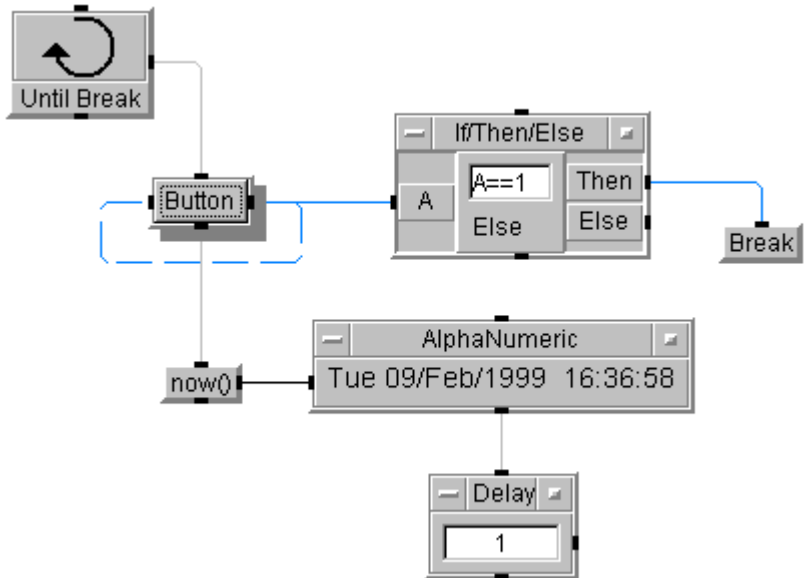


図8-13. If/Then/Elseを使って連続ループを停止

Toggleの出力は、自分自身のReset制御入力に接続されています。デフォルトの初期値は0で、ループのたびにIf/Then/Elseにこれが出力されます。Toggleの値が0の間は、プログラム・フローはNowオブジェクトに到達します。ToggleのButtonをクリックすると、値が1に切り替わり、If/Then/Elseの式の条件が満足されて、Breakオブジェクトがアクティブになります。

対話的なプログラムの作成

図8-14のプログラムは、すでに説明したループの技法を使って、対話的なプログラムの一般的なアーキテクチャを作成する方法を示します。ユーザが適切なToggleボタンをクリックすることにより、2つの動作のどちらかを選ぶか、プログラムを終了できる単純なプログラムを考えます。

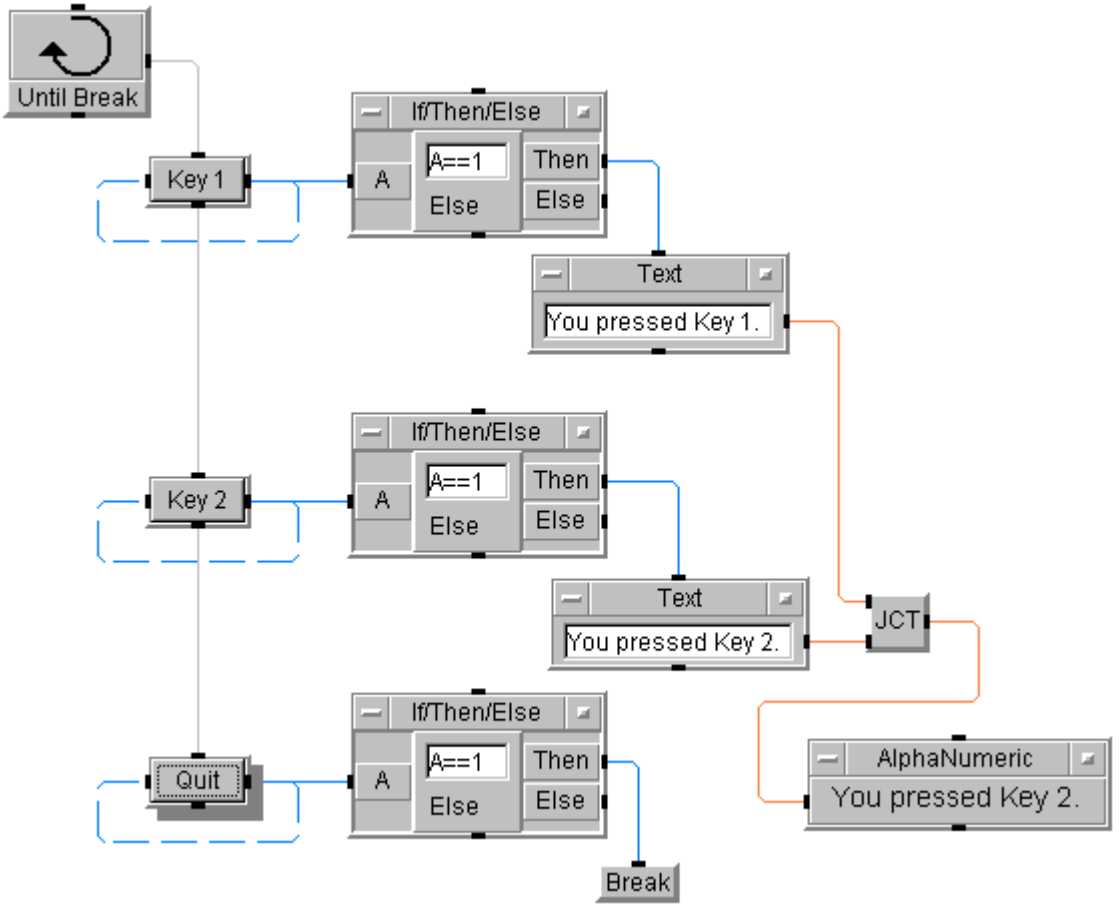


図8-14. Until Breakループを使ってプログラムのサブスレッドを選択

考え方としては、それぞれの異なる動作に1つの並列スレッドを割り当てます。ループは各`Toggle`オブジェクトの出力(初期値は0)をそれぞれの`If/Then/Else`式で連続的にチェックします。ボタンが押されると、`Toggle`の出力が1に変わり、対応する`Text`の出力が`AlphaNumeric`表示に送られるか、プログラムが終了します。I/Oや計算を実行するために必要なだけの並列スレッドを追加できます。

このアーキテクチャの特徴として、実行中のスレッドが終了しないと他のスレッドは実行されません。実行中のスレッドの終了までに長い時間がかかると、このスレッドが終了するまで他のスレッドの実行が待たされることになります。

高度なプログラム制御

これまでの技法を拡張することにより、より複雑な制御作業を実行できるようになります。

例: プログラム作業の初期化

例として、図8-15のプログラムでは、ユーザが複数の作業から1つを選択するか、機器からサービス・リクエスト(SRQ)が発生することで作業が開始されるようになっています。

プログラムを起動したあと、Task 1またはTask 2ボタンを押すことにより対応する表示出力を得ることができます。パネル・ドライバのStatus Panel でClear Statusを押すと、SRQ!メッセージが出力され、他の作業が実行できる状態に戻ります。Quitボタンをクリックするとプログラムが停止します。

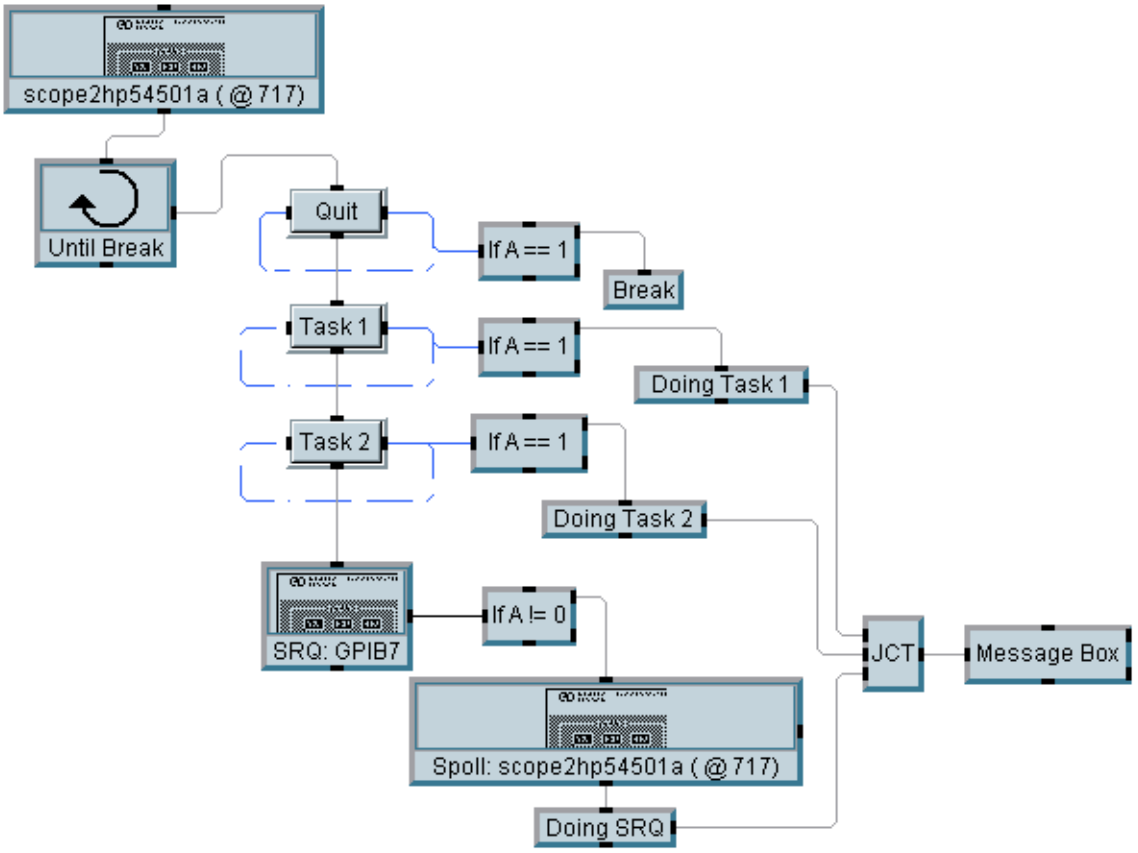


図8-15. Until Breakループを使って機器のサービス・リクエストを検出

Until Breakオブジェクトは、プログラム内の4つの並列スレッドを駆動します。これらは3個のToggleボタン(Task 1、Task 2、Quit)とInterface Eventオブジェクト(SRQ:GPIB7)によって制御されます。

Task 1ボタンとTask 2ボタンで定義される2つのスレッドは、Doing Task 1およびDoing Task 2というテキストをMessageダイアログ・ボックスに表示します。Quitボタンで定義されるスレッドは、プログラムを停止して表示をクリアします。

ここで興味深いのは、SRQに関するスレッドです。Until Breakオブジェクトに接続されたHP 54501Aパネル・ドライバは、SPoll Enableが(ステータス・パネルで)

セットされているため、サービス・リクエストを生成することができます。ステータス・パネルでClear Statusボタンが押されると、サービス・リクエストが送られます。SRQを処理するスレッドは、図8-16の設定を使って、Interface Eventオブジェクトでマスクを設定してSRQを待ちます。

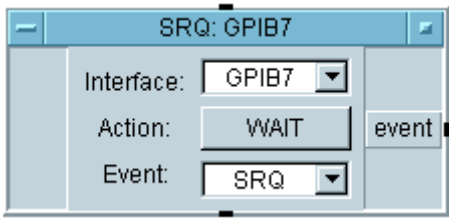


図8-16. SRQ設定

SRQが発生すると、Interface EventオブジェクトはInstrument Eventオブジェクトにシリアル・ポールを実行させます。これによりオシロスコープのSRQがクリアされます(図8-17参照)。

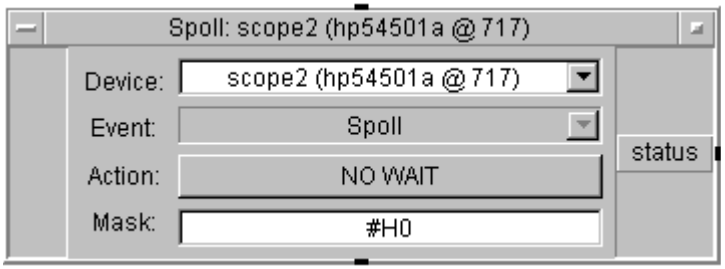


図8-17. SRQのクリア

NO WAITが設定されているので、オブジェクトはシリアル・ポールを実行したあとTextオブジェクトを実行します。SRQ!メッセージがMessageダイアログ・ボックスに送られ、SRQが発生したことを示します。Messageダイアログ・ボックスはオペレータの応答を待つように設定されています。マスク値は無意味です。

関数の呼出し

図8-18のプログラムも同様のものです。ユーザは3つのユーザ定義関数A、B、Cのうち1つを呼び出して動作を起動し、選択した関数の実行をプログラムに継続させます。

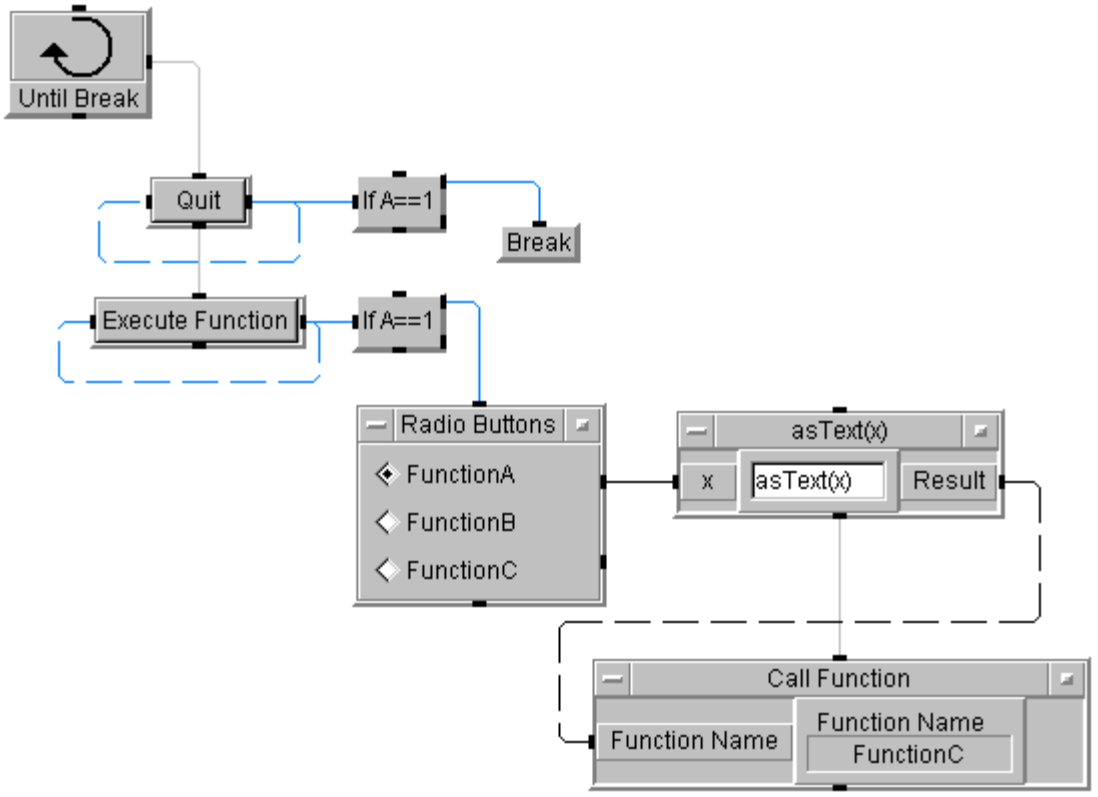


図8-18. Until Breakループを使ってUserFunctionを呼び出す

Radio Buttonsオブジェクトには、使用可能な関数のリストが表示されます。Toggleボタンは、選択された関数を実行するか、プログラムを終了するかを選ぶためのものです。関数の実行を選択した場合、関数名がasText Formulaオブジェクトに出力されます。asTextは入力をテキスト・データ型に変換する組込み関数です。

テキストの関数名がCall Functionオブジェクトの制御入力Function Nameに出力され、選択された関数が呼び出されます。プログラムが実行されると、オペレータはRadio Buttonsオブジェクトで関数名を選択し、Execute Functionボタンを押します。

注記

制御入力はオブジェクトの伝搬には影響しません。asTextのシーケンス出力がCall Functionのシーケンス入力に接続されているのは、Function Nameを受け取るまでCall Functionが伝搬しないようにするためです。

ストリップ・チャートのクリア

関連のあるプログラム・フローの問題として、図8-19のプログラムは、ストリップ・チャートを生成し、特定の数のポイントがカウントされるか、ユーザがボタンをクリックしたときにストリップ・チャートをクリアします。

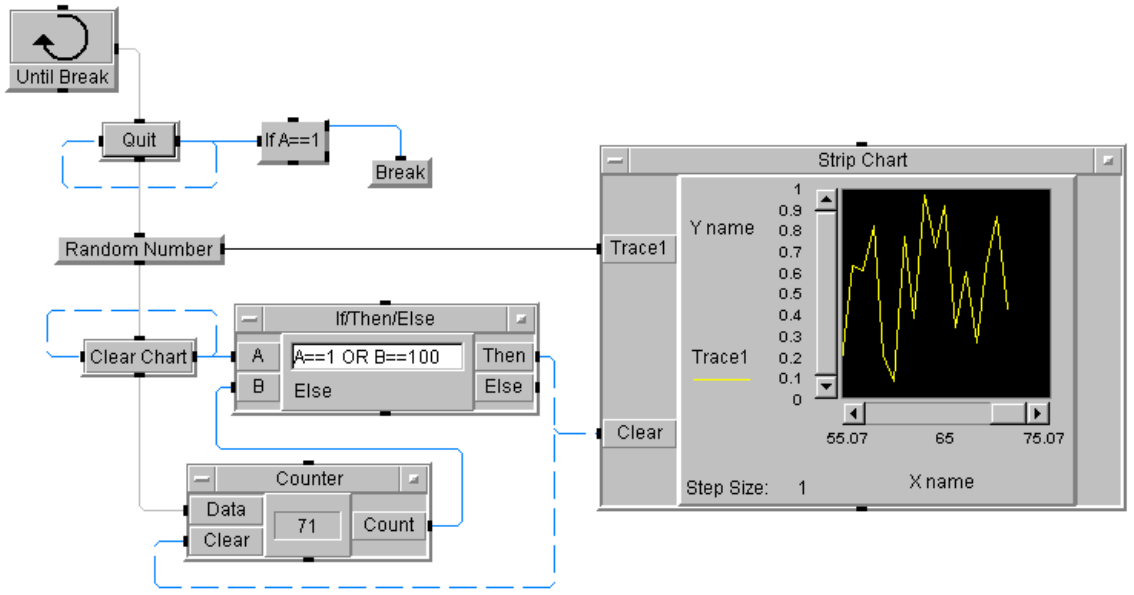


図8-19. Until Breakループを使ってストリップ・チャートのデータ収集を制御

Until Breakオブジェクトがプログラムを駆動します。Quit Toggle、Random Number、Clear Chart Toggle、Counterの各オブジェクトが、シーケンス入出力ピンを通じてUntil Breakから制御されます。Random Number(random(high, low)組込み関数)オブジェクトは、Strip Chart表示に値を送ります。Counterはループの繰返しをカウントして、If/Then/Elseにカウントを出力します。If/Then/Elseは、ユーザがClear Chartボタンを押すか、Countが100になったときにStrip ChartとCounterをクリアします。

Toggle Controlオブジェクトは Until Breakによって連続的に駆動され、ほとんどの場合0を生成します。Toggle Controlをクリックすると値が1に変わり、フィードバック接続によってリセットされます。この1はIf/Then/Elseに入力されます。Toggle Controlオブジェクトのデフォルトの外観を変更するには、Properties (オブジェクト・メニュー)を使ってタイトル・バーを非表示にし、Reset端子を追加します。

伝搬に関するトラブルの扱い

場合によっては、プログラムの結果が予期したものにならないことがあります。原因としては、制御ピンの使用法、ループ内部のオブジェクトの伝搬方法、並列スレッドの伝搬方法などが考えられます。問題を発見するために以下の指針が役立ちます。

エラー処理

エラー処理はVEEの重要な概念の1つです。エラー処理を使えば、何らかの動作を行ってエラーが発生したときに、動作を繰り返すか先に進むかを選択できるようになります。図8-20のプログラムがその例で、異なる結果を生じる動作をダイアログ・ボックスによって表しています。

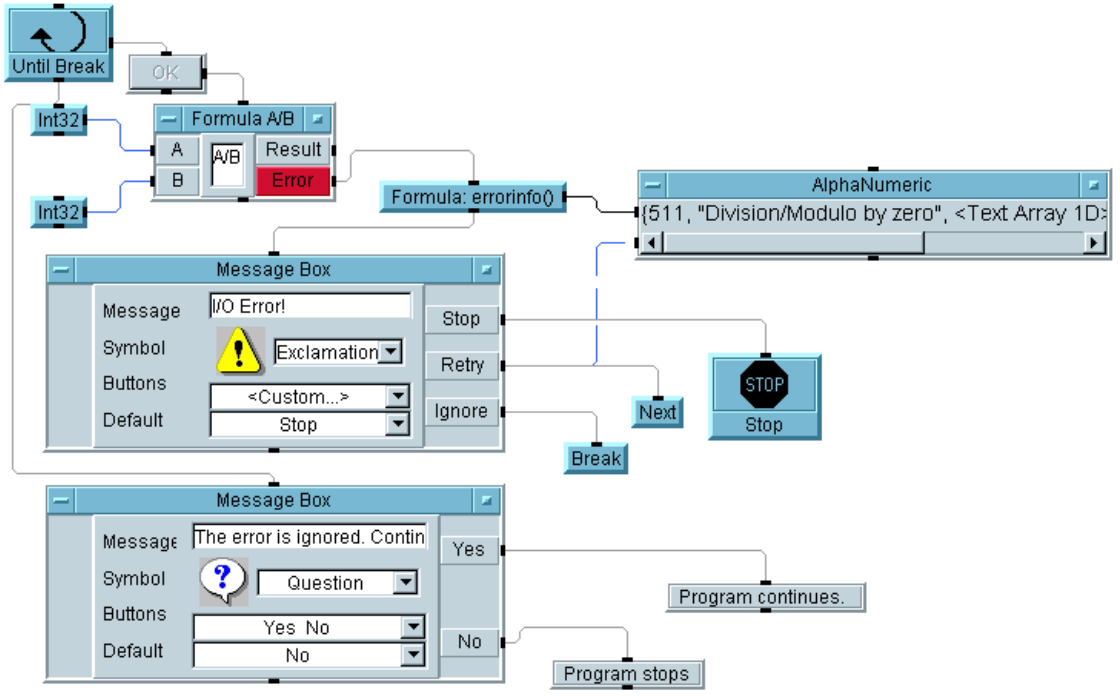


図8-20. Until Breakループを使ってエラー条件を処理

このプログラムでは、Exclamation Message Boxをポップアップして、停止、再試行、無視のいずれかをユーザが選択できるようにします。

- Stopを選択すると、Stopオブジェクトが実行され、プログラムが停止します。
- Retry を選択すると、Next が実行され、ループが繰り返されて、同じメッセージ・ボックスが再表示されます。
- Ignoreを選択すると、Breakオブジェクトが実行されて、Until Breakループが停止します。

Until Breakが停止すると、Question Message Boxが実行され、別のプログラム制御オプションが提示されます。

"I/O"ループのあとで順次実行されるエレメントが、Until Breakのシーケンス出力ピンに接続されていることに注意してください。ループ・エレメントには接続されていません。

エラー処理を標準的な習慣にするのは避けてください。特に、複雑な数式がトランザクションに含まれるトランザクション・オブジェクトの場合は注意が必要です。これらの式の実行のためにVEEはメモリを割り当てるので、実行中にエラーが発生するとメモリが解放されず、そのたびにメモリ・リークが増えていきます。

制御ピン・エラーの捕捉

制御ピンはただちに実行され、オブジェクトを伝搬させないので、場合によってはプログラムが正しく動作しない原因となることがあります。
制御ピンのためにエラーが生じる場合、プログラムからエラーを捕捉するには特別なプログラミング技法が必要です。

制御ピンが原因でオブジェクトがエラーを生成すると、プログラムは停止し、エラー・ダイアログが表示されます。プログラムでエラーを捕捉して解決する代表的な方法は、オブジェクトにError出力を設けることです。この方法はほとんどの場合に有効ですが、制御ピンからエラーが生じる場合だけはうまくいきません。制御ピンはオブジェクトの伝搬に影響しないので、オブジェクトはエラー情報を伝搬しません。すなわち、制御ピンはオブジェクトを実行させないため、オブジェクトは伝搬できないのです。オブジェクトが動作しない限り、Errorピンを含めてすべての出力ピンは伝搬しません。制御ピンから生じたエラーを捕捉するには、オブジェクトを含むコンテキストにError出力を追加する必要があります。

図8-21のプログラムは、制御ピンから生じるエラーを捕捉する間違った方法を示します。このプログラムは、XY Trace表示に波形を表示します。XY TraceにはScales制御入力があり、表示のスケールを変更するためにRecordデータ型を必要とします。Scales制御入力に接続されたBuild Recordに、TextとIntegerから値が供給されます。XY TraceのError出力は、エラー条件を捕捉してerrorInfo()関数に送ることを意図しています。

このプログラムは、エラー番号とメッセージを表示することによりエラーを処理します。プログラムはXY Traceから生成されたエラーを捕捉しますが、Scales制御入力でエラーが生じた場合はそれができません。

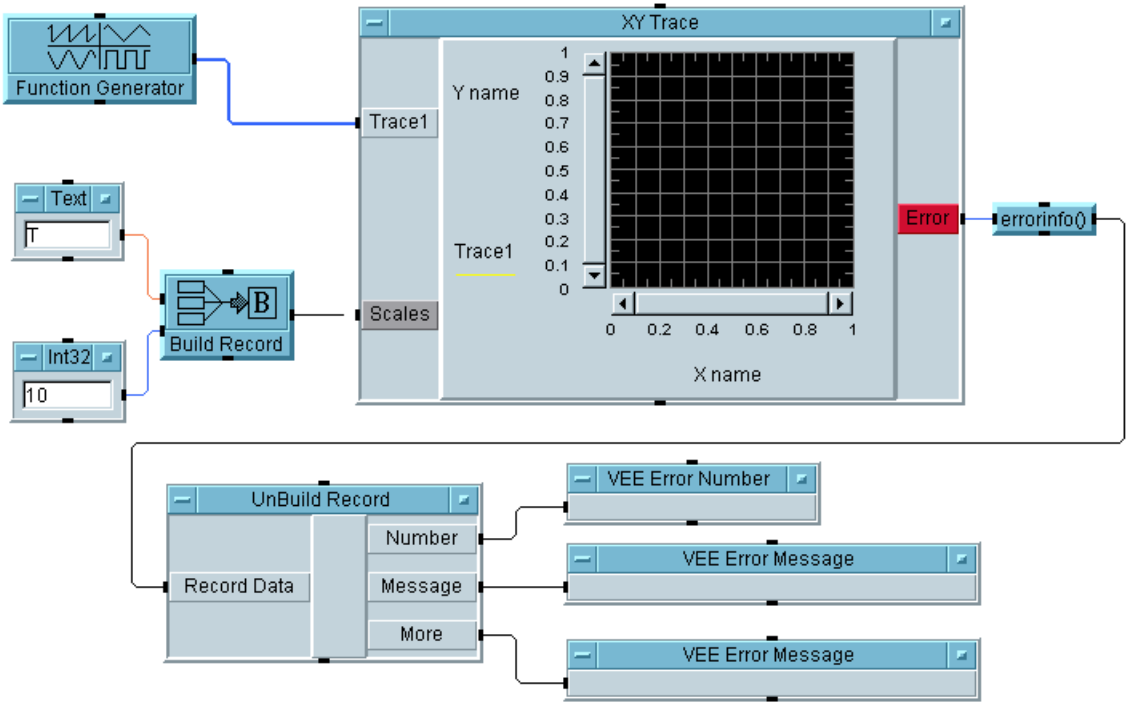


図8-21. 制御ピンのエラーを捕捉する間違った方法

プログラムを実行すると、制御ピンから生じるエラーをプログラムで捕捉しようとする場合の問題が明らかになります。Scales制御ピンは、受け取るRecordに少なくとも変更するスケールを識別する値が含まれていると想定します。使用できる値は、X、Y、Y1、Y2、Y3です。

Textから正しくない値Tが渡されるため、制御ピンはエラーを発生します。これは制御ピンのエラーなので、XY Traceからの伝搬は起こらず、Error出力はエラー情報を受け取りません。プログラムは突然停止し、図8-22のエラー・ダイアログが表示されます。

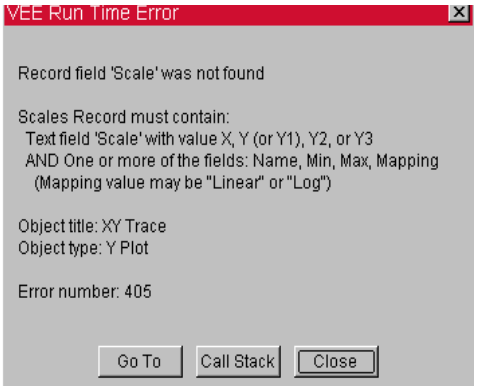


図8-22. エラー・ダイアログ・ボックス

すでに説明したように、制御ピンのエラーを捕捉する正しい方法は、オブジェクトを含むコンテキストにError出力ピンを追加することです。図8-23のプログラムでは、Build RecordとXY TraceをUserObjectに入れることでこれを実現しています。XY Trace表示のError出力は削除され、UserObjectにError出力が追加されていることに注目してください。UserObjectのError出力はerrorInfo()に接続されています。

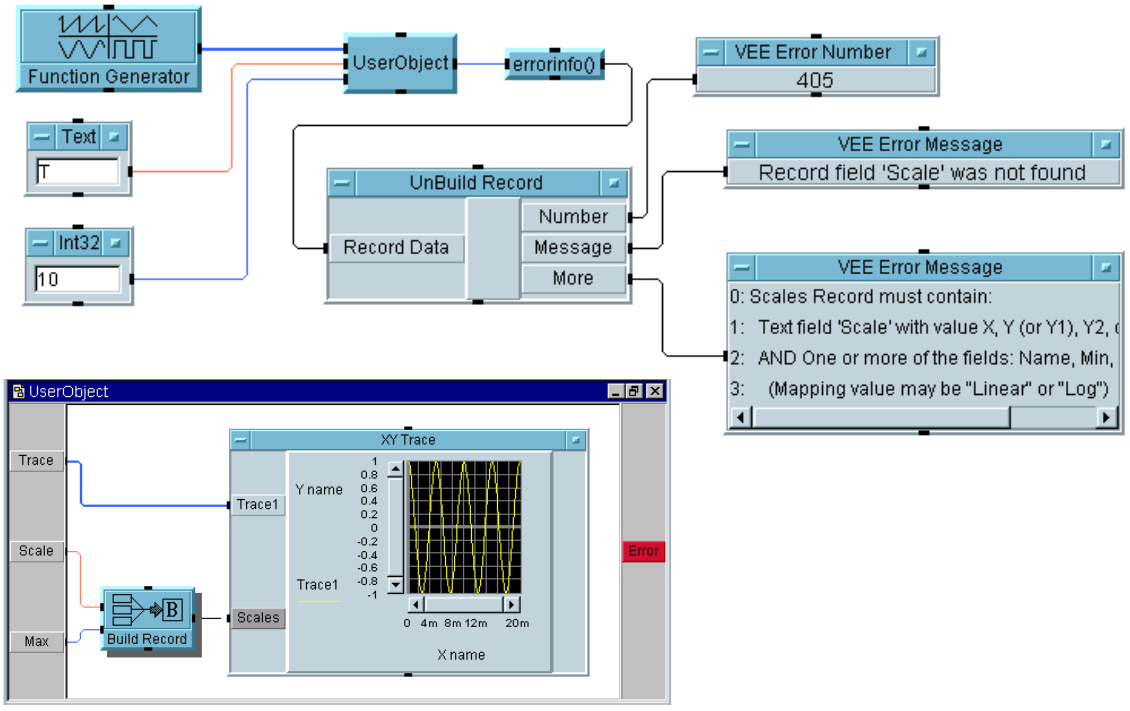


図8-23. 制御ピンのエラーを捕捉する正しい方法

制御ピンへのデータ伝搬

オブジェクトの制御ピンがデータ(ファイル名やデフォルト値など)を受け取る場合、オブジェクトのシーケンス入力ピンを接続しないと、プログラムが正常に動作しない場合があります。制御ピンはオブジェクトの伝搬に影響しないため、データ入力がデータを受け取ると、制御ピンの値が設定されていなくてもオブジェクトは伝播するからです。

図8-24のプログラムは、このような順序の問題を示しています。`To File`の`File Name`制御ピンにファイル名`dataFile2`が送られる前に、`Alloc Array`が`To File`にデータを送ります。データを受け取った`To File`は、ただちに`dataFile1`の内容を更新します。ところが、実際に更新したいのは`dataFile2`です。`To File`は新しいファイル名を制御ピンで受け取りますが、それは更新してしまったあとなのです。

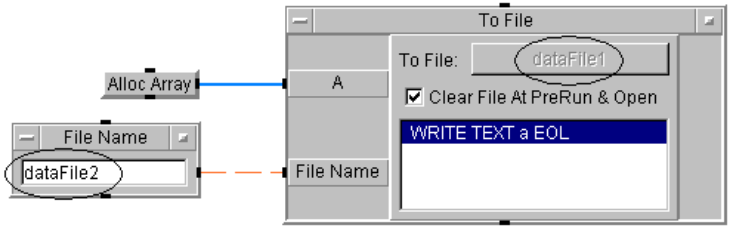


図8-24. 制御ピンを持つオブジェクトにおける順序の問題

この問題を解決するには、To Fileのシーケンス入力ピンを使って、制御ピンにデータが入るまでオブジェクトが動作しないようにします。図8-25のプログラムでは、File Nameのシーケンス出力をTo Fileのシーケンス入力に接続することにより、正しいファイルにデータが書き込まれるようにしています。

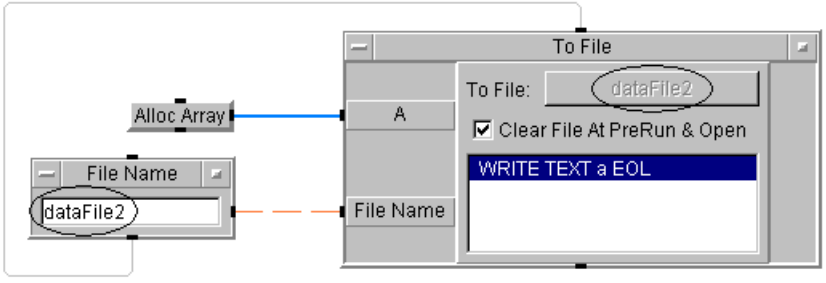


図8-25. 制御ピンを持つオブジェクトでのシーケンス入力の使用

レコードの構築

図8-26のプログラムでは、3つの波形からなるレコードを構築しようとしていますが、Build Recordオブジェクトは伝搬しません。Function GeneratorがDeMultiplexerに出力を送ったあとに、For Rangeオブジェクトがループを開始して0から2までカウントします。カウントのたびに、対応するAddrがDeMultiplexerからそれぞれのBuild Record入力に送られます。

注記

DeMultiplexerのように複数の出力を持つオブジェクトが、ループ内部で1回の反復ごとに1つの出力からデータを送出する場合、他の出力の値はループが先頭に戻るたびに無効になります。これは、不適切な古いデータが次のオブジェクトに伝搬するのを防ぐためです。このことは、Build Recordなど複数の入力を持つオブジェクトにも当てはまります。

ループの1回の反復ごとに1つの入力がデータを受け取る場合、他の入力の値はループが先頭に戻るたびにすべて無効になります。これは、プログラムが現在の値でなく過去の値で動作することにより、正しくない結果が生じるのを防ぐためです。

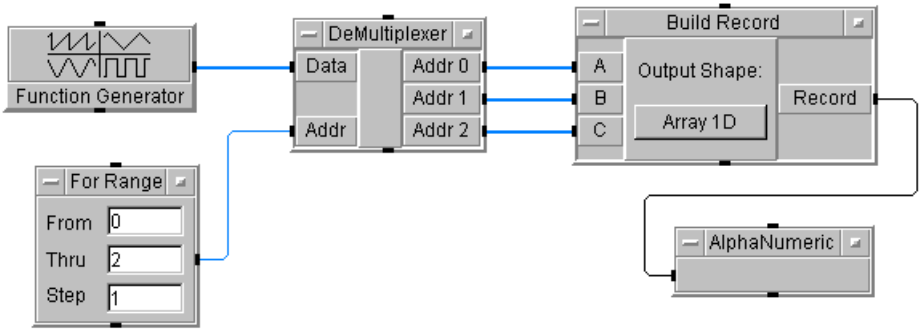


図8-26. 無効なデータ入力のためにループ内のBuild Recordが伝搬しない例

Build Recordオブジェクトが Record出力を伝搬しない理由は、VEEのループの動作方法にあります。このプログラムでは、For Rangeの反復のたびに、前回の反復でBuild Recordの入力に置かれたデータが無効になります。Build Recordオブジェクトはループの1回の反復ごとに1個の入力しか受け取らないので、同時に有効な入力は1つだけであり、Record出力は生成されないのです。

このプログラムを実際に作成し、Show Data Flowをオンにして、ループの1回の反復ごとにDeMultiplexerから1個のデータだけが伝搬することを確認してみてください。

図8-27のプログラムでは、UserObjectを使うことで、この方法が意図したとおりに動作するようにしています。この例では、反復がすべて終了するまでUserObjectの出力端子にデータが保持されます。UserObjectが3つの出力を同時に送出するので、Build

Recordへの入力データは有効になります。3つの入力すべてに有効なデータが含まれる場合、Build Recordオブジェクトは意図したとおりの結果を出力します。

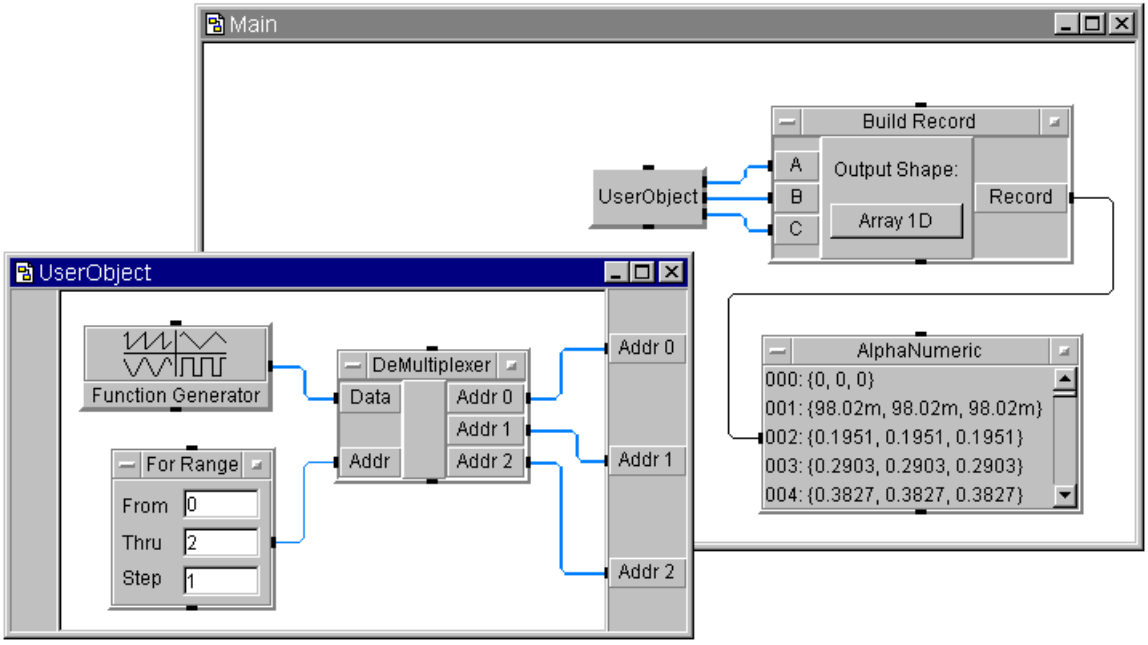


図8-27. データ入力が無効になっても伝搬を行わせる方法

図8-28のプログラムでは、DeMultiplexerの代わりにShift Registerを使うことで問題を解決しています。DeMultiplexerでは同時に1つの入力だけが有効であるのに対して、Shift Registerの3つの出力はすべて同時に有効であり、Build Recordの3つの入力に同時に送られます。Shift Registerの出力にデータが存在しない場合、nilが伝搬されます。

このプログラムでは、3つの波形をShift Registerに入力してからBuild Recordを使ってそれらをレコードにします。レコードの代わりに配列を出力したい場合は、Collectorを使います。

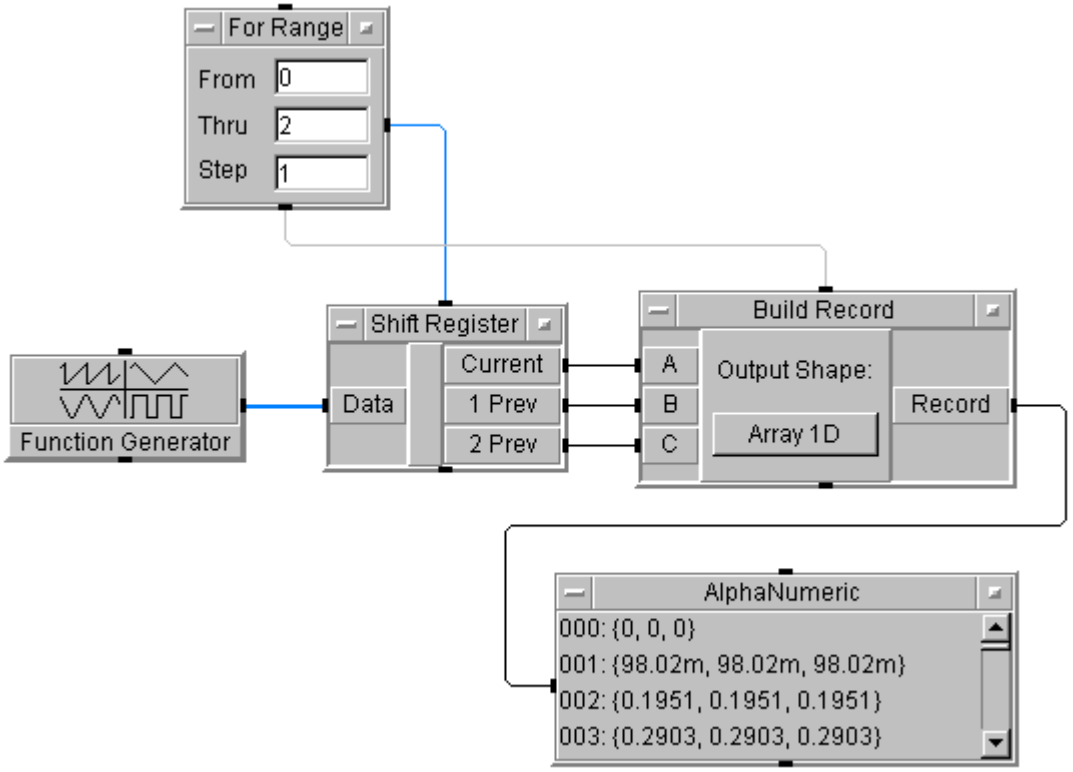


図8-28. データ入力が無効になるのを防ぐことで伝搬を行わせる方法

Formulaに対する複数の入力

ループ内部でFormulaオブジェクトの複数の入力に値を送る場合、異なるループ・サイクルで値が送られると伝搬の問題が生じることがあります。前のBuild Recordオブジェクトの例で示したように、どれかの入力端子に無効なデータがあるとFormulaは動作しません。ループ内部でDeMultiplexerオブジェクトを使うと、さらに問題が複雑になります。図8-29のプログラムはこの問題を示しています。

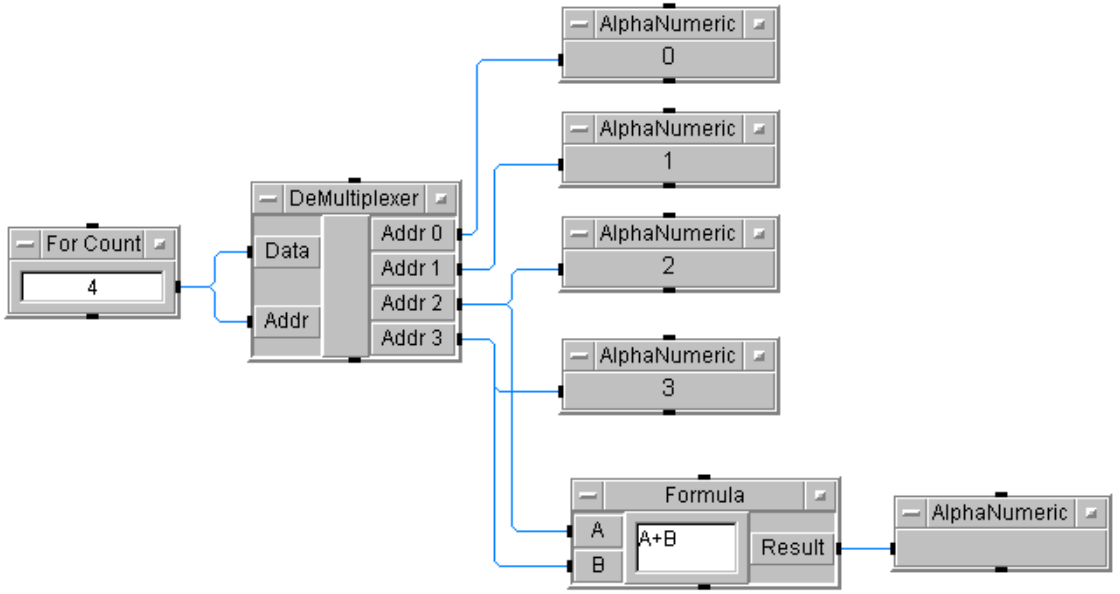


図8-29. ループ内のFormulaで無効なデータ入力のために伝搬が停止する例

DeMultiplexerはFor Countオブジェクトによって駆動され、カウントの各サイクルごとに1つの値(0～3)を出力します。1回のサイクルで出力される値は1個だけなので、Formulaオブジェクトへの入力は各サイクルの終わりで無効になります。Formulaの2つの入力AとBは同時に有効になることがないので、Formulaは実行されず、Formulaからの出力は得られません。

プログラムを書く側は、ピンAの値2がループの次の反復でも有効であると決めていますが、VEEにはそれがわかりません。このループは、同じ式に対する複数の係数を計算しているかもしれません。係数のうち半分が古い値で、半分が新しい値のときに、式を計算しても何の意味もありません。一般的には、前回の反復で生成されたデータが「期限切れ」であると想定するほうが、プログラミング言語にとっては安全です。このためにVEEは、ループの各回の反復の先頭でオブジェクトの入力を無効にするのです。

ループの操作

変化しない入力の値がループの反復ごとに無効になるという状況に対処するには、いくつかの方法があります。前のセクションの例では、Shift Registerによって複数の有効な出力を同時に生成する方法を示しました。ほとんどの作業に直接適用

できる方法として、変数の使用があります。図8-30のプログラムは、変数を使って Formulaオブジェクトに有効な値を供給する方法を示します。

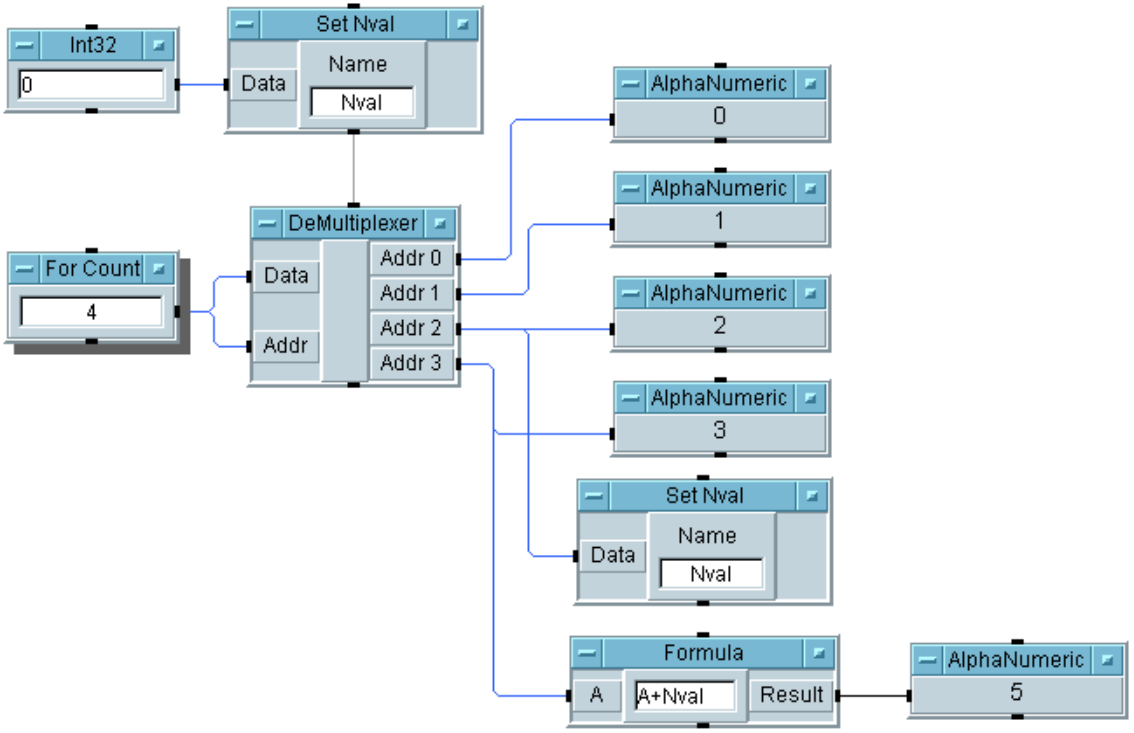


図8-30. 変数を使ってFormulaのデータ入力が無効になるのを防ぐ例

このプログラムでのグローバル変数Nvalの使用方法には、2つの重要なポイントがあります。第1に、プログラムが動作を開始したときにNvalは初期化されます。このプログラムに限っては、VEEが使用前にこの変数を初期化するので、最初の初期化は不要ですが、初期化しておくのがプログラミング上のよい習慣です。第2に、NvalがFormulaによって使用される前に、必ず新しい値がNvalに設定されます。

変数には、オブジェクトによって使用される前に必ず値を設定する必要があります。そうしないと、無効なデータの場合と同様な問題が生じます。変数が初期化されている場合、値が正しくない可能性があります。変数が初期化されていない場合、Variable was not foundのようなエラーが生じます。

イベントの時間測定

Timerオブジェクトは、プログラムでの接続のしかたによっては奇妙な(おそらくは誤った)結果を表示することがあります。図8-31のプログラムは、VEEの伝搬の問題を考慮してプログラム内のオブジェクトを接続しなければならないことを示しています。

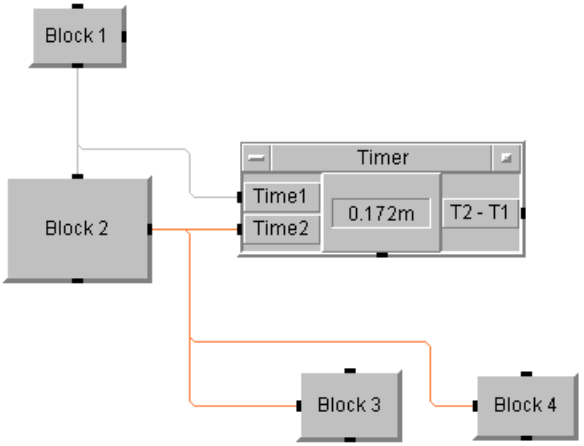


図8-31. 制御されないタイマ入力によってタイミング・エラーが発生する例

このプログラムの"Block"は、何らかのVEEオブジェクト群を含む任意のスレッドです。Block 2の実行時間を測定するためにTimerが使われています。このプログラムは、動作する場合も、エラーを生じる場合もあります。
プログラムが動作した場合でも、Timerの結果は正しくない可能性があります。2つの問題が、タイマの測定開始と終了のタイミングに影響します。

第1に、Block 1のシーケンス出力は、Block 2と、TimerのTime1入力の両方にデータを送ります。どちらに先にデータを送るかをプログラムは指定していないので、どちらかが任意に選ばれます。Block 2が先に選ばれた場合、Block 2が終了するまでTime1にはデータが送られません。この場合、誤った時間測定結果が得られるか、Time1にデータが送られる前にBlock 2の出力からTime2にデータが送られればエラーが発生します。

問題を発見するためにShow Data Flowデバッグ機能をオンにしても、データ・フロー・インジケータからは実際の伝搬の様子が完全に分からないおそれがあります。

この場合の伝搬の順序は、Block 1のシーケンス出力からTime1とBlock 2へのラインが接続された順序に依存します。

2番目の問題は、Block 2の出力からTimerのTime2入力への接続に関するものです。これはBlock 3と4の入力にも接続されているため、どの入力が最初に動作するかは保証されません。このため、誤った時間測定結果が得られるおそれがあります。

図8-32は、正しい伝搬と正確な時間測定が得られるようにプログラムを修正する方法を示します。Block 1とBlock 2との間にDoオブジェクトを挿入し、Block 2のシーケンス出力をTime2入力に接続します。Doオブジェクトは、プログラムでDoオブジェクトの周りに記されている番号の順序でオブジェクトが実行されるようにします。

Block 2の出力ピンは、周りに示された番号の順序で動作します。Block 2のシーケンス出力ピンをTime2入力に接続することにより、TimerはBlock 2とその出力ピンが駆動するすべてのブロックが終了したあとで結果を表示します。同様に、Block 2のシーケンス出力ピンに他のブロックが接続されている場合は、正しい伝搬を保証するためにさらにDoオブジェクトを追加します。

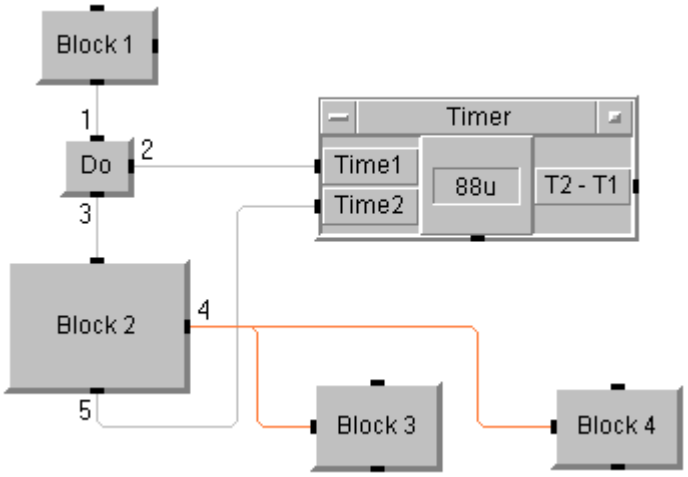


図8-32. Doオブジェクトと Timerを組み合わせて正しい結果を得る方法

9

演算機能

演算機能

この章では、スカラと配列に対する演算機能について説明します。下記の内容があります。

- データ・コンテナの理解
- データ型変換
- データの処理
- VEEでの配列操作
- VEEでの配列操作

データ・コンテナの理解

VEEプログラムでのデータ伝搬は、あるオブジェクトから別のオブジェクトにデータ・コンテナが移動することによって起こります。データ・コンテナは、VEEの内部的なデータ・フォーマットです。すべてのデータ・コンテナには、データ型(テキスト、実数など)とデータ形状(スカラ、1次元配列など)があります。

データ・コンテナの操作

データ・コンテナには、1個の値だけが含まれる場合も、いくつかの値の配列が含まれる場合もあります。どちらの場合も、オブジェクトが動作したときに1本のデータ出力ピンから伝搬するデータ・コンテナは1個だけです。

図9-1のサンプル・プログラムで、`Real64`定数オブジェクトは1次元配列として構成されています。`Int32`定数オブジェクトはスカラとして構成されています。

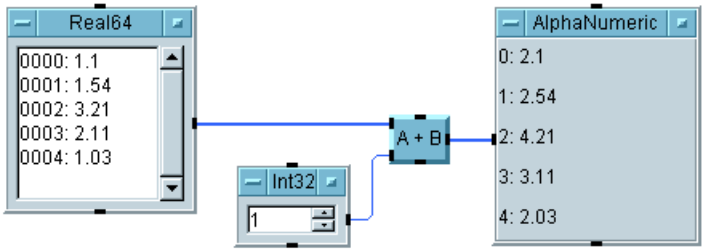


図9-1. VEEは必要に応じてデータ型を自動変換

プログラムが動作すると、`Real64`定数オブジェクトは1次元実数配列のデータ・コンテナを伝搬します。`Int32`定数オブジェクトは整数スカラ(値`1`)のデータ・コンテナを出力します。VEEは自動データ型変換によってこれら2つのコンテナを加算します。

VEEは整数値`1`を等価な実数値(`1.0`)に変換します。`a+b`オブジェクトは、1次元実数配列のすべての要素に実数値`1.0`を加算し、結果の1次元実数配列を上記のように出力します。

VEEデータ・ライン上で送られる実際のコンテナを調べたい場合、Line Probeを使います。目的のラインの上にマウス・ポインタを動かすと、ラインが強調表示されるので、左マウス・ボタンを押します。Line Valueボックスが表示されます。例えば、上記の例のReal64定数オブジェクトのデータ出力ライン上で送られるコンテナは、図9-2のように表示されます。

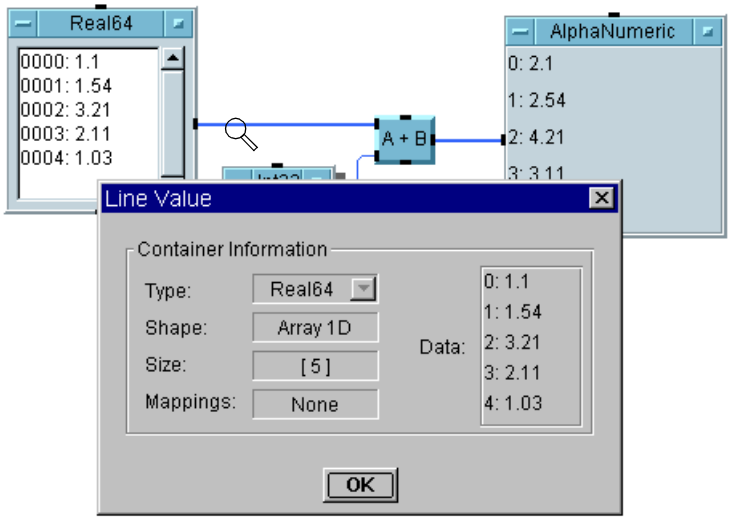


図9-2. ラインを左クリックしてデータ・コンテナを表示

一般に、VEEはデータ型を自動的に変換し、可能ならデータ形状を解決します。これがどのように行われるかについて、通常は考える必要はありません。変換プロセスの技術情報について知りたい場合は、302ページ「データ型変換」を参照してください。

端子情報

端子には、オブジェクト入力が要求する型および形状と、入出力コンテナに関する情報が表示されます。ピンは端子の接続ポイントです。端子が存在するのに表示されない場合は、オブジェクト・メニューから表示させることができます。Propertiesをクリックし、GeneralタブでShow Terminalsをチェックします。

端子の属性を表示または変更するには、端子の情報エリア(ピンではなく)をダブルクリックします。端子情報を記載したダイアログ・ボックスが表示されます。

ダイアログ・ボックスのすべてのフィールドが淡色表示されている場合、端子は変更できません。入力フィールド(白の背景)やボタンがある場合は、値を変更できます。

端子には下記の特性があります。

- Nameは端子の名前です。このフィールドは通常変更可能です。Formulaの式では、端子名を式の中で使用できます。
- Modeには端子のタイプ(Data、Control、Trigger、Errorなど)が表示されます。このフィールドは変更できません。
- Required TypeとRequired Shape(入力端子のみ)は、オブジェクトが要求する入力データに関する情報を指定します。一部のオブジェクトではRequired TypeまたはRequired Shapeを変更できますが、通常はその必要はありません。
- Container Informationには、オブジェクトが処理するコンテナの情報(入力端子の入力要件に基づく)、または処理したコンテナの情報(出力端子の場合)が記載されます。この情報には、データ型、データ形状、サイズ(データが配列の場合)、マッピング、データ自体が含まれます。

データ型変換

従来のプログラミング言語では、データ型の間の変換は手動で行わなければならないのが普通です。VEEではほとんどのデータ型がオブジェクトの入力端子で自動変換されます。組込みの型変換関数やオブジェクトを使用することもできます。

注記

入力端子ではデータ形状の変換は行われませんが、演算機能の中ではデータ型と形状の両方が自動変換される場合があります。312ページ「データの処理」を参照してください。機器I/Oトランザクションにおけるデータ型の変換は特別な場合です。詳細については、310ページ「機器I/Oのデータ型変換」を参照してください。

VEEのデータ型

VEEには15のデータ型が用意されています。データ型とデータ型変換の詳細については、302ページ「データ型変換」を参照してください。ActiveXオートメーションおよびコントロールに対するVEEのサポートについては、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

注記

VEEオブジェクトの入力端子で`Any`(ほとんどの場合のデフォルト)が指定されている場合、任意のVEEデータ型のコンテナが受け入れられます。複合データ型(Waveform、Spectrum、Record、Coord、Object)には特定のデータ形状が対応します。

データ型の説明

表9-1に示すデータ型が、すべてのVEE動作に用いられます。すなわち、VEEオブジェクトの間で受け渡されるすべてのVEEデータ・コンテナはこれらの型のどれかです。

表9-1. VEEデータ型

型	説明
Complex	直交座標形式の複素数。1つの複素数は、(real,imag)という形式の実数部と虚数部を持ちます。実数部と虚数部はともにReal64です。例えば、複素数1 +2iは(1,2)と表されます。
Coord	(x, y, ...)という形式の2個以上の成分を持つ複合データ型。各成分はReal64です。Coordのデータ形状はScalarまたはArray 1Dでなければなりません。
Enum	特定の整数値に対応するテキスト文字列。Enumデータ型を伝搬するのは、Data ⇒ Selection Controlの下にあるオブジェクト(Radio Buttonsオブジェクトなど)です。 整数値にアクセスするには、オブジェクトのordinal出力ピンまたはordinal(x)関数を使います。Enumのデータ形状はScalarでなければなりません。Enumは必須データ入力型になることはできません。
UInt8	8ビット2の補数符号なし整数(0～255)
Int16	16ビット2の補数整数(-32768～32767)
Int32	32ビット2の補数整数(-2147483648～2147483647)
Object	VEE 5以上の実行モードにおいて、ActiveXオートメーション・オブジェクトおよびコントロール用に予約されたデータ型。ObjectはUserObjectおよびUserFunctionとの間で入出力として受け渡すことができますが、リモートUserFunctionやコンパイル関数との間で受け渡すことはできません。 Object変数の値には、Dispatchインタフェース名("Range"、"Application"など)と、オートメーション・オブジェクトからエクスポートされたDispatchのポインタ値が含まれます。Objectのデータ形状はScalarでなければなりません。

表9-1. VEEデータ型

型	説明
PComplex	(mag, @phase)という形式の振幅成分と位相成分。位相は現在の三角関数単位で表されます。例えば、Trig ModeがDegreesに設定されていれば、振幅4、位相30度のPComplex数値は(4, @30)と表されます。各成分はReal64です。
Real32	IEEE 754標準に適合する32ビット実数値 (有効数字10進約8桁: ±3.40282347E±38)。
Real64	IEEE 754標準に適合する64ビット実数値 (有効数字10進約16桁: ±1.7976931348623157E308)。
Record	いくつかのフィールドから構成されるデータ型。各フィールドは、名前と、任意の型(Recordを含む)でデータ形状がScalarまたは1D Arrayのコンテナを持ちます。
Spectrum	周波数領域の値を表す複合データ型で、PComplex値のポイント群と、最小/最大周波数値からなります。周波数領域データは対数または直線スケールで一様にマッピングされます。Spectrumのデータ形状はArray 1Dでなければなりません。
Text	英数字からなる文字列。
Variant	Variantデータ型は、特定の種類のデータに固定されません。必要に応じて、他のどんなデータ型にもなることができます。ByRef Variantパラメータを使用するActiveXオートメーション・メソッドで用いられます。
Waveform	時間領域の値を表す複合データ型で、直線スケール上に等間隔でマッピングされたReal64値のポイント群と、波形の全タイム・スパンからなります。Waveformのデータ形状はArray 1D(1次元配列)でなければなりません。

データ型とライン・カラー

VEE 4以上の実行モードでは、データ・ラインを通るデータの型に応じて、異なるカラーがラインに割り当てられます。下に示すのは、デフォルト・カラーとカラー・プロパティ名(File ⇒ Default Preferencesで変更可能)の一覧です。

- 青: 数値(IntegerまたはReal型)
- 青: 複素数(ComplexおよびPComplex型)
- オレンジ: 文字列(String型)
- グレー : シーケンス出力(nil値、通常はシーケンス出力ラインから)
- 黒: 未知の型または最適化されない型(Record型など)

注記

データ型が配列の場合、太いラインが表示されます。プログラムの速度を改善するには、カラー付きのラインに注目してください。黒以外のラインが多いほど、プログラムは高速に動作します。

VEEのデータ形状

複合データ型(Waveform、Spectrum、Record、Coord、Enum、Object)には特定のデータ形状が対応します。

- WaveformおよびSpectrumデータ型は、常に1次元配列です。
- RecordおよびCoordデータ型は、スカラまたは1次元配列です(2次元以上の配列にはなれません)。
- ObjectおよびEnumデータ型は常にスカラです。

他のデータ型はすべて、ScalarまたはArrayのデータ形状を持つことができます。

- `Scalar`は単一の数(`10`、(`32, @10`)など)です。
- `Array`は1～10次元の配列です。

配列にはマッピングが必要です(マッピングとは、配列の独立変数を表す連続値または離散値の集合です)。

多くの場合、VEEオブジェクトのデータ・ピンの入力データ形状要件は`Any`であり、さまざまなデータ形状のコンテナをオブジェクトが受け入れることを示します。

データ型の変換

このセクションでは、入力端子でデータ型を変換する方法、オブジェクトや関数を使ってデータ型を変換する方法、機器I/Oのデータ型変換について説明します。

入力端子でのデータ型変換

ほとんどのオブジェクトのデータ入力端子は任意のデータ型を受け入れますが、一部のオブジェクトでは特定のデータ型または形状が要求されます。これらのオブジェクトの場合、データ入力端子は入力コンテナを必要なデータ型に自動的に変換しようとします。

例えば、Magnitude Spectrum表示にはSpectrumデータが必要です。Function Generatorの出力(Waveform)がMagnitude Spectrum表示に接続された場合、Magnitude Spectrumの入力端子は自動的にFFTを実行して、時間領域データを周波数領域データに(WaveformをSpectrumに)変換します。

型変換にはプロモーションとデモーションがあります。プロモーションとは、情報の少ないデータ型から情報の多いものへの変換です。例えば、Int32からReal64への変換はプロモーションです。プロモーションは必要に応じて自動的に行われます。

デモーションとは、データの一部が失われるような変換です。例えば、Real64からInt32への変換はデモーションです。実数値の小数部が失われるからです。データ型のデモーションが行われるのは、オブジェクトの入力に対してユーザが特定のデータ型を指定した場合に限られます。

データ型を指定した場合、デモーションが必要で可能ならば、デモーションは自動的に行われます。

例えば、Formulaオブジェクトの必須入力型をInt32に変更し、他のオブジェクトがその入力にReal64の数値(28.2など)を供給した場合、値はInt32(28)に変換されます。Formula入力のデータ型をAnyからInt32に変更するには、入力端子の情報エリア(ピンではなく)をダブルクリックし、Required Typeフィールドをクリックします。ドロップダウン・リストでInt32をクリックして型を変更します。

次にプログラムを実行したときに、データ変換が行われます。供給されたデータの型が、入力で選択したデータ型に変換できないものだった場合、エラーが返されます。

オブジェクトおよび関数によるデータ型の変換

VEEには、データの型を変換するためのオブジェクトと組込み関数が用意されています。これらは入力端子での型変換では不可能な特殊な型変換を処理するために用いられます。

VEEに組み込まれている型変換関数をプログラムに追加するには、Function & Object Browserを使うか、式が使用できるオブジェクトに関数名を入力します。Browserでは、Type: Built-in FunctionsとCategory: Type Conversionを選択すると、使用できる関数が表示されます。

例えば、asText(x)関数は入力xをTextデータ型に変換し、xと同じデータ形状で返します。xは任意の形状、任意の型を取ることができます。式asText(3.4)の結果はText値"3.4"になります。

注記

VEE 5.0より前では、Formulaなどのオブジェクトの中の式にIntegerを直接入力した場合、Realに変換されていました。VEE 5.0以降でExecution ModeがVEE 5以上に設定されている場合、式の中のIntegerがRealに変換されることはなくなりました。入力端子でのデータ型変換にはこの変更は無関係です。例えば、Formula式"2^4"はVEE 4以前のモードではReal64の値16.0を生成しますが、VEE 5以上のモードではInt32の値16を生成します。

優先度が最も高いのはRecordデータ型です。ただし、Recordデータ型との間のプロモーションやデモーションは自動的には行われません。RecordとRecord以外のデータとの間の変換には、Build Recordオブジェクトと Unbuild Recordオブジェクトを使います。

316ページ「式でのレコードの使用」で説明するシンタックスを使った式でも、同様の結果を得ることができます。レコードの詳細については、第11章「レコードとデータセットの使用」を参照してください。

Coordデータ型に関しては、下記の特別な規則が存在します。

- Int32 および Real データ型の配列を Coord にプロモートすることは可能ですが、Coordを他の数値型に変換することはできません。
- マッピングのない配列を Coord に変換する場合、Coord の独立値 (最後以外のフィールド)は配列のインデックスから作成され、従属値(最後のフィールド)に要素の値が入ります。例えば、配列AをCoordに変換する場合、Aの内容が[1,5,7]なら、[(0,1),(1,5),(2,7)]という内容のCoord配列に変換されます。

■ マッピングのある配列をCoordに変換する場合、Coordの独立パラメータはマッピングの最小値から`Xmin+(Xmax-Xmin/N)*(N-1)`までの値を取ります。

Objectデータ型に対しても自動プロモーション/デモーションは行われず、この型を他のデータ型に変換することはできません。Objectデータ型はActiveXオートメーション・オブジェクトに用いられます。オートメーション・オブジェクトの作成と使用には、`CreateObject`関数と`GetObject`関数を使います。ActiveXオートメーションおよびコントロールに対するVEEのサポートについては、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

自動データ型変換

表9-2に示すのは、入力端子での自動型変換と関数による型変換、およびエラーを起こす型変換の一覧です。「可」は型変換が可能なことを、「不可」はエラーが返ることを示します。陰付きで示したのはデモーションです。

新しい"Variant"データ型は、特定のデータ型ではありません。この型は、式の中で、その表現する値が他のデータ型のどれかに属することを示すために用いられます。その型は、その時点で保持されている値のデータ型となります。任意のデータ型がVariantにプロモートまたはデモートでき、Variantからのプロモーションまたはデモーションは、その保持しているデータと、そのデータが従う規則に基づいて行われます。表9-2にはVariantは示されていません。

表9-2. データ型のプロモーションとデーモン

変換後→ ↓変換前	UInt8	Int16	Int32	Real32	Real64	Complex	PComplex	Waveform	Spectrum	Coord	Enum	Text
UInt8	n/a	可	可	可	不可[5]	不可	不可	不可	不可	不可	不可	不可
Int16	可	n/a	可	可	可	可[2]	可[2]	不可	不可	可[3]	不可	可
Int32	可	可	n/a[1]	可	可	可[2]	可[2]	不可	不可	可[3]	不可	可
Real32	可	可	可[4]	n/a	可	可[2]	可[2]	不可	不可	可[3]	不可	可
Real64	可	可	可	可	n/a	可[2]	可[2]	不可	不可	可[3]	不可	可
Complex	不可	不可	不可	不可[5]	不可[5]	n/a	可	不可	不可	不可	不可	可
PComplex	不可	不可	不可	不可[5]	不可[5]	可	n/a	不可	不可	不可	不可	可
Waveform	不可	可[4]	可[4]	可[6]	不可[5]	不可	不可	n/a	可[7]	可	不可	可
Spectrum	不可	不可	不可	不可	不可[5]	可[6]	可[6]	可[7]	n/a	不可	不可	可
Coord	不可	不可	不可	不可	不可[5]	不可	不可	不可	不可	n/a	不可	可
Enum	不可	不可[8]	不可[8]	不可	不可[5]	不可	不可	不可	不可	不可	n/a	可
Text	不可	可[9]	可[9]	可[9]	可[9]	可[9]	可[9]	不可	不可	可[9]	不可	n/a

1. n/a = Not applicable(該当せず)
2. Int32またはReal*の*value*は*、Complex*の*(*value*、0)またはPComplex*の*(*value*、@0)にプロモートされます。
3. 独立成分、すなわちn個のフィールドを持つCoordの最初のn-1個のフィールドは、配列がマッピングされていなければ配列のインデックスになります。配列がマッピングされている場合、独立成分は各次元のマッピングから得られます。従属成分yは配列の要素になります。コンテナがスカラ(配列でない)の場合、変換はエラーになります。
4. 値が変換後の型の範囲を超えていた場合、これらのデモーションはエラーになります。
5. このデモーションは自動的には行われませんが、re(x)、im(x)、mag(x)、phase(x)の各オブジェクトまたは`Data ⇒ Build Data/Unbuild Data ⇒`のオブジェクトを使って実行できます。
6. WaveformとSpectrumのマッピングはこれらのデモーションで保持されます。
7. FFTまたは逆FFTが自動的に実行されます。
8. このデモーションは自動的には行われませんが、`ordinal(x)`オブジェクトを使って実行できます。

9. このデモーションは、テキスト値が数値(34、42.6など)でない場合、または受入れ可能な数値フォーマットでない場合にはエラーになります。受入れ可能なフォーマットは下記の通りです(スペースは、数値内部にある場合を除いて、無視されます)。
   - Int32またはReal型にデモートされるテキストには、下記のものが含まれていてもかまいません。
     - 前置符号。-34など。
     - サフィックスeまたはE、その後に任意指定の符号またはスペース、その後に整数。42.6E-3など。
   - Complexにデモートされるテキストは、(数値, 数値)という形式でなければなりません。
   - PComplexにデモートされるテキストは、(数値, @数値)という形式でなければなりません。位相(2番目の成分)は、この変換では常にラジアンと見なされ、`Trig Mode`設定には影響されません。
   - Coord型にデモートされるテキストは、(数値, 数値, ...)というフォーマットでなければなりません。

機器I/Oのデータ型変換

整数を使用するI/Oトランザクションに対して、VEE 5およびそれ以前の実行モードでは自動データ型変換が行われます。これによってプログラムの動作が変化する例については、17ページ「VEE実行モードの使用」を参照してください。VEEは下記のように自動型変換を実行します。

- VEE 5およびそれ以前の実行モードの`READ`トランザクションにおいて、機器からの`Int16`または`Byte`値は符号拡張を伴って`Int32`値に変換されます。また、機器からの`Real32`値は64ビット`Real`値に変換されます。VEE 6実行モードでは、`Int16`、`Int32`、`Byte(UInt8)`の本来の値が保持されます。
- VEE 5およびそれ以前の実行モードの`WRITE` トランザクションにおいて、`Int32`または`Real`値は下記の方法で機器に対する適切な出力フォーマットに変換されます。VEE 6実行モードでは、`Int16`、`Int32`、`Byte(UInt8)`の本来の値が書き込まれます。
  - 機器が `Real32` フォーマットをサポートする場合、64 ビットの `Real` 値は`Real32`値に変換されて機器に出力されます。`Real`値が`Real32`値の範囲を超えている場合、エラーが発生します。
  - 機器が`Int16`フォーマットをサポートする場合、`Int32`値は`Int16`値に切りつめられて機器に出力されます。`Real`値はまず`Int32`値に変換され、VEE 5実行モードでは切りつめられてから出力されます。VEE 6実行モードでは、数値を切りつめずに出力します。`Real`値が`Int32`の範囲を超えている場合、エラーが発生します。

- 機器がByteフォーマットをサポートする場合、VEE 5実行モードではInt32値がByte値に切りつめられて機器に出力されます。Real値はまずInt32値に変換され、VEE 5実行モードでは切りつめられてから出力されます。VEE 6実行モードでは、数値を切りつめずに出力します。Real値がInt32の範囲を超えている場合、エラーが発生します。

データの処理

データを処理するには、Function & Object Browserに用意されている演算子と関数を使います。ブラウザをオープンするには、Function & Object Browserツールバー・ボタンを使います。関数を組み合わせることで、演算式を作成できます。

Function & Object Browser

Function & Object Browserには、データをさまざまな方法で処理するための演算関数が揃っています。これらの関数は、対応するタイトルと入出力を持つFormulaオブジェクトに式として入力します。Formulaオブジェクトのオープン・ビューでは、式やオブジェクトのプロパティを変更できます。

Function & Object Browserに記載されたすべての関数は、式が使用できる他のメニューの任意のオブジェクトで使用できます。式が使用できる他のメニューのオブジェクトとしては下記のものがあります。

- Data ⇒ Access Array ⇒ Set Values
- Data ⇒ Access Array ⇒ Get Values
- Data ⇒ Access Record ⇒ Get Field
- Data ⇒ Access Record ⇒ Set Field
- Device ⇒ Sequencer
- Flow ⇒ If/Then/Else
- Flow ⇒ Conditional(すべての条件オブジェクト)
- トランザクションを使用するI/Oオブジェクト

一般的概念

Constantオブジェクトなどの数値入力フィールドを使えば、プログラムを実行する前にデータを処理できます。一部のオブジェクトの数値入力フィールドでは、任意の式が使用できます。この式はただちに評価され、結果の値がフィールドに設定されます。Constantで変数を使用することはできません。

入力する式は、オブジェクトが要求する型、またはそれに変換可能な型のスカラ値に評価される必要があります。一般的には、任意の2項演算子、ネストのための括弧、関数呼出し、既定義の数値定数PI(3.1416...)が数値入力フィールドで使用できます。

式と関数

式で使用できるのは、データ入力端子名、データ出力端子名(I/Oトランザクションと Formulaのみ)、変数(任意のスコープの宣言済み、および未宣言)、ユーザ定義関数(コンパイル、リモート、UserFunction)、およびFunction & Object Browserに用意されている任意の演算式です。データ入力端子名は変数として用いられます。

USASCIIキーボードを使用する場合、式の中の入力変数名の大文字小文字は区別されません。USASCII以外のキーボードを使用する場合、7ビットASCII文字に関してのみ大文字小文字が区別されません。式は実行時に評価されます。

関数に配列を渡すと、特に記述がない限り、関数は配列の各要素に対して動作します。例えば、スカラのsqrtはスカラを返すので、sqrt(4)は2を返します。一方、配列のsqrtは同じサイズの配列を返すので、sqrt([1,4,9,64])は配列[1,2,3,8]を返します。

VEE 5以上の実行モードでは、式フィールドに整数として入力された数値はすべてInt32と見なされます。VEE 3およびVEE 4実行モードでは、これらの数値は括弧を使ってComplexまたはPComplexを指定しない限りReal64と見なされていました。したがって、2は実行モードによってRealまたはInt32と見なされます。(1, @2)はPComplex数値であり、(1, 2)は直交座標のComplex数値です。

注記

VEEでは括弧の中の値はすべてComplexまたはPComplexと見なされます。式でCoord値を使用する場合、coord(x, y)関数を使います。coord関数は2個以上のパラメータを取ります。coord(1, 2)は2個のフィールドを持つスカラCoordコンテナを返します。

Coordデータ型を操作するすべての関数は、Coordの従属フィールド(最後のフィールド)だけを対象とします。例えば、abs(coord(-1, -2, -3))は(-1, -2, 3)というCoordを返します。

Enumコンテナは、ordinal(x)関数を除くすべての演算でTextに変換されます。Enumの配列はサポートされません。Enumの配列を作成しようとすると、Textの配列が作成されます。

各データ型の定義については、302ページ「VEEのデータ型」を参照してください。

式での文字列の使用

式の中の文字列は、2重引用符で囲む必要があります。文字列内では表9-3のエスケープ・シーケンスが使用できます。

表9-3. エスケープ・シーケンス文字

エスケープ文字	意味
\n	改行
\t	水平タブ
\v	垂直タブ
\b	バックスペース
\r	キャリッジ・リターン
\f	改ページ
\"	2重引用符
\'	シングル引用符
\\	バックスラッシュ
ddd	文字コード値。*d*は8進数字。

式での変数の使用

変数の作成と設定には`Declare Variable`オブジェクトと`Set Variable`オブジェクトを使用し、変数にアクセスするには`Get Variable`オブジェクトを使います。詳細については、**VEEオンライン・ヘルプ**で`Declare Variable`、`Set Variable`、`Get Variable`を参照してください。

また、演算式の中に変数名を書くことによっても変数にアクセスできます。`Formula`オブジェクトと、遅延評価式フィールドを持つすべてのオブジェクトの演算式で変数が使用できます。

このようなオブジェクトとしては、`If/Then/Else`、`Get Values`、`Get Field`、`Set Field`、およびトランザクションで式を使用するすべての機器I/O(`To File`、`From File`、`From DataSet`、`From Stdin`、`To/From Named Pipes`、`To/From Socket`、`Sequencer`、`Direct I/O`)があります。

式で変数を使用するには、入力変数と同じように変数名を使用します。例えば、プログラムでSet Variableオブジェクトを使ってnumFilesという変数が定義されている場合、プログラムの他の場所にある、入力Aを持つFormulaオブジェクトではnumFiles+3*Aという式が使用できます。

注記

変数名の大文字小文字は区別されません。大文字と小文字のどちらを使用してもかまいません。例えば、GLOBALAとglobalAは同じものと見なされます。

エラーや予期しない結果を避けるため、式で変数を使用する際には下記の2つの制限に注意してください。

1. **ローカル入力変数はグローバル変数よりも優先されます。**すなわち、変数名が重なる場合、グローバル変数でなくローカル変数が用いられます。例えば、Freqという入力(ローカル変数)を持つFormulaオブジェクトにFreq*10という式が指定され、同時にFreqというグローバル変数も存在する場合、この式はグローバル変数でなくローカル変数のFreqを使って評価されます。名前が重なってもエラーは発生しません。

2. **プログラムのフローによっては、変数を含む式を評価するオブジェクトが変数定義よりも先に実行されるおそれがあります。**例えば、変数globalAがSet Variableオブジェクトで設定され、Formulaオブジェクトに式globalA*X^2が含まれる場合を考えます。

   プログラムのフローによっては、Formulaオブジェクトが Set Variableオブジェクトよりも先に実行される可能性があります。この場合、globalAが未定義なのでFormulaオブジェクトは式を評価できません。エラー・メッセージが表示されます。

   この場合、正しい伝搬を保証する、すなわちSet Variableが先に実行されるようにするための手段を取る必要があります。このためには、Set Variableオブジェクトのシーケンス出力ピンをFormulaオブジェクトのシーケンス入力ピンに接続します。この変数を評価する式を含むオブジェクトが他にあれば、それにも同様に接続します。

Get Variableオブジェクトが存在する場合、そのシーケンス入力ピンにもSet Variableのシーケンス出力ピンを接続します。また、Declare Variableオブジェクトで変数を宣言する場合、Set Variableで変数を初期化する必要があります。詳細については、第10章「変数」を参照してください。

注記

デフォルトでは、Default Preferencesダイアログ・ボックスのDelete Variables at PreRunがチェックされてオンになっているため、プログラムを実行したときに変数の値が削除されます。これは、変数に古いデータが含まれるために予期しない結果が生じるのを避けるためです。

変数は配列の値を取ることもできます。変数配列にアクセスする方法は、入力変数にアクセスする場合の配列シンタックスと同じです。例えば、GlobAry[2]のようにします。変数がRecordの場合、レコード・アクセス・シンタックスを使ってglobRecord.numFilesのようにします。

式でのレコードの使用

式を使って、レコードのフィールドまたはサブフィールドにアクセスできます。レコードAのBフィールドにアクセスするには、サブフィールド・シンタックスA.Bを使います。AがレコードでBというフィールドを持ち、BもレコードでCというフィールドを持つ場合、A.Bシンタックスを再帰的に使用してCフィールドにアクセスできます。すなわち、A.B.Cという式を使います。AにBというフィールドがない場合、あるいはBにCというフィールドがない場合、エラーが発生します。

式で使用するA.B.C.D.E.Fといった再帰の数に制限はありません。フィールド名の大文字小文字は区別されません(小文字と大文字は同じと見なされます)。フィールド名はサブレコードと重複していてもかまわないので、A.a.Aといった式も可能です。

レコードを使うと、1つの変数に複数の異なる値を保持できるので便利です。Formulaオブジェクトは Get Variable の代わりに使用できます。例えば、GlobRec.numFilesにアクセスするのに、Get VariableとFormulaオブジェクトを使ってレコードを分解しなくても、1個のオブジェクトでアクセスできます。

レコードと配列のシンタックスを式の中で組み合わせることにより、レコード配列のフィールドにアクセスしたり(A[1].Bなど)、レコードのフィールドとなっている配列の一部にアクセスしたり(A.B[1]など)できます。A[1].bとA.b[1]の違いに注意してください(どちらもサポートされます)。

- 前者は、スカラ・フィールドbを持つレコードの1次元配列に対して使用します。A[1].bは、レコード配列Aの2番目のレコード要素のフィールドbにアクセスします。
- 後者は、スカラ・レコードにフィールドbがあって、それが1次元配列の場合に使用します。A.b[1]は、レコードAのフィールドbの2番目の要素にアクセスします。

レコードのフィールドを変更するには、Formulaオブジェクトで代入演算子を使います。例えば、レコードRにフィールドAがあって、R.Aの値をsin(R.A)に変更したいとします。この場合、R.A = sin(R.A)という式を使います。レコードR(フィールドAの値が更新されたもの)は同じVEEプログラムで引き続き使用できます。

注記

式でオブジェクトを使って ActiveX オートメーション・オブジェクトおよびコントロールを操作する方法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

代入演算の使用

代入演算を使った式をFormulaオブジェクトで使用することにより、配列やレコードの一部の値を変更したり、ローカル変数やグローバル変数に値を代入したりできます。Result出力端子には、配列やレコード全体ではなく、その変更された部分だけが出力されます。

例えば、式A[2] = 4を持つFormulaのResult端子には、値4を持つ配列要素A[2]が出力されます。配列A全体が出力されるのではありません。Function & Object BrowserのType: Operators、Category: Assignmentには、あらかじめ代入を設定したFormulaオブジェクトが用意されています。

注記

代入演算はFormulaオブジェクトでのみ使用できます。

代入演算を使ってActiveXオートメーション・オブジェクトおよびコントロールを操作する方法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

使用可能なシンタックス. Formulaでは、複数の式をセミコロンで区切って指定できます。式の左辺で使用できるのは、配列およびレコードの要素と変数の値を変更するシンタックスです。右辺は左辺の変更対象の部分と**正確に**対応する必要がありま

す。レコードの場合、スキーマ(フィールドの型、形状、サイズ)は変更できず、値だけが変更できます。

下の例は、配列に使用できる左辺シンタックスを示します。

```
A[2]=
A[2, 3, 4:5]=
A[2:4, 4:6, 3:*, *]=
```

下の例は、レコードに使用できる左辺シンタックスを示します。

```
RecA.B=
RecA.B.C.D=
RecA[1].B=
RecA.B[1]=
RecA.B.C.D[1]=
RecA[1].B[2].C[3].D=
RecA.B[2:3].C[3:4]=
```

下の左辺シンタックスは、グローバル/ローカル変数を直接設定したり、宣言済みのローカル変数を初期化したりするために使用できます。

```
GLOBAL=2
TMP_LOCAL=4
```

例. 以下の代入の例では、左辺の変更対象の部分に右辺が正確に対応しなければならないことを示します。右辺のデータ型は左辺のデータ型に変換可能でなければなりません(IntegerからRealへ、RealからStringへなど)。ComplexからRealへなどの変換はできません。

- `ArrayA[2:4] = ArrayB`
  (`ArrayB`は3個の要素を持つ1次元配列でなければなりません)

- `ArrayA[2, 3, 4:5, 7, 8:9] = ArrayB`
  (`ArrayB`は2×2の2次元配列でなければなりません)

- `Rec[3:4].field = ArrayB`
  (`ArrayB`は2個の要素を持つ1次元配列でなければなりません)

- `Rec[3:4].field[4:5] = ArrayB`
  (`ArrayB`は2×2の2次元配列でなければなりません)

レコードの配列を代入の左辺で暗黙に使用することはできません。Recがレコードの配列の場合、Rec.A=2はエラーになり、Rec[*].A=2という完全に明示的なシンタックスを使ってくださいというメッセージが出ます。Recがスカラ・レコードでBが配列の場合、Rec.B=2に対しても同様のエラーが生じます。この場合、Rec.B[*]=2と明示的に指定することを求められます。

注記

VEE 4.0以降のバージョンでは、Set ValuesオブジェクトとSet Fieldオブジェクトは実際には代入の式を持つFormulaオブジェクトです。これらのオブジェクトの定義は以前のバージョンから変更されています。VEE 3.x以前で作成された既存のプログラムでSet Valuesオブジェクトや Set Fieldオブジェクトを使っている場合、VEE 3実行モードで実行すれば従来の定義に従って動作します。

エラーの回復

Formulaには代入やその他の演算を含む複数の式が含まれるため、エラーは特別な方法で処理されます。代入やその他の演算が実行されてから、あとの式でエラーが発生した場合、前の式の結果は取り消されません。

FormulaにGlobal[2]=24; 2/0という式がある場合を考えます。

最初にGlobal[2]が24に設定され、次に0除算によりエラーが発生します。この場合、Global[2]は前の値に戻されません。

注記

代入演算の使用手順について知るには、**VEEヘルプ**(Help ⇒ Contents)を使います。How Do I ...セクションで、Work with Dataセクションをオープンし、Working with Arraysの下のトピック(To Change Values in an Arrayなど)と、Working with Variablesの下のトピックを参照してください。

グローバル変数とローカル変数の使用

Formulaオブジェクトでの代入では、A=2またはGlobalA[5]=2といった式を使ってグローバル変数やローカル変数の値を変更できます。変数には、未宣言のグローバル、宣言済みグローバルまたはローカル、直接設定ローカルという種類があるので、Formulaオブジェクトは下記の優先順序で変数Aを探します。

1. 入力端子であるローカル変数。これは、入力端子の値(型と形状を含む)を上書きします。

2. 出力端子であるローカル変数。この変数は作成されて出力端子上に置かれます。

3. グローバル変数。変数はあらかじめ存在しなければなりません。変数の値は、型と形状を含めて完全に変更されます。

上記の規則により、例えばtmp=2という式がFormulaにあり、tmpが上記のどの基準も満たさない場合には、エラーが発生します。

代入におけるグローバル変数とローカル変数

グローバルな配列やレコードに値を代入する場合、特別な注意が必要です。これらの変数は宣言されている場合といない場合があるからです。グローバル変数が存在するようになるのは、Set Variableなどのオブジェクトで作成されるか、Declare Variableで宣言されたときです。

未宣言のグローバル変数の型と形状は、代入式で変更できます。これに対して、宣言済みのグローバル変数の場合、代入の右辺はグローバル変数の宣言された型に変換可能でなければなりません。

注記

きわめて重要！ Declare Variableオブジェクトを使ってグローバル配列またはレコード変数を宣言する場合、変数全体をまとめて初期化してからでないと、その一部を変更することはできません。図9-3のプログラムでは、変数GlobalAryが宣言され、初期化されてから、個々の部分がFormulaでの代入によって変更されています。

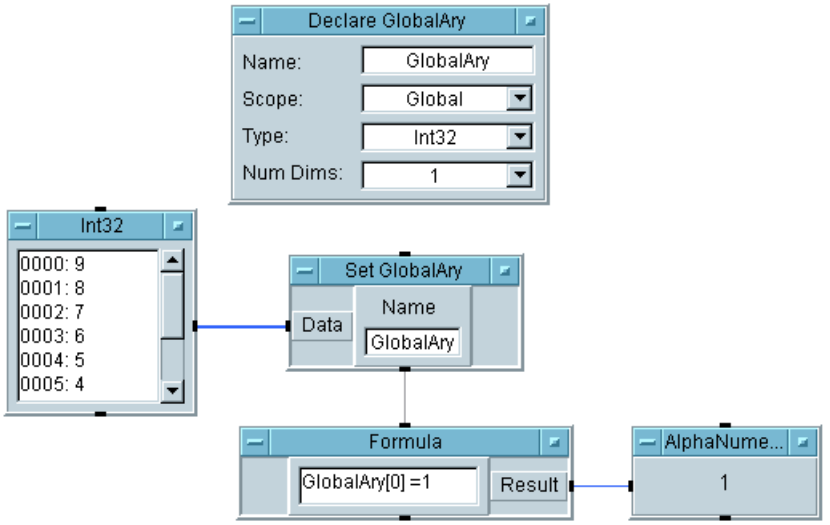


図9-3. 宣言済みグローバル変数の初期化

端子上のデータ・コンテナの内容

Formulaオブジェクトの入出力端子によって、変数の値が代入によってどのように変化するかが影響されます。Result出力端子のデータ・コンテナには、代入演算子の左辺の変更された部分が(変更後のデータ型と形状で)入ります。

式ArrayA[2:4] = ArrayB[5:6]を持つFormulaの場合、Result出力端子にはArrayB[5:6]の値を持つArrayA[2:4]が入ります。ArrayA[2]=4のような式の場合、ArrayAのデータ型がComplexなら、出力端子のArrayA[2]の値は(4,0)というComplexに変換されます。

下記の出力端子名が使用できます。

- Result。これはFormulaのデフォルト出力端子です。これは予約名であり、代入式の結果が入ります。Resultを削除して他の出力端子を使い、必要ならあとで再びResultを追加することもできます。ただし、Result出力端子の名前を変更することはできません。

- 入力端子名。この値は入力からコピーされます。代入で変更された場合、新しい値が出力されます。変数を入力端子名として使用する場合、変数は存在してあらかじめ値を持たなければなりません。

- ローカル変数名。tmp=2などの代入式で作成されるTmpといった名前です。

Formulaでその他の出力ピンを使うことにより、変更された配列やレコードのさまざまな部分をオブジェクトから出力することができます。これにより、変数の変更された部分を1つの端子から出力し、変数全体を別の端子から出力することができます。配列またはレコード全体を出力端子から得るには、代入式でグローバル変数を使うか、その配列またはレコードを出力する端子をFormulaオブジェクトに追加します。値が割り当てられない出力端子があると、エラーが発生します。

コンテナ全体を変更する場合を除いて、配列のマッピングは無視されます。例えば、ArrayA[2:4] = ArrayB[2:4]はArrayAのマッピングを変更しません。これに対して、A = ArrayB[2:4]の場合は、AがArrayBで置き換えられるので、Aのマッピングが変更されます。

2項演算子の使用

2項演算子に関しては、いくつかの付加的な条件と指針があります。2項演算子はFunction & Object BrowserのType: Operatorsで下記のカテゴリに分類されています。

2項演算子のカテゴリ

- Category: Arithmetic(算術演算)
  - a + b
  - a - b
  - a * b
  - a / b
  - a ^ b (累乗)
  - a mod b (剰余、除算の余りを返す)
  - a div b (整数除算 - 余りなし)
- Category: Comparison(比較)
  - a ~= b
  - a == b
  - a != b

- a < b
- a > b
- a <= b
- a >= b

■ Category: Logical(論理演算)

- a AND b
- a OR b
- a XOR b
- NOT a (単項演算子だが、2項演算子と同じ指針に従う)

配列に対して2項演算子を使用する場合、2つの配列のサイズ、形状、マッピング(存在する場合)が一致する必要があります。Coordsの場合、2つのCoordの独立変数の値が一致する必要があります。

2項演算子の優先順位

このリストは、2項演算子の優先順位を示します。順位の高いものが先で、同じ優先順位のものは同じレベルに記載されています。

1. 式をグループ化するための括弧( )
2. ^
3. 単項マイナス -
4. * / MOD DIV
5. + -
6. ~= == != < > <= >=
7. NOT
8. AND
9. OR XOR

2項演算子のデータ型変換

2項演算では、高い方のデータ型に入力値がプロモートされたあと、演算が実行されます。出力のデータ型は入力データ型のうち高いほうになります。例えば、複素数(2, 3)と文字列"Dog"を加算すると("Dog"+(2,3))、結果は文字列"Dog(2, 3)"になります。

注記

この規則には例外が1つあります。Text文字列とInt32を乗算すると、結果は文字列の繰返しになります。例えば、"Hello"*3はHelloHelloHelloを返します。この場合、データ型のプロモーションは発生しません。

データ型の順序(高いものが先):

1. Object
2. Record
3. Text (EnumはTextとして扱う)
4. Spectrum
5. PComplex
6. Complex
7. Coord (他の数値型との変換は不可)
8. Waveform
9. Real64
10. Real32
11. Int32
12. Int16
13. UInt8
14. Variant

Variantデータ型は、特定の種類のデータに固定されません。必要に応じて、他のどんなデータ型にもなることができます。関数sin(var)では、Variantデータ型(var)のデータは整数、実数、波形など、そのとき割り当てられている値の型になります。a=sin(var)という式では、aのデータ型はデータ(var)の内容の型になります。

2項演算子に関する考慮事項

Objectに関する考慮事項. Objectは他の型に自動的に変換されません。Object自体に対しては2項演算はサポートされませんが、ほとんどのObjectにはStringまたはIntegerの値を取るデフォルト・プロパティが存在するため、Objectに対して大部分の演算を実行することができます。

異なる点は、実際に演算が実行されるのがデフォルト・プロパティに対してであることです。Objectの詳細と、VEEでのActiveXオートメーションの使用法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

Recordに関する考慮事項. Recordはすべてのデータ型の中で優先順位が最高ですが、他のデータ型をRecordデータ型に変換するには`Build Record`などの特別なオブジェクトが必要です。Recordが他の型に自動的に変換されることも、他の型が自動的にRecord型に変換されることもありません。ObjectとVariantはRecordのフィールドにはなれません。

2項演算子でレコードと他のデータ型を組み合わせることは可能ですが、この場合は必ずレコードが返されます。レコードとレコードでないデータとの2項演算では、レコードの各フィールドとレコードでないデータとの間で演算が行われます。

例えば、レコードRに2個のフィールドAとBがあり、Aの値がスカラReal値(2.0)、Bの値がスカラComplex値(3,30)であるとします。この場合、R+2という式は、2個のフィールドAとBを持つレコードRを生成し、Aの値はスカラReal値4、Bの値はスカラComplex値(5,30)になります。レコードのどれかのフィールドに対して演算が実行できない場合、エラーが発生します。

レコードと他の型に対して2項演算を行うと、同じ「スキーマ」のレコードが返されます。すなわち、返されたレコードには同じ名前、型、形状のフィールドが同じ数存在します。2項演算でレコード・フィールドの型や形状が変更されることはありません。

例えば、レコードRに2つのフィールドAとBがあり、AがスカラReal、BがスカラComplexだとします。この場合、R+(2,3)という式はエラーになります。VEEは最初に(2,3)をR.Aに加算しようとし、次にR.Bに対して同じことをしようとします。

エラーが発生するのは、R.AフィールドがRealであり、R.A+(2,3)の結果がComplexだからです。ComplexをRealにデモートしてR.Aに書き戻すことはできないのです。

レコードと配列との2項演算では、レコードのほうが配列より優先順位が高いものとして扱われます。例えば、[1,2,3] + [3,4,5]が[4,6,8]を生成するように、配列は要素ごとに演算されます。ところが、レコードは配列より高い優先順位にあります。このため、Rが2つのフィールドAとBを持つレコードの場合、式R + [1,2]は配列[1,2]をRの各フィールドに加算しようとします。1とR.A、2とR.Bが加算されるのではありません。

配列とレコード配列を組み合わせるともっと複雑になります。例えば、Rがレコードの1次元配列で、長さが2、フィールドがA、B、Cの3つだとします。式R + [1,2,3]や式R + [1,2]は、Rの各要素のフィールドA、B、Cに配列全体を加算しようとします。Rが配列であるにもかかわらず、それがレコードであるという事実のほうが重視されるのです。

2個のレコードの2項演算では、対応するフィールド同士が演算されるので、2個のレコードは「スキーマ」が一致しなければなりません。すなわち、フィールドの数が同じで、各フィールドの名前、型、形状が同じ順序で一致しなければなりません。

フィールドAに1、フィールドBに2……を加算したい場合、最も簡単なのは複合代入を使うことです(VEE ヘルプで「代入」を参照)。Formula オブジェクトにR.A=R.A+1,R.B=R.B+2と入力します。こうすれば、新しい値を持つR.AとR.Bをプログラム中で使用できます。

Spectrumに関する考慮事項. dBスケールを使用する場合は注意が必要です。dBスケールのデータは(Waveform (Time)表示以外では)正しく表示されますが、多くの演算関数(fft(x)、ifft(x)など、およびPComplex値を扱うもの)はdBスケールのデータを正しく扱えません。

これらの演算を使用する場合、演算の前にdBスケールのデータを明示的にリニア・スケールに変換してください。VEEのインストール位置(通常は下記の位置)に、dB変換用のライブラリ・プログラムが用意されています。

Windowsの場合:

```
C:\Program Files\Agilent\VEE Pro 6.0\lib\convert
```

UNIXの場合:

```
/opt/veetest/lib/convert (HP-UX 10.20)
```

特定のdB単位を使用する場合、演算関数によっては意味のある結果が得られますが、その意味は使用する単位によって決まります。例えば、dBWスケールのSpectrumに20を加算すると、各要素の振幅に20が加算されます(これは、Spectrumをリニア・スケールに変換し、各要素を100倍して、dBWに戻すのと同じ意味です)。

データ形状に関する考慮事項. 2つのオペランド(入力)が配列である2項演算では、2つの配列のサイズと形状が一致しなければなりません。演算の結果は、入力配列と同じサイズと形状の配列になります。ただし、比較演算子(==、<など)は例外で、常にスカラを返します。配列の次元数が異なっていたり、サイズが違っていたりすると、エラーが返されます。例えば、[1,2] + [1,2,3]はエラーを返します。

スカラと配列を演算する場合、スカラが配列オペランドと同じサイズと形状の定数配列であるかのように扱われます。例えば、2 + [1,2,3]は[2,2,2] + [1,2,3]のように扱われます。結果は[3,4,5]です。

*n*次元の配列をCoordに変換すると、Coordのデータ形状はArray 1Dになり、各Coord要素には*n*+1個のフィールドができます。

Variantに関する考慮事項. 2項演算(+-*/など)と関数の結果はVariantにはなりません。

- 式"b=a"では、"b"がVariantだったとしても、"a"と同じ型になります。
- 式"b=a+2"では、"a"の型に関わらず、"b"がVariantになることはありません。
- 式"b=sin(a)"では、"a"の型に関わらず、"b"がVariantになることはありません。
- 関数"func(2+a)"には、"a"の型に関わらず、決してVariantが渡されることはありません。

単なる変数名"a"、単項演算子"-a"、括弧(a)は、"a"のデータ型を変えません。

- 関数"func(a)"には、"a"がVariantならVariantが渡されます。

VEEでの配列操作

VEEは配列操作に最適化されています。従来のループ構文で配列操作を実行することもできますが、一般にプログラム速度は低下します。このセクションでは、Formulaや他のオブジェクトを使って配列に対する演算を実行する方法を説明します。317ページ「代入演算の使用」で説明した代入演算子も、配列の一部の値を変更するために使用できます。

注記

このセクションの例にならい、代入演算子を使用することで、時間のかかる計算ループを避けることができます。このような技法はわかりにくい場合があるので、プログラムの詳細なドキュメントを作るようにしてください。

配列操作の技法

このセクションでは、VEEでの配列操作の技法について説明します。ここでは、配列操作技法の比較、式での配列アクセス、配列操作の実行、式での変数の使用について扱います。

配列操作技法の比較

図9-4と図9-5のプログラム・セグメントは、0～360度の範囲のサイン値とコサイン値を1度刻みで記録した配列を生成する技法を比較したものです。これらの技法を実際に試してみれば、最初のほうが時間がかかることがわかるはずです。

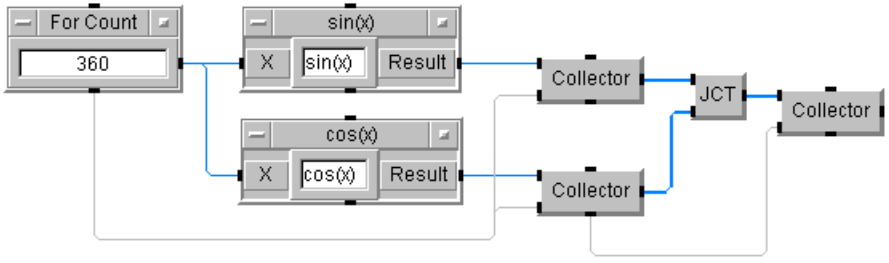


図9-4. 個々のオブジェクトを使って配列を生成

このオブジェクト群が実現しているロジックを演算式に変換すると、図9-5のFormulaオブジェクトに示された式になります。この技法は同じ計算を短い時間で実現します。

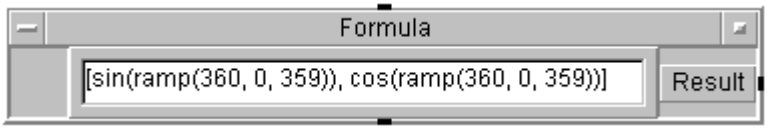


図9-5. 演算式を使って配列を生成

この技法は前のものよりはるかに高速ですが、同じ計算をしていることが必ずしも明白ではありません。この式の動作を説明すると次のようになります。

1. ramp()関数が、0から359までの360個の値からなる配列を生成します。

2. 2つのramp関数は同じ配列を生成し、それぞれがsin()とcos()の三角関数に渡されます。VEE関数のうち、通常スカラ値を受け入れるもののほとんどが、配列をパラメータとして受け取り、返すことができます。

3. Formulaオブジェクトには2つの式が含まれますが、それらの出力は角括弧に囲まれているので配列フォーマットに変換されます。

この技法がすべてのプログラムで最善というわけではありません。サインとコサインの演算はramp()関数から生成された配列全体に対して行われるのであって、個々の値が生成されるたびに演算が行われるわけではありません。個々の値が生成されるたびに演算を行う必要がある場合、配列全体を処理する式を使わず、ループ構文を使います。

式での配列アクセス

式の中では配列をスカラと同様に使用できます。配列は名前で参照します。配列定数は式の中に直接指定できます([1,2,3]など)。VEEでは配列要素の間にカンマが必要です。

注記

配列のインデックスは0起源です。インデックスは0から始まり、*n*-1で終わります。ここで、*n*はその次元の要素数を表します。

式を使って配列の一部にアクセスできます。式でサブ配列を指定したら、そのサブ配列を出力することも、計算に使用することもできます。

アクセスするサブ配列は、1つの配列の中の連続した部分に限ります。サブ配列にアクセスするには、配列のすべての次元に対してパラメータを指定します。配列パラメータの指定には下記の文字を使用します。

- カンマ","は配列の次元を区切ります。サブ配列を操作するときは、すべての配列次元に対して1つずつの指定が必要です。
- コロン":"は配列の1つの次元の要素の範囲を指定します。
- アステリスク "*" はワイルドカードであり、その配列次元のすべての要素を指定します。

注記

波形の時間スパン、スペクトラムの周波数スパン、配列のマッピングは、サブ配列のポイント数に基づいて調整されます。例えば、256ポイントの波形(`WF`)に対して`WF[0:127]`を要求すると、波形の最初の半分が得られ、時間スパンは元の半分になります。

例: 配列から返される値

下記の式は、`A`が10要素の1次元配列(Array 1D)の場合に返される値を示します。

- `A[1]`は`A`の2番目の要素にアクセスし、スカラを返します。
- `A[0:5]`は`A`の最初の6要素を含む1次元のサブ配列を返します。
- `A[1:1]`は、`A`の2番目の要素だけを含む1要素の1次元サブ配列を返します。これと最初の例(`A[1]`)との違いに注意してください。
- `A[2:*]`は、`A`の3～10番目の要素を含む1次元サブ配列を返します。
- `A`または`A[*]`は配列`A`全体を返します。

- A[1,2]は2次元配列に対するパラメータ指定なので、エラーを返します。

Bは5×5の行列(Array 2D)とします。

- B[*]はエラーを返します。2次元配列に対して1個のパラメータしか指定されていないからです。
- B[1,2]は2行目の3番目の要素のスカラ値を返します。
- B[1,*]は、1行目全体をArray 1Dとして返します。
- B[1,1:*]は、1行目の最初の要素以外をArray 1Dとして返します。
- B[4,1:4]は、4行目の2～5番目の要素からなる4要素のArray 1Dを返します。
- B[5,5]はエラーを返します。配列は0起源なので、アクセス可能な要素はB[4,4]までです。
- B[1 1]は次元の間にカンマがないのでエラーを返します。

式での配列の作成

他の配列やサブ配列の要素から配列を作成できます。式の各要素は、同じ数の次元を持ち、各次元に同じ数の値を持たなければなりません。

例えば下記の式は、Aが10要素の1次元配列(Array 1D)でBが5×5の行列の場合に返される値を示します。

- [1,2,3]は、値1、2、3を持つ3要素のReal Array 1Dを返します。
- [A[0], A[5:7], A[9]]はエラーになります。スカラの要素とArray 1Dの要素が混在しているからです。
- [A[0:4], B[0,*]]は、Aの最初の5要素を1行目とし、Bの1行目を2行目とする10要素のArray 2D(サイズ2×5)を返します。

- `[A[0], A[1], B[2,3], A[5]]`は、Aの最初と2番目の要素、Bの3行4列の要素、Aの6番目の要素を含む4要素のArray 1Dを返します。

配列操作の実行

配列操作に関しては、いくつかの簡単な付加的規則があります。配列に対して基本スカラ算術演算を実行すると、単に配列の各要素に対して演算が実行されます。

- A*2は配列の各要素を2倍します。
- A-4は配列の各要素から4を引きます。
- `sin([1,2,3])`のように関数で配列を使用すると、`sin`関数が配列の各要素に適用されます。

サイズと次元が等しい2個の配列に対して演算を実行すると、配列の対応する要素の間で演算が行われます。

- A*Bは、配列Aの各要素と配列Bの対応する要素とを乗算します。これは行列の乗算とは異なります。行列の乗算は、行と列をかけて結果を合計し、スカラを生成するというもっと複雑な演算です。VEEでこれを行うには、組込みの行列関数`matMultiply(A,B)`を使います。

基本的な配列操作

VEEには配列のアクセスや操作を柔軟に実行できる方式が用意されています。配列の一部を取り出したり、単純な配列操作を実行したりする方法については、314ページ「式での変数の使用」と332ページ「配列操作の実行」を参照してください。

配列関数の動作

VEEの基本的な演算関数のうち、`log()`、`sin()`、`cos()`などほとんどのものは、配列をパラメータとして受け取り、配列を返すことができます。配列の演算と操作に便利な専用の関数もいくつか用意されています。配列に対して使用できない関数もあります。配列操作に役立つ関数としては下記のものがあります。

- `ramp()`はループ・カウントの生成に使用します。
- `concat()`は2個の配列を連結し、1次元配列を返します。

- totSize()は配列の全要素数を返します。
- signof()は値の符号を調べます(負なら-1、0なら0、正なら1を返す)。
- abs(x)は絶対値を返します。
- rotate()は配列の要素を回転します。
- sum()は配列の全要素を合計します。
- sort()は配列をソートします。
- randomize()は乱数の配列を生成します。
- min()はデータ・セットの最小値を求めます。
- max()はデータ・セットの最大値を求めます。
- clipUpper()は最大値より下に値をクリッピングします。
- clipLower()は最小値より上に値をクリッピングします。

Formulaオブジェクトの便利な機能として、式を配列として定義できることがあります。配列要素の間にはカンマが必要であることに注意してください。下記の式は、Bという入力の2倍、逆数、2乗、自然対数を含む配列を生成します。

```
[2*B, 1/B, B*B, log(B)]
```

下記の例は、Formulaオブジェクトの式で配列を操作する方法を示します。配列を持つオブジェクトをFormulaオブジェクトの入力ピンに接続します。

配列の値の変更

既存の配列を受け取って、代入式によりその一部の要素に対して演算を実行することができます。図9-6の例は、2次元配列の1つの行の値を変更します。Formulaオブジェクトの式は、入力配列A=[1,2,3,4]に対してこの操作を実行します。ここで、行0には1と2、行1には3と4が含まれます。

この式は2番目の行の要素を4倍し、結果を配列Aに代入します。Result端子は変更された値だけを出力するのに対して、A端子は新しい値が設定された配列全体を出力します。

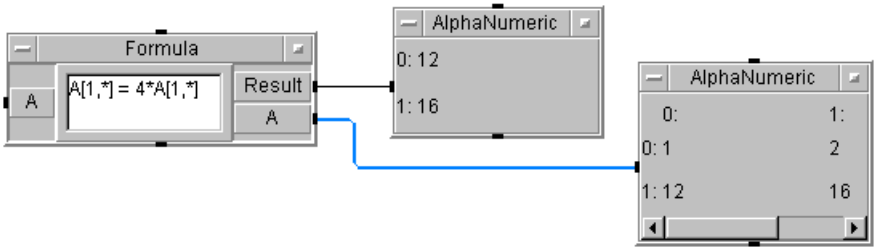


図9-6. 代入式を使って配列の値を変更

大きな配列の分割

2048要素の1個の配列を、128要素×16組として表示できます。ここではデータの生成よりも表示に焦点を当てていますが、配列を分割する方法としても参考になります。

図9-7のプログラムは、目的の表示を実現するために配列を分割する方法を示します。

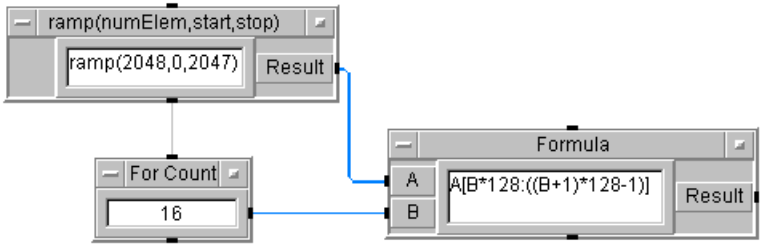


図9-7. 式を使って大きな配列の値を再構成

`ramp()`関数は単に0～2047の値を持つ2048要素の配列をテスト用に生成するだけです。`For Count`オブジェクトは生成する16個の配列をカウントし、`Formula`ボックスはカウントから生成したインデックスによって下記のサブ配列を選択します。

```
A[0:127], A[128:255], A[256:384], ..., A[1920:2047]
```

もちろん、ここでは配列のサイズ、サブ配列の数、サブ配列のサイズは固定されています。これらのいずれかが一致しない場合はエラーが発生します。

配列の結合

次の例は、多次元配列を結合する方法を示します。concat(x,y)関数は1次元の出力しか生成しませんが、Formulaオブジェクトで使えばこの目的に役立ちます。ただし、この方法は行または列の数が固定でないと使えません(実用上はこの制約は満たされるのが普通です)。図9-8のFormulaオブジェクトに示されているのは、2行の配列2個を結合する式です。

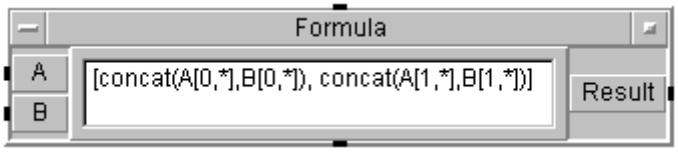


図9-8. 式を使って2個の配列を結合

concat()関数は2個の配列を結合し、1次元の配列を生成します。この式は、配列のそれぞれの行を抽出して結合したあと、再び2次元配列に戻します。結果の配列には2つの行があり、その要素数は元の配列の行の要素数を合計したものになります。

ベクトルと行列の乗算

下記の例は、ベクトル

```
[ X1, X2, X3, X4 ]
```

と行列

```
[[ y11, y12, y13, y14 ],
 [ y21, y22, y23, y24 ],
           ...
 [ y51, y52, y53, y54 ]]
```

とを乗算して、下記の結果を得る方法を示します。

```
[[ X1*y11, X2*y12, X3*y13, X4*y14 ],
 [ X1*y21, X2*y22, X3*y23, X4*y24 ],
                ...
 [ X1*y51, X2*y52, X3*y53, X4*y54 ]]
```

VEEでは、ベクトルのスカラ倍と、行列乗算が実行できます。スカラ乗算は、行列の各要素をスカラ倍することで、元と同じサイズの行列を生成します。行列乗算は、M×N行列とN×M行列との演算で、スカラを生成します。この例の演算は上記のどちらでもありません。

この例の演算は実質的に、行列の各行をベクトルの各要素でスカラ倍するものです。この実装では、配列操作の技法を使っています。下記のベクトルV

```
[ 1, 2, 3, 4 ]
```

および下記の行列Mからなるデータ・セットを考えます。

```
[  [1,    2,    3,    4,    5,    6,    7,    8    ],
   [10,   20,   30,   40,   50,   60,   70,   80   ],
   [100,  200,  300,  400,  500,  600,  700,  800  ],
   [1000, 2000,3000,4000,5000,6000,7000,8000 ]   ]
```

必要な結果は下記の通りです。

```
[  [1,    2,    3,    4,    5,    6,    7,    8    ],
   [20,   40,   60,   80,   100,  120,  140,  160  ],
   [300,  600,  900,  12K,  15K,  18K,  21K,  24K  ],
   [4000, 8000,12K,  16K,  20K,  24K,  28K,  32K  ]   ]
```

図9-9のFormulaオブジェクトの式で乗算が行われます。行列配列は端子Mに、ベクトル配列は端子Vに接続されます。テストの結果によれば、これは同じ配列のスカラ乗算に比べて約50%遅いだけです。

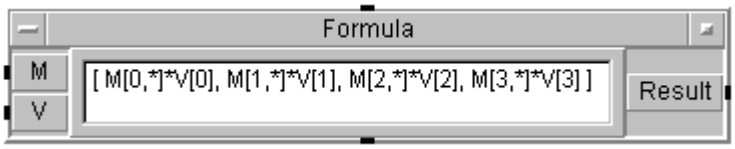


図9-9. ベクトル配列と行列配列の乗算

配列への要素の挿入

図9-10は、既存の配列にデータ要素を挿入する式を示します。入力は下記の通りです。

Aはインデックス値
Bは新規データ
Cは元配列

変更された配列がResultに出力されます。

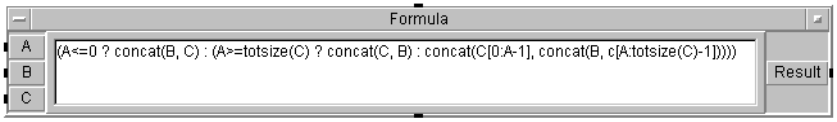


図9-10. 既存の配列に要素を挿入する式

Aのインデックス値は、新規データの開始インデックスを示します。Aが0以下の場合、Bの新規データはCの元配列の先頭に結合されます。

Aが元配列の長さ以上の場合、新規データは元配列の末尾に結合されます。Aがその中間の値の場合、元配列はインデックス値のところで分割され、新規データがその間に挟み込まれます。

図9-11の例は前の例に似ています。これは配列操作によってデータ・キューを実現します。キューとは本質的には固定長の配列で、新しい要素が一端に追加されると、既存の値は1つずつずれ、反対の端から押し出された値は失われます。

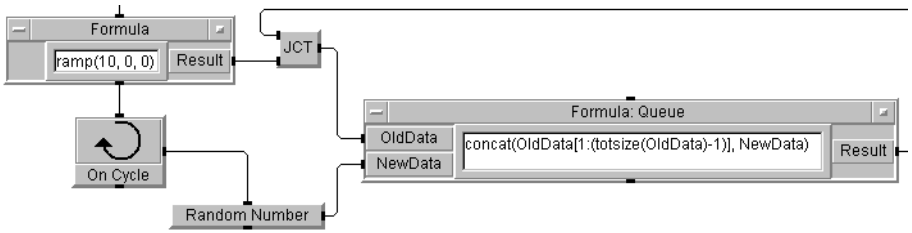


図9-11. ループを使って既存の配列に要素を挿入

ramp()式は、キューの役割を果たす10要素の空の配列を割り当てます。On Cycleループ(サイクルは1秒に設定)が開始されると、1秒ごとにキューの先頭に乱数が送られます。Formula: Queueは、OldData入力の最新の9個の要素を取り出し、NewData入力にある新しい乱数と結合します。Resultの配列出力は次のOldDataとしてフィードバックされ、次のプログラム・セグメントに送られます。次のサイクルでは新しい乱数がNewDataに送られます。

ベクトルから行列への変換

場合によっては、配列関数の代わりにトランザクション・オブジェクトを使って必要な結果を得られることがあります。図9-12の例は、From Stringオブジェクトを使ってベクトル(1次元配列)を行列(2次元配列)に変換します。次のプログラムに示すように、ベクトルをFrom Stringに通し、出力の配列フォーマットを指定します。RealとTextの間のデータ型変換はVEEが自動的に行います。

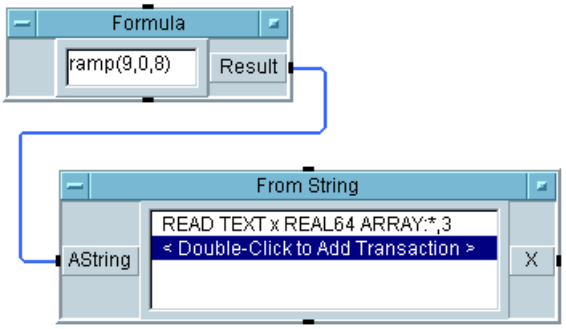


図9-12. 1次元配列を2次元に変換

例えば、9要素の1次元配列を3×3の2次元配列に変換する場合を考えます。

ramp()関数のベクトル出力は下記の通りです。

```
0,1,2,3,4,5,6,7,8
```

From Stringトランザクションは、これを行順序の行列に変換します。

```
0,1,2
3,4,5
6,7,8
```

カラム順序の行列にしたい場合は、transpose()(行列の転置)関数を使って下記の結果を得ます。

```
0,3,6
1,4,7
2,5,8
```

ベクトルを行列に変換するもう1つの方法は、Formulaの配列生成シンタックスを使うことです。シンタックス[1,2,3]は3要素の1次元配列を生成します。同様に、aが10要素の1次元配列の場合、シンタックス[a]は1×10の2次元配列を生成します。ここでも、10×1の行列が必要ならtranspose()を使用します。

高度な配列操作

このセクションでは、高度な配列操作をいくつか紹介します。配列全体のデータの比較などを扱います。

異種の要素を1つの配列に結合

図9-13のプログラムは、デバイスからいくつかのデータ・セットを取得して、それぞれの最大値からなるデータ・セットを生成する方法を示します。

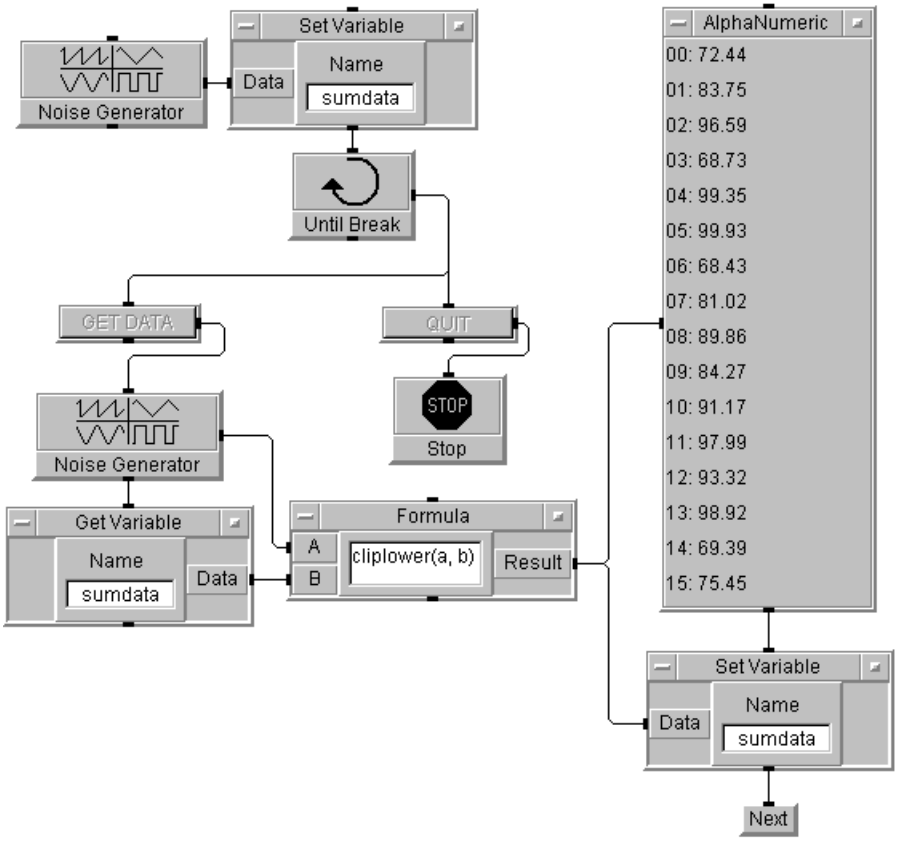


図9-13. 複数の配列の最大値を収集

このプログラムでは、Noise Generatorを使って入力データをシミュレートしています。プログラムは初期データ・セットを生成し、グローバル変数sumdataに格納したあと、ループに入って新しいデータを取得し、最後にプログラムを終了します。

Get Dataボタンを押すと、Noise Generatorから新しい波形が生成され、Get Variableオブジェクトでsumdataのデータが読み出されます。2つの波形はFormulaオブジェクトに渡されて処理され、結果がSet Variableオブジェクトでsumdataに書き込まれます。

Formulaオブジェクトは、新しい配列データをピンAに、sumdataの配列をピンBに受け入れます。式cliplower(a,b)は、A > Bの場合はA、それ以外の場合はBの値を持つ配列を結果として出力します。この代わりにclipUpperを使うと、最小値が得られます。

2個の配列の比較

図9-14の例は、2個の乱数配列を比較し、最初の配列の要素のうちで2番目の配列の要素よりも大きいものがいくつあるかを数えます。値の比較には比較演算子(==、!=、<=など)や3項演算子(A < B ? C : D)が通常用いられますが、この問題にはどちらも使用できません。これらは任意のデータ形状を受け入れますが、結果はスカラだけです。この例では配列の結果を返す比較演算の方法が必要です。

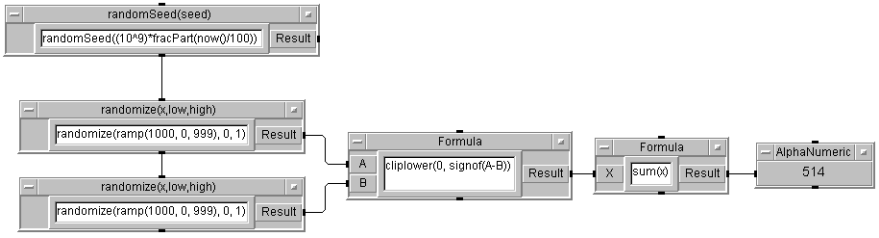


図9-14. 2個の配列の比較

randomSeed()関数は、時間とともに高速に変化するシード(種)を乱数発生器に供給します。これにより、プログラムを実行するたびに異なった結果が得られます。2つのrandomize()関数は、それぞれ0～1の範囲の1000個の乱数からなる配列を生成します。Formulaオブジェクトの中の式は、下記の合計演算を行います。

```
sum(clipLower(0,signof(A-B)))
```

式の各部分の働きを下に示します。

1. A-Bは、A > Bなら正、A == Bなら0、A < Bなら負の値を持つ配列を生成します。

2. signof(A-B)は、正の数を1、負の数を-1に変換し、0は0のままにします。

3. clipLower(0,signof(A-B))は、-1の値を配列から取り除きます。結果として、A > Bなら1、それ以外なら0の配列が得られます。

4. sum()は1を合計し、比較結果がA > Bとなる数を返します。

代替式の使用

前のセクションで示したように、通常の比較演算子の動作を配列に対して他の方法で実現することができます。

- A == B: (1-abs(signof(A-B)))
- A != B: abs(signof(A-B))
- A > B: clipLower(0,signof(A-B))
- A < B: clipLower(0,-signof(A-B))
- A >= B: (1-clipLower(0,-signof(A-B)))
- A <= B: (1-clipLower(0,signof(A-B)))

1と0からなる配列を1から引くと、配列に対するNOT演算を実現できます。配列同士だけでなく配列とスカラの比較にも、同様の方法が使えます。結果の1と0の配列に対して論理演算を行うこともできます。

例えば、A1とA2がともに1と0からなる配列だとすると、下記の論理演算が成り立ちます。

- NOT A1:   1 - A1
- A1 AND A2: A1 * A2
- A1 OR A2: signof(A1 + A2)
- A1 XOR A2: 1 - abs(signof((A1 + A2) - 1))

これらの結果を利用して、乗算により元の値の配列を「マスキング」することができます。0に対応する値は除去され、1に対応する値だけが残ります。

効率的な技法の選択

今までに説明した技法を使った場合、Formulaオブジェクトにたくさんのロジックが詰め込まれ、保守が困難なプログラムになるおそれがあります。複数のオブジェクトをFormulaオブジェクトに置き換えることで、ループをなくしたり減らしたりすることが目標ですが、UserFunctionを使う方法もあります。配列操作の技法をよく理解しておくことで、複雑な形式論理操作を省いてより直接的な解法を実現することができます。

下記の例は、配列操作のためにいくつかの技法が選択できることを示します。I/Oデバイスから受け取った8ビット符号なしデータの配列が、VEEによって32ビットの符号付き整数データに変換された場合に、元の値を復元したいとします。これを1つの式で実現するには、配列のすべての値に256を加算します。

```
(A + (clipLower(-1,clipUpper(0,A))) * (-256))
```

この式は下記の動作を実行します。

1. clipUpper(0,A)は正の値をすべて0に変換します。

2. clipLower(-1,clipUpper(0,A))は、負の値をすべて-1に変換します(入力は実数でなく整数であることに注意)。これにより、負の値に対しては-1、それ以外に対しては0を値とする配列が作成されます。

3. clipLower(-1,clipUpper(0,A)) * (-256)は、上記の配列に-256をかけることにより、負の値に対しては256、それ以外に対しては0を値とする配列を作成します。この配列を元の配列に加算することにより、負の値を元の正の値に嵩上げします。

もう1つの方法として、前に示したように比較演算子を用いることもできますが、この直接的方法よりははるかに複雑になります。

配列計算に非常に役立つオブジェクトとして、Comparatorがあります。これは、特定の基準を満たす配列要素を抽出するものです。例えば、下記のデータ・ストリームの遷移点を求めたいとします。

```
0 0 0 0 1 1 0 0 1 1 0
```

図9-15の解法は、値が0から1、または1から0に遷移する配列要素のインデックスを見つける最も簡単な方法です。

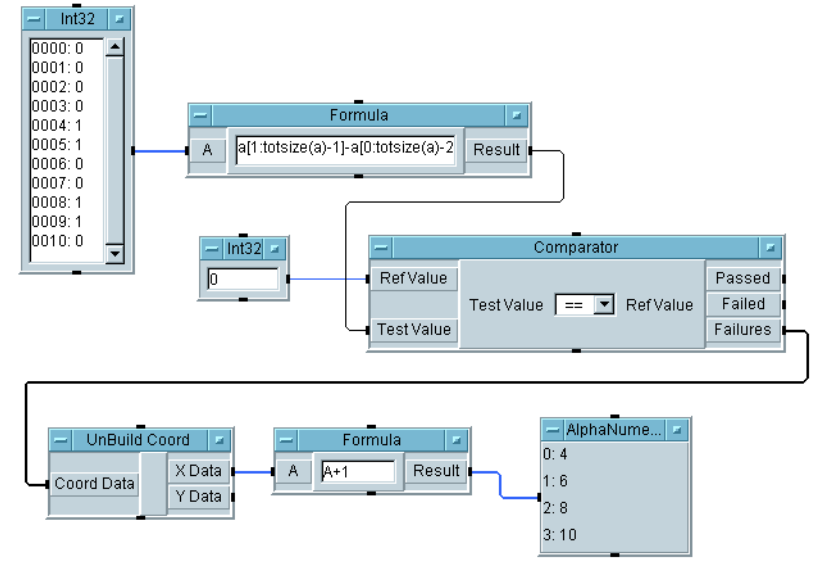


図9-15. 配列の遷移点を見つける

このプログラムにはComparatorが用いられていますが、鍵となるのは下記の式を持つFormulaオブジェクトです。

```
A[1:(totSize(A)-1)] - A[0:(totSize(A)-2)]
```

この式の動作を理解するため、データ・ストリームに配列インデックスを添えてみます。

```
B0:0 1:0 2:0 3:0 4:1 5:1 6:0 7:0 8:1 9:1 10:0
```

遷移が起きる位置の配列インデックスにはマークが付いています。上記の式は、入力配列をインデックス1つ分後ろにずらしたものを入力配列自体から減算することにより、下記の新しい配列を生成します。

```
0:0  1:0  2:0  3:0  4:1  5:1  6:0  7:0  8:1  9:1  10:0
  -  0:0  1:0  2:0  3:0  4:1  5:1  6:0  7:0  8:1   9:1  10:0
     0:0  1:0  2:0  3:1  4:0 5:-1  6:0  7:1  8:0  9:-1
```

ComparatorはResult配列をチェックして、0に等しい要素を調べます。テストに通らなかった配列要素が、求める答えであり、該当位置のインデックスと値からなるX-Y座標の配列としてFailures端子に返されます。インデックス値だけを得るため、Unbuild Coordオブジェクトでxとyの値を分離します。得られたxのインデックス値に1を足して、減算の際にずらした分を補正します。

減算で得られたデータは、遷移のインデックスだけでなく、その方向も示しています。1は正の遷移、-1は負の遷移を表します。この操作は、差分方程式を使って微分を行う方法と基本的に同じです。

10

変数

変数

この章では、VEEの変数について説明します。下記の内容があります。

- 変数について
- 変数の使用

注記

ActiveXオートメーション・オブジェクトおよびコントロールと変数を組み合わせて使用する方法については、第13章「ActiveXオートメーション・オブジェクトおよびコントロールの使用」を参照してください。

変数について

VEEの変数には、未宣言と宣言済みの2種類があります。どちらの変数にも、波形やレコードなどの複合データ型を含む任意のデータ型を格納できます。また、スカラと配列の任意のデータ形状が使用できます。

未宣言変数について

未宣言変数は簡単に使用できますが、動作が遅く、スコープ(このあとの「宣言済み変数について」で説明)も使用できません。未宣言変数には下記のものがあります。

- **グローバル変数は、プログラムの任意の場所で使用できます。**これらはSet Variableオブジェクトで作成されます。Delete Variables at PreRunプロパティがセットされていれば、グローバル変数はプログラムの実行前に削除されます。グローバル変数にGet Variableオブジェクトでアクセスしたり、グローバル変数を式で使用したりするためには、あらかじめ変数が作成されている必要があります。作成されていないと、エラーが発生します。

  未宣言のグローバル変数が役立つのは、値のデータ型や形状が前もってわからない場合、あるいは型や形状が変化する場合です。スコープ付きの変数(ローカル変数)が必要な場合は、宣言済み変数を使います(348ページ「宣言済み変数について」を参照)。

- **テンポラリ変数は、*Formula*オブジェクトでのみ用いられます。**テンポラリ変数(tmpなど)を作成するには、Formulaに出力端子を追加します。例えば、Formulaオブジェクトの端子aとbに入力された値を入れ替えるには、テンポラリ変数tmpを使います。式はtmp=a, a=b, b=tmpのようになります。テンポラリ変数の詳細については、**VEEオンライン・ヘルプ**のReference ⇒ Math Functions and OperatorsにあるAssignmentを参照してください。

- 端子名は、オブジェクト(トランザクション・オブジェクトやFormulaオブジェクト)の内部で変数として用いられます。

宣言済み変数について

宣言済み変数は、使用前に定義されます。宣言済み変数にはスコープが使えるという利点があり、データ型と形状が実行以前に知られているため高速に動作します。ただし、宣言済みの変数に宣言で指定した以外のデータ型や形状を設定しようとすると、エラーが発生します。

変数を宣言するには、Data ⇒ Variable ⇒ Declare Variableオブジェクトを使用します。コンテキスト内にこのオブジェクトを配置すると、他のオブジェクトが実行される前に変数が宣言されます。宣言された変数は、Set VariableまたはFormulaオブジェクトで設定されるまで値を持ちません。

宣言済み変数のスコープは、Declare Variableオブジェクトで指定します。スコープには下記の種類があります。

- Global - 変数はプログラムの任意の場所で使用できます。
- Local to Context - 変数は、1個のUserObjectまたはUserFunction、またはMainの中でのみ使用できます。この変数が使用できるのは、Declare Variableオブジェクトが存在するコンテキストと、その中にネストされたUserObjectの中だけです。そのコンテキストから呼び出されるUserFunctionの中では使用できません。
- Local to Library - 変数は、Declare Variableオブジェクトが存在するUserFunctionライブラリの中でのみ使用できます。Declare Variableはどれかのユーザファンクションの中になければなりません。

同じ名前とスコープを持つ変数を複数定義することはできません。定義しようとするとエラーが発生します。

変数の名前について

変数の名前としては、任意の有効な変数名が使用できます。最初の文字は英字でなければなりません。2文字目以降には、英字、数字、下線文字のどれかが使用できます。変数名の大文字小文字は区別されません(大文字と小文字は同じものと見なされます)。スペースなどの特殊文字は使用できません。

変数の値を読み取るには、変数を宣言または設定したときに指定した名前を使います。

Default PreferencesでExecution ModeがVEE 5モード以上に設定されている場合、一部の名前は一意でなければなりません。VEE 5モードでの変数名の使用法については、17ページ「VEE実行モードの使用」を参照してください。Execution ModeがVEE 4またはVEE 3モードに設定されている場合、他の変数と同じ名前の変数を作成すると、優先度の問題が生じます。優先順位は下記の通りです(優先順位が高いものが上)。

1. 入力端子名(Formulaオブジェクトやトランザクション・オブジェクト)
2. テンポラリ変数(Formulaオブジェクト)
3. Local to Contextと宣言された変数
4. Local to Libraryと宣言された変数
5. 宣言済みグローバル変数
6. 未宣言グローバル変数

1つのオブジェクトに同じ名前の変数が2つある場合、衝突が生じます。この場合、優先順位の高い方が用いられます。

変数の使用

このセクションでは、VEEでの変数の使用に関する指針を記します。初期値の設定、変数値へのアクセス、変数の削除、ライブラリでの変数の使用などの内容があります。

初期値の設定

変数にアクセスするには、あらかじめ初期値を設定しておく必要があります。初期値が設定されていないと、エラーが発生します。図10-1に変数の例を示します。

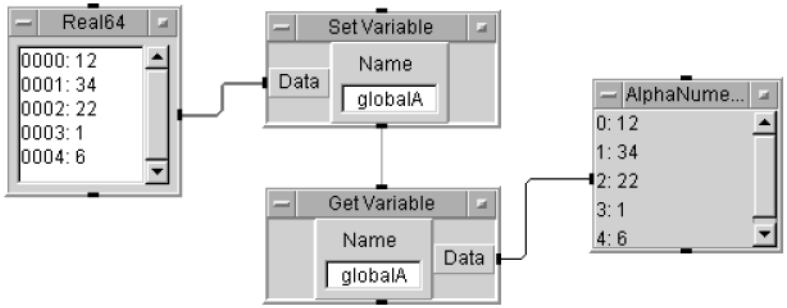


図10-1. 変数の例

グローバル変数の値をGet Variableで読み取る前に、Set Variableで設定しておく必要があります。このためには、Set Variableオブジェクトのシーケンス出力ピンをGet Variableオブジェクトのシーケンス入力ピンに接続します。これをしないと、存在しないグローバル変数にGet Variableがアクセスしようとしてエラーが発生するおそれがあります。

Delete Variables at PreRunプロパティが設定されていない場合、エラーが発生せず、代わりに古い値が読み取られる可能性があります。

宣言済み変数は作成された時点では初期化されていないので、Get Variableオブジェクトでアクセスしたり式で使用したりするためには値を設定しておく必要があります。そうしないと、プログラムでエラーが発生します。値を設定するには、Set Variableオブジェクトまたは Formulaオブジェクトを使います。

変数が配列またはレコードの場合、Formulaオブジェクトでどれかの要素にアクセスするには、配列またはレコード全体の値を設定しておく必要があります。図10-2の例では、Formulaオブジェクトで値を初期化する2つの方法を示します。

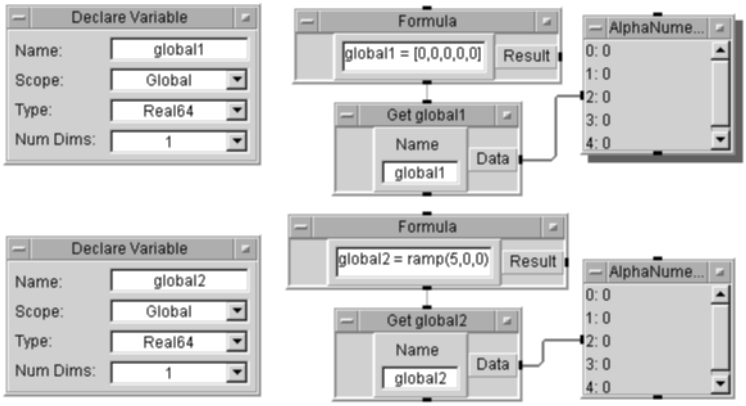


図10-2. 配列の値の設定

変数値へのアクセス

変数に名前を付けたら、必要なだけ何回でもプログラムからその値にアクセスできます。変数の値を読み取るにはいくつかの方法があります。図10-3の例では、グローバル変数globalAに記憶された値を、1回目はGet Variableオブジェクトで、2回目はFormulaオブジェクトの式にglobalAという名前を指定することで、3回目はTo FileオブジェクトのトランザクションにglobalAという名前を指定することで読み取っています。

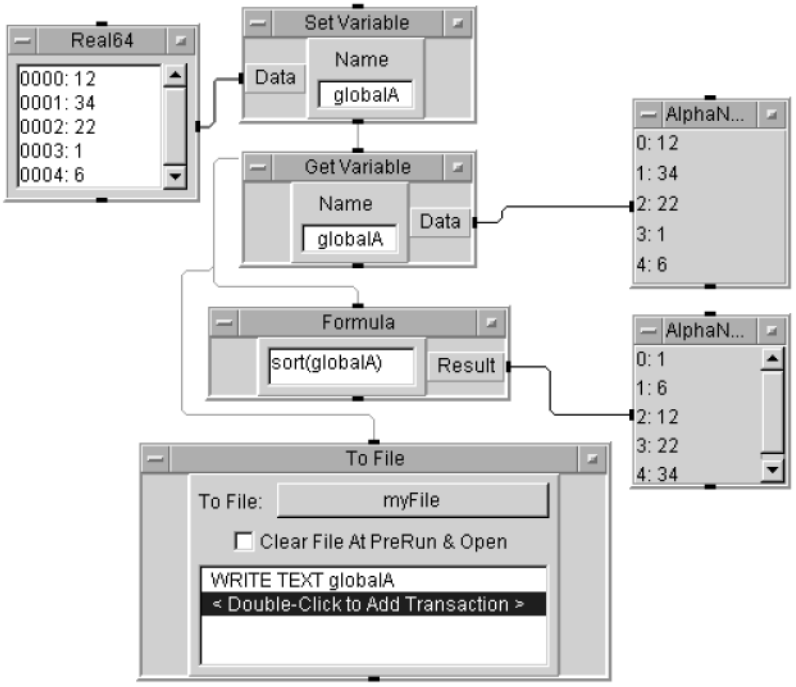


図10-3. 変数にアクセスするさまざまな方法

注記

グローバル変数の名前は、Formula オブジェクトの任意の式や、実行時に評価される他の任意の式で使用できます。

変数の削除

メモリを効率的に使うには、変数が不要になったときにDelete Variableオブジェクトを使ってメモリ空間を解放します。未宣言の変数を削除すると、値と定義の両方が削除されます。宣言済みの変数を削除すると、値が未初期化状態にリセットされ、定義はそのまま残ります。

Delete VariableをBy Nameに設定すると、指定した名前の最も近い変数が削除されます。最も近い変数は優先順位によって決まります。

Delete VariableをAllに設定すると、すべてのスコープにある宣言済みおよび未宣言のすべての変数が対象になります。インポートされたライブラリにある変数も含まれます。すでに説明したように、宣言済みの変数は未初期化状態になり、未宣言の変数は削除されます。

すべての変数を削除しても、すべてのメモリが解放されたり、すべてのActiveXオートメーション・ポインタが解放されたりするわけではありません。詳細については、423ページ「オートメーション・オブジェクトの削除」を参照してください。

プログラムが実行される前にすべての変数が削除されるようにするには、File ⇒ Default Preferencesを選択し、Delete Variables at PreRunチェック・ボックスをクリックします。このチェック・ボックスが選択されていないと、すべての変数の値が保持され、変数の宣言によっても値は再初期化されません。

ライブラリでの変数の使用

ライブラリをインポートしたときにはUserFunctionだけがロードされるので、Declare Variableオブジェクトを使用する場合は、ライブラリのメイン・ウィンドウでなく、UserFunctionのどれかに入れておく必要があります。

Globalスコープの変数は、ローカル・プログラムだけから使用できます。リモート関数を呼び出していても、その中からは使用できません。

ライブラリがインポートされると、インポートされたUserFunctionで(Declare Variableオブジェクトによって)宣言されたすべての変数が、指定されたスコープにその時点で定義されます。例えば、変数のスコープがGlobalと指定されている場合、ライブラリが削除されるまでプログラムの任意の部分からアクセスできるようになります。ライブラリが削除されると、そのUserFunctionで宣言された変数もすべて削除されます。

11

レコードとデータセットの使用

レコードとデータセットの使用

この章では、Recordデータ型とデータセットの2つの概念を紹介します。データセットとは、Recordコンテナの集合をファイルに保存してあとで読み取れるようにしたものです。本章の内容は下記の通りです。

- レコードの使用
- データセットの使用

レコードの使用

このセクションでは、オブジェクトによるレコードの作成と操作に関する指針を記します。レコード・コンテナの理解、レコードへのアクセス、プログラムによるレコードの作成、レコード・フィールドの編集などの内容があります。

レコード・コンテナの理解

レコードの作成と操作のために、Record、Build Record、UnBuild Record、Merge Record、SubRecord、Set Field、Get FieldなどのVEEオブジェクトが用意されています。これらのオブジェクトはすべてDataメニューに存在します。

Recordデータ型のコンテナには、データを表す名前付きのフィールドがあります。1つのレコードに存在する名前付きフィールドの数に制限はありません。フィールドの値は、他のレコード、スカラ、配列のどれでもかまいません。

Recordオブジェクトを使うと、各フィールドの値を手動で入力することによりレコードを作成できます。Recordオブジェクトのオブジェクト・メニューからPropertiesダイアログ・ボックスにアクセスし、オブジェクトをスカラ(配列要素 = 0)または配列(配列要素 = 0以外)として構成します。

図11-1のRecordオブジェクトは、4個の配列要素を持つレコード配列として構成されています。このレコードには5個のフィールドがあり、Textフィールド(Name、Address、City)とInt32フィールド(EmplNoおよびZip)からなります。Recordオブジェクトでは、First、Prev、Next、Lastボタンを使って配列要素の間を移動することができます。その途中でフィールドを編集することもできます。

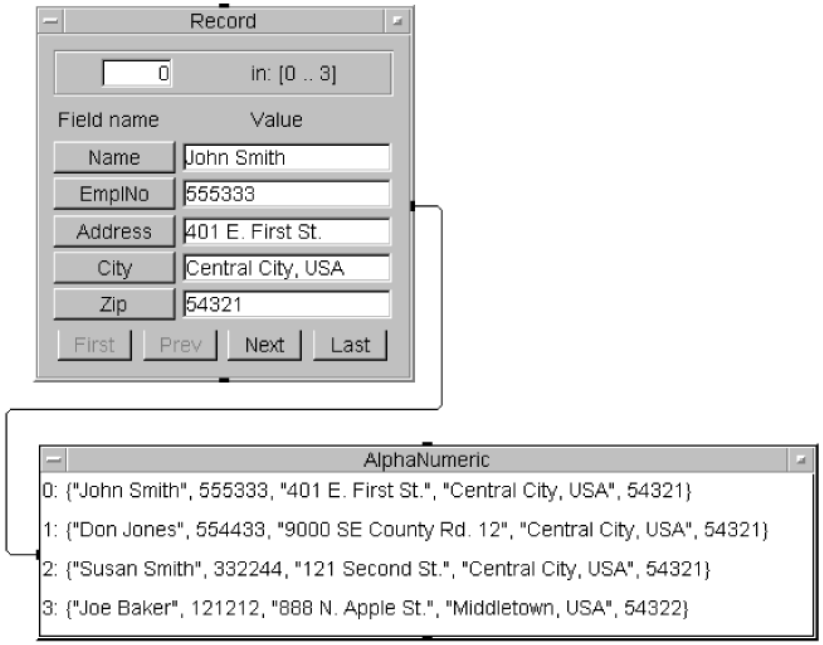


図11-1. 例: レコード・コンテナ

プログラムが実行されると、レコード全体が`Record`出力ピンに出力されます。`AlphaNumeric`表示には、4個の配列要素(0〜3)のそれぞれが5個のレコード・フィールド("`{}`"内)を持つレコードの全体が表示されます。

レコードへのアクセス

図11-2と図11-3のサンプル・プログラムは、レコードにアクセスして個々のフィールドを抽出する1つの方法を示します。

レコードから1つのフィールドを抽出するには、`Get Field`オブジェクトを使います。`Get Field`は`Data ⇒ Access Record`の下にあります。図11-2の例では、`Get Field`オブジェクトを使って`Name`フィールドと`EmplNo`フィールドの全体を抽出しています。`Get Field`オブジェクトは、デフォルトの式として`rec.field`を持つ`Formula`オブジェクトです。

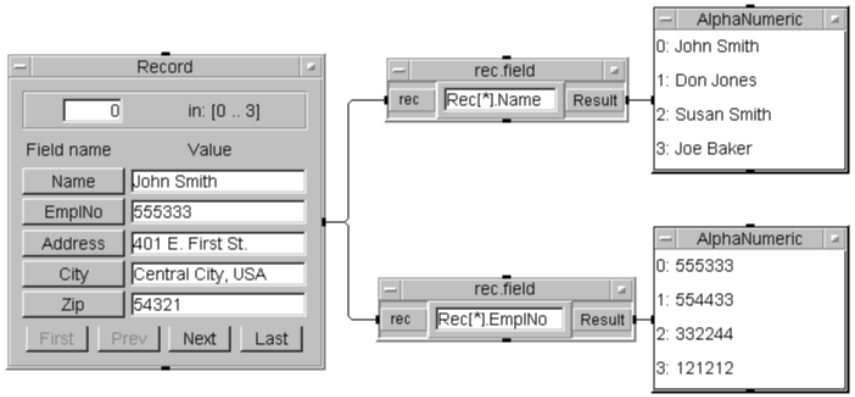


図11-2. Get Fieldによるレコード・フィールドの読取り

個々のフィールドにアクセスするには、ドット・シンタックスを使います。例えば、`Rec[*].Name`や`Rec[*].EmplNo`のようにします。このシンタックスについての説明は、VEEヘルプの`Tell Me About`の下の`Mathematically Processing Data ⇒ General Concepts`にあります。

`Rec[*].Name`は、「`Rec`入力ピンのレコード配列の全要素から`Name`フィールドを取得する」という意味です。このシンタックスは、`Formula`オブジェクトの式または実行時に評価される他の任意の式で使用できます。例えば、`To String`オブジェクトのトランザクションでもこのシンタックスを使用できます。

各フィールドの2 番目の要素(要素1) だけを取得するには、図11-3 のように`Rec[1].Name`および`Rec[1].EmplNo`というシンタックスを使います。

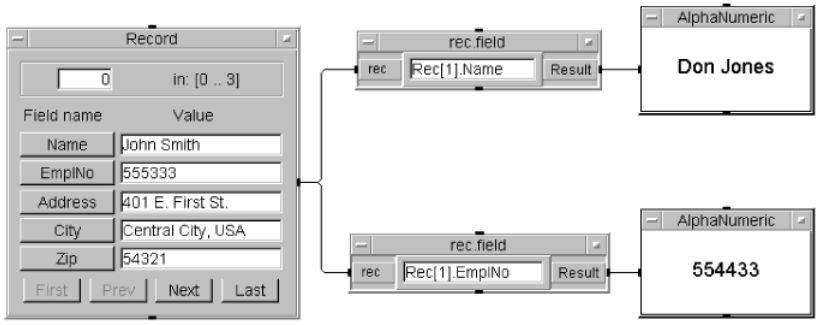


図11-3. Get Fieldでの配列シンタックスの使用

レコードの複数のフィールドの値を取得するには、図11-4のようにUnBuild Recordオブジェクトを使います。

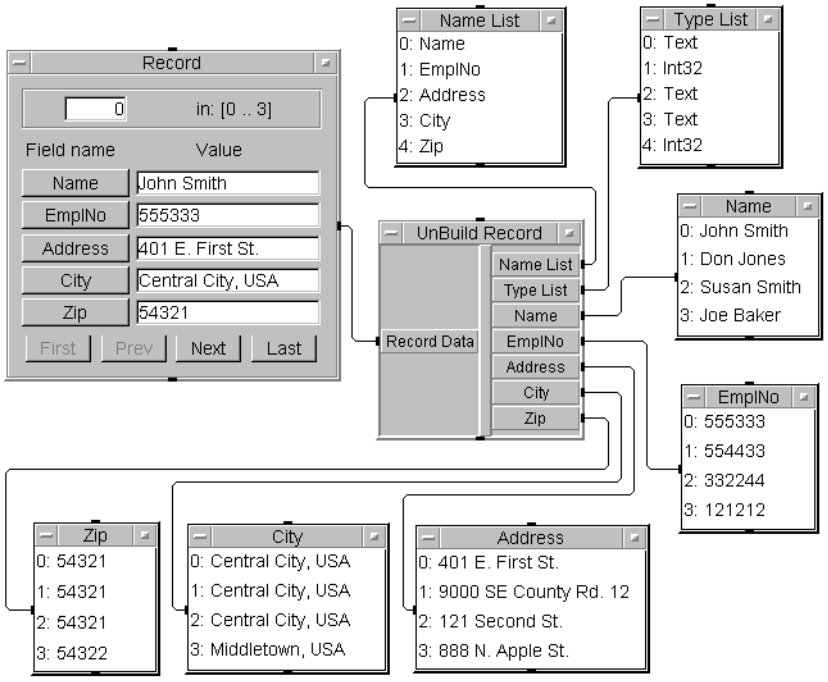


図11-4. UnBuild Recordによるレコード・フィールドの取得

UnBuild Recordオブジェクトには、レコードの各フィールドに対する出力を追加でき、そのほかにName ListとType Listの2つの出力があります。これらの出力は、レコードの各フィールドの名前と型のリストです。

このプログラムは、examplesディレクトリにあるファイルmanual38.veeに保存されています。

注記

Record データ型との間では、データ型の自動変換は行われません。例えば、Recordデータを Real入力端子に送るには、すでに説明したようにUnbuild Recordオブジェクトを使ってレコードのフィールドを抽出するか、Get FieldでRec.Aのシンタックスを使う必要があります。

プログラムによるレコードの作成

Recordオブジェクトは、単純なレコードの作成と編集に便利ですが、大規模なレコードの作成は面倒です。既存のデータからレコードを作成したい場合もあるでしょう。このような場合、Build Recordを使ってレコードを作成します。

Build Recordを使って個々のデータ・コンポーネントからレコードを作成する場合、出力レコード・コンテナのデータ形状を指定する必要があります。Build Recordオブジェクトには、Output ShapeとしてScalarとArray 1Dの2つの選択肢があります。ほとんどの場合、デフォルトのScalarが最適な選択です。

図11-5の例は、2個の入力配列から出力レコードを作成する場合のScalarとArray 1Dの違いを示します。

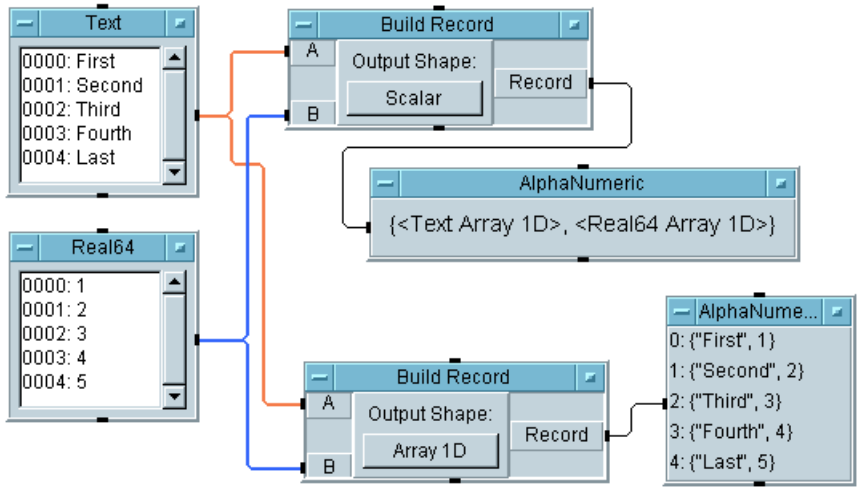


図11-5. Build RecordのOutput Shapeによる違い

図11-5で、Scalarが選択されている場合、出力レコードは2個のフィールドを持つスカラ・レコードで、フィールドのそれぞれが入力配列に対応します。一方、同じ入力データに対してArray 1Dを選択すると、2個の入力配列に要素数が等しいレコード配列が出力されます。出力レコードのデータは要素単位で組み合わされています。

2個の入力配列の要素数が異なる場合、Output ShapeとしてはScalarだけが使用できます。Array 1Dの出力レコードを作成する場合、すべての入力配列が同じ数の要

素を持たないと、エラーが発生します。ただし、図11-6の例のようにスカラと配列の入力データを混在させることはできます。

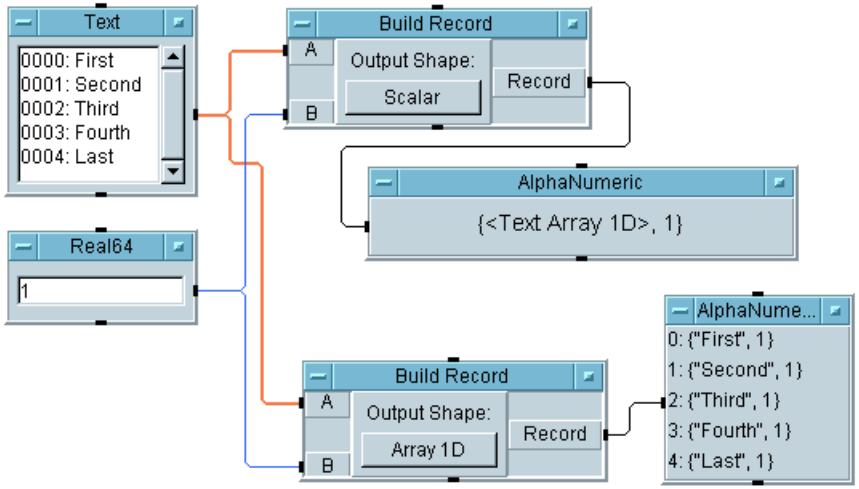


図11-6. スカラと配列の入力データの混在

この場合、`Array 1D`を選択すると、スカラReal値1が出力レコード配列で5回繰り返されます。

レコード・フィールドの編集

`Set Field`オブジェクトを使って、レコードのフィールドを変更できます。`Set Field`オブジェクトは、**左辺式**を**右辺式**に等しく設定する代入ステートメントです。左辺式は変更するフィールド、右辺式は新しい値を指定します。

右辺式が評価され、左辺式で指定されるレコード・フィールドにその値が設定されます。図11-7に例を示します。

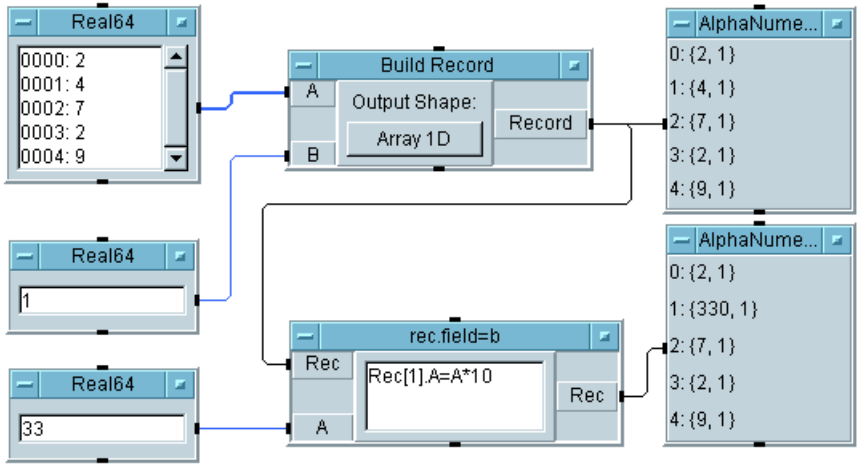


図11-7. Set Fieldによるレコードの編集

この例では、Build Recordを使って5要素のレコード配列を作成しています。Set Fieldオブジェクト(タイトルrec.field = b)は、フィールドRec[1].A(レコード要素1のフィールドA)に値A*10を代入することを示します。

このプログラムには混乱を招く要素があります。左辺式Rec[1].AのAは、レコードのAフィールドを指します。一方、右辺式A*10のAは、Set Fieldオブジェクトの入力の値を指します。この例から、変数やレコード・フィールドに適切な名前を付けることの重要さがわかります。

変数Aの値は33なので、A*10は330と評価され、その値が図11-7のようにRec[1].Aに代入されます。レコードの他の値は変化していないことに注意してください。

Set FieldはFormulaオブジェクトです。詳細については、VEEオンライン・ヘルプのReferenceにあるMath Functions and Operatorsを参照してください。

データセットの使用

VEEのデータ(波形など)は、レコードにまとめて後から読み取ることができます。データセットと呼ばれるファイルにレコードを記録することもできます。データセットにレコードを記録し、読み取るためには、To DataSetオブジェクトとFrom DataSetオブジェクトを使います。これらのオブジェクトはI/Oメニューにあります。

データセットとは、あとから読み取れるようにファイルに保存されたレコード・コンテナの集合です。To DataSetオブジェクトは、レコード・データを入力で収集し、そのデータを名前付きのファイル(データセット)に書き込みます。図11-8に例を示します。

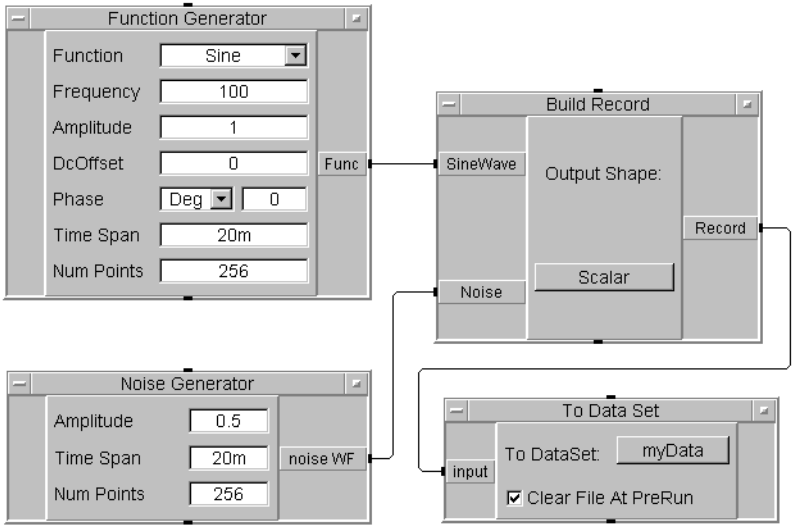


図11-8. To DataSetによるレコードの保存

正弦波とノイズ波の2つの波形がBuild Recordオブジェクトに出力され、レコードが作成されます。このレコードはTo DataSetオブジェクトに出力され、ファイルmyDataに書き込まれます。Clear File at PreRunがチェックされているため、過去にmyDataに記録されたデータはクリアされます。

データがデータセットとして保存されたら、From DataSetを使ってレコードを読み取り、必要なら分解します。図11-9のプログラムにその方法を示します。

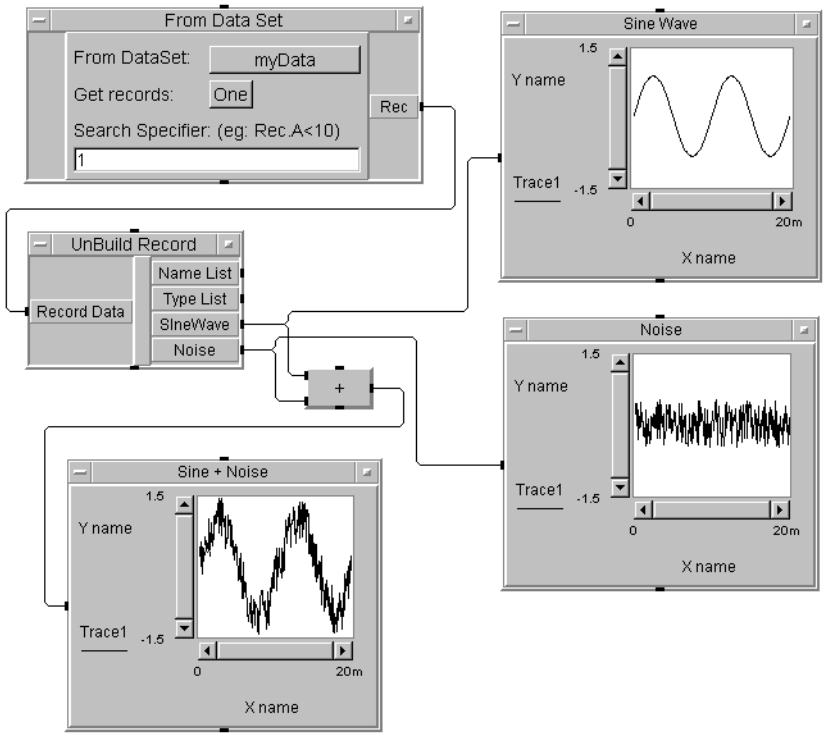


図11-9. From DataSetによるレコードの読取り

From DataSetオブジェクトは、myDataからレコード・データを読取り、Unbuild Recordにデータを出力します。Unbuild Recordは正弦波とノイズ・データの各フィールドを分離します。この例では、正弦波、ノイズ波、2つの波形の和がそれぞれ別のXY Traceオブジェクトに表示されます。

この最後の例の2つのプログラムは、examplesディレクトリにあるファイルmanual40.veeとmanual41.veeに保存されています。

12

ユーザ定義関数/ライブラリ

ユーザ定義関数/ライブラリ

VEEには、プログラムで使用できる19のカテゴリの組込み関数が用意されています。プログラムの必要を満たす組込み関数が見つからない場合、独自の関数を定義することができます。

この章では、UserFunctionによってカスタム関数を作成する方法を説明します。

VEE Proでは下記の3種類のユーザ定義関数がサポートされます。

- UserFunction
- コンパイル関数
- リモート関数

この章では、UserFunction、コンパイル関数、リモート関数について説明します。下記のセクションがあります。

- UserFunctionについて
- 関数ライブラリの使用
- コンパイル関数について
- リモート関数について

UserFunctionについて

UserFunctionは、ユーザ定義関数を作成するための特別な機能です。UserFunctionを作成するには、Deviceメニューで UserFunctionを選択するか、既存のオブジェクトまたは既存のUserObjectをUserFunctionに変換します。このセクションでは、UserFunctionを作成する方法を説明します。次のセクションでは、UserObjectをUserFunctionに変換する方法を示します。

UserFunctionを作成するには、Device ⇒ UserFunctionをクリックします。空のUserFunctionウィンドウがワーク・エリアに表示されます。必要に応じて端子とオブジェクトを追加して関数を作成します。名前は任意のものに変更できます(スペースは使用できません)。その他の詳細については、『VEEユーザーズ・ガイド』または**VEEオンライン・ヘルプ**のHow Do Iを参照してください。

UserFunctionが完成したら、アイコン化するかクローズしておけば、プログラムの他の部分が見やすくなります。UserFunctionを呼び出すには、プログラムでCallオブジェクト(Device ⇒ Call)を使うか、このあとで紹介する他のオブジェクトを使います。UserFunctionをライブラリに保存して、Import Libraryオブジェクトでプログラムにインポートすることもできます。

UserObjectに対するUserFunctionの利点は、1つのUserFunctionをプログラムから何回でも呼び出せることです。このため、UserObjectの複数のインスタンスの代わりに、1つのUserFunctionだけを編集したり管理したりすればよいのです。

VEE 4以上の実行モードでは、Call、Formula、If/Then/Else、Sequencerの各オブジェクト(Functionフィールドからのみ)から呼び出されたUserFunctionはタイムスライス実行されます。

To File、To String、その他類似のオブジェクトから呼び出された場合、あるいはFormulaオブジェクトの式が制御ピンから供給される場合、UserFunctionはタイムスライス実行されません。

UserObjectとUserFunctionの間の変換

UserObjectをUserFunctionに変換するには、UserObjectのオブジェクト・メニューからMake UserFunctionを選択します。UserObjectウィンドウは同じ入出力端子を持つUserFunctionウィンドウに置き換わります。UserObjectオブジェクトはUserFunctionへのCallオブジェクトに置き換わります。

UserFunctionをUserObjectに戻すには、UserFunctionウィンドウのオブジェクト・メニューからMake UserObjectを選択します。UserFunctionへの呼出しは残りますが(削除は手動で行う必要があります)、UserFunctionは自動的にUserObjectに変換されます。

式からのUserFunctionの呼出し

Formulaオブジェクトの式や、ToFileオブジェクトなどの実行時に評価される任意の式からUserFunctionを呼び出すことができます。図12-1のプログラムは、UserFunctionを呼び出すいくつかの方法を示します。

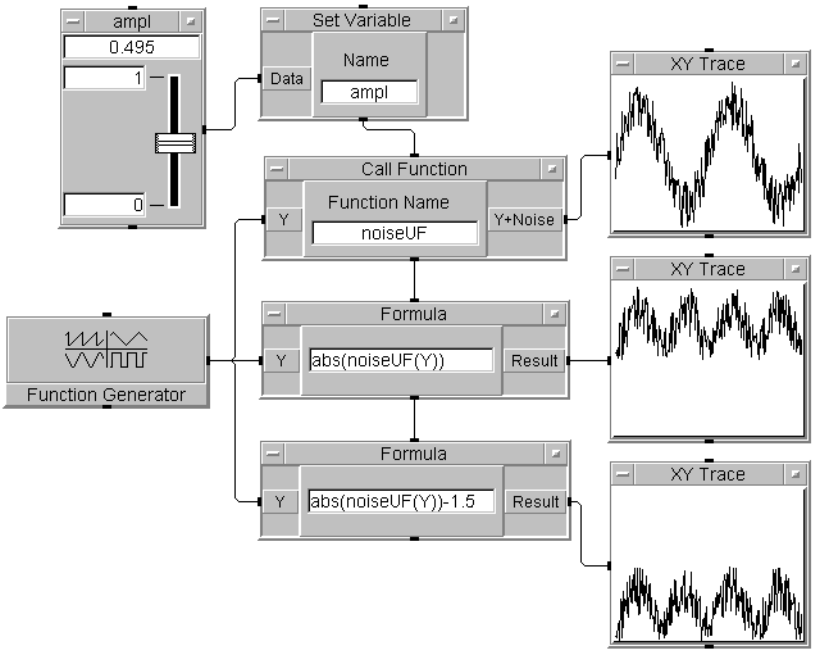


図12-1. 式からのUserFunctionの呼出し

このプログラムで、Callオブジェクトは`noiseUF`というUserFunctionを呼び出し、正弦波にノイズ成分が加わったものを返します。最初のFormulaオブジェクトの式`abs(noiseUF(Y))`は、UserFunctionから返された波形の絶対値を返します。その結果、ノイズの載った正弦波が正方向に整流されて表示されます。

2番目のFormulaオブジェクトの式abs(noiseUF(Y))-1.5も同じUserFunctionを呼び出しますが、波形に負のDCオフセットを加えています。UserFunctionでグローバル変数が用いられるため、シーケンス・ピンを使って正しい伝搬順序を保証しています。

このプログラムは、examplesディレクトリにあるファイルmanual43.veeに保存されています。

関数ライブラリの使用

どのタイプのユーザ定義関数でも、作成してVEEプログラムで使用する方法は似通っています。これらの関数はすべてCallオブジェクトまたはある種の式(Sequencerオブジェクトや Formulaオブジェクトなど)から呼び出されます。3種類のユーザ定義関数はすべてライブラリで使用できます。関数ライブラリを使用するには、下記の手順を実行します。

1. ライブラリをインポートします。

   Device ⇒ Import Libraryオブジェクトを使います。Library Type (UserFunction、Compiled Function、Remote Functionのいずれか)を選択し、必要なフィールドを指定します。個々のフィールドに関しては、この章の関連するセクションで説明します。

2. ライブラリに含まれる関数を呼び出します。

   DeviceメニューのCall、Formula、Sequencerオブジェクトを使用します。実行時に式を呼び出す他のオブジェクト、例えばIf/Then/ElseやTo Fileなども使用できます。関数から複数の値を返す場合は、Callオブジェクトを使う必要があります。

3. ライブラリを削除します。

   メモリを効率的に使いたい場合や、プログラムを高速に実行したい場合は、Device ⇒ Delete Libraryオブジェクトを使ってプログラムでライブラリをメモリから解放します。特にそのような理由がなければ、プログラムで解放しなくてもVEEの終了時にライブラリは自動的に解放されます。

個々の種類のライブラリの使用法については、このあとで説明します。

式からのUserFunctionの呼出しは非常に役立ちます。特に、Sequencerオブジェクトのトランザクションの式で使う場合は便利です。詳細については、第14章「シーケンサ・オブジェクトの使用」を参照してください。

UserFunctionライブラリの作成

ここまでは、作成されたのと同じプログラムで用いられるローカルUserFunctionについて説明してきました。このほかに、複数のUserFunctionのライブラリを外部に作成して、あとでプログラムにインポートすることもできます。

UserFunctionのライブラリを作成するには、空のVEEワーク・エリアでUserFunctionを作成し、ファイルに保存します。例えば、2つのUserFunction、`myRand1`と`myRand2`(入力値に乱数を加算)のライブラリを作成するには、図12-2のように2つのUserFunctionを作成します。

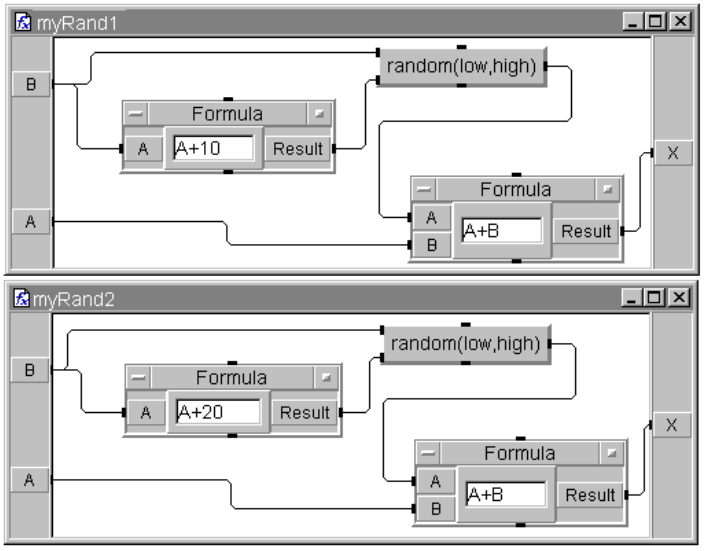


図12-2. ライブラリ用のUserFunctionの作成

UserFunctionライブラリを作成するには、ライブラリを識別する名前でプログラムを保存します。例えば、.veeの代わりに.vlbを拡張子として使います。

注記

プログラムには通常はUserFunctionだけを置きます。他のオブジェクト(`Main`など)がプログラムに存在しても、ライブラリをインポートするときに無視されます。`Declare Variable`オブジェクトを使用する場合、ライブラリのメイン・ウィンドウではなく、どれかのUserFunctionの中に置きます。

UserFunctionのインポートと呼出し

UserFunctionライブラリをプログラムにインポートするには、Import Libraryオブジェクトを使います。図12-3のプログラムは、ファイルuser_func_libからライブラリをインポートし、UserFunction myRand1とmyRand2を呼び出します。

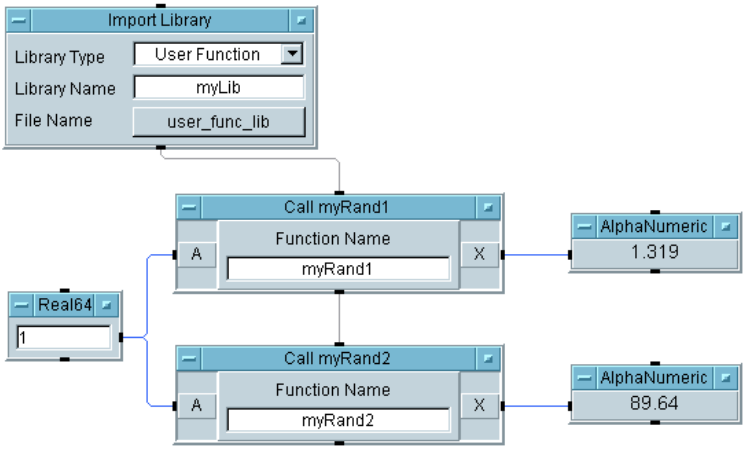


図12-3. UserFunctionライブラリのインポート

Import Libraryオブジェクトでは、ライブラリのタイプをUserFunction、Compiled Function、Remote Functionの中から指定します。UserFunctionを選択した場合、Library NameとFile Nameも指定します。

Library Nameフィールドは、このプログラム内でライブラリに付けるローカル名を指定します。これにより、Delete Libraryオブジェクトを使ってプログラムからライブラリを削除することが可能になります。この例では、ファイルuser_func_libからインポートしたライブラリにmyLibという名前をImport Libraryで割り当てています。

File Nameフィールドは、ライブラリをインポートするために読み取るファイル名(この例ではuser_func_lib)を指定します。File Nameフィールドをクリックすると、すべてのライブラリ・ファイルのリストからファイルを選択できます。

このプログラムは単純なので、使用後にUserFunctionライブラリを削除する必要はありません。大規模なプログラムから大規模なライブラリを呼び出す場合、使い終わっ

たライブラリを削除することにより、プログラムのメモリ要求量を減らすことができます。

注記

`Device ⇒ Import Library`でインポートしたUserFunctionは編集できませんが、内容を表示してデバッグ用のブレークポイントを設定することはできます。インポートしたUserFunctionの内容を表示するには、`Program Explorer`を使います。

`File ⇒ Merge Library`を使ってUserFunctionのライブラリを**マージ**することができます。プログラムにライブラリをマージすると、`Edit ⇒ Edit UserFunction`で個々のUserFunctionを編集できるようになります。

UserFunctionのマージ

UserFunctionをマージすると、UserFunctionはプログラムの一部になります。インポートされたものではないので、必要に応じて編集できます。マージされたUserFunctionはVEEプログラムといっしょに保存され、元のライブラリが変更されても変化しません。

UserFunctionをプログラムにマージするには、`File ⇒ Merge Library`を選択します。ライブラリ・ディレクトリのファイル一覧を表示したダイアログ・ボックスがオープンします。必要なUserFunctionライブラリを含むファイルを選択してOpenをクリックします。

コンパイル関数について

コンパイル関数を作成するには、C、C++、FORTRAN、Pascalなどで作成されたライブラリをVEEプロセスに動的にリンクします。コンパイル関数のライブラリは、UNIXでは共有ライブラリと呼ばれ、Microsoft Windowsではダイナミック・リンク・ライブラリ(DLL)と呼ばれます。

コンパイル関数の作成は、UserFunctionの作成よりもはるかに高度な作業です。Cなどの言語で関数のライブラリを書いたあと、コンパイルしてDLLまたは共有ライブラリを作成します。このほかに、関数呼出しに必要な情報をVEEに伝えるための定義ファイルも作成する必要があります。

コンパイル関数の使用

コンパイル関数を使用するには、下記の手順を実行します。

1. 外部プログラムを作成します。
2. DLL(Windows)または共有ライブラリ(UNIX)と定義ファイルを作成します。
3. ライブラリをインポートし、関数をVEEから呼び出します。
4. 使い終わったライブラリをVEEのメモリから削除します。

注記

Pascalの共有ライブラリは、HP 9000シリーズ700コンピュータでのみサポートされます。

コンパイル関数ライブラリをインポートして関数を呼び出す手順は、UserFunctionライブラリの場合とほとんど同じです。`Import Library`オブジェクトはVEEプロセスにDLLをリンクし、定義ファイルの宣言を解釈します。

定義ファイルには、外部ライブラリとVEEとの間で受け渡されるデータの型が定義されています(このファイルについてはこのあとで説明します)。コンパイル関数は`Call`オブジェクトまたは`Formula`、`If/Then/Else`などのオブジェクトから呼び出すことができます。

コンパイル関数の設計上の考慮事項

コンパイル関数を使うと、時間的要求の厳しいルーチンを他の言語で作成し、VEEプログラムに直接統合できます。また、独自ルーチンのセキュリティを守るためにコンパイル関数を使うこともできます。

コンパイル関数はタイムスライス実行されない(実行終了まで中断されない)ため、VEEの他の機能では実現できない特定の目的に限って使用すべきです。

コンパイル関数を使えばVEEプログラムの機能を拡張できますが、VEEのプロセスが複雑になります。設計にあたっては下記の点を第1に考慮してください。

- 外部ルーチンの役割は、特定の作業に厳密に限定してください。
- コンパイル関数を使うのは、必要な機能や性能がVEEの他の機能では得られない場合に限るようにしてください。代わりの方法として、UserFunctionや、`Execute Program`によるオペレーティング・システムの呼出しを検討します。

リンクするプログラムから使用できるあらゆるオペレーティング・システム機能(演算ルーチン、機器I/Oなど)が使用できますが、リンクする外部関数からVEEの内部関数を使用することはできません。

コンパイル関数を使うとVEEの機能を拡張できますが、いくつかの問題が起きるおそれがあります。主要なものを以下に挙げます。

- 外部ルーチンで発生したエラーは VEE でトラップできません。外部ルーチンはVEEプロセスの一部となるため、外部ルーチンで発生したエラーはすべてVEEにも波及します。外部ルーチンの障害によってVEEがハングしたり、その他の異常が発生したりするおそれもあります。外部ルーチンの役割を明確にし、ルーチン内部でエラー・チェックを行う必要があります。外部ルーチンでプログラムを終了すると、VEEも終了します。
- 外部ルーチンは、必要なメモリをすべて自分で管理する必要があります。ルーチンの実行中に割り当てたメモリは必ず解放してください。

- 外部ルーチンでは、VEEのデータ型変換が利用できません。外部ルーチンで使用できる型と形状のデータだけを受け入れるように、Callオブジェクトの入力端子を設定する必要があります。

- 外部ルーチンが配列を受け入れる場合、参照するデータ型に対する有効なポインタを使用する必要があります。また、処理する配列のサイズをチェックする必要があります。このための最善の方法は、配列自体とは別に配列のサイズをルーチンへの入力としてVEEから渡すことです。渡された配列の値をルーチンで上書きする場合、関数の戻り値を使って配列要素のどれだけが有効なのかを示すようにします。

- システムのI/Oリソースはロックされる可能性があります。タイムアウトなどの管理は外部ルーチンが自分で行う必要があります。

- 外部ルーチンが不正な動作をした場合、例えば配列の末尾を超えてメモリを上書きしたり、ヌル・ポインタや無効なポインタを参照したりした場合、VEEが終了したり、一般保護違反(MS Windows)またはセグメンテーション違反(UNIX)といったエラーが発生したりするおそれがあります。

- 外部ルーチンに配列またはchar*のパラメータがある場合、ルーチンに渡すメモリはVEEで割り当てる必要があります。メモリの割当ては下記の方法で行います。

  - 配列入力の場合、適切な型のAlloc Arrayオブジェクトを使い、サイズを正しく設定します。

  - 文字列入力の場合、Formulaオブジェクトを使います。Formulaオブジェクトのデータ入力端子を削除し、256*"a"のような式を入力します。これにより、256文字のa(+ヌル・バイト)からなる文字列が作成されます。大部分のVXI*plug&play*関数はテキスト・パラメータに256文字より多くは書き込みませんが、念のためテキスト入力を必要とする関数パネルのヘルプをチェックしてください。

コンパイル関数のインポートと呼出し

Import Libraryオブジェクトを使ってVEEプログラムにDLLをインポートしたあと、Callオブジェクトを使ってコンパイル関数を呼び出します。このプロセスは、この章の初めに説明したUserFunctionライブラリのインポートと関数の呼出しによく似ています。

コンパイル関数ライブラリをインポートするには、Library Typeフィールドで Compiled Functionを選択します。

UserFunctionの場合と同様、Library Nameフィールドにはプログラム内でライブラリを識別する名前を指定し、File Nameフィールドにはライブラリをインポートするために読み取るファイル名を指定します。コンパイル関数の場合、4番目のフィールドDefinition Fileに定義ファイルの名前を指定します(図12-4参照)。

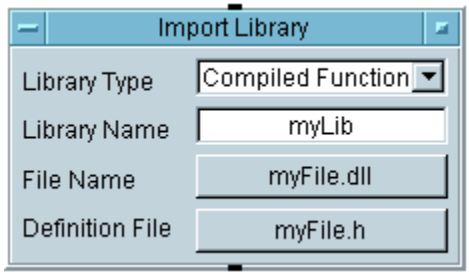


図12-4. コンパイル関数をImport Libraryから使用

定義ファイルには、外部ルーチンとVEEとの間で受け渡されるデータの型が定義されています。これには関数のプロトタイプが記載されています。

Import Libraryでライブラリをインポートしたら、Callオブジェクトに関数名を指定することでコンパイル関数を呼び出すことができます。例えば、図12-5のCallオブジェクトはmyFunctionという名前のコンパイル関数を呼び出しています。

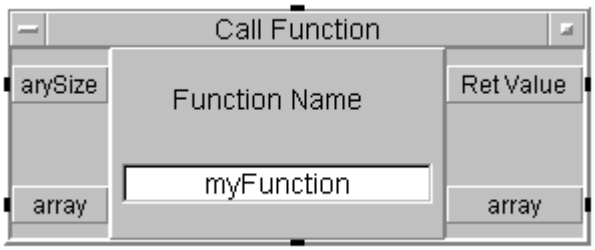


図12-5. コンパイル関数をCallから呼び出す

目的の関数を選択するには、Callオブジェクト・メニューまたはFunction & Object Browser(Device ⇒ Function & Object Browser)でSelect Functionを使うか、Callオブジェクトに名前を入力します。

VEEが関数を認識すると、Callオブジェクトの入出力端子がその関数に合わせて自動的に構成されます(必要な情報は定義ファイルから得られます)。Callの入出力端子の構成を変更するには、オブジェクト・メニューでConfigure Pinoutを選択します。

Callオブジェクトには、関数が必要とする入力端子と、関数の戻り値に対応する出力端子Ret Valueが作成されます。参照渡しの入力については、それぞれに対応する出力端子も作られます。

Formulaオブジェクトの式や、実行時に評価されるその他の式でコンパイル関数の名前を指定して呼び出すこともできます。例えば、SequencerやToFileのトランザクションの式にコンパイル関数の名前を指定して呼び出すことができます。

ただし、式の中で利用できるのはコンパイル関数の戻り値(CallオブジェクトのRet Value)だけです。他のパラメータを関数から取得するには、Callオブジェクトを使う必要があります。

定義ファイル

CallオブジェクトやFormulaの式は、関数に渡すデータの型を定義ファイルの内容に基づいて決定します。定義ファイルには、関数の戻り値のデータ型、関数名、関数の引数が記載されています。データは下記の形式を取ります。

```
<return type> <function name> (<type> <paramname>, <type>
<paramname>, ...) ;
```

ここで:

- <return type>は、int、short、long、float、double、char*、voidのいずれかです。
- <function name>は、先頭が英字で2文字目以降が英数字の文字列で、長さは最大512文字です。
- <type>は、int、short、long、float、double、int*、char*、short*、long*、float*、double*、char**、voidのいずれかです。
- <paramname>は、先頭が英字で2文字目以降が英数字の文字列で、長さは最大512文字です。パラメータ名は指定しなくてもかまいませんが、指定することを推奨します。パラメータを参照渡しにする場合、パラメータ名の前に間接参照記号(*)を付けます。

戻り値の型としては下記のものが使用できます。

- 文字列(char*、VEEのTextデータ型に対応)
- 整数(short、int、long、VEEのInt16およびInt32データ型に対応)
- 単精度および倍精度の浮動小数点実数(floatおよびdouble、VEEのReal32およびReal64データ型に対応)

パラメータの前に*を付けて「参照渡し」を指定すると、情報のアドレスが関数に渡されます。*を付けないで「値渡し」を指定すると、アドレスでなく値自体がコピーされて関数に渡されます。参照渡しは、外部ルーチンでそのデータを変更してVEEに情報を伝えるために使います。**配列はすべて参照渡しです。**

コンパイル関数に参照渡しで渡されるパラメータは、Callオブジェクトに出力端子としても追加されます。出力端子としては、関数の戻り値を出力するRet Valueと、参照渡しの入力パラメータのそれぞれに対応する出力があります。

VEEはスタック上に144バイトをプッシュします。したがって、参照渡しで最大36個のパラメータをコンパイル関数に渡すことができます。値渡しの場合、長整数パラメータなら36個、倍精度浮動小数点パラメータなら18個まで渡すことができます。

注記

HP-UXでは、コンパイル関数から共有ライブラリを作成するのに必要な位置独立コードを生成するために、ANSI Cコンパイラを使用する必要があります。

VEEの定義ファイルでは、「囲み型」のコメントと、「行末まで」のコメントの両方が使用できます。

「囲み型」のコメントは、/*コメント*/という区切りシーケンスを使用します。/*と*/がそれぞれコメントの始まりと終わりを示します。

「行末まで」のコメントは、区切り文字//を使ってコメントの始まりを示します。その行の終わりまでがコメントになります。

C関数の作成

下記のC関数は、実数配列を受け取って、配列の各要素に1を加算します。変更された配列はArray端子からVEEに返され、配列のサイズがRet Value端子に返されます。この関数をVEEにリンクすると、図12-6のようにVEEプログラムから呼び出されるコンパイル関数になります。

```
/*
   C code from manual49.c file
*/

#include <stdlib.h>

#ifdef WIN32
#  define DLLEXPORT __declspec(dllexport)
#else
#  define DLLEXPORT
#endif

/* The description will show up on the Program Explorer when you select
"Show Description" from the object menu and the Function Selection
dialog box in the small window on the bottom of the box.
*/
DLLEXPORT char myFunc_desc[] = "This function adds 1.0 to the array passed in";

DLLEXPORT long myFunc(long arraySize, double *array) {
   long i;

   for (i = 0; i < arraySize; i++, array++) { *array += 1.0; }

   return(arraySize);
}
```

この関数の定義ファイルは下記の通りです。

```
/*
definition file for manual49.c
*/

long myFunc(long arraySize, double *array);
```

(この定義は、Cファイル中のANSI Cプロトタイプ定義と同じです)

ルーチンが依存するヘッダ・ファイルがあれば、インクルードする必要があります。ライブラリから呼び出すシステム関数を解決するために他のシステム・ライブラリが必要なら、それをリンクする必要があります。

サンプル・プログラムではANSI Cの関数プロトタイプが使われています。関数プロトタイプは、VEEが関数に渡すデータの型を宣言します。

配列はポインタ変数として宣言されています。Callのデータ入力端子にある情報のアドレスがこの変数に格納されます。配列サイズは長整数として宣言されています。配列のサイズの値(アドレスではなく)がこの変数に格納されます。データ入力端子と変数宣言の位置は重要です。データ入力ピン(上から下の順番)に与えられたデータ・アイテムのアドレス(または値)は、左から右の順番で関数プロトタイプの変数に渡されます。

C関数の1つの変数(およびそれに対応するCallオブジェクトの1つのデータ入力端子)が、配列のサイズを表すために用いられます。このarraySize変数は、配列の末尾を超えてデータが書き込まれるのを防ぐ働きをします。配列の境界を超えてメモリを上書きした場合、結果は使用する言語に依存します。Pascalでは境界チェックが行われるので、ランタイム・エラーが発生し、VEEは停止します。Cなどの言語では、境界チェックがないので、結果は予測できませんが、可能性が高いのは間欠的なデータ破壊が起きることです。

この例では配列へのポインタを渡しているので、ポインタが指すデータを参照して情報を読み取る必要があります。

arraySize変数は値渡しなので、データ出力端子にはなりません。しかし、ここでは関数の戻り値を使って出力配列のサイズをVEEに返しています。この技法は、入力配列よりも小さいサイズの配列を返す場合に有効です。

図12-6のプログラムは、サンプルCプログラムから作成されたコンパイル関数を呼び出しています。

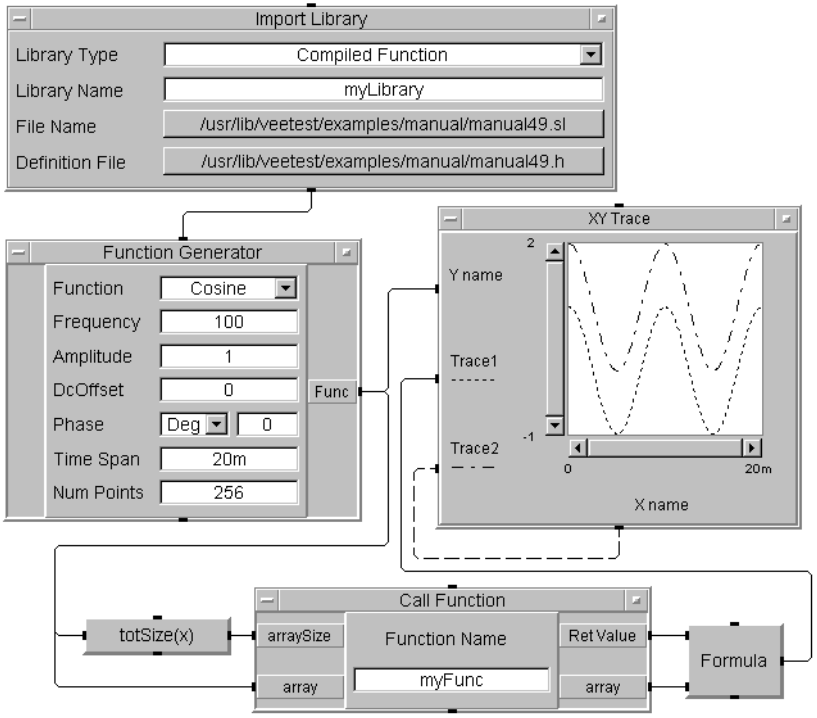


図12-6. コンパイル関数を呼び出すプログラム

図12-6の例は、`examples`ディレクトリにあるファイル`manual49.vee`に保存されています。Cファイルは`manual49.c`、定義ファイルは`manual49.h`、共有ライブラリは`manual49.sl`です。

コンパイル関数の作成(UNIX)

コンパイル関数を作成するには、C、C++、FORTRAN、Pascal(HP 9000シリーズ700のみ)のどれかでプログラムを書き、関数の定義ファイルを作成する必要があります。次にコンパイル関数を含む共有ライブラリを作成し、共有ライブラリをVEEプロセスにバインドします。

共有ライブラリの作成

共有ライブラリを作成するには、関数が位置独立コードとしてコンパイルされている必要があります。すなわち、ルーチンのエントリ・ポイントが絶対アドレスで存在するのではなく、関数名へのシンボル参照がルーチンのシンボル・テーブルに入っている必要があります。

VEE環境に関数がバインドされた時点で、参照される関数の絶対アドレスによってシンボル・テーブルが更新されます。共有ライブラリを作成するための特別なオプションをリンク時に指定する必要があります。

サンプルCルーチンがファイルdLink.cにあるとします。このファイルを位置独立でコンパイルするには、コンパイラ・オプション+zを使います。また、-cオプションを指定して、コンパイラがリンク・フェーズを実行しないようにします。コンパイル・コマンドは下記のようになります。

```
cc -Aa -c +z dLink.c
```

これによりdLink.oという出力ファイルが生成されます。これを下記のコマンドで共有ライブラリにリンクします。

```
ld -b dLink.o
```

-bオプションは、位置独立コードから共有ライブラリを生成する指示です。これにより、a.outという共有ライブラリが生成されます。あるいは、下記のコマンドも使用できます。

```
ld -b -o dLink.sl dLink.o
```

この場合、出力ファイル名(-oオプションで指定)はdLink.slになります。

共有ライブラリのバインド

VEEは共有ライブラリをVEEプロセスにバインドします。必要な手順は、Import Libraryオブジェクトをプログラムに追加し、インポートするライブラリを指定し、(Callオブジェクトなどを使って)関数を名前で呼び出すことです。Import Libraryが実行されると、共有ライブラリがバインドされ、Callオブジェクトに適切な入出力端子が追加されます。

Callオブジェクトのオブジェクト・メニューのコマンド(Configure PinoutおよびSelect Function)を使って、Callオブジェクトを正しく構成します。共有ライブラリは、VEEが終了するか、ライブラリが明示的に削除されるまでVEEにバインドされた状態になります。

ライブラリをVEEから削除するには、Import Libraryオブジェクト・メニューからDelete Libを選択するか、プログラムにDelete Libraryオブジェクトを追加し

ます。1つの共有ライブラリ・ファイルを複数のライブラリ名で参照している場合もあります。この場合、それぞれのライブラリをDelete Libraryオブジェクトで削除します。共有ライブラリは、それを参照している最後のライブラリが削除されるまでバインドされたままになります。

Import Libraryオブジェクト・メニューのDelete Libコマンドを使うと、他のImport Libraryオブジェクトには関わりなく共有ライブラリのバインドが解除されます。

共有ライブラリをバインドすると、コンパイル関数に必要な入出力端子がVEEによって定義されます。Callオブジェクトでコンパイル関数を選択するか、Configure Pinoutを実行すると、適切な端子が自動的にCallに設定されます。アルゴリズムは下記の通りです。

- 関数に渡される入力パラメータ(参照渡しと値渡しの両方)に対してそれぞれ入力端子が作成されます。
- コンパイル関数の戻り値を出力する出力端子Ret Valueが作成されます。これは常にいちばん上の出力ピンです。
- **参照渡し**の入力に対してそれぞれ出力端子が作成されます。

入出力端子の名前(Ret Value以外)は、定義ファイルのパラメータ名によって決められます。ただし、出力端子に出力される値は名前でなく位置で決まります。最初(いちばん上)の出力ピンは必ず戻り値です。

2番目の出力ピンは、参照渡しの最初のパラメータの値を返します。以下すべて同様です。自動ピン構成の後で端子を追加しない限り、名前と位置が対応しなくなることはありません。

ダイナミック・リンク・ライブラリの作成(MS Windows)

Windows版VEEでは、Callオブジェクトおよびオブジェクトを使ってDLLにアクセスできます。

注記

このセクションではDLLの呼出し方法を扱い、DLLの作成方法は説明しません。VEEバージョン3.2以上は32ビットDLLだけをサポートし、16ビットDLLはサポートしません。

DLLの作成

VEEプログラムを作成する前にDLLを作成します。DLLの作成方法は他のDLLと同じですが、Cの型の一部しか使用できない点に注意してください(388ページ「定義ファイルの作成」参照)。

DLL関数の宣言. Microsoft Visual C++バージョン2.0以上を使う場合、関数定義は下記のようにします。

```
__declspec(dllexport) long myFunc (...);
```

この定義を使えば、関数をDLLからエクスポートするための.DEFファイルが不要になります。DLLをコンパイルしてリンクするには下記のコマンド行を使います。

```
cl /DWIN32 $file.c /LD
```

/LDはDLLを作成する指示です。デバッグ情報を生成するには/Ziを指定します。

MSリンカは、デフォルトでCマルチスレッド・ランタイム・ライブラリとリンクします。GetComputerName()などの関数を使用する場合、Kernel32.libとリンクする必要があります。コンパイル/リンク行は下記のようになります。

```
cl /DWIN32 file.c /LD /link Kernel32.lib
```

DLL関数の宣言. Microsoft C++バージョン2.0以上では、VEEから呼び出すDLL関数を__declspec(dllexport)と宣言すれば、.DEFファイルが不要になります。例として、一般的な関数を下記のように定義します。

```
__declspec(dllexport) long genericFunc(long a) {return (a*2); }
```

Microsoft Visual C++以外を使用する場合、.DEFファイルに下記のように指定します。

```
EXPORTS genericFunc
```

関数定義は下記のようになります。

```
long genericFunc(long a);
```

定義ファイルの作成. 定義ファイルには、インポートされる関数のプロトタイプのリストを記載します。VEEはこのファイルを使って、Callオブジェクトを構成し、DLL関数に対するパラメータの渡し方を決定します。プロトタイプのフォーマットは下記の通りです。

```
<return type> <modifier> <function name> (<type> <paramname>, <type> <paramname>,
...) ;
```

ここで:

- <return type>は、int、short、long、float、double、char*、voidのいずれかです。
- <function name>は、先頭が英字で2文字目以降が英数字の文字列で、長さは最大512文字です。
- <modifier>は_cdecl、_pascal、_stdcallのいずれかです。
- <type>は、int、short、long、float、double、int*、char*、short*、long*、float*、double*、char**、voidのいずれかです。
- <paramname>は、先頭が英字で2文字目以降が英数字の文字列で、長さは最大512文字です。パラメータ名は指定しなくてもかまいませんが、指定することを推奨します。パラメータを参照渡しにする場合、パラメータ名の前に間接参照記号(*)を付けます。

例:

4個のパラメータを渡し、longを返す:

```
long aFunc(double *,long param2,long *param3, char *);
```

パラメータなし、doubleを返す:

```
double aFunc();
```

文字列を渡し、longを返す:

```
long aFunc(char *aString);
```

文字列の配列を渡し、longを返す:

```
long aFunc(char **aString);
```

パラメータの制限

VEEから呼び出されるDLL関数は、最大144バイトをスタックにプッシュします。関数が使用するパラメータの数はこれによって制限されます。144バイトの制限を超えない限り、任意の組合わせのパラメータが使用できます。longは4バイト、doubleは8バイト、ポインタは4バイトを使用します。例えば、1つの関数で36個のlong、または18個のdouble、または20個のポインタと8個のdoubleを使用できます。

Import Libraryオブジェクト

CallオブジェクトやFormulaボックスを使ってDLL関数を実行するには、Import Libraryオブジェクトを使ってVEE環境に関数をインポートする必要があります。Import LibraryオブジェクトのLibrary TypeでCompiled Functionを選択します。Definition Fileボタンを使って正しい定義ファイル名を入力します。最後に、File Nameボタンを使って正しいファイルを選択します。Library Nameボタンは関数の組に論理名を割り当てるもので、変更の必要はありません。

Callオブジェクト

DLL関数をCallオブジェクトから呼び出すには、Callオブジェクトを構成する必要があります。最も簡単な方法は、Import Libraryオブジェクト・メニューでLoad Libを選択してVEE環境にDLLファイルをロードすることです。次に、Callオブジェクト・メニューでSelect Functionを選択します。

定義ファイルに記載されたすべての関数のリストがダイアログ・ボックスとして表示されます。関数を選択すると、Callオブジェクトの入出力端子と関数名が自動的に正しく構成されます。

Callオブジェクトを手動で構成するために、関数名を変更し、適切な入出力端子を追加することもできます。

1. 関数に渡されるパラメータと同じ数の入力端子を追加します。いちばん上の入力端子が最初のパラメータ、そのすぐ下の端子が2番目のパラメータに対応します。

2. 参照渡しのパラメータを出力端子としてCall オブジェクトに追加します。値渡しのパラメータを出力端子にすることはできません。いちばん上の出力端子は、関数の戻り値です。そのすぐ下の端子が最初の参照渡しのパラメータです。

3. Function NameフィールドにDLL関数名を正しく入力します。

例えば、DLL関数が下記のように定義されている場合、

```
long foo(double *x, double y, long *z);
```

x、y、zに対応する3つの入力端子と、戻り値に1つ、xとzに1つずつの合計3つの出力端子が必要です。Function Nameフィールドの値はfooです。入力端子と出力端子の数が関数のパラメータの数と一致しない場合、エラーが発生します。

DLLライブラリがロードされているときにFunction Nameフィールドに関数名を入力した場合、Callオブジェクト・メニューのConfigure Pinoutコマンドを使って端子を構成することができます。

Delete Libraryオブジェクト

非常に大規模なプログラムの場合は、使用後にライブラリを削除したほうがよい場合があります。Delete Libraryオブジェクトはメモリからライブラリを削除します。これは、Import Libraryオブジェクト・メニューでDelete Libを選択した場合の動作と同様です。

Formulaオブジェクトでの DLL関数の使用

FormulaオブジェクトでDLL関数を使用することもできます。Formulaオブジェクトの場合、戻り値だけが式で使用できます。参照渡しのパラメータにアクセスすることはできません。例えば、上の定義のDLL関数は次の式のように用いられます。

```
4.5 + foo(a, b, c) * 10
```

ここで、aはFormulaオブジェクトのいちばん上の入力端子、bはその下、cはいちばん下の入力端子です。fooへの呼出しのパラメータ数が正しくないと、エラーが発生します。

リモート関数について

リモート関数とは、リモート・ホスト・コンピュータ上の別のVEEプロセスで動作するUserFunctionです。リモート関数はUserFunctionの特別な場合です。UserFunctionに関する一般的な情報については、369ページ「UserFunctionについて」を参照してください。

リモート関数の使用

リモート関数はローカルVEEプロセスからLAN経由で呼び出されます。UserFunctionやコンパイル関数の場合と同様、リモート関数のライブラリを`Import Library`オブジェクトでインポートします。

インポートしたリモート関数は、`Call`オブジェクトを使うか、式の中で関数名を使用することにより呼び出すことができます。リモート関数をプログラムで呼び出す方法は、UserFunctionの場合と同様です。ただし、このあとで説明するいくつかの違いや、ネットワークに関連する技術的な注意事項があります。

リモート関数のライブラリは、UserFunctionのライブラリと同様の方法で作成しますが、保存先は目的のリモート・ホスト・コンピュータにします。リモート・ホスト・コンピュータ上にもVEE ProまたはVEE Proランタイムがインストールされている必要があります。

リモート関数のライブラリは、ローカルVEEプロセスではなく、リモート・ホスト上で動作する「サービス」と呼ばれる特別なVEEのプロセスにインポートされます。ローカルVEEプロセスのことを「クライアント」と呼びます。

クライアントVEEプロセスは、`Import Library`オブジェクトを使ってリモート関数ライブラリをインポートします。`Library Type`フィールドで`Remote Function`を選択すると、いくつかの新しいフィールドが図12-7のように表示されます。

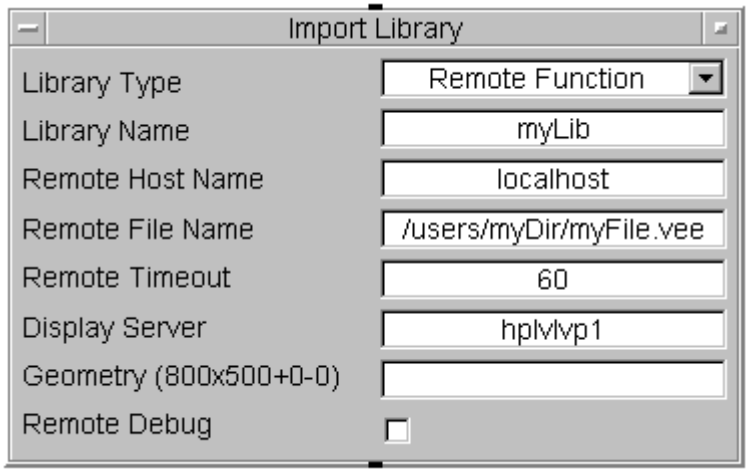


図12-7. リモート関数用のImport Library

`Library Type`フィールドと`Library Name`フィールドの動作は、UserFunctionおよびコンパイル関数の場合と同じです。その他のフィールドは下記の通りです。

- `Remote Host Name` - サービスVEEプロセスが動作するホスト(リモート・ホスト)の名前。ホストのシンボル名(`myhost`など)か、IPアドレス(`14.13.29.99`など)を指定します。

- `Remote File Name` - リモート関数ライブラリ・ファイルの名前。`Remote File Name`はUserFunctionライブラリの`File Name`フィールドと似た役割を果たします。ただし、ファイルの**絶対**パスを指定する必要があります。このため、パスとファイル名は長くなる場合があります。すべてのユーザが、例えば下記のような共通の場所にリモート関数ライブラリ・ファイルを置くようにしておくと便利です。`/users/remfunc/`または`C:\USERS\REMFUNC`

注記

クライアントから起動されるリモートVEEサービスは、Import Libraryオブジェクトの Host Nameに依存します。Host Nameが一致する2個のImport Libraryオブジェクトが存在する場合、サービス・プロセスは1つしか起動されません。Library NameとRemote File Nameが異なっていても、2つのオブジェクトは同じサービスと通信します。一方、2つのImport LibraryのHost Nameが異なっていれば、2つのサービスが起動されます。

- Remote Timeout - VEEサービスとの通信に用いられるタイムアウト(秒単位)。VEEサービスがリモート関数の結果をこの時間内に返さないと、エラーが発生します。

- Display Server - 解決可能なホスト名またはIPアドレスを指定します。このホストではXサーバが実行されており、指定したマシン上でXクライアント・ディスプレイを使用するパーミッションが設定されている必要があります。サービスがMS Windowsマシンで動作している場合、Display Serverフィールドは Remote Host Nameと一致しなければなりません。HP-UXの場合、別々のホストを指定できます。

- Geometry - リモートVEEのビューを表示するウィンドウの初期ジオメトリを標準ジオメトリ・フォーマットで指定します(800x500+0-0など)。

- Remote Debug - このチェック・ボックスを選択すると、ライブラリ内のすべてのUserFunctionがデバッグ・モードで実行されます(ブレークポイントの設定やライン・プローブの実行などのデバッグ操作が可能になります)。この設定は、UserFunctionにパネル・ビューがあるかどうかに関わらず動作します。

Import Libraryオブジェクトを実行すると(オブジェクト・メニューからLoad Libを選択するか、通常のプログラム実行による)、Host Nameフィールドに指定されたリモート・ホスト上でVEEサーバ・プロセスが開始されます。クライアント・プロセスとサーバ・プロセスがネットワーク経由で接続され、通信が可能になります。

クライアントVEEのCallオブジェクトがリモート関数を呼び出すと、引数(Callオブジェクトのデータ入力ピン)がネットワーク経由でリモート・サービスに送られ、リモート関数が実行され、結果がCallオブジェクトに送り返されて、データ出力ピンに出力されます。

プログラムでDelete Libraryオブジェクトを使ってリモート関数のライブラリを削除すると、そのライブラリに対応するリモート関数が削除されます。VEEサーバ・プロセスに複数のライブラリをロードし、サービス接続を維持したまま必要に応じてライブラリを個々に削除できます。VEEクライアント・プロセスが動作している間は、VEEサーバ・プロセスも存在します。

サービスVEEプロセスは、クライアントと同じホスト上に存在することも、ネットワークで接続された別のホスト上に存在することもできます。最も一般的なのはLANで接続されたホストの場合ですが、ネットワーク経路さえ存在すれば、海の向こうのホストでもかまいません。

VEEサービス・プロセスには、下記のように通常のVEEプロセスと異なる性質があります。

1. VEEサービス・プロセスは、Import Libraryで指定されたリモート関数ライブラリに含まれるリモート関数だけを実行します。

2. リモート関数にはビューが割り当てられています。UserFunctionがパネル・ビューを持っていれば、リモート関数を呼び出したときに、リモート・ホスト上にVEEウィンドウが表示されるようにできます。

3. グローバル変数(宣言済み、未宣言とも)はプロセス間では共有されません。

4. 呼び出されたリモート関数はタイムスライス動作しません。

5. リモート関数との間でobject型のパラメータを受け渡しすることはできません(ActiveXオートメーション・オブジェクトや、ActiveXコントロールへのポインタを含む)

6. サービスVEEプロセスが使用する実行モードは、インポートされたファイルに保存されているものでなく、ユーザの.veercファイルに指定されたものになります。

7. サービスVEEプロセスがインポートしたファイルに含まれる埋込み.veeioファイル構成は無視されます。グローバルI/O構成ファイルだけが用いられます。

Windowsを使用している場合、下記の手順でVEEサービス・マネージャを手動で起動する必要があります。

1. VEEインストール・ディレクトリに移動します。

2. *veesm.exe*を実行します。

3. コンソール・ウィンドウが表示されたら、邪魔にならないように最小化してかまいません。

4. VEEサービス・マネージャ・プロセスを停止するには、コンソール・ウィンドウをオープンし、**Ctrl+C**を押します。

VEEサービス・マネージャを自動的に起動する手順:

1. *veesm.exe*へのショートカットを作成します。

2. Start ⇒ Programsを選択します。

3. ショートカットをStartupフォルダに移動します。

UNIXのセキュリティ、UID、名前

クライアントVEEプロセスがリモート・ホスト上のサービスVEEプロセスを動作させるには、いくつかのセキュリティ上の要件が満たされる必要があります。基本的な要件として、サービスVEEプロセスを起動するには、クライアントVEEプロセスを実行するコンピュータ上のユーザ名と同じユーザ名がリモート・ホスト上に存在する必要があります(ただし、パスワードが一致する必要はありません)。

また、リモート・ホスト上の`/users`ディレクトリに自分のディレクトリがなければなりません。さらに、2つのホストの間でネットワーク接続を確立するには、クライアント・ホストのエントリを記入した`/etc/hosts.equiv`ファイルがリモート・ホストに存在するか、リモート・ホストの`$HOME`ディレクトリにクライアント・ホストのエントリを記入した`.rhosts`ファイルが存在する必要があります。

以下に例を示します。

クライアント・ホストが下記のように識別されるとします。

クライアント・ホスト: `myhost`

ユーザ: `mike`

パスワード: `twoheads`

また、サービス・ホストが下記のように識別されるとします。

サービス・ホスト:remhost

ユーザ:mike

パスワード:arebetter

ディレクトリ: /users/mike

この場合、下記のどちらかがサービス・ホスト上に必要です。

■ エントリmyhostを持つ/etc/hosts.equivファイル

または

■ エントリmyhost mikeを持つ/users/mike/.rhostsファイル

/etc/hosts.equivファイルを変更できるのはスーパユーザ(通常はシステム管理者)だけですが、.rhostsファイルはユーザが変更できます。一般的なやり方として、1つのサブネットに属するすべてのコンピュータに同一の/etc/hosts.equivファイルを置き、これらのコンピュータをすべてリストに記載する場合があります。最初に、クライアント・ホストのエントリが/etc/hosts.equivファイルにあるかどうかが調べられます。見つからない場合、.rhostsファイルが調べられます。

注記

サービスVEEプロセスを呼び出す際に、パスワードは不要であり、要求されることもありません。リモート・ホスト上のhosts.equivファイルまたは.rhostsファイルにクライアントのエントリが存在することが条件です。

UNIXセキュリティのもう1つの要素は、ユーザID(UID)とグループID(GID)です。UIDとは、1つのホスト上のすべてのユーザに/etc/passwdファイルによって割り当てられる固有の整数です。GIDとは、ユーザのグループに割り当てられる固有の整数です。すべてのUNIXプロセスには、UIDとGIDが対応付けられています。どのファイルやディレクトリに対してユーザが読取り、書込み、実行を行えるかが、UIDとGIDによって決まります。

サービス・ホスト上のVEEサービスには、クライアント・ホストからプロセスを起動したユーザのGIDとUIDが対応付けられます。すなわち、ファイルのパーミッションは、通常の対話的なVEEセッションをユーザが実行している場合と同じです。

リソース・ファイル

VEEサービス・プロセスが使用するVEE.IOまたは.veeioファイルおよびVEE.RCまたは.veercファイルは、リモート・ホスト上でプロセスを起動したユーザが所有するものです。前の例のユーザmikeの場合、VEEサービス・プロセスはホストremhost上の下記のファイルを読み取ります。

```
/users/mike/.veeio /users/mike/.veerc
```

(VEEが読み取るのはVEE.IOまたは.veeioファイルだけです。VEE.RCまたは.veercファイルは、trig設定と実行モードだけに用いられます)

エラー

リモートVEEサービスで発生するエラーには下記の2つの種類があります。

- **致命的エラー** - すでに説明したタイムアウト違反のように、エラーのためにサービスが使用不可能な状態になっている可能性が高いもの。致命的エラーがVEEサービスで発生すると、エラーは致命的であるというメッセージが表示されます。この場合、リモート関数ライブラリをインポートし直す必要があります。VEEクライアントはリモート・サービスを終了しようとします。

  ほとんどの場合、致命的エラーが起きるのは、ネットワークまたはリモート・サービスの呼出しに何か問題が発生したためです。リモート関数自体の問題で致命的エラーが発生することはまずありません。

- **非致命的エラー** - ほとんどの場合はリモート関数自体で発生したエラー(0除算など)です。この種のエラーは、関数がローカルでもリモートでも変わらずに発生するのが普通です。通常のエラー・メッセージが表示され、エラーが発生したリモート関数の名前が示されます。

注記

リモート関数が無限ループなどのためにハングすることもあります。この場合、リモート関数はタイムアウトし、致命的エラーのメッセージが表示されます。VEEクライアントはサービスを終了しようとしますが、サービスが応答しないために失敗します。このようなときは、リモート・マシン上でプロセスを終了する必要があります。UNIX版VEEの場合、リモート・ホストにログオンし、psでプロセスIDを調べ、killでプロセスを終了します。

13

ActiveXオートメーション・オブジェクトおよびコントロールの使用

ActiveXオートメーション・オブジェクトおよびコントロールの使用

Windows版VEEは、Windows 95、98、2000、NT 4.0を搭載したPC上のActiveXオートメーションおよびコントロールをサポートします。ActiveXテクノロジはUNIXではサポートされません。この章では、VEEでActiveXオートメーションおよびコントロールを使用する方法を説明します。ActiveXテクノロジ自体については解説しません。この章の内容は下記の通りです。

- VEEでのActiveXオートメーションの使用
- ActiveXオートメーション・オブジェクトの 使用
- ActiveXオートメーション・コントロールの 使用

注記

推薦図書

Microsoft Office 97 Visual Basic Programmer's Guide.
Microsoft Press, 1997. ISBN 1572313404.

Microsoft Office 2000 Visual Basic Programmer's Guide.
Microsoft Press, 1999. ISBN 1572319526.

VEEのActiveXサポートの実装は、Microsoft Visual Basicで確立された標準に基づいています。ActiveXテクノロジの初歩については、上記の書籍でオブジェクト・モデルとActiveXコントロールを扱った部分を参照してください。これらの概念を理解すれば、VEEのActiveX機能を使うために役立ちます。

VEEでのActiveXオートメーションの使用

ActiveXオートメーションを使えば、VEEをオートメーション・コントローラとして使って、Microsoft Word、Excel、Access、Seagate Crystal Reportsなどの他のWindowsアプリケーションを制御できます。アプリケーションにデータを送ってレポートを作成したり、アプリケーションからデータを読み取ったりできます。オートメーション機能を持つアプリケーションに対しては、DDE(Dynamic Data Exchange)でなくこの機能を使うことが推奨されます。

ActiveXコントロールはさまざまなベンダから提供されています。これらのコントロールは、ActiveXオートメーション・プロパティ、メソッド、イベントを通じて各ドメインに固有のサービスを提供することにより、VEEの機能を拡張する役割を果たします。大部分のActiveXコントロールにはユーザ・インタフェースも装備されており、「スライダ」などのコントロールを操作することで、VEEのスライダ・オブジェクトの場合と同様に、プログラムに値を入力することができます。

注記

ActiveXサポートを利用するには、VEEの`Default Preferences`ダイアログ・ボックスの`General`タブで、実行モードをVEE 5またはVEE 6モードに設定する必要があります。現在のモードはVEEウィンドウ下部のステータス・バーに表示されています。VEE 3.x、4.x、5.xで開発したプログラムにActiveX機能を追加する場合、新しい機能を追加する前に、VEE 5またはVEE 6実行モードでプログラムが動作することを確認してください。詳細については、17ページ「VEE実行モードの使用」を参照してください。

ActiveXオートメーションおよびActiveXコントロールの使い方を示すために、いくつかのサンプルが用意されています。サンプルがある場所は、VEEインストール・ディレクトリの`\examples\ActiveXAutomation`および`\ActiveXControls`です。これらのサンプルをオープンして実行するには、`Help ⇒ Open Example...`を使います。

ActiveXオートメーション・オブジェクトの使用

ActiveXサポートを有効にするには、VEEの実行モードを(`Default Preferences`で)VEE 5またはVEE 6モードに設定します。

オートメーション・オブジェクトをVEEで使用可能にする

Windowsアプリケーションをインストールすると、多くの場合ActiveXタイプ・ライブラリがいっしょにインストールされ、アプリケーションをオートメーション・サーバとして使えるようになります。タイプ・ライブラリにはActiveXオブジェクトの機能が記述されており、システムに存在すれば使用可能です。下記の理由で、VEEで使用するタイプ・ライブラリはVEEで選択しておくのが便利です。

- ActiveX オブジェクト用に宣言された変数でオブジェクト型が指定されている場合に、VEEに型チェックを行わせるため(404ページ「オートメーション・オブジェクト変数の宣言」を参照)。
- オートメーション・オブジェクトが生成したイベントを捕捉するため(423ページ「オートメーション・オブジェクト・イベントの処理」を参照)。
- ActiveXオブジェクト・ブラウザに情報を表示するため(411ページ「ActiveXオブジェクト・ブラウザの使用」を参照)。

プログラムで参照するタイプ・ライブラリを選択するには、`Device ⇒ ActiveX Automation References...`をクリックします。`ActiveX Automation References`ダイアログ・ボックスがオープンし、Windowsのレジストリに登録されているすべてのタイプ・ライブラリのリストが表示されます。図13-1のダイアログ・ボックスでは、Microsoft Accessライブラリが選択されています。

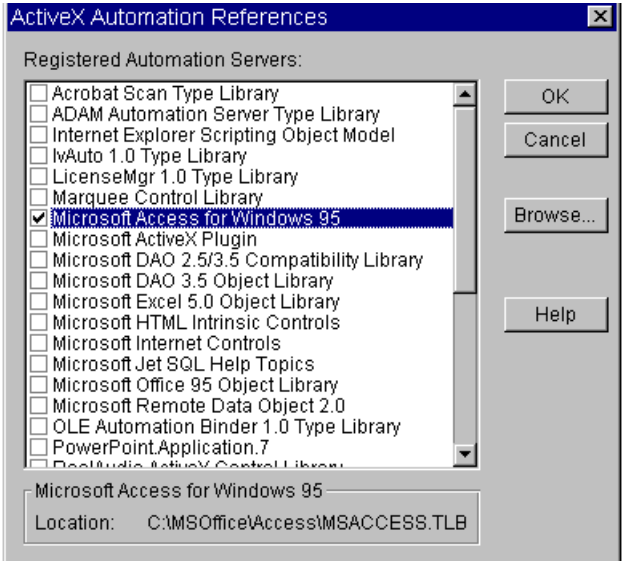


図13-1. ActiveXオートメーション・タイプ・ライブラリの選択

実際に表示されるリストは、インストールされているアプリケーションによって異なります。ライブラリ名を強調表示すると、ダイアログ・ボックスのステータス・エリアにその位置が表示されます。使用するオートメーション・サーバが見つかったら､ライブラリ名の隣のチェック･ボックスをクリック(または名前をダブルクリック)して、チェック・マークを表示させます。その後、OKをクリックします。

これにより、選択したタイプ・ライブラリがロードされ、ライブラリからエクスポートされているオブジェクト・クラス、ディスパッチ・インタフェース、イベントが検索されます。複数のライブラリを選択することができますが、選択したライブラリはメモリを使用するため、実際に使用するものだけを選択するようにしてください。

特定のオートメーション・サーバのタイプ・ライブラリ・ファイルが、存在するはずなのにリストに表示されない場合、アプリケーションをインストールしたときにタイプ・ライブラリがレジストリに登録されていない可能性があります。Browseボタンを押せば、リストにないタイプ・ライブラリを探すことができます。タイプ・ライブラリ・ファイルを見つけてオープンすると、VEEはそのタイプ・ライブラリの登録を試み、リストに追加します。

オートメーション・オブジェクト変数の宣言

ActiveXオートメーション・オブジェクト用の変数を宣言するには、新しいObjectデータ型を使います(`Data ⇒ Variable ⇒ Declare Variable`)。宣言された変数は、他のプロセスに存在するオブジェクトへの参照です。例えば、AccessのComboBoxを指します。図13-2に示すように、変数の`Type`を`Object`に設定すると、ダイアログ・ボックスが拡張され、ライブラリ名、クラス、イベントのイネーブル状態が表示されます。

図13-2. ActiveXオートメーション変数の宣言

オブジェクト変数の型をさらに詳しく指定するには、`Specify Object Type`をクリックしてチェック・マークを表示させます。次に、`Edit`ボタンをクリックして`Specify Object Type`ダイアログ・ボックスを表示させ、ライブラリ名とクラス名を設定し、このクラスで使用できるイベントをイネーブルします。

Accessオブジェクト・ライブラリを使用する場合、`combo`という変数を宣言し、図13-3のようにオブジェクト型を`Library: Access`、`Class: ComboBox`と指定します。この例では、クラス`ComboBox`にイベントが存在します。イベントを使用するには、`Enable Events`をクリックします。使用できるイベントがクラスにない場合、チェックボックスは淡色表示になります。

オブジェクト型を指定したら、`OK`をクリックしてダイアログ・ボックスを閉じ、`Declare Variable`に戻って表示された情報を確認します。

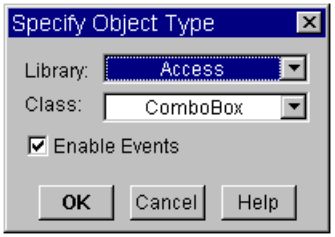


図13-3. オートメーション・オブジェクト型の指定

イベントをイネーブルした場合、捕捉するイベントのそれぞれに対してイベント・ハンドラ用UserFunctionを作成します。イベントの使用法については、423ページ「オートメーション・オブジェクト・イベントの処理」を参照してください。

あらゆるVEE変数と同様、変数の宣言は任意であり、宣言してもプログラム内にオートメーション・オブジェクトが作成されるわけではありません。ただし、オートメーション・オブジェクト用の変数を宣言してオブジェクト型の詳細を指定すると、VEEによって型チェックが自動的に行われ、指定したライブラリおよびクラス型のオブジェクト以外は宣言された変数に代入できなくなります。

注記

Windowsでプログラムを開発してActiveXオブジェクト用の変数を宣言し、そのプログラムをHP-UXでオープンした場合、変数宣言はプログラムにそのまま残りますが、オブジェクト型指定は無視されます。`Declare Variable`オブジェクトはオブジェクト型指定を保持しており、変更はできません。

プログラムでのオートメーション・オブジェクトの作成

VEEからサーバ・アプリケーションを制御するには、プログラムでオートメーション・オブジェクトを作成する必要があります。このためには*`CreateObject`*関数を使います。この関数をプログラムで使用するには、***fx***ツールバー・ボタンをクリックして`Function & Object Browser`を表示させ、下記のように選択します。

```
Type: Built-in Functions
Category: ActiveX Automation
Member: CreateObject
```

Create Formulaをクリックして、Formulaオブジェクトをプログラムに配置します。Formulaには下記の式が入っています。

```
CreateObject(ProgID)
```

この式を必要に合わせて変更します。

注記

*ProgID*は、作成するオートメーション・オブジェクトを識別する文字列です。オートメーション・オブジェクトのProgIDを調べるには、ベンダのドキュメントを参照してください。

ほとんどの場合は、サーバ・アプリケーションの新しいインスタンスにオートメーション・オブジェクトの新しいインスタンスを作成します。例えば、下記のVEEの式は、Excelの新しいインスタンスを(すでにExcelが動作している場合でも)作成し、新しい"Workbook"オブジェクトへの参照を返してexcel変数に代入します。

```
SET excel = CreateObject("Excel.Sheet")
```

DCOM(Distributed Component Object Model)の使用

DCOMを使うと、オートメーション・クライアント(VEE)からオートメーション・サーバ(Excel、Accessなど)をリモート制御できます。例えば、VEEが動作しているのとは別のコンピュータで動作しているExcelを操作できます。別のコンピュータにはVEEがインストールされている必要はなく、VEEが制御するアプリケーションがあればかまいません。

このためには、CreateObject関数に任意指定の2番目のパラメータを渡します。これはオブジェクトのインスタンスを作成するリモート・ホスト・コンピュータ(サーバ)の名前を指定します(この機能を使う場合、クライアントとサーバの両方のコンピュータにDCOMがインストールされており、dcomcnfg.exeを使ってセキュリティが正しく設定されている必要があります)。追加のパラメータを加えたCreateObject()の指定は下記のようになります。

```
CreateObject ("ProgID", ["hostName"])
```

ここで、*hostName*はText型です。*hostName*は有効なUNCまたはDNSドメイン名です。有効な*hostName*指定の例を下に示します。

```
Set obj = CreateObject ("ProgID", "server"
Set obj = CreateObject ("ProgID", "\\\\server")
```

```
Set obj = CreateObject ("ProgID", "server.domain.com")
Set obj = CreateObject ("ProgID", "135.5.33.19")
```

リモート・コンピュータ上でオブジェクトのインスタンスを作成するために必要なセキュリティ設定を、VEEからプログラムで制御する方法はありません。この設定は*dcomcnfg.exe*プログラムによって静的に行います。

既存のオートメーション・オブジェクトの取得

オートメーション・オブジェクトをすでに作成してある場合、GetObject関数を使うことにより、アクティブ・オブジェクトを取得したり、既存のオブジェクトをファイルからロードしたりできます。この関数をプログラムで使用するには、***fx***ツールバー・ボタンをクリックしてFunction & Object Browserを表示させ、下記のように選択します。

```
Type: Built-in Functions
Category: ActiveX Automation
Member: GetObject
```

Create Formulaをクリックして、Formulaオブジェクトをプログラムに配置します。Formulaには下記の式が入っています。

```
GetObject("fileName", "ProgID")
```

この式を必要に合わせて変更します。

下記の式は、アクティブ・オブジェクトを取得し、現在動作しているExcelアプリケーションのApplicationオブジェクトへの参照を返します。Excelが動作していない場合、この呼出しは失敗します。

```
SET excel = GetObject("","Excel.Application")
```

下記の式は、既存のオブジェクトをファイルからロードします。ProgIDパラメータは任意指定です。

```
SET excel = GetObject("d:/tmp/TestData.xls","Excel.Sheet")
```

または

```
SET excel = GetObject("d:/tmp/TestData.xls")
```

これらはいずれも、現在動作しているExcelアプリケーションの、d:/tmp/TestData.xlsに対応するシート・オブジェクトへの参照を返します。Excelが動作していない場合、Excelが起動されてからオブジェクトがロードされます。ProgIDを省略すると、VEEはCOM(Component Object Model)ライブラリを使ってファイルが関連づけられているアプリケーションを調べます。

オートメーション・オブジェクトの操作

オートメーション・オブジェクトを作成したら、オブジェクトを操作することによりサーバ・アプリケーションを制御することができます。オートメーション・オブジェクトの操作は、3つの基本的な動作からなっています。プロパティの取得、プロパティの設定、メソッドの呼出しです。このセクションでは、初期化済みのオブジェクト変数cell、sheet、excelを使ってこれらの使い方を説明します。VEEの新しいキーワードとしてSETとByRefが登場します。

プロパティの取得と設定

このセクションの式は、オブジェクトのプロパティを取得したり設定したりする例です。下記の式はプロパティの取得で、valueプロパティがセルの内容を返します。

```
contents = cell.value
```

次の式でも、valueプロパティがセルの内容を返します。

```
contents = sheet.cells(1,1).value
```

次の式は、前の式と同じくプロパティの取得ですが、.valueが隠れています。これは.valueがデフォルト・プロパティだからです(このあとで説明)。

```
contents = sheet.cells(1,1)
```

通常は右辺の式は内容、値、デフォルト・プロパティを取得しますが、値でなくオブジェクトのポインタを取得したい場合もあります。オブジェクト・ポインタを取得するには、SETキーワードを使ってデフォルト値でなくポインタを取得することを指示します。次の式はオブジェクト参照を設定しており、セルの集合の中の1つのセルを指すようにcell変数が設定されます。

```
SET cell = sheet.cells(1,1)
```

これらの例の違いは、デフォルト・プロパティでなく右辺のオブジェクト自体を左辺に代入することをSETによって指定していることです。

下記の式は、オブジェクトのプロパティを**設定する**例です。下の3つの式は、デフォルト・プロパティを使っているためすべて等価です。

```
cell.value = "Test Data2:"
sheet.cells(1,1).value = "Test Data2"
sheet.cells(1,1) = "Test Data2"
```

デフォルト・プロパティについて. ほとんどのオートメーション・オブジェクトは、デフォルト・プロパティまたはデフォルト・メソッドの概念をサポートします。これを利用して上記の例のようにオートメーション・オブジェクトを操作できます。cellの場合、デフォルト・プロパティはvalueです。したがって、さきほどのプロパティ取得の最初の例は、この概念を利用して.valueプロパティを隠すことで次のようになります。

```
contents = cell
```

この理由で、下記の式

```
contents = sheet.cells(1,1)
```

はセルの集合の中の1つのセルを返すだけでなく、そのセルのデフォルト・プロパティ (.value)を評価し、下記の式と同じ役割を果たします。

```
contents = sheet.cells(1,1).value
```

セルの集合の中の1つのセル自体を取得するには、次のように式の中でキーワードSETを使います。

```
SET cell = sheet.cells(1,1)
```

これにより、cellはExcelの対応するセルへのポインタに設定されます。これを次の式(既出)と比較してください。

```
contents = sheet.cells(1,1)
```

この式では、Excelの対応するセルの内容が取得されます。また、.valueプロパティはプロパティを設定する場合も隠すことができるので、下記の2つの式は同じ機能を果たします。

```
cell.value = "Test Data"
cell = "Test Data"
```

メソッドの呼出し

メソッドの呼出しはプロパティの取得と似ていますが、メソッドは関数と同様に括弧()を付けて使い、パラメータの指定が可能です。プロパティは通常、オブジェクト

の属性の値を取得、設定するために用いられます。メソッドは一般に何らかの動作を行うために用いられます。

下記の式は、オブジェクトのメソッドを呼び出す例です。

```
result = excel.CheckSpelling("aardvark")
```

デフォルトではパラメータは値渡しになります。例えば、cells(1,1)は実際にはメソッドを呼び出し、2個のパラメータ(1と1)を渡します。値渡しによりパラメータは単にExcelに送られ、戻り値が返されます。パラメータの値は変更されません。

オートメーション・メソッドの中には、参照渡しのパラメータを持つものもあります。パラメータの値がオートメーション・サーバによって変更され、新しい値がVEEに返されます。例えば、機器制御用のActiveXコントロールには下記の式で呼び出されるオートメーション・メソッドがあるかもしれません。

```
passed = Scanner.GetReading (ByRef Reading)
```

ここで、passedに格納されるメソッドの戻り値は、getReadingが成功した場合は真、失敗した場合は偽で、それ以外の値はByRefパラメータReadingによって返されます。変数Readingは関数に渡す前に初期化する必要があり、この式を含むFormulaオブジェクトには戻り値を利用するための出力端子が必要です。

VEEはByRefキーワードをサポートしており、ByRefで渡されるパラメータはFunction & Object Browserの情報エリアに表示されます。ByRefでは一部のデータ型がサポートされません。419ページ「VEE 5実行モードでのVEEデータ型からオートメーション・スカラ・データ型への変換」を参照してください。

列挙型の使用

タイプ・ライブラリに列挙型が定義されている場合、VEEのFunction & Object BrowserのClassエリアに表示されます。列挙型はオブジェクトのメソッドやプロパティを簡単に使えるようにするためのものです。例えば、ExcelのWindowオブジェクトにはWindowStateプロパティがあります。このプロパティはXlWindowState列挙型です。この列挙型には下記の3つの値があります。

```
xlMaximized (-4137)
xlMinimized (-4140)
xlNormal (-4143)
```

VEEは列挙型をサポートしており、オブジェクトのメソッドやプロパティを使う際に下記のような式が使用できます。

```
Window.WindowState = xlMinimzed
```

ActiveXオブジェクト・ブラウザの使用

ActiveXオブジェクト・ブラウザは、ツールバーの*fx*を押すとオープンするFunction & Object Browserの一部です。Type: ActiveX Objectsを選択すると、ブラウザの構成が変化します。ActiveXオブジェクトに用意されているプロパティ、メソッド、イベントをブラウザで調べることができるようになります。ActiveX情報をここに表示するためには、あらかじめオートメーションまたはコントロール・タイプ・ライブラリを選択しておく必要があります(Device ⇒ ActiveX Automation ReferencesまたはActiveX Control References)。図13-4は、Function & Object BrowserにAccessタイプ・ライブラリのActiveX情報を表示したところです。

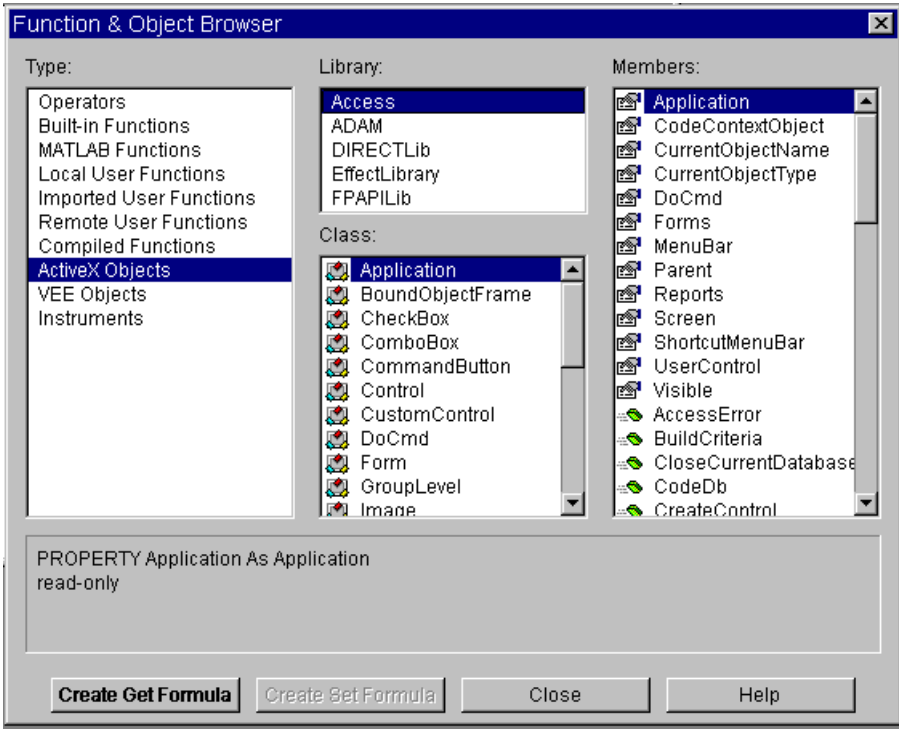


図13-4. ActiveXオブジェクト・ブラウザの使用

Library名を選択すると、使用可能なディスパッチ・インタフェース(dispinterfaces)と列挙型がClassエリアに表示されます。dispinterfaceを選択すると、使用可能なプロパティ、メソッド、イベントがMemberエリアに表示されます。列挙型を選択すると、定数の一覧が表示されます。図13-4には、Accessライブラリの機能の一部が表示されています。選択されているComboBoxディスパッチ・インタフェースのプロパティ、メソッド、イベントがMemberエリアに表示されています。図13-5に示すのは、ブラウザのClassesエリアとMembersエリアのエントリの間の関係と、それぞれの識別用アイコンです。

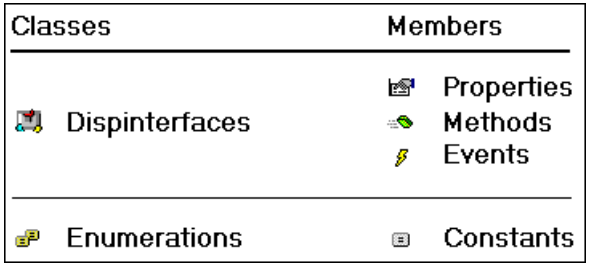


図13-5. Function & Object Browserに表示される要素

ブラウザの情報領域(ボタンの上の部分)には、プロパティ、メソッド、イベント、定数に関するヘルプ文字列が(オートメーション・オブジェクトにヘルプ文字列が登録されていれば)表示されます。ここには、パラメータ・リストの形でオブジェクトの型情報が示されます。角括弧[ ]で囲まれたパラメータは任意指定です。アプリケーションによってはヘルプ文字列が用意されていない場合もあります。

型情報は、ActiveXオブジェクトのプロパティ、メソッド・パラメータ、戻り値について用意されています。パラメータ型が指定されない場合、デフォルトの型はVT_VARIANTです。ほとんどの場合、VT_VARIANTに対する型変換はVEEが自動的に行います。オートメーション・データ型とVEEデータ型の間の変換については、表13-1と、このあとの部分を参照してください。

プロパティに関しては、読取り専用または書込み専用のプロパティかどうか、デフォルト・プロパティかどうかといった型情報がブラウザに表示されます。プロパティの取得や設定を行うには、Formulaオブジェクトを作成します。下に示すのは、ブラウザの情報エリアに表示されるプロパティ情報の例です。

```
DEFAULT PROPERTY Name as Text
```

メソッドに関しては、パラメータ・リスト内の各パラメータと戻り値に関する型情報がブラウザに表示されます。メソッドがデフォルト・メンバである場合には、そのことも示されます。メソッドを呼び出すには、Formulaオブジェクトを作成します。下に示すのは、ブラウザの情報エリアに表示されるメソッド情報の例です。

```
METHOD Void SetData(vValue, vFormat)
```

イベントに関しては、メソッドと同じ型情報がブラウザに表示されます。ただし、イベントに対応するイベント・ハンドラは、通常はクライアント・アプリケーションが呼び出します。Accessをオートメーションによって制御する場合、Accessがイベント・ハンドラUserFunctionを呼び出します。ActiveXコントロールを使用する場合、ActiveXコントロールがイベント・ハンドラUserFunctionを呼び出します。

プログラムやVEEからコールバック・イベント・ハンドラを呼び出すことはないので、Create Formulaボタンは淡色表示になります。イベントに関しては情報の表示だけが可能です。イベントは特定のActiveXオートメーション変数またはActiveXコントロールと結びついていなければならないので、Function & Object Browserからイベント・ハンドラUserFunctionを作成することはできません。

イベント・ハンドラを作成するには、Declare VariableまたはActiveXコントロールのオブジェクト・メニューを使用します。下に示すのは、ブラウザの情報エリアに表示されるイベント情報の例です。

```
EVENT Void Click()
```

列挙型の定数に関しては、定数の値がブラウザに表示されます。下に示すのは、ブラウザの情報エリアに表示される定数情報の例です。

```
CONSTANT tvwRootLines = 1
```

定数値が0より小さい場合と1024より大きい場合は、下記のように定数値の16進表現も表示されます。

```
CONSTANT xlNormal = -4143 (#HFFFFEFD1)
```

Helpボタンをクリックすると、選択したActiveXオブジェクト・メンバに関するヘルプ・ファイルとトピックがオープンします(オブジェクトにヘルプ情報が用意されている場合)。ヘルプ情報が用意されていない場合、選択したメンバに関するヘルプが存在しないというダイアログ・ボックスが表示されます。このヘルプ情報はアプリケーション・ベンダが提供するもので、**VEEオンライン・ヘルプ**の一部ではありません。

データ型の互換性

ActiveXオートメーションは、特定のデータ型をサポートします。このセクションでは、VEEデータ型とActiveXオートメーション・データ型との間の型変換について説明します。型変換は自動的に行われます。

表13-1に示すのは、サポートされるオートメーション・データ型と、対応するVEE 6実行モードのデータ型です。

表13-1. VEE 6実行モードでのオートメーション・スカラ・データ型から VEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_EMPTY	Text	空文字列("")のTextが返されます。VariantがVT_EMPTY型だったかどうかを調べるには、`isVariantEmpty()`を使います。
VT_NULL	Text	空文字列("")のTextが返されます。VariantがVT_NULL型だったかどうかを調べるには、`isVariantNull()`を使います。
VT_UI1	UInt8	
VT_I2	Int16	
VT_I4	Int32	
VT_R4	Real32	
VT_R8	Real64	
VT_CY	Real64	8バイト固定小数点整数で、小数点の右4桁がVEEのReal64に変換されます。VariantがVT_CY型だったかどうかを調べるには、`isVariantCurrency()`を使います。

表13-1. VEE 6実行モードでのオートメーション・スカラ・データ型から VEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_DATE	Real64	1899年12月30日からの日数が、VEEの日付/時間表現である0001年1月1日からの秒数に変換されます。
VT_BSTR	Text	
VT_DISPATCH	Object	
VT_ERROR	Int32	scodeの値を持つInt32が返されます。VariantがVT_ERROR型だったかどうかを調べるには、`isVariantError()`を使います。
VT_BOOL	Int16	VariantがVT_BOOL型だったかどうかを調べるには、`isvariantBool()`を使います。
VT_VARIANT	*	ByRefの場合(VT_VARIANT | BYREF)のみ有効。この場合、Variantは別のVariantを指します。VEEは埋め込まれたVariantの型に基づいてコンテナを作成します。
VT_UNKNOWN	Object	

表13-2に示すのは、サポートされるオートメーション・データ型と、対応するVEE 5実行モードのデータ型です。

表13-2. VEE 5実行モードでのオートメーション・スカラ・データ型から VEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_EMPTY	Text(空文字列)	
VT_NULL	Text(空文字列)	
VT_UI1	<エラー発生>	unsigned char
VT_I2	Int32	
VT_I4	Int32	
VT_R4	Real64	
VT_R8	Real64	
VT_CY	Real64	8バイト固定小数点整数で、小数点の右4桁がVEEのReal64に変換されます。
VT_DATE	Real64	1899年12月30日からの日数が、VEEの日付/時間表現である0001年1月1日からの秒数に変換されます。
VT_BSTR	Text	
VT_DISPATCH	Object	
VT_ERROR	<エラー発生>	
VT_BOOL	Int32	
VT_VARIANT	*	ByRefの場合(VT_VARIANT | BYREF)のみ有効。この場合、Variantは別のVariantを指します。VEEは埋め込まれたVariantの型に基づいてコンテナを作成します。
VT_UNKNOWN	Object	

表13-3に示すのは、サポートされるVEE 6実行モードのデータ型と、対応するオートメーション・データ型です。前の表に示された反対方向のマッピングと異なり、これは固定された1対1のマッピングではありません。ほとんどのオートメーション・サーバ・オブジェクトは必要なデータ型にデータを変換できます。

例えば、ターゲット・プロパティが長整数の場合(ポイントのX座標など)、正確に一致する`Int32`だけでなく、`Real`やText(数字の文字列の場合)も渡すことができます。ただし、配列の場合(常にVARIANTとして渡される)、使用可能なデータ型と配列形状は、ターゲットCOMオブジェクトの実装に依存します。

**表13-3. VEE 6実行モードでのVEEデータ型から
オートメーション・スカラ・データ型への変換**

変換前のVEEデータ型	変換後のオートメーション・データ型	注記
Uint8	VT_UI1	
Int16	VT_I2	
Int32	VT_I4	
Real32	VT_R4	
Real64	VT_R8	
Text	VT_BSTR	
<型*のスカラ>	VT_BOOL	Int16にキャストできるすべてのVEEスカラ・データ型(UInt8、Int16、Int32、Real32、Real64、Text)に対して`asVariantBool()`を使用します。

**表13-3. VEE 6実行モードでのVEEデータ型から
オートメーション・スカラ・データ型への変換**

変換前のVEEデータ型	変換後のオートメーション・データ型	注記
<型*のスカラ>	VT_CY	Real64にキャストできるすべてのVEEスカラ・データ型(UInt8、Int16、Int32、Real32、Real64、Text)に対して`asVariantCurrency()`を使用します。
<型*のスカラ>	VT_DATE	Real64にキャストできるすべてのVEEスカラ・データ型(UInt8、Int16、Int32、Real32、Real64、Text)に対して`asVariantDate()`を使用します。
<型*のスカラ>	VT_ERROR	Int32にキャストできるすべてのVEEデータ型(UInt8、Int16、Int32、Real32、Real64、Text)に対してasVariantError()を使用します。Int32の値が、エラーのscode(エラー番号)になります。
Variant	<Variant型*のスカラ>	VEEコンテナの現在の型が、適切なVariant型に変換されます。例えば、VariantコンテナにVEEのInt32が入っている場合、VEEはVT_I4型のVariantを作成します。
Object	VT_DISPATCH	オブジェクトのIDispatchポインタをVEEが持っている場合。
Object	VT_UNKNOWN	VEEがIUnknownポインタを持っているが、IDispatchポインタを持っていない場合。

表13-4に示すのは、サポートされるVEE 5実行モードのデータ型と、対応するオートメーション・データ型です。

表13-4. VEE 5実行モードでのVEEデータ型からオートメーション・スカラ・データ型への変換

変換前のVEEデータ型	変換後のオートメーション・データ型	注記
Int32	VT_I4	
Real64	VT_R8	
Text	VT_BSTR	
Object	VT_DISPATCH	オブジェクトのIDispatchポインタをVEEが持っている場合。
Object	VT_UNKNOWN	VEEがIUnknownポインタを持っているが、IDispatchポインタを持っていない場合。

表13-5に示すのは、オートメーション・データ型からVEE 6実行モードのデータ型への配列変換です。

表13-5. VEE 6実行モードでのオートメーション配列データ型からVEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_UI1	UInt8の配列	
VT_I2	Int16の配列	
VT_I4	Int32の配列	
VT_R4	Real32の配列	
VT_R8	Real64の配列	
VT_BSTR	Textの配列	
VT_BOOL	Int16の配列	Variant配列がVT_BOOL型だったかどうかを調べるには、`isVariantBool()`を使います。

表13-5. VEE 6実行モードでのオートメーション配列データ型からVEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_CY	Real64の配列	8バイト固定小数点整数で、小数点の右4桁がVEEのReal64に変換されます。Variant配列がVT_CY型だったかどうかを調べるには、`isVariantCurrency()`を使います。
VT_DATE	Real64の配列	1899年12月30日からの日数が、VEEの日付/時間表現である0001年1月1日からの秒数に変換されます。Variantが VT_DATE型かどうかを調べるには、isVariantDate()を使います。
VT_ERROR	Int32の配列	scodeの値を持つInt32が返されます。Variant配列が VT_ERROR型だったかどうかを調べるには、isVariantError()を使います。
VT_VARIANT	<同一の型の配列またはレコード>	配列要素がすべて同じ基本データ型の場合、VEEは表13-1のスカラ・マッピングで決まる型の配列を作成します。混合データ型の配列がVariantに含まれる場合、VEEのレコードが作成されます。
VT_DISPATCH	N/A	Object型の配列はサポートされません。
VT_UNKNOWN	N/A	Object型の配列はサポートされません。

表13-6に示すのは、オートメーション・データ型からVEE 5実行モードのデータ型への配列変換です。

表13-6. VEE 5実行モードでのオートメーション配列データ型からVEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_UI1	Int32の配列	
VT_I2	Int32の配列	
VT_I4	Int32の配列	
VT_R4	Real64の配列	
VT_R8	Real64の配列	
VT_BSTR	Textの配列	
VT_BOOL	Int32の配列	
VT_CY	Real64の配列	8バイト固定小数点整数で、小数点の右4桁がVEEのReal64に変換されます。
VT_DATE	Real64の配列	1899年12月30日からの日数が、VEEの日付/時間表現である0001年1月1日からの秒数に変換されます。
VT_ERROR	<エラー発生>	

表13-6. VEE 5実行モードでのオートメーション配列データ型からVEEデータ型への変換

変換前のオートメーション・データ型	変換後のVEEデータ型	注記
VT_VARIANT	Record	混合データ型の配列がVariantに含まれる場合、VEEのレコードが作成されます。
VT_DISPATCH	N/A	Object型の配列はサポートされません。
VT_UNKNOWN	N/A	Object型の配列はサポートされません。

表13-7は、オートメーション・データ型の修飾子を示します。

表13-7. オートメーション・データ型の修飾子

オートメーション型修飾子	VEEの型	注記:
VT_BYREF	上記のスカラ・マッピング表に示された型のスカラまたは配列	例えば、VT_BYREF | VT_BSTRはVEEのTextコンテナを生成します。スカラVT_VARIANTの場合、Variantは別のVariantを指します。VEEは埋め込まれたVariantのデータ型に基づいてコンテナを作成します。

オートメーション・オブジェクトの削除

オートメーション・オブジェクトへの参照をVEEが解放したときには、オブジェクトは自分自身を削除する責任があります。オートメーション・オブジェクトへの参照が存在しなくなった時点で、VEEはオブジェクトに対して参照が解放されたことを通知します。通知を受けたオブジェクトは、他のActiveXオートメーション・コントローラ・アプリケーションによってまだ使用されていない限り、自分自身を削除します。オートメーション・オブジェクトへの参照をVEEが解放するのは下記の場合です。

- オートメーション・オブジェクトの変数名に対して`Delete Variable`オブジェクトが実行された場合。ただし、そのオブジェクトに対する他のポインタがVEEにまだ保持されている可能性もあります。詳細については、**VEEオンライン・ヘルプ**の"Delete Variable"を参照してください。
- `Default Preferences`で`Delete Variables at Prerun`が選択されているときに、プログラムを再始動した場合。
- VEEが終了するか、`File` ⇒ `New`コマンドまたは`File` ⇒ `Open`コマンドが用いられた場合。

オートメーション・オブジェクト・イベントの処理

オートメーション・オブジェクトはイベントを生成することがあります。オートメーション・コントローラであるVEEには、UserFunctionを使ってイベントを捕捉できる機構が備わっています。オートメーション・オブジェクトからのイベントを処理するイベント・ハンドラUserFunctionを作成するには、適切な型の変数を宣言し、そのイベントを有効にしておく必要があります。オブジェクトが生成するイベントの1つ1つに対してイベント・ハンドラUserFunctionを作成できます。

オブジェクト用の変数を宣言し、そのイベント(存在する場合)を有効にしたら、イベント・ハンドラUserFunctionを作成できます。

1. 変数を宣言してその型を指定し、イベントを有効にしたら、`Declare Variable`オブジェクト・メニューをオープンします。
2. オブジェクト・メニューで、`Create Event Handler...`をクリックします。`Create Event Handler UserFunction`ブラウザが表示されます。図13-6を参照してください。

`Class`エリアに表示されたディスパッチ・インタフェースで使用できるすべてのイベントが`Member`エリアに表示されます。

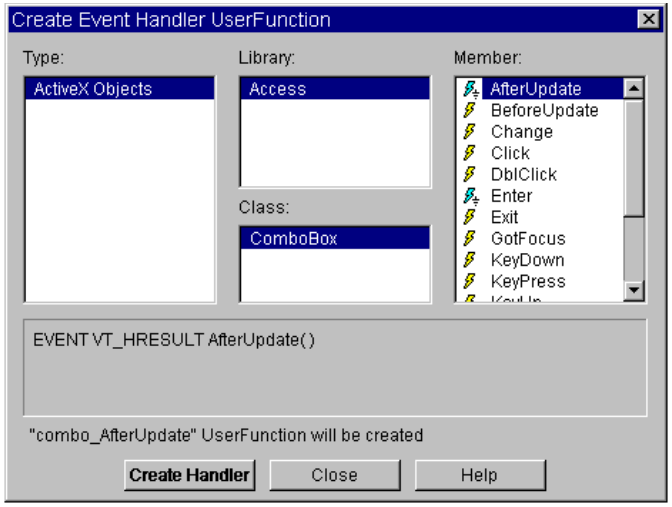


図13-6. `Create Event Handler UserFunction`**ブラウザ**

3. イベント名をクリックして選択します。

   イベントを選択すると、ブラウザの情報エリアにイベントの詳細が示され、VEEが作成するUserFunctionのタイトルがステータス・エリアに表示されます。`Help`ボタンを押すと、イベントの使い方に関する情報が表示されます。オンライン・ヘルプはライブラリ・ベンダから供給されるので、イベントによっては存在しない可能性もあります。イベントのオンライン・ヘルプは**VEEオンライン・ヘルプ**の一部ではありません。

4. `Create Handler`をクリックします。新しいUserFunctionウィンドウが表示されます。このダイアログ・ボックスをもう一度オープンして別のイベント・ハンドラを作成すると、ハンドラを作成済みのイベントのアイコンが別の色になっていることが分かるはずです。

   新しいイベント・ハンドラUserFunctionは、必要な入出力以外は空になっています。イベントを処理するプログラムをここに作成します。既存のイベントを編集するには、`Declare Variable`オブジェクト・メニューで`Edit Event Handler...`をクリックします。

イベントは宣言された変数名に結びついています。UserFunctionのタイトルは、変数名とイベント名を組み合わせたものです。例えば、`combo`という名前の変数を宣言

し、Access.ComboBox型と指定した場合、イベント・ハンドラUserFunctionの名前は下記のようになります。

```
combo_AfterUpdate
combo_Change
combo_DblClick
combo_KeyDown
```

イベントはコールバック関数です。生成されたUserFunction(コールバック関数)が各イベントを正しく処理するようにプログラムする必要があります。オートメーション・オブジェクトがイベントを生成すると、対応するUserFunctionが呼び出されてイベントを処理します。オートメーション・オブジェクトはイベントを生成したときにVEEからの戻り値を要求する場合もあります。この場合、UserFunctionが値を返すようにプログラムする必要があります。

オブジェクトが戻り値を要求する場合、VEEが値を返すまでオブジェクトは待ちます。イベント・ハンドラUserFunctionが戻るまでVEEとオートメーション・サーバ(Accessなど)の両方が待たされるため、イベント・ハンドラUserFunctionの処理はすばやく行う必要があります。

イベント・ハンドラUserFunctionが戻るまでオートメーション・サーバは待ち状態になるため、UserFunctionは非タイムスライス・モードで実行されます。すなわち、UserFunctionが実行を終了するまで、他のVEEプログラムとのタイムスライス実行は行われません。タイムスライス実行が行われないので、イベント・ハンドラUserFunctionではブレークポイントが使用できません。また、エラーが発生してもVEEは停止しません。エラーは注意に変換され、実行は継続します。

ActiveXオートメーション・コントロールの使用

ActiveXサポートを有効にするには、`Default Preferences`で実行モードがVEE 5またはVEE 6に設定されている必要があります。ActiveXサポートの詳細については、401ページ「VEEでのActiveXオートメーションの使用」を参照してください。

注記

VEEはすべてのActiveXコントロールをサポートするわけではありません。VEEと互換性のないコントロールの場合、プログラムにコントロールを追加したとき、あるいはコントロールの使用中にエラーが発生する可能性があります。**VEEオンライン・ヘルプ**に、問題が発生することが知られているコントロールの一覧が記載されています。

ActiveXコントロールの選択

VEEでActiveXコントロールを使用するには、どのActiveXコントロールを使用するかをVEEに知らせる必要があります。`Device ⇒ ActiveX Control References...`をクリックします。`ActiveX Control References`ダイアログ・ボックスがオープンし、Windowsレジストリに登録されている使用可能なコントロール・タイプ・ライブラリのリストが表示されます。図13-7に、いくつかのコントロールが選択されたダイアログ・ボックスを示します。

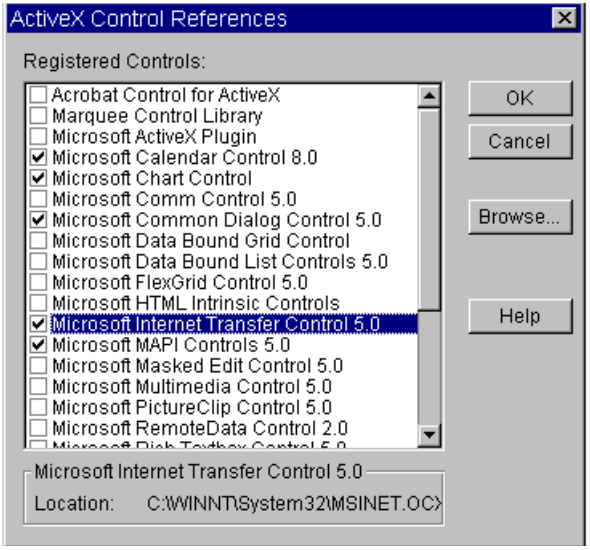


図13-7. ActiveXコントロールの選択

実際に表示されるリストは、インストールされているアプリケーションやコントロールによって異なります。コントロールは個別にインストールされることも、他のアプリケーションの一部としてインストールされることもあります。コントロール名を強調表示すると、ダイアログ・ボックスのステータス・エリアにその位置が表示されます。

使用するコントロールが見つかったら、コントロール名の隣のチェック・ボックスをクリック(または名前をダブルクリック)して、チェック・マークを表示させます。次に、OKをクリックしてコントロールをメモリにロードします。これにより、コントロールのオブジェクト・クラス、ディスパッチ・インタフェース、エクスポートされたイベントが検索され、VEEで使用可能になります。複数のコントロールを選択することができますが、選択したライブラリはメモリを使用するため、実際に使用するものだけを選択するようにしてください。

特定のコントロールのコントロール・タイプ・ライブラリが、存在するはずなのにリストに表示されない場合、インストール時にライブラリがレジストリに登録されていない可能性があります。Browseボタンを押せば、リストにないタイプ・ライブラリを探すことができます。タイプ・ライブラリ・ファイルを見つけてオープンすると、VEEはそのタイプ・ライブラリの登録を試み、リストに追加します。

VEEへのコントロールの追加

VEEプログラムにコントロールを追加する方法は、他のオブジェクトの場合と同様です。上記の手順でActiveXコントロールを選択すると、プログラムに追加できるようになります。`Device ⇒ ActiveX Controls`をクリックすると、選択されているコントロールのカスケード・メニューが表示されます。図13-8に例があります。

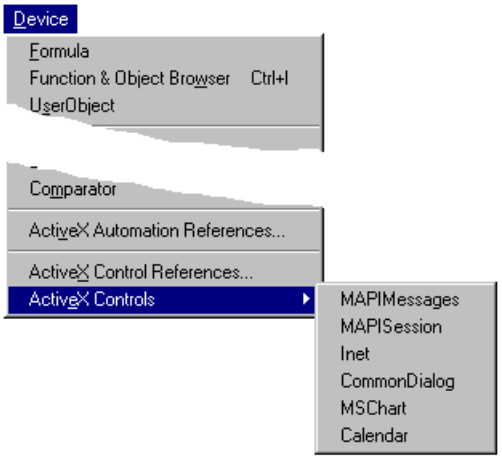


図13-8. Device MenuからのActiveXコントロールの追加

注記

図13-7と図13-8を比べると、`ActiveX Control References`ダイアログ・ボックスでは5個のコントロールが選択されているのに、`Device ⇒ ActiveX Controls`カスケード・メニューに表示されているのは6個です。このように、1つの選択肢によって複数のActiveXコントロールがメニューに追加されることは珍しくありません。

コントロールを選択し、生成されるオブジェクトをVEEワーク・エリアの詳細ビューに配置します。コントロールは、メイン、UserObject、UserFunctionのどのコンテキストにも配置できます。コントロールを削除するには、オブジェクト・メニューで`Cut`を選択するか、オブジェクトのコンテキスト・メニュー・ボタンをダブルクリックします。図13-9を参照してください。

ActiveXコントロール・ホストの違い

ActiveXコントロールと他のVEEオブジェクトとの間には大きな違いがあります。他のVEEオブジェクトと異なり、ActiveXコントロールには入出力ピンがなく、シーケンス入出力ピンもありません。制御はデータ・フローによって行われるのではありません。

他のオブジェクトと同様の方法でコントロールにアクセスできるように、VEEはコントロールのホストとなる特殊なコンテナをプログラム中に作成します。このコンテナは、コントロールの開発者によって組み込まれたコントロール固有のプロパティにアクセスできるようにする役割も果たします。これらの付加的な機能も含めてActiveXコントロールと呼んでいます。

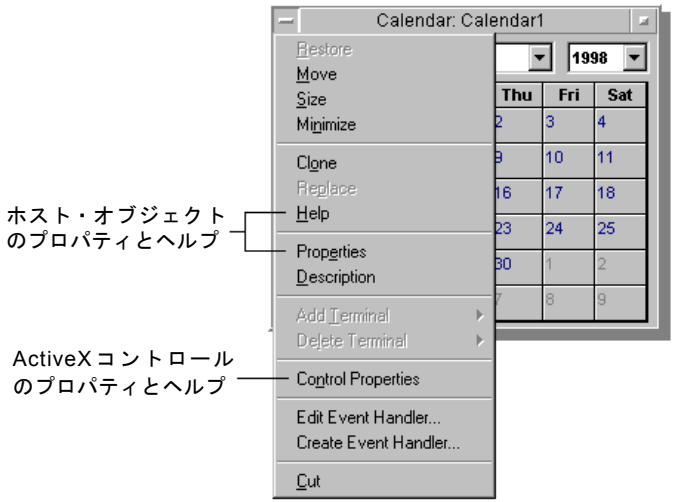


図13-9. ActiveXコントロールのプロパティとヘルプへのアクセス

オブジェクト・メニューの違いについて説明します。`Properties`と`Control Properties`の2つのメニュー・アイテムが存在し、それぞれ異なるプロパティ群へのアクセスを可能にします。ホスト・コンテナのプロパティはコントロールのプロパティとは別です。VEEオブジェクト(この場合はホスト・コンテナ)の一般的なプロパティを見るには、`Properties`をクリックします。コントロールの開発者が用意したActiveXコントロールのプロパティを表示したり変更したりするには、`Control Properties`をクリックします。

コントロールの`Properties`ダイアログ・ボックスの`Help`ボタンを使うと、そのコントロールのオンライン・ヘルプ・ファイルがオープンします(開発者から提供されている場合)。オブジェクト・メニューの`Help`アイテムは、**VEEオンライン・ヘルプ**のホスト・コンテナに関するトピックをオープンします。`Create Event Handler...`および`Edit Event Handler...`は、423ページ「オートメーション・オブジェクト・イベントの処理」で説明したActiveXオートメーション・オブジェクトに関する機能と同等のものを実現します。

VEEでのActiveXコントロールの使用

VEEのワーク・エリアにコントロールを追加すると、ローカル変数名が割り当てられてコントロールのタイトル・バーに表示されます。割り当てられた変数を変更するには、コントロールのタイトル・バーをダブルクリックして、ActiveX Control Propertiesダイアログ・ボックスを表示します。GeneralタブでName:の隣の値を変更します。

コントロールにはピンがなく、プログラムの他のオブジェクトとラインによって接続することができないので、コントロールを操作するにはFormulaオブジェクトの式でコントロールの変数名を参照します。コントロールの変数名のスコープはコンテキストにローカルなので、式はコントロールと同じコンテキストに存在する必要があります。

割り当てられたローカル変数の使用

プログラムにCalendarコントロールを追加すると、Calendarというローカル変数名が割り当てられます。タイトル・バーにはCalendarと表示されます。コントロールを操作するには、Calendarコントロールと同じコンテキストにFormulaオブジェクトを追加します。下の例は、Calendarという名前のVEEローカル変数で参照されるActiveXコントロールのプロパティの設定、プロパティの取得、メソッドの呼出しの方法を示します。

```
Calendar.Day = 24;
Month = Calendar.Month;
Calendar.AboutBox()
```

コントロールのためのグローバル変数の宣言

変数名をグローバルにするには、Declare Variable(Data ⇒ Variable ⇒ Declare Variable)を使って新しい変数名を宣言します。これは404ページ「オートメーション・オブジェクト変数の宣言」で説明した変数宣言と同様です。コントロールの変数名(Calendarなど)はすでに存在するので、同じ名前をグローバルとして宣言すると衝突してしまいます。一般的な方法としては、ローカル変数名を(g_localNameのように)一部に利用して、g_calendarのような名前を付けます。

Declare Variableに新しい変数名を入力し、ScopeをGlobalに、Type:をObjectに設定します。必須ではありませんが、Specify Object Typeをチェックして LibraryとClassを指定すれば、自動的に型チェックが行われ、指定したLibraryとClassだけがこの変数に代入できるようになります。

グローバル変数を宣言したら、Formula式を使ってコントロールのローカル変数名(Calendarなど)を宣言した変数名(g_calendarなど)に代入します。下の式のように必ずSETキーワードを使用します。

```
SET g_calendar = Calendar
```

ActiveXコントロールの操作

ActiveXコントロールのプロパティの設定と取得、メソッドの呼出し、イベントの処理の方法は、408ページ「オートメーション・オブジェクトの操作」と423ページ「オートメーション・オブジェクト・イベントの処理」で説明したActiveXオートメーション・オブジェクトに対する方法と同じです。

VEEはActiveXコントロールをホスト・オブジェクトに入れることによってアクセス可能にしていますが、プログラム実行時のコントロールの動作には多少の違いがあります。基本的に、ある時点ではコントロールは詳細ビューとパネル・ビューのどちらか一方でしか表示できません。

例えば、プログラムの詳細ビューにコントロールを追加したとします。プログラムが動的にパネルを表示して、その上にコントロールがShow Panel on ExecuteまたはshowPanel()によって表示された場合、パネルがクローズするまで詳細ビューのコントロールは空白になります。パネルがクローズすると、詳細ビューにコントロールが再表示されます。

14

シーケンサ・オブジェクトの使用

シーケンサ・オブジェクトの使用

この章では、シーケンサ・オブジェクトを使用するための指針について説明します。下記の内容があります。

- シーケンサ・オブジェクト
- シーケンサ・オブジェクトの使用

シーケンサ・オブジェクト

シーケンサ・オブジェクトはDeviceメニューにあります。シーケンサ・オブジェクトを使いこなすには、本書やその他のマニュアルに記載された下記の内容を理解しておく必要があります。

- 機器I/O操作(第2章「機器制御の基本」を参照)
- トランザクション(第4章「トランザクションI/Oの使用」を参照)
- UserObject(267ページ「UserObject内部の伝搬」を参照)
- レコードとデータセット(第11章「レコードとデータセットの使用」を参照)
- UserFunction(第12章「ユーザ定義関数/ライブラリ」を参照)

シーケンサ・オブジェクトとは何か

Sequencerオブジェクトは、一連のテストの実行順序を制御するものです。そのためにSequencerオブジェクトは、テストを実行し、テスト結果を仕様と比較し、比較結果に基づいて次の動作を決定します。

Sequencerオブジェクトは、あらかじめ決められた一連のシーケンス・トランザクションを実行します。それぞれのトランザクションは、UserFunction、コンパイル関数、リモート関数、その他のVEE関数などの呼出しを含むVEEの式を評価します。式が返した値をトランザクションは既存のテスト仕様と比較します。テストの合否に応じて、トランザクションは異なる式を評価し、次に実行するトランザクションを選択します。トランザクションの動作を指定するには、トランザクションをクリックしてSequence Transactionダイアログ・ボックスを表示します。

トランザクションは、結果をLog出力ピンまたはUserFunction、コンパイル関数、リモート関数に出力することができます。結果はできたものから読み取ることも、Logレコードに蓄積することも、両方を組み合わせることもできます。ロギング動作を指定するには、Sequencer Propertiesダイアログ・ボックスのLoggingタブを使います。

テスト結果のロギング

Sequencerのログ・レコードをレコード配列に収集する際には、注意が必要になることがあります。第11章「レコードとデータセットの使用」で説明したように、レコードの配列を作成する場合、特定のフィールドのすべての配列要素は同じ型、形状、サイズでなければなりません。SequencerのLog出力端子から生成されるレコードのレコードの場合は、サブレコードの各フィールドの型、形状、サイズも一致する必要があります。

例えば、Sequencerを何回か実行し、ロギング結果を収集するために、Collectorを使って配列を作成するか、結果をデータセットに送るとします。どちらの場合も、トランザクションからロギングされる値の型、形状、サイズがシーケンサの実行のたびに変化すると、エラーが発生します。これは、レコードの配列が作成できないためにCollectorまたはTo DataSetオブジェクトがエラーを生成するからです。

このような状況は、シーケンサが実行されるたびにトランザクションが実行されるとは限らない場合、例えばENABLED IF条件が指定されている場合などによく起こります。
トランザクションが実行されない場合、ログ・レコードは生成されますが、NAMEフィールドとDESCRIPTIONフィールドは空文字列であり、その他のフィールドにはRealスカラ値0が入っています。

Sequencerが次に実行されたときにこのトランザクションが実行され、Realスカラ以外の結果をロギングした場合、エラーが発生します(このような場合、To DataSetを使わずに、To Fileを使ってファイルにコンテナ・フォーマットでロギング・レコードを書き込む方法があります)。

また、異なるサイズの配列がテストから返される場合、例えばフェールしたデータ・ポイントの配列が返される場合などにも、エラーが発生します。この場合、配列要素をパディングして常に同じサイズの配列を返すようにテストを作成する方法があります。

シーケンサ・オブジェクトの使用

Sequencerオブジェクトを使用する下記の4つの例を紹介します。

- 例: シーケンサ・トランザクション
- 例: テスト結果のロギング
- 例: データセットへのロギング
- 例: ビン・ソート

例: シーケンサ・トランザクション

Sequencerオブジェクトのオープン・ビューには、シーケンス・トランザクションのリストが表示されます。各トランザクションは、第4章「トランザクションI/Oの使用」で紹介した他のタイプのトランザクションと似ています。図14-1のプログラムには、Sequencerがトランザクションを使用して式を実行し、関数を呼び出すようすが示されています。

図14-1には、バックグランドにある2個のUserFunctionが示されていません。1つはmyRand1で、0～1の乱数を入力の値に加算します。もう1つはmyRand2で、0～100の乱数を入力に加算します。トランザクションを展開すると、これらのUserFunctionへの呼出しが見られます。UserFunctionの作成と使用の詳細については、第12章「ユーザ定義関数/ライブラリ」を参照してください。

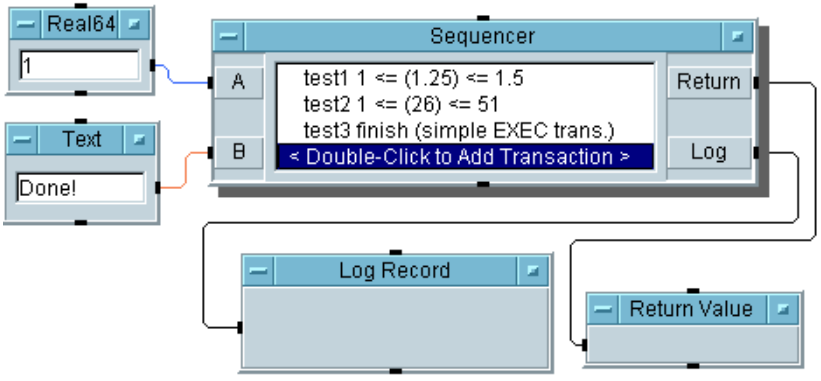


図14-1. 例: シーケンサ・トランザクション

シーケンサ・オブジェクトの使用

トランザクションをクリックすると、ダイアログ・ボックスにトランザクションが展開され、内容を見て編集することができるようになります。図14-2のダイアログ・ボックスは最初のトランザクションtest1を表示しています。

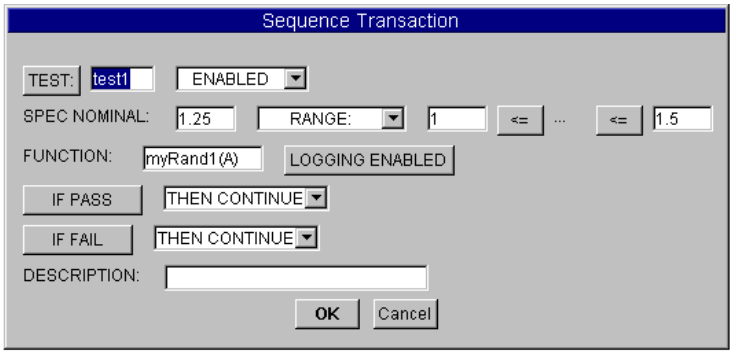


図14-2. test1**のSequence Transactionダイアログ・ボックス**

シーケンス・トランザクションには、TESTトランザクションとEXECトランザクションの2種類があります。このトランザクションのタイプはTESTです。名前フィールドはtest1、公称仕様は1.25、RANGE:仕様を使用、レンジは1 <= ... <= 1.5です。

テストに合格するのは1～1.5の範囲の値だけです。式myRand1(A)は、Sequencerの入力端子Aの値をパラメータとしてUserFunctionを呼び出します。

トランザクションでロギングがオンになっているので、このテストの結果のログ・レコードを記録するローカル変数Test1が自動的に作成されます。このログ・レコードは、Log出力端子からも得られます。IF PASS条件とIF FAIL条件はともにTHEN CONTINUEに設定されています。その意味は、合否に関わらず、test1が終了したら次のトランザクションtest2を実行するということです。

DESCRIPTIONフィールドは、このテストに関するコメントを入れるところです。

注記

RANGEまたはLIMITの場合、SPEC NOMINALの値はドキュメンテーション用にしか用いられません。一方、TOLERANCEまたは%TOLERANCEの値に基づいてテストを行う場合、SPEC NOMINALの値を基準に許容範囲が計算されます。

シーケンサ・オブジェクトの使用

図14-3に示す2番目のトランザクションtest2もTESTトランザクションです。

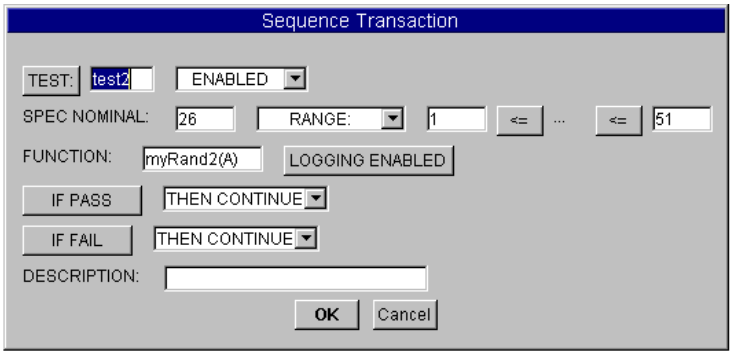


図14-3. test2のSequence Transactionダイアログ・ボックス

この2番目のテストは1番目とよく似ています。UserFunction myRand2が式myRand2(A)によって呼び出されます。結果の値が1～51の範囲内であるかどうかがテストされ、公称仕様は26です。ここでも、合否に関わらずSequencerは次のトランザクションに進みます。

図14-4に示す3番目のトランザクションはEXECトランザクションです。

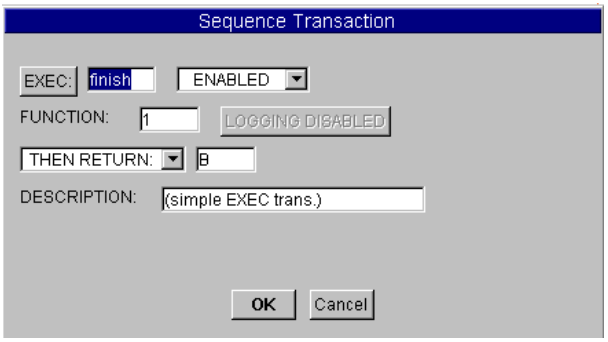


図14-4. EXECトランザクションのダイアログ・ボックス

EXECトランザクションは、TESTトランザクションと異なり、関数の結果を仕様やレンジと比較しません。EXECトランザクションの目的は、合否テストを必要としない動作を実行することです。

シーケンサ・オブジェクトの使用

例えば、TESTトランザクションを実行する前に外部構成をセットアップするルーチンをEXECトランザクションで呼び出したり、一連のテストのあとでパワー・ダウン手順を実行したりすることができます(EXECトランザクションは、常に合格するテスト条件を指定するのと同等の簡便な方法です)。

この例では、`finish`という名前のトランザクションが`Sequencer`オブジェクトの`Return`出力端子に`B`の値を返しています。EXECトランザクションではテストが実行されないので、ロギングは行われません。

`DESCRIPTION`フィールドを使ってトランザクションの簡単な説明を記述できます。

プログラムを実行すると、図14-5のように3つのトランザクションが順番に実行されます。

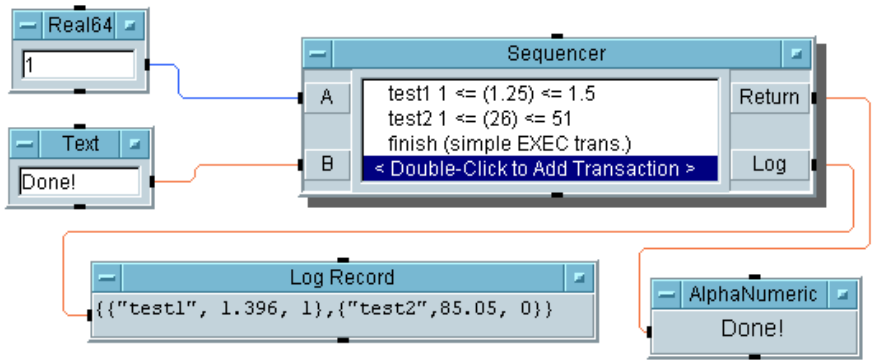


図14-5. プログラムの実行

ロギングされたテスト結果は`Log`出力端子に出力されて表示されます。結果はRecordデータ型(レコードのレコード)で記録されます。この場合、`test1`は値`1.396`で合格し、`test2`は値`85.05`で不合格になっています。3番目のトランザクションは`B`入力の値(文字列`Done!`)を返します。

ロギングがオンになっているトランザクションは、トランザクション名と同じ名前のログ・レコードを生成します。この例では、最初の2つのテストでロギングがオンになっているので、ローカル変数`Test1`と`Test2`にこれらのトランザクションのログ・レコードが格納されます。

ログ・レコードに含まれるフィールドは、`Properties`ダイアログ・ボックスで定義されます。ロギング構成にアクセスするには、`Sequencer`オブジェクト・メニュー

シーケンサ・オブジェクトの使用

でPropertiesをクリックし、Loggingタブをクリックします。デフォルトでは、Name、Result、Passの各フィールドがログ・レコードに含まれます。

Sequencer内部の任意の式でローカル変数Test1とTest2を使用することにより、現在または前に実行されたトランザクションの結果にアクセスすることができます。例えば、Test3でTest1.Resultを使って関数を呼び出すことにより、最初のテストの結果をパラメータとして渡すことができます。また、Test2.Passという式は、Test2が合格していれば1、不合格なら0に評価される式になります。

ロギング・レコードにアクセスするために、もう1つのローカル変数thisTestが用意されています。thisTestの値は常に、現在実行中のトランザクションのロギング・レコードです。これにより、同じ式を多数のトランザクションで使用する場合に、いちいちトランザクションの名前を指定する手間が省けます。

SequencerのLog出力端子から生成されるデータ構造は、図14-6に示すようなレコードのレコードです。

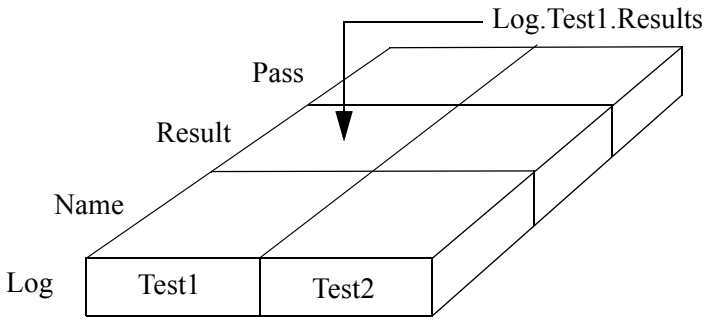


図14-6. ロギング結果のレコードのレコード

Log出力ピンから生成されるレコードには、ロギングがオンになっているトランザクションのそれぞれに対応するフィールド(この例ではTest1とTest2)があります。これらのフィールドはそれぞれ、対応するトランザクションのログ・レコードであり、Name、Result、Passの各フィールドを持ちます。

このレコードのレコードはLog出力ピンに出力され、レコードのドット・シンタックスによって他のオブジェクトから使用できます。例えば、Log.Test1.Resultという式は、この例の場合、値1.396を返します(図14-5を参照)。同様に、Log.Test1.Nameはtest1を返し、Log.Test1.Passは1を返します。

シーケンサ・オブジェクトの使用

Log出力ピンにロギングされるデータは、常に各トランザクションの**最後の**実行によるデータです。各トランザクションの**すべての**実行の結果をロギングしたい場合、Sequencer Propertiesダイアログ・ボックスのLoggingタブでLogging Modeを Log Each Transaction To:に設定します。このオプションを選択すると、トランザクションが終了するたびに指定された関数(または式)が呼び出されます。

このオプションは、Sequencerが終了してからでなく、テスト結果が得られるたびにファイルやプリンタに出力する場合に便利です。ローカル変数thisTestをロギング関数へのパラメータとして使うことにより、終了したばかりのトランザクションのログ・レコードを渡すことができます。

例: テスト結果のロギング

図14-7に示すのは、テスト結果のロギングに関するもう1つの例であり、反復子を使ってSequencerにテストを繰り返させ、結果をロギングします。

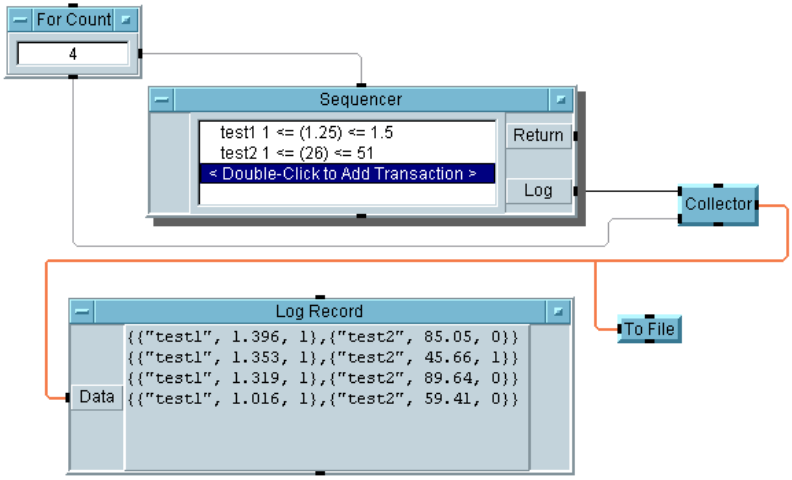


図14-7. 例: テスト結果のロギング

シーケンサ・オブジェクトの使用

この例では、For Countオブジェクトによって Sequencerが一連のテスト(前の例の test1と test2)を4回繰り返します。例えば、アセンブリ・ラインで4個の部品をテストする場合、Sequencerの1回の実行によって1個の部品がテストされます。

Log出力端子から順次出力されるレコードがCollectorに収集され、レコードの配列として表示されます。To Fileオブジェクトで WRITE CONTAINER I/Oトランザクションを使ってこの配列をファイルに出力することも、データセットを使うこともできます。

この例でのCollectorの出力は、図14-8に示すようなレコードのレコードの配列と考えることができます。

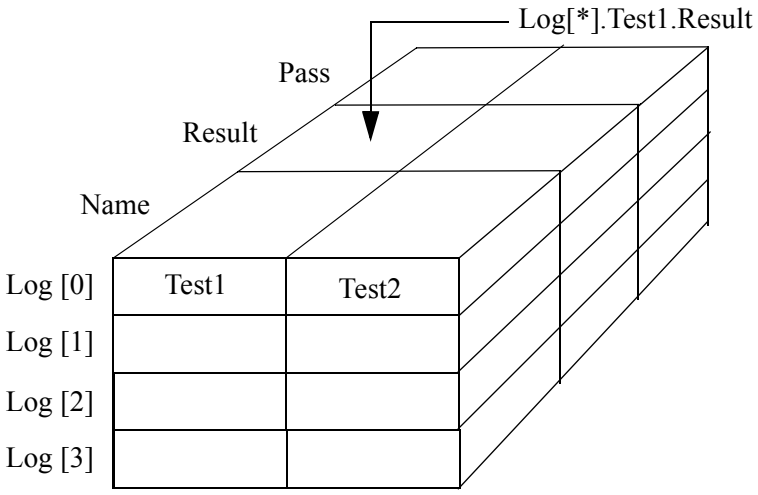


図14-8. ロギング結果のレコードのレコードの配列

各配列要素(Log[0]、Log[1]など)がシーケンサの1回の反復に対応し、レコードのレコードになっています。ロギング出力は式を使って解析できます。この例では、Log[*].Test1.Resultが配列の「コア・サンプル」です。Log[*].Test1.Result は値の配列(図14-5の例の結果では1.396、1.353、1.319、1.016)を返します。

シーケンサ・オブジェクトの使用

注記

ロギング結果は3次元配列ではなく、レコードのレコードの配列です。これが重要なのは、レコードの各フィールドは異なるデータ型でもかまわないからです。`Name`フィールドがTextなのに対し、`Result`フィールドはWaveformなどでもかまいません。また、`Test2.Result`フィールドがWaveformで、`Test1.Result`フィールドがReal値でもかまいません。

ただし、それぞれのフィールドは配列全体を通して一貫したデータ型でなければなりません。例えば、`Test1.Result`が`Log[0]`ではReal値、`Log[1]`ではWaveformということは不可能です。

図14-9の例は、`Sequencer`の反復を10回に増やし、ロギング・データの解析を加えたものです。図14-9で、`Formula`オブジェクトの式`log[*].test1.result`は、`test1`の結果を記録した10要素のReal配列を返します。この配列に対して`min(x)`、`max(x)`、`mean(x)`、`sdev(x)`オブジェクトを使って統計的解析が行われます。

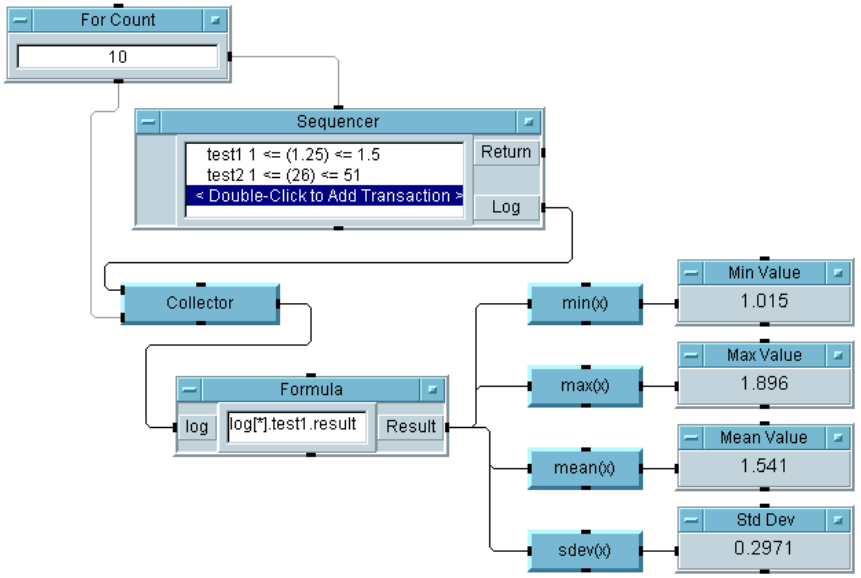


図14-9. ロギングされたテスト結果の解析

シーケンサ・オブジェクトの使用

この例は、examplesディレクトリにあるファイルmanual44.veeに保存されています。

例: データセットへのロギング

ロギングされたテスト結果を、データセットに記録することができます。図14-10のプログラムでは、Sequencerオブジェクトの Log出力端子がTo DataSetオブジェクトに接続されています。

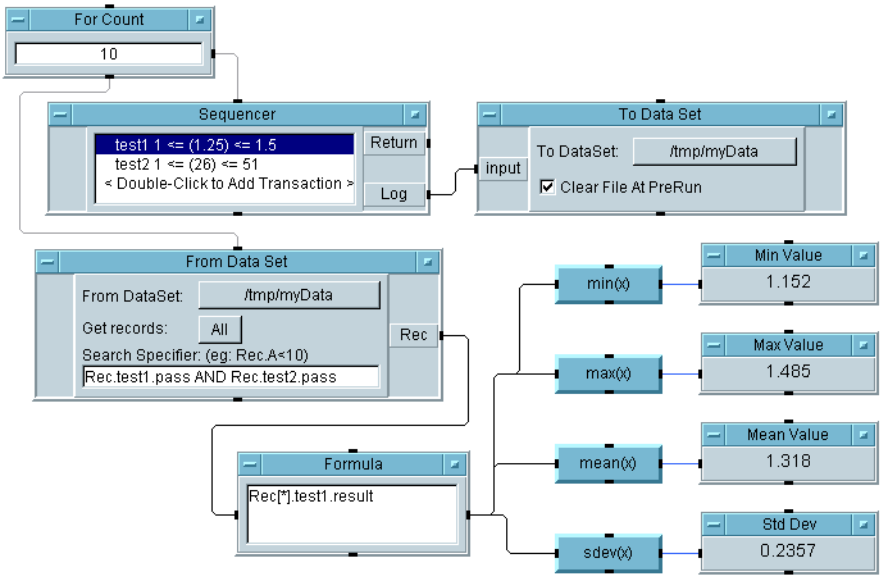


図14-10. 例: データセットへのロギング

For Countオブジェクトが終了すると、記録されたデータセット(myDataSet)をFrom DataSetオブジェクトが読み取ります。From DataSetは、myDataSetのすべてのレコードを読み取り、各レコードを条件Rec.test1.pass AND Rec.test2.passによってテストするように設定されています。これにより、test1とtest2の**両方**にパスしたレコードだけが読み取られます。

読み取られたレコードから、式Rec[*].test1.resultによってtest1.resultレコード・フィールドがすべて返され、その結果が統計的に解析されます(式Rec.test1.pass AND Rec.test2.passを満たすレコードがない場合、プログラムはエラーになります)。

シーケンサ・オブジェクトの使用

この例は、examplesディレクトリにあるファイルmanual45.veeに保存されています。

例: ビン・ソート

次の例は、カーボン抵抗の抵抗値を測定するものです。

以前には、カーボン抵抗は比較的精度の低いプロセスで製造され、テストのあとソートされてマーキングされていました。すべての抵抗を利用できるように、標準の抵抗値(220 Ω、270 Ω、330 Ωなど)は10%の許容範囲で重なり合うように選ばれていました。例えば、220 Ωより10%大きい抵抗値を持つ抵抗は、270 Ωの抵抗として使用できます。

図14-11に示すプログラムでは、抵抗値を返すUserFunctionをSequencerから呼び出しています。Sequencerは一連のテストを実行して、抵抗が満たす公称抵抗値と許容範囲のパーセンテージを求めます。これは「ビン・ソート」の問題です。すなわち、どのビンに抵抗を分類すればよいかがシーケンサから返されます。

シーケンサを使ってUserFunctionを呼び出すことの利点の1つは、異なるUserFunctionを置き換えて使えることです。この例の場合、開発中には所定の範囲内のランダムな抵抗値を返すUserFunction(simResist)を使い、機器I/Oを行って実際の抵抗値を返す別のUserFunctionにあとで置き換えることができます。

この問題を解決する最も簡単な方法は、多数のシーケンス・トランザクションを並べ、それぞれ1つずつの公称値と許容範囲に対して抵抗をテストすることです。

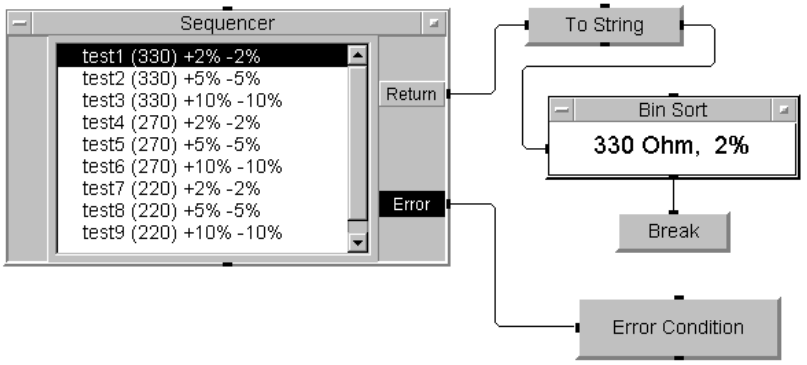


図14-11. ビン・ソートの例

この例では、最初のシーケンス・トランザクション(test1)がsimResist()という式でUserFunction simResistを呼び出しています(このUserFunctionには入力がありません)。図14-12に最初のシーケンス・トランザクションを示します。

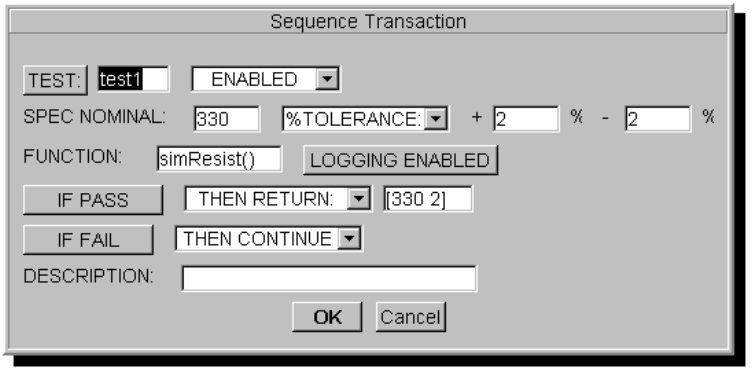


図14-12. test1トランザクション

test1は、simResistが返した抵抗値が公称値330 Ωの±2%以内にあるかどうかをテストします。値が範囲内の場合、2要素のReal配列[330 2]がReturn出力端子に返され、To Stringオブジェクトがこの値を文字列330 Ohm, 2%に変換します。テストが不合格になった場合、Sequencerは次のテストに進みます。

シーケンサ・オブジェクトの使用

2番目のトランザクションtest2はtest1と同様に動作しますが、simResistを呼び出すのではなく、FUNCTIONフィールドに式test1.resultが入っています。図14-13に2番目のシーケンス・トランザクションを示します。

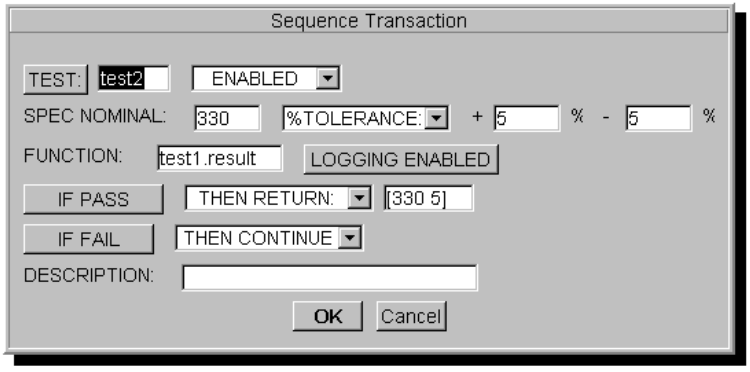


図14-13. test2 **トランザクション**

注記

トランザクションでロギングをオンにすると、テストと同じ名前のローカルなレコード変数が作成されます。このレコードには、ロギング・レコードで指定されたフィールドが含まれます。トランザクションtest1(図14-12)の場合、式test1.resultはtest1で呼び出された関数の戻り値を返します。

この例で式test1.resultを使用するには2つの理由があります。第1に、test2からtest9までのトランザクションでtest1.resultを使うことにより、すべてのトランザクションで同じ関数の結果が用いられることが保証されます(例えば、test1で別の関数を呼び出すようにあとで変更した場合でも)。

この例ではさらに重要な理由があって、このUserFunctionは呼び出されるたびに別の抵抗値を返すのです。ここでは新しい値が必要なのではなく、最初の値を別の公称値と許容範囲に対してテストし続けるのが目的です。このため、test2からtest9までのトランザクションはすべてFUNCTIONフィールドに式test1.resultを使用しています。これらのトランザクションは最初のものと同様に動作し、合格した場合はそれぞれの配列([330 5]、[330 10]、[270 2]など)を返します。

最初の8個のテストが不合格の場合、次のテストに進みます。しかし、**すべての**テストが不合格の場合には何らかの方法でわかるようにしなければなりません。このた

シーケンサ・オブジェクトの使用

め、test9はIF FAIL THEN ERRORに設定されています。Error出力端子によりError Conditionというタイトルの AlphaNumeric表示が実行され、Out of Rangeというテキストが表示されます。

この方法は簡単ですが、効率的とはいえません。各ケースで3つの許容範囲を使用して複数の抵抗値をテストするには、膨大な数のシーケンス・トランザクションが必要になります。図14-14に、ビン・ソートの例を改良したものを示します。

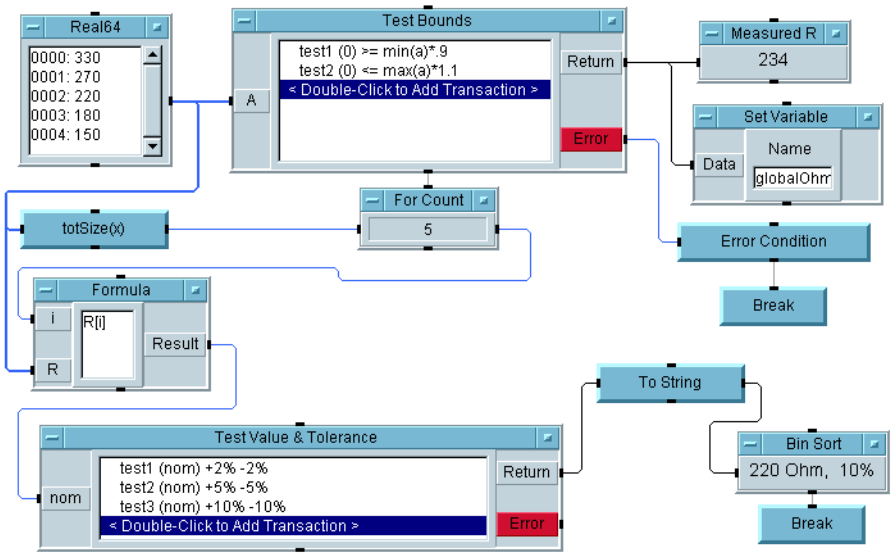


図14-14. ビン・ソートの例を改良したもの

この例は、examplesディレクトリにあるファイルmanual46.veeに保存されています。

シーケンサ・オブジェクトの使用

このプログラムの核心部分を以下に説明します。

- このプログラムでは2個のSequencerオブジェクトが用いられています。最初のもの(Test Boundsというタイトル)は、2番目のもの(Test Value & Toleranceというタイトル)のテストを再使用しています。

- プログラムの左上隅にあるReal64配列には5個の要素があり、それぞれが標準の抵抗値に対応します。ただし、この例ではリストの値を増やすこともできます。配列の要素数がいくつであっても、TotSize(x)関数が要素数を返すので、For Countオブジェクトは適切な回数だけ反復します。Formulaオブジェクトの式R[i]により正しいインデックスが使用されます。

- Test Boundsという名前のSequencerでは、図14-15に示すように、最初のトランザクション(test1)が式simResist()でUserFunction simResistを呼び出しています。

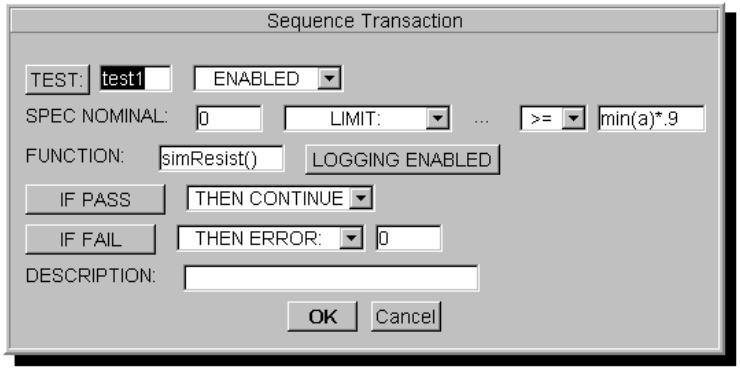


図14-15. 改良されたtest1トランザクション

シミュレートされた抵抗値が返され、配列の最小値(150 Ω)の90%以上であるかどうかがテストされます(シーケンス・トランザクションのすべての値フィールドにはmin(a)*.9のような式が使用できます)。

シーケンサ・オブジェクトの使用

2番目のトランザクション(test2)は、値(test1.result)が配列の最高値(330 Ω)の110%以下であるかどうかをテストします。図14-16にこのトランザクションを示します。

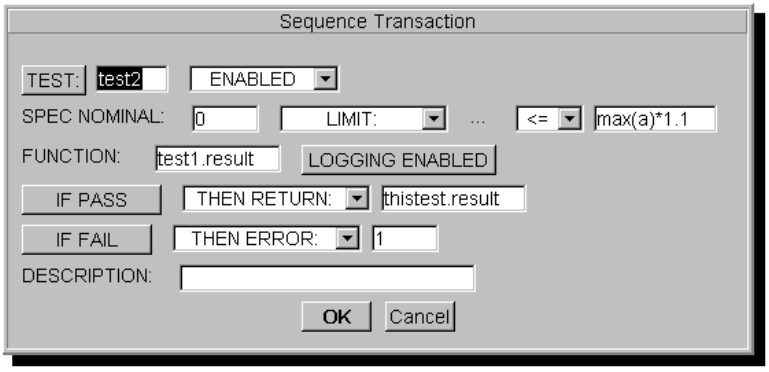


図14-16. 改良されたtest2トランザクション

どちらかのテストが不合格だと、エラーが発生します。

- エラーが発生した場合、Error Conditionという名前のUserObjectで、Out of Range: LOWとOut of Range: HIGHのどちらを表示するかが3項式によって決定されます。このUserObjectはShow Panel on Execに設定されているので、どちらかのエラー条件が発生すると、エラー表示がポップアップします。UserFunction simResistは100～400の範囲の乱数を返すので、プログラムを実行すると何回に1回かこのエラーが発生します(ポップアップ・ボックスでOKを押すと継続できます)。

シーケンサ・オブジェクトの使用

■ UserFunction simResistは､最初のSequencerのトランザクションtest1で1回だけ呼び出されます(test2はsimResistでなく式test1.resultを使います)。これは、単一の抵抗値に対して複数のテストを実行するために必要です。UserFunctionを複数回呼び出すと、そのたびに別の値が返されてしまいます。ただし、理由はもう1つあります。

UserFunction simResistは1回しか呼び出されていないので、別のUserFunctionへの呼出しに置き換えることができます。この例(図14-14のmanual46.vee)にはmeasResistというもう1つのUserFunctionがあります。これはパネル・ドライバを使ってHP 3478Aディジタル電圧計を呼び出し、抵抗測定を実行します。HP 3478A電圧計をお持ちなら、GPIBで接続し、test1のFORMULAフィールドを式measResist()に変更して、プログラムを実行してみてください。

■ シミュレートされた測定値と実測値のどちらを扱う場合でも､Test Boundsの戻り値が表示され、グローバル関数(globalOhms)に設定されます。例えば、Test Value & Toleranceという名前のシーケンサの3つのトランザクション(図14-14)は、それぞれ式globalOhmsによってこのグローバル変数を呼び出します。最初のトランザクションを展開したところを図14-17に示します。

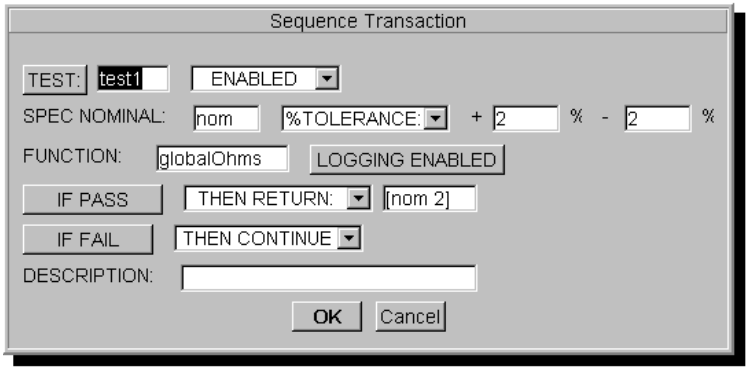


図14-17. globalOhms **トランザクション**

シーケンサ・オブジェクトの使用

テストが合格した場合、対応する実数配列([220 2]など)が出力されます。To Stringオブジェクトがデータを文字列(220 Ohm, 2%など)に変換します。Bin Sortの結果が見つかるまで、必要な回数だけSequencerが実行されます。

- Log出力端子はどちらのSequencerでも用いられないので、実行速度を上げるため削除されています。
- このプログラムのフローを見たい場合、Show Execution FlowおよびShow Data Flowをオンにしてプログラムを何回か実行してみてください。

Sequencerを使ったその他のサンプルが、examplesディレクトリに用意されています。

シーケンサ・オブジェクトの使用

A

I/Oトランザクション・リファレンス

I/Oトランザクション・リファレンス

この付録では、VEEのI/Oトランザクションの動作、エンコーディング、フォーマットについて説明します。下記の内容があります。

- I/Oトランザクション一覧
- WRITEトランザクション
- READトランザクション
- その他のトランザクション

I/Oトランザクション一覧

表A-1はVEEのI/Oトランザクション・タイプの一覧です。表A-2はVEEのI/Oトランザクション・オブジェクトの一覧です。

表A-1. I/Oトランザクション・タイプ一覧

動作	説明
WRITE	オブジェクトで指定された出力先にデータを書き込みます。
READ	オブジェクトで指定されたソースからデータを読み取ります。
EXECUTE	オブジェクトに対応するファイル、機器、インタフェースを制御する低レベル・コマンドを実行します。ファイル・ポインタの調整、パイプやファイルのクローズ、機器やハードウェア・インタフェースに対する低レベル制御の実現に用いられます。
WAIT	指定された秒数だけ待ってから次のトランザクションを実行します。 Direct I/Oオブジェクトの場合、特定のシリアル・ポール応答またはアクセス可能なVXI機器レジスタの特定の値を待つためにも使用できます。
SEND	IEEE 488で定義されたバス・メッセージ(バス・コマンドおよびデータ)をGPIBインタフェースに送信します。
READ(REQUEST)	他のアプリケーションからDDEデータを読み取ります。
WRITE(POKE)	他のアプリケーションにDDEデータを書き込みます。

表A-2. I/Oトランザクション・オブジェクト一覧

オブジェクト	サポートされるトランザクション				
	EXECUTE	WAIT	READ	WRITE	SEND
To File	X	X		X	
From File	X	X	X		
To Printer		X		X	
To String		X		X	
From String		X	X		
To StdOut		X		X	
From StdIn		X	X		
To StdErr		X		X	
Execute Program (UNIX)[a]	X	X	X	X	
To/From Named Pipe	X	X	X	X	
To/From Socket	X	X	X	X	
Direct I/O	X	X	X	X	
MultiInstrument Direct I/O	X	X	X	X	
Interface Operations	X				X
To/From Rocky Mountain Basic[b]	X	X	X	X	
To/From DDE[c]	X	X	X	X	

a. Execute Program (PC)はトランザクション・ベースではありません。
b. HP-UX版VEEのみ
c. Windows版VEEのみ

WRITEトランザクション

このセクションでは、WRITEトランザクションについて表A-3で説明します。すべてのWRITEエンコーディングに当てはまる項目をセクションの最初に記載します。

パス固有の動作

一部のWRITEトランザクションは、出力先のI/Oパスによって異なる動作をします。例えば、WRITE TEXT HEXトランザクションは、出力先がUNIXファイルか機器かによって異なる方法で16進数をフォーマットします。これらの動作を区別するため、このセクションでは下記の用語を使います。

- **UNIXパス**とは、機器以外のすべての出力先、すなわちUNIXファイル、文字列、プリンタ、UNIXパイプなどを表します。
- **MS-DOSパス**とは、機器以外のすべての出力先、すなわちMS-DOSファイル、文字列、プリンタなどを表します。
- **Direct I/Oパス**とは、Direct I/Oによってアクセスされる機器を表します。

すべてのパスに対する動作

以下の部分で説明する動作は、表A-3に特に記したものを除いて、すべてのパスに該当します。

表A-3. WRITE**エンコーディングおよびフォーマット**

エンコーディング	フォーマット
TEXT	DEFAULT STRING QUOTED STRING INT16, INT32 OCTAL HEX REAL32, REAL64 COMPLEX PCOMPLEX COORD TIME STAMP
BYTE	使用不可
CASE	使用不可
BINARY	STRING BYTE INT16 INT32 REAL32 REAL64 COMPLEX PCOMPLEX COORD
BINBLOCK	BYTE INT16 COMPLEX INT32 PCOMPLEX REAL32 REAL64 COORD
CONTAINER	使用不可
STATE[a]	使用不可

表A-3. WRITE**エンコーディングおよびフォーマット**

エンコーディング	フォーマット
REGISTER[b]	BYTE WORD16 WORD32 REAL32
MEMORY[b]	BYTE WORD16 WORD32 REAL32 WORD32*2 REAL64
IOCONTROL[c]	使用不可

a. Direct I/OからGPIBのみ
b. Direct I/OからVXIのみ
c. Direct I/OからGPIOのみ

TEXTエンコーディング

WRITE TEXTトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList [Format]
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

*Format*は任意指定の設定で、表A-4に記載されたフォーマットの1つを指定します。

表A-4. WRITE TEXT**トランザクションのフォーマット**

フォーマット	説明
DEFAULT	書き込むアイテムのデータ型に応じて、適切なテキスト表現をVEEが自動的に決定します。
STRING	Textデータを無変換で書き込みます。数値データ型は、最高の数値精度を持つTextとして書き込みます。
QUOTED STRING	STRINGと同じフォーマットでデータを書き込みますが、データを2重引用符(ASCIIの10進34)で囲みます。
INT16	16ビットの2の補数整数として10進形式でデータを書き込みます。
INT32	32ビットの2の補数整数として10進形式でデータを書き込みます。
OCTAL	32ビットの2の補数整数として8進形式でデータを書き込みます。
HEX	32ビットの2の補数整数として16進形式でデータを書き込みます。
REAL32	32ビットの浮動小数点数として、固定小数点や科学的記数法などさまざまな表記法でデータを書き込みます。
REAL64	64ビットの浮動小数点数として、固定小数点や科学的記数法などさまざまな表記法でデータを書き込みます。
COMPLEX	複素数を表す、カンマで区切った64ビット浮動小数点数の対を書き込みます。最初の数値が実数部、2番目の数値が虚数部を表します。
PCOMPLEX	複素数を表す、カンマで区切った64ビット浮動小数点数の対を書き込みます。最初の数値が大きさ、2番目の数値が位相角を表します。位相角はトランザクションで指定された位相単位で表されます。
COORD	直交座標を表す、カンマで区切った64ビット浮動小数点数の列を書き込みます。
TIME STAMP	実数(now ( ) 関数の出力など)を意味のある形式に変換し、年、月、日、時刻のさまざまな組合わせで書き込みます。

DEFAULTフォーマット

WRITE TEXT(デフォルト)トランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

このトランザクションは、*ExpressionList*の各アイテムを意味のある文字列に変換し、書き込みます。図A-1には、スカラ変数Xを書き込む単純な例を示します。

```
WRITE TEXT X
```

図A-1. WRITE TEXT**トランザクション**

図A-1のXに下記のようなテキストが入っている場合、

```
bird cat dog
```

変換は行われず、トランザクションは正確に12文字を書き込みます。

図A-1のXに下記のようなスカラ整数が入っている場合、

```
8923  Xの値(10進表記)
```

数値がテキストに変換され、VEEは正確に4文字を書き込みます。

図A-1のXに下記のようなスカラ実数値が入っている場合、

```
1.2345678901234567    Xの値(17桁のスカラ実数値)
```

最大16桁までの有効数字が書き込まれます。VEEの内部バイナリ形式と10進表記との変換のため、最も小さい桁は近似値になります。このスカラ実数値を下記のトランザクションで使用した場合、

```
WRITE TEXT a EOL
```

下記のように書き込まれます。

```
1.234567890123457    16桁の値
```

数値の絶対値がある程度大きいか小さい場合、指数表現が用いられます。Coord、Complex、PComplexの構成要素であるRealも同じ方法で出力されます。

WRITE TEXT DEFAULTトランザクションにEOL ONが指定された場合、そのオブジェクトのEOL Sequenceフィールドに指定された文字が、*ExpressionList*の最後の文字のあとに書き込まれます。

STRINGフォーマット

WRITE TEXT STRINGトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList STR
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

WRITE TEXT STRINGトランザクションは、基本的にはWRITE TEXT(デフォルト)トランザクションと同じ動作をします(例外についてはあとで説明します)。重要な違いは、STRINGではフィールド幅、ジャスティフィケーション、文字数などの出力フォーマットを細かく指定できることです。

フィールド幅とジャスティフィケーション. トランザクションにDEFAULT FIELD WIDTHと指定されている場合、*ExpressionList*のアイテムからTextへの変換から生じた文字だけが書き込まれます。

トランザクションにFIELD WIDTH: *F*と指定されている場合、変換されたTextがF文字分の空間の中で右揃えまたは左揃えで出力されます。

図A-2のトランザクションは、すべての文字を20文字分のフィールドの中で左揃えで出力するように指定しています。

```
WRITE TEXT X STR FW:20 LJ EOL
WRITE TEXT Y STR FW:20 LJ EOL
```

図A-2. 2つのWRITE TEXT STRINGトランザクション

図A-2のXとYが下記の値を持つとします。

```
bird cat dog          XのText値
12345678901234567     YのReal値
```

この場合、VEEは次のように書き込みます。

```
bird cat dog
12345678901234567
^                  ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。dogの右と2番目の7の右にある文字はスペース(ASCIIの10進32)です。

ジャスティフィケーションをRIGHT JUSTIFYに変更すると、トランザクションは図A-3のようになります。

```
WRITE TEXT X STR FW:20 RJ EOL
WRITE TEXT Y STR FW:20 RJ EOL
```

図A-3. 2つのWRITE TEXT STRINGトランザクション

図A-3のXとYが下記の値を持つとします。

```
bird cat dog            XのText値
12345678901234567       YのReal値
```

VEEは次のように書き込みます。

```
          bird cat dog
    12345678901234567
^                   ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。birdの左と最初の1の左にある文字はスペース(ASCIIの10進32)です。

指定されたフィールド幅より文字列が長い場合、文字列全体が書き込まれます。フィールド幅指定で文字が切り捨てられることはありません。文字が切り捨てられるのはMAX NUM CHARSを指定した場合だけです。

図A-4のトランザクションは、すべての文字を4文字分のフィールドの中で左揃えで出力するように指定しています。

```
WRITE TEXT X STR FW:4 LJ
```

図A-4. WRITE TEXT STRINGトランザクション

図A-4のXが下記の値を持つとします。

```
bird cat dog    XのText値、12文字
```

VEEは次のように書き込みます。

```
bird cat dog    12文字全部
```

指定したフィールド幅は4文字ですが、トランザクションは文字列の12文字全部を書き込みます。

文字数. ALL CHARSを指定すると、*ExpressionList*の各アイテムの変換で生じたすべての文字が書き込まれます。MAX NUM CHARS: *M*を指定すると、*ExpressionList*の各アイテムの最初の*M*文字だけが書き込まれます。

図A-5のトランザクションは、各フィールドに最大7文字を書き込み、フィールド幅が20文字で、フィールド・エントリを左揃えにすることを指定します。

```
WRITE TEXT X STR:7 FW:20 LJ EOL
WRITE TEXT Y STR:7 FW:20 LJ EOL
```

図A-5. 2つのWRITE TEXT STRINGトランザクション

図A-5のXとYが下記の値を持つとします。

```
bird cat dog        XのText値
12345678901234567   YのReal値
```

VEEは次のように書き込みます。

```
bird ca
1234567
^                   ^
```

Yの数値がTextに変換されたあと、文字が切り捨てられます。MAX NUM CHARSでは数値の丸めは行われません。

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。birdの右と最初の1の右にある文字はスペース(ASCIIの10進32)です。

Direct I/Oによる配列の書込み. ダイレクトI/OパスにWRITE TEXT STRトランザクションは、Direct I/Oオブジェクトの配列を書き込むArray Separator設定を無視します。配列の各要素(文字列)を書き込んだあとの区切りには、ラインフィード(ASCIIの10進10)が常に用いられます。この動作は、大部分の機器の要求に適合しています。

注記

配列に対するこの特別な動作は、他のトランザクション・タイプには当てはまりません。

QUOTED STRINGフォーマット

WRITE TEXT QUOTED STRINGトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList QSTR
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

一般に、STRINGフォーマットのところで説明した動作はQUOTED STRINGフォーマットにも当てはまります。STRINGとQUOTED STRINGの違いは下記の2点です。

- QUOTED STRINGの場合、文字列の先頭と末尾に2重引用符(ASCIIの10進34)が付加されます。2重引用符が付加されるのは、指定されたフィールド幅の中で文字列を位置揃えするためのスペースが付加されるより前です。
- 制御文字(ASCIIの10進0～31)、エスケープ文字(表A-5)、文字'(ASCIIの10進39)および"(ASCIIの10進34)が2重引用符で囲まれた文字列の内部にある場合、特別な扱いを受けます。

フィールド幅とジャスティフィケーション. DEFAULT FIELD WIDTHを指定した場合、*ExpressionList*のアイテムからTextへの変換から生じた文字とその前後の2重引用符だけが書き込まれます。

FIELD WIDTH: *F*と指定した場合、変換されたTextと前後の引用符が*F*文字分の空間の中で右揃えまたは左揃えで出力されます。

図A-6のトランザクションは、すべての文字を引用符で囲まれた文字列として20文字分のフィールドの中で左揃えで出力するように指定しています。

```
WRITE TEXT X QSTR FW:20 LJ EOL
WRITE TEXT Y QSTR FW:20 LJ EOL
```

図A-6. 2つのWRITE TEXT QUOTED STRINGトランザクション

図A-6のXとYが下記の値を持つとします。

```
bird cat dog        XのText値
12345678901234567   YのReal値
```

VEEは次のように書き込みます。

```
"bird cat dog"
"12345678901234567"
^                   ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。dog"の右と7"の右にある文字はスペース(ASCIIの10進32)です。

ジャスティフィケーションをRIGHT JUSTIFYに変更すると、トランザクションは図A-7のようになります。

```
WRITE TEXT X QSTR FW:20 RJ EOL
WRITE TEXT Y QSTR FW:20 RJ EOL
```

図A-7. 2つのWRITE TEXT QUOTED STRINGトランザクション

図A-7のXとYが下記の値を持つとします。

```
bird cat dog         XのText値
12345678901234567    YのReal値
```

VEEは次のように書き込みます。

```
     "bird cat dog"
"12345678901234567"
^                  ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。"birdの左と"1の左にある文字はスペース(ASCIIの10進32)です。

指定されたフィールド幅より文字列が長い場合、文字列全体が出力されます。フィールド幅指定で出力文字列が切り捨てられることはありません。文字が切り捨てられるのはMAX NUM CHARSを指定した場合だけです。

図A-8のトランザクションは、すべての文字を4文字分のフィールドの中で左揃えで出力するように指定しています。

```
WRITE TEXT X QSTR FW:4 LJ
```

図A-8. WRITE TEXT QUOTED STRING**トランザクション**

図A-8のXが下記の値を持つとします。

```
bird cat dog    XのText値、12文字
```

VEEは次のように書き込みます。

```
"bird cat dog"    12文字全部
```

文字数. ALL CHARSを指定すると、*ExpressionList*の各アイテムの変換で生じたすべての文字とその前後の2重引用符が書き込まれます。MAX NUM CHARS: *M*を指定すると、*ExpressionList*の各アイテムの最初の*M*文字と前後の2重引用符だけが書き込まれます。すなわち、*ExpressionList*の各アイテムに対して合計*M*+2文字が書き込まれます。

図A-9のトランザクションは、MAX NUM CHARS:7(フィールド幅20、左揃え)を指定しています。

```
WRITE TEXT X QSTR:7 FW:20 LJ EOL
WRITE TEXT Y QSTR:7 FW:20 LJ EOL
```

図A-9. 2つのWRITE TEXT QUOTED STRING**トランザクション**

図A-9のXとYが下記の値を持つとします。

```
bird cat dog          XのText値
12345678901234567     YのReal値
```

VEEは次のように書き込みます。

```
"bird ca"
"1234567"
^                   ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。ca"の右と7"の右にある文字はスペース(ASCIIの10進32)です。

埋め込まれた制御文字とエスケープ文字. この説明では、**制御文字**および**エスケープ文字**という用語を特別な意味に使用します。制御文字とは、ASCII文字の10進0～31に対応する1バイトのデータを表します。例えば、ラインフィードはASCIIの10進10であり、この説明では記号<LF>でラインフィード文字を表します。文字列\nはラインフィード文字を表す人間が読めるエスケープ文字であり、VEEによって認識されます。引用符で囲まれた文字列の中では、VEEはエスケープ文字を使って制御文字を表します。

エスケープ文字については表A-5を参照してください。

表A-5. エスケープ文字

エスケープ文字	ASCIIコード(10進)	意味
\n	10	改行
\t	9	水平タブ
\v	11	垂直タブ
\b	8	バックスペース
\r	13	キャリッジ・リターン
\f	12	改ページ
\"	34	2重引用符
\'	39	シングル引用符
\\	92	バックスラッシュ
ddd		3桁の8進数*ddd*に対応するASCII文字

図A-10のトランザクションを例にとって、さまざまなエスケープ文字を埋め込んだ場合の影響を見てみましょう。

```
WRITE TEXT X QSTR EOL
```

図A-10. `WRITE TEXT QUOTED STRING`**トランザクション**

図A-10のXが下記の値を持つとします。

```
bird\ncat dog
```

UNIXパスに対してはVEEは下記のように書き込みます。

```
"bird\ncat dog"
```

同じトランザクションとデータを使っても、ダイレクトI/OパスにはVEEは下記のように書き込みます。

```
"bird<LF>cat dog"
```

ここで、<LF>は1文字のラインフィード(ASCIIの10進10)を表します。

図A-10のXが下記の値を持つとします。

```
bird \"cat\" dog
```

UNIXパスとシリアル・インタフェースに対するDirect I/Oパスの場合、VEEは下記のように書き込みます。

```
"bird \"cat\" dog"
```

同じトランザクションとデータを使っても、GPIBインタフェースに対するダイレクトI/OパスにはVEEは下記のように書き込みます。

```
"bird ""cat"" dog"
```

GPIBインタフェースに対するこの特別な動作は、IEEE 488.2の要件をサポートすることが目的です。

INTEGERフォーマット

WRITE TEXT INTEGERトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList INT16
WRITE TEXT ExpressionList INT32
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

このトランザクションで生成される整数の型は、16ビットまたは32ビットの2の補数整数です。16ビット整数の範囲は-32766～+32767、32ビット整数の範囲は-2 147 483 648～+2 147 483 647です。数値を表現するために用いられる文字は+-0123456789だけです。

VEEは、*ExpressionList*の各アイテムをInt32またはInt16データ型に変換してから、Textに変換して最終的なフォーマットを行います。通常の変換規則が用いられます。詳細については、**VEEオンライン・ヘルプ**のTell Me About...にあるData Type Conversionのトピックを参照してください。

Real32をINT16またはINT32フォーマットで書き込んだ場合:

- Int16の有効範囲外のReal値はエラーを生成します。
- Int32またはInt16の有効範囲内のReal値は、Realの小数部分を切り捨てることによって変換されます。

Real64をINT16またはINT32フォーマットで書き込んだ場合:

- Int32またはInt16の有効範囲外のReal値はエラーを生成します。
- Int32またはInt16の有効範囲内のReal値は、Realの小数部分を切り捨てることによって変換されます。

桁数. DEFAULT NUM DIGITSを指定すると、整数の値を表現するのに必要な桁だけが書き込まれ、先頭に0は付きません。

MIN NUM DIGITS: *M*を指定すると、全部で*M*桁になるように必要に応じて出力の先頭に0が付けられます。

図A-11の2つのInt16またはInt32トランザクションは、出力桁数の指定だけが異なっています。

```
WRITE TEXT X INT16  EOL デフォルトの桁数
WRITE TEXT X INT16:6 EOL          6桁
                または
WRITE TEXT X INT32  EOL デフォルトの桁数
WRITE TEXT X INT32:6 EOL          6桁
```

図A-11. 2つのWRITE TEXT INTEGERトランザクション

図A-11のXが下記の値を持つとします。

```
4567
```

VEEは次のように書き込みます。

```
4567
004567
```

MIN NUM DIGITSで出力文字列が切り捨てられることはありません。図A-12のトランザクションは、最小桁数を1に指定しています。

```
WRITE TEXT X INT16:1 EOL
または
WRITE TEXT X INT32:1 EOL
```

図A-12. WRITE TEXT INTEGER**トランザクション**

図A-12のXが下記の値を持つととします。

```
12345678
```

VEEは次のように書き込みます。

```
12345678    8桁全部
```

符号プレフィックス. WRITE TEXT INTトランザクションの一部として、表A-6に示す符号プレフィックスの1つを任意に指定できます。

表A-6. 符号プレフィックス

プレフィックス	説明
/-	正の数はプレフィックスなし(+もスペースもなし)で書き込まれます。負の数は-プレフィックス付きで書き込まれます。
+/-	正の数は+プレフィックス付きで、負の数は-プレフィックス付きで書き込まれます。
" "/-	正の数はスペース(ASCIIの10進32)プレフィックス付きで、負の数は-プレフィックス付きで書き込まれます。

符号プレフィックスはMIN NUM DIGITSの桁数には含まれません。図A-13のトランザクションは、正と負の両方の数の前に明示的に符号を置くことを指定しています。

```
WRITE TEXT X INT16:6 SIGN:"+/-" EOL
WRITE TEXT Y INT32:6 SIGN:"+/-" EOL
```

図A-13. 2つのWRITE TEXT INTEGER**トランザクション**

図A-13のXとYが下記の値を持つとします。

```
123    Xの整数値
-123   Yの整数値
```

VEEは次のように書き込みます。

```
+000123    6桁+符号
-000123
```

OCTALフォーマット

WRITE TEXT OCTALトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList OCT
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

このトランザクションで書き込まれる整数の型は32ビットの2の補数整数です。整数の範囲は-2 147 483 648～+2 147 483 647です。8進数値の表現に用いられる文字は01234567だけです。他の文字を含むプレフィックスを任意に指定できます。

VEEは、OCTALフォーマットで書き込まれるデータをInt32データ型に変換してから、Textに変換して最終的なフォーマットを行います。VEEの通常の変換規則が用いられます。

OCTALフォーマットでRealを書き込んだ場合:

- Int32の有効範囲外のReal値はエラーを生成します。
- Int32の有効範囲内のReal値は、Realの小数部分を切り捨てることによって変換されます。

桁数. DEFAULT NUM DIGITSとMIN NUM DIGITSの動作は、474ページ「桁数」でのWRITE TEXT INTEGERトランザクションに関する説明と同じです。

8進プレフィックス. WRITE TEXT OCTALトランザクションの一部として、表A-7に示すプレフィックスの1つを指定できます。

表A-7. 8進プレフィックス

プレフィックス	説明
NO PREFIX	8進数はプレフィックスなしで書き込まれます。出力には01234567の数字だけが用いられます。
DEFAULT PREFIX	ダイレクトI/Oパスの場合、8進数の前には#Qが付けられます。これはIEEE 488.2で定義された8進の非10進数値データ・フォーマットをサポートするためです。 UNIXパスの場合、8進数の前には0が付けられます。MIN NUM DIGITSの指定を満たすために先頭に0が付加された場合、DEFAULT PREFIXによってさらに先頭に0が付加されることはありません。
PREFIX:*string*	*string*で指定した文字が8進数の前に付けられます。

図A-14のトランザクションは、デフォルト・プレフィックスと6桁を指定しています。

```
WRITE TEXT X OCT:6 PREFIX EOL
```

図A-14. WRITE TEXT OCTALトランザクション

図A-14のXが下記の値を持つとします。

```
15      10進15の値
```

ダイレクトI/Oパスに対してはVEEは下記のように書き込みます。

```
#Q000017      6桁+プレフィックス
```

同じトランザクションとデータを使っても、UNIXパスに対してはVEEは下記のように書き込みます。

```
000017    6桁
```

図A-15のトランザクションは、カスタム・プレフィックスと10桁を指定しています。

```
WRITE TEXT X OCT:10 PREFIX:"oct>" EOL
```

図A-15. `WRITE TEXT OCTAL`**トランザクション**

図A-15のxが下記の値を持つとします。

```
15
```

10進15の整数値

UNIXパスに対してもダイレクトI/Oパスに対してもVEEは下記のように書き込みます。

```
oct>000017
```

`DEFAULT PREFIX`によって書き込まれるプレフィックスは出力先に依存しますが、`PREFIX:`文字列によって書き込まれる文字列は出力先に依存しません。

HEXフォーマット

`WRITE TEXT HEX`トランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList HEX
```

このトランザクションで書き込まれる整数の型は32ビットの2の補数整数です。整数の範囲は`-2 147 483 648`～`+2 147 483 647`です。16進数値の表現に用いられる文字は`0123456789abcdef`だけです。他の文字を含むプレフィックスを任意に指定できます。

`WRITE TEXT HEX`の動作は`WRITE TEXT OCTAL`とほぼ同一です。異なる点は、使用可能なプレフィックスと、`DEFAULT PREFIX`の動作だけです。

16進プレフィックス. `WRITE TEXT HEX`トランザクションの一部として、表A-8に示すプレフィックスの1つを指定できます。

表A-8. 16進プレフィックス

プレフィックス	説明
`NO PREFIX`	16進数はプレフィックスなしで書き込まれます。出力には `0123456789abcdef` の数字だけが用いられます。
`DEFAULT PREFIX`	ダイレクトI/Oパスの場合、16進数の前には`#H`が付けられます。これはIEEE 488.2で定義された16進の非10進数値データ・フォーマットをサポートするためです。 UNIXパスの場合、16進数の前には`0x`が付けられます。
`PREFIX:`*`string`*	*`string`*で指定した文字が16進数の前に付けられます。

図A-16のトランザクションは、デフォルト・プレフィックスと6桁を指定しています。

```
WRITE TEXT X HEX:6 PREFIX EOL
```

図A-16. `WRITE TEXT HEX`**トランザクション**

図A-16のXが下記の値を持つとします。

```
15      10進15の整数値
```

ダイレクトI/Oパスに対してはVEEは下記のように書き込みます。

```
#H00000f      6桁+プレフィックス
```

同じトランザクションとデータを使っても、UNIXパスに対してはVEEは下記のように書き込みます。

```
0x00000f    6桁+プレフィックス
```

図A-17のトランザクションは、カスタム・プレフィックスと3桁を指定しています。

```
WRITE TEXT X HEX:3 PREFIX:"hex>" EOL
```

図A-17. `WRITE TEXT HEX` **トランザクション**

図A-17のxが下記の値を持つとします。

```
15     10進15の整数値
```

UNIXパスに対してもダイレクトI/Oパスに対してもVEEは下記のように書き込みます。

```
hex>00f     3桁+プレフィックス
```

`DEFAULT PREFIX`によって書き込まれるプレフィックスは出力先に依存しますが、`PREFIX:`文字列によって書き込まれる文字列は出力先に依存しません。

REAL32およびREAL64フォーマット

`WRITE TEXT REAL32`トランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList REAL32
```

このトランザクションで生成される実数の型は、32ビットのIEEE 754浮動小数点数です。数値の範囲は下記の通りです。

```
-3.40282347E38
        ～
 3.40282347E38
```

`WRITE TEXT REAL64`トランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList REAL64
```

このトランザクションで生成される実数の型は、64ビットのIEEE 754浮動小数点数です。数値の範囲は下記の通りです。

```
-1.797 693 134 862 315E+308
   ～
1.797 693 134 862 315E+308
```

これらの数値を表現するために用いられる文字は下記のものだけです。

```
+-.0123456789E
```

表記法と桁数. WRITE TEXT REALトランザクションの一部として、表A-9に示す表記法の1つを任意に指定できます。

表A-9. REAL**の表記法**

表記法	説明
STANDARD	各Real値を固定小数点表現(必要に応じて小数点を使用、指数部なし)と指数表現のどちらで書き込むかをVEEが自動的に決定します。有効数字でない0は書き込まれません。
FIXED	各Real値が固定小数点数として書き込まれます。小数部を持つ数値は、NUM FRACT DIGITSで指定された小数点以下の桁数に収まるように自動的に丸められます。指定された小数点以下の桁数を満たすように、必要なら末尾に0が付加されます。
SCIENTIFIC	各Real値が指数表現で書き込まれます。指数部には明示的な符号(+または–)と大文字のEが常に用いられます。小数部を持つ数値は、NUM FRACT DIGITSで指定された小数点以下の桁数に収まるように自動的に丸められます。指定された小数点以下の桁数を満たすように、必要なら末尾に0が付加されます。

図A-18のトランザクションは、STANDARD表記法と4桁の有効数字を指定しています。

```
WRITE TEXT X REAL32 STD:4 EOL
WRITE TEXT Y REAL64 STD:4 EOL
WRITE TEXT Z REAL32 STD:4 EOL
```

図A-18. 3つのWRITE TEXT REAL**トランザクション**

図A-18のX、Y、Zが下記の値を持つとします。

```
1.23456E2     XのReal32値
1.23456E09    YのReal64値
1.23          ZのReal32値
```

VEEは次のように書き込みます。

```
123.5         仮数部を必要に応じて丸める
1.235E+09     大きな数は指数表現
1.23          末尾に0はなし
```

図A-19のトランザクションは、FIXED表記法と小数点以下4桁を指定しています。

```
WRITE TEXT X REAL64 FIX:4 EOL
WRITE TEXT Y REAL32 FIX:4 EOL
WRITE TEXT Z REAL64 FIX:4 EOL
```

図A-19. 3つのWRITE TEXT REALトランザクション

図A-19のX、Y、Zが下記の値を持つとします。

```
1.2345678E2       XのReal64値
1.2345678E-09     YのReal32値
1.23              ZのReal64値
```

VEEは次のように書き込みます。

```
123.4568    仮数部を必要に応じて丸める
0.0000      小さな数は0に丸められる
1.2300      必要に応じて末尾に0を付加
```

図A-20のトランザクションは、SCIENTIFIC表記法と小数点以下4桁を指定しています。

```
WRITE TEXT X REAL32 SCI:4 EOL
WRITE TEXT Y REAL64 SCI:4 EOL
WRITE TEXT Z REAL32 SCI:4 EOL
```

図A-20. 3つのWRITE TEXT REALトランザクション

図A-20のX、Y、Zが下記の値を持つとします。

```
1.2345678E2        XのReal32値
-1.2345678E-09     YのReal64値
0                  ZのReal32値
```

VEEは次のように書き込みます。

```
1.2346E+02     指数部はEと2桁の符号付き数値
-1.2346E-09    最後の桁は必要に応じて丸める
0.0000E+00     必要に応じて末尾に0を付加
```

COMPLEX、PCOMPLEX、COORDフォーマット

COMPLEX、PCOMPLEX、COORDは、同じ名前のVEEのマルチフィールド・データ型に対応します。これら3つのフォーマットの動作はよく似ています。このセクションで説明する動作は、特に記す場合を除いて3つのフォーマットすべてに当てはまります。

VEEデータ型のComplex、PComplex、Coordが複数のReal数から構成されるのに対応して、COMPLEX、PCOMPLEX、COORDの各フォーマットは基本的にREAL64フォーマットの複合形式です。マルチフィールド・データ型の構成要素のReal値は、個々の値をREAL64フォーマットで出力した場合と同じ規則に従って書き込まれます。

マルチフィールド・フォーマットを扱うトランザクションの最終的な出力は、対応するオブジェクトのMulti-Field Formatの設定に影響されます。Multi-Field Format設定にアクセスするには、Direct I/Oオブジェクトの場合はI/O ⇒ Instrument Manager、他のオブジェクトの場合はオブジェクト・メニューのConfigを使います。Multi-Field Formatでは下記の2つの設定が可能です。

- Data Only - マルチフィールド・データ・フォーマットは、カンマで区切られた数値のリストとして、括弧なしで書き込まれます。
- (...) Syntax - マルチフィールド・データ・フォーマットは、カンマで区切られた数値のリストとして、括弧でくくられて書き込まれます。

これらの動作の例を以下に示します。

COMPLEXフォーマット. WRITE TEXT COMPLEXトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList CPX
```

図A-21のトランザクションは、固定小数点表現、明示的に符号を前置、10文字のフィールド幅、右揃えを指定しています。

```
WRITE TEXT X CPX FIX:3 SIGN:"+/-" FW:10 RJ EOL
```

図A-21. WRITE TEXT COMPLEX**トランザクション**

Multi-Field Formatが(...) Syntaxに設定され、図A-21のXが下記の値を持つとします。

```
( -1.23456 , 9.8 )    XのComplex値
```

VEEは次のように書き込みます。

```
(   -1.235 ,     +9.800)
 ^        ^   ^         ^
```

Multi-Field FormatがData Onlyに設定され、図A-21のXが同じ値を持つとすると、VEEは下記のように書き込みます。

```
      -1.235,    +9.800
^         ^ ^         ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。+の左にある文字はスペース(ASCIIの10進32)です。

(...) Syntaxでは、Complex数の実数部と虚数部を含む10文字幅のフィールドが、スペース-カンマ-スペースのシーケンスによって区切られます。どちらのMulti-Field Formatでも、実数部と虚数部のそれぞれに対して別々の10文字のフィールドが用意されます。括弧や区切りのカンマ、スペースはフィールドに含まれません。

PCOMPLEXフォーマット. WRITE TEXT PCOMPLEXトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList PCX
```

PCOMPLEXフォーマットでは、極座標形式の複素数出力に使用する位相単位を指定できます。この位相単位は、PropertiesのTrig Modeで設定する単位とは無関係です。表A-10を参照してください。

表A-10. PCOMPLEX**の位相単位**

単位	説明
DEG	度
RAD	ラジアン
GRAD	グラジアン

図A-22の最初のトランザクションは度単位の位相測定値を、2番目のトランザクションはラジアン単位の位相測定値を指定します。

```
WRITE TEXT X PCX:DEG STD EOL
WRITE TEXT X PCX:RAD STD EOL
```

図A-22. 2つのWRITE TEXT PCOMPLEX**トランザクション**

Multi-Field FormatがData Onlyに設定され、図A-22のxが下記の値を持つとします。

```
(-1.23456, @90)
```
XのPComplex値、度単位の位相

VEEは次のように書き込みます。

```
1.23456,-90
1.23456,-1.570796326794897
```

図A-23のトランザクションは、ラジアン単位の位相測定値、固定小数点表現、小数点以下3桁、明示的に符号を前置、10文字のフィールド幅、右揃えを指定しています。

```
WRITE TEXT X PCX:RAD FIX:3 SIGN:"+/-" FW:10 RJ EOL
```

図A-23. WRITE TEXT PCOMPLEX**トランザクション**

Multi-Field Formatが(...) Syntaxに設定され、図A-23のXが下記の値を持つとします。

```
(-1.23456 , @9.8)    XのPComplex値、ラジアン単位の角度
```

VEEは次のように書き込みます。

```
(    +1.235 , @    +0.375)
 ^         ^    ^         ^
```

PComplex数値はすべて、大きさが正、位相が+πと-πの間になるように正規化されます。

Multi-Field FormatがData Onlyに設定され、図A-23のXが同じ値を持つとすると、VEEは下記のように書き込みます。

```
      +1.235,    +0.375
^         ^ ^         ^
```

キャレット文字(^)は実際に書き込まれるわけではなく、フィールド幅を示すためのものです。-の左と+の左にある文字はスペース(ASCIIの10進32)です。

COORDフォーマット. WRITE TEXT COORDトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList COORD
```

COORDフォーマットの動作はCOMPLEXフォーマットと同じです。異なる点は、COORDには任意の数のフィールドがあるのに対して、COMPLEXのフィールドは常に2つしかないことだけです。

TIME STAMPフォーマット

WRITE TEXT TIME STAMPトランザクションは下記の形式を取ります。

```
WRITE TEXT ExpressionList [DATE:DateSpec] [TIME:TimeSpec]
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

*DateSpec*は下記の定義済みの日付と時刻の組合わせです。

- Date(日付)
- Time(時刻)
- Date&Time(日付と時刻)
- Time&Date(時刻と日付)
- Delta Time(差分時刻)

Dateを含むトランザクションを指定する場合、*DateSpec*としてWeekday DD/Month/YYYYまたはDD/Month/YYYYを指定することもできます。

Timeを含むトランザクションを指定する場合、*TimeSpec*を指定することもできます。*TimeSpec*は下記の定義済みの時刻フォーマットの組合わせです。

- HH:MM (時:分)
- HH:MM:SS (時:分:秒)
- 12 HOUR(12時間制)
- 24 HOUR(24時間制)

*ExpressionList*の各アイテムがRealに変換され、日付と時刻として解釈されます。このReal数は、紀元1年1月1日午前0時(UTC)からの経過秒数を表します。このReal数は一般にTime Stampオブジェクトからの出力として得られます。TIME STAMPフォーマットを使うことにより、このReal数を人間にとって意味のある日付と時刻の文字列に変換することができます。

TIME STAMPはさまざまな日付と時刻の表記法をサポートします。Real変数に下記の値が含まれるとします。

```
62806574669.31164
```

TIME STAMPを使えば、表A-11に示す任意のTimeとDateの表記法を使ってこれを書き込むことができます。

表A-11. `Time`と`Date`**の表記法**

表記法	結果
Date+Weekday DD/Month/YYYY	Thu 04/Apr/1999
Time+HH:MM:SSおよび24 HOUR	15:44:29
Date&Time+Weekday DD/Month/YYYY、HH:MM:SS、24 HOUR	Thu 04/Apr/1999 15:44:29
Time&Date+HH:MM:SS、24 HOUR、Weekday DD/Month/YYYY	15:44:29 Thu 04/Apr/1999
Delta Time+HH:MM:SS	17446270:44:29
Date+Weekday DD/Month/YYYY	Thu 04/Apr/1999
Date+DD/Month/YYYY	04/Apr/1999
Time+HH:MM:SSおよび24 HOUR	15:44:29
TIME+HH:MMおよび24 HOUR	15:44
TIME+HH:MM:SSおよび24 Hour	15:44:29
TIME+HH:MM:SSおよび12 Hour	3:44:29 PM

BYTEエンコーディング

`BYTE`トランザクションは下記の形式を取ります。

```
WRITE BYTE ExpressionList
```

*`ExpressionList`*は単一の式またはカンマで区切った式のリストです。

VEE 5およびそれ以前の実行モードでは、*`ExpressionList`*の各アイテムが`Int16`(16ビットの2の補数整数)に変換され、下位8ビットが書き込まれます。VEE 6実行モードでは、*`ExpressionList`*の各アイテムが`UInt8`(8ビットの2の補数整数)に変換されて書き込まれます。このトランザクションの目的は、機器に1文字を書き込むことです。*`ExpressionList`*のアイテムはスカラでなければなりません。

VEE 6実行モードでは、256より大きい値を指定するとエラーになります。例えば、VEE 5およびそれ以前の実行モードでは、図A-24のトランザクションは下記のデータ出力を生成します。

```
ABCAA
```

```
WRITE BYTE 65,66,67
WRITE BYTE 65+1024,65+2048
```

図A-24. 2つのWRITE BYTEトランザクション

VEE 6実行モードでは、図A-24の2番目のトランザクションはエラーになります。

CASEエンコーディング

WRITE CASEトランザクションは下記の形式を取ります。

```
WRITE CASE ExpressionList1 OF ExpressionList2
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

*ExpressionList1*の各アイテムが整数に変換され、*ExpressionList2*へのインデックスとして用いられます。インデックスで指定された*ExpressionList2*のアイテムが、WRITE TEXT(デフォルト)と同じ文字列フォーマットで書き込まれます。

*ExpressionList2*のアイテムのインデックスは0起源です。

図A-25のトランザクションは、CASEフォーマットの動作を示します。

```
WRITE CASE 2,1 OF "Str0","Str1","Str2"
WRITE CASE X OF 1,1+A,3+A
```

図A-25. 2つのWRITE CASEトランザクション

図A-25の変数が下記の値を持つとします。

```
2       XのReal32値
0.1     AのReal64値
```

VEEは次のように書き込みます。

```
Str2Str1
3.1
```

BINARYエンコーディング

WRITE BINARYトランザクションは下記の形式を取ります。

```
WRITE BINARY ExpressionList DataType
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

*DataType*は下記の定義済みVEEデータ型のどれかです。

- BYTE - 8ビット符号なしバイト
- INT16 - 16ビット2の補数整数
- INT32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- REAL64 - 64ビットIEEE 754浮動小数点数
- STRING - ヌル終止文字列
- COMPLEX - 2個のREAL64と同じ
- PCOMPLEX -2個のREAL64と同じ
- COORD - 2個以上のREAL64と同じ

注記

VEE 5以下の実行モードでは、整数値はすべてINT32データ型として、実数値はすべてRealデータ型(REAL64とも呼ぶ)として格納され、操作されます。したがって、INT16とREAL32のデータ型はI/O専用です。VEE 5以下の実行モードでは、機器I/Oの出力トランザクションで下記のデータ型変換が実行されます。

INT32の値は、個々にINT16の値に変換され、機器に出力されます。ただし、INT16データ型の範囲は-32768～32767なので、範囲外の値は16ビットに切り捨てられます。

REAL64の値は、個々にREAL32の値に変換され、機器に出力されます。ただし、REAL32データ型の範囲はREAL64より狭いので、範囲外の値はREAL32に変換できず、エラーになります。

VEE 6実行モードでは、データは適切な型に変換され、データが範囲外の場合にはエラーが発生します。

BINARYエンコーディングのトランザクションでは、*ExpressionList*に指定された値のそれぞれが、*DataType*で指定されるVEEデータ型に変換されます。変換されたアイテムはそれぞれ指定されたバイナリ・フォーマットで書き込まれます。ただし、書き込まれるバイナリ・データはコンピュータ・メモリ上の表現のコピーなので、異なるコンピュータ・アーキテクチャやハードウェアで利用するには手間がかかります。

BINARYエンコーディングのデータの利点は、非常にサイズが小さいことです。READ BINARYトランザクションを使えば、対応するWRITE BINARYで書き込まれたデータを読み取ることができます。

BINARYエンコーディングでは、各データ型の数値部分だけが書き込まれます。例えば、ComplexデータやCoordデータをTEXTエンコーディングで書き込む際に出力される括弧やカンマは、BINARYエンコーディングでは書き込まれません。

同様に、BINARYエンコーディングで配列を書き込む場合、Array Separatorsは書き込まれません。WRITE BINARYトランザクションでEOL ONを指定することは可能です。数値データ型は固定長であり、文字列はヌルで終わるので、BINARYトランザクションでEOLを指定することはほとんどありません。

BINBLOCKエンコーディング

WRITE BINBLOCKトランザクションは下記の形式を取ります。

```
WRITE BINBLOCK ExpressionList DataType
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

*DataType*は下記の定義済みVEEデータ型のどれかです。

- BYTE - 8ビット符号なしバイト
- INT16 - 16ビット2の補数整数
- INT32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- REAL64 - 64ビットIEEE 754浮動小数点数
- COMPLEX - 2個のREAL64と同じ
- PCOMPLEX -2個のREAL64と同じ
- COORD - 2個以上のREAL64と同じ

BINBLOCKは、*ExpressionList*の**各アイテム**を別々のデータ・ブロックとして書き込みます。用いられるブロック・ヘッダは、WRITEを実行するオブジェクトのタイプと、オブジェクトの構成に依存します。

GPIB以外のBINBLOCK

GPIBへのDirect I/O以外のオブジェクトの場合、WRITE BINBLOCKは常にIEEE 488.2の固定長任意ブロック応答データ・ブロックを書き込みます。このデータ・フォーマットは主にDirect I/Oを使ってGPIB機器と通信するために用いられますが、他のオブジェクトでもサポートされます。

固定長任意ブロックは下記の形式を取ります。

```
#<Num_digits><Num_bytes><Data>
```

ここで:

#はそのまま#文字です。

<Num_digits>は1個の数字(10進表記)を表すASCII文字で、<Num_bytes>の桁数を表します。

<Num_bytes>は数字(10進表現)を表すASCII文字のリストで、このあとの<Data>のバイト数を表します。

<Data>は任意の8ビット・データ・バイトのシーケンスです。

GPIBのBINBLOCK

GPIBに対するDirect I/Oオブジェクトの場合、WRITE BINBLOCKトランザクションの動作はDirect I/O ConfigurationのConformanceとBinblockの設定に依存します。これらの設定にアクセスするには、メニューでI/O ⇒ Instrument Managerを選択します。

ConformanceをIEEE 488.2に設定すると、WRITE BINBLOCKは常にIEEE 488.2の固定長任意ブロック応答データ・ブロックを書き込みます。

ConformanceをIEEE 488に設定すると、Binblockで指定したタイプのヘッダが用いられます。Binblockには、IEEE 728 #A、#T、#Iのいずれかのブロック・ヘッダを指定できます。BinblockがNoneの場合、WRITE BINBLOCKはIEEE 488.2の固定長任意ブロック応答データ・ブロックを書き込みます。

IEEE 728のブロック・ヘッダは下記の形式を取ります。

```
#A<Byte_Count><Data>
#T<Byte_Count><Data>
#I<Data><END>
```

ここで:

#はそのままの文字です。

A、T、Iはそれぞれそのままの文字です。

<Byte_Count>は16ビットの符号なし整数を表す2バイトであり、このあとの<Data>のバイト数を表します(この値はVEEが自動的に計算します)。

<Data>は任意バイトのストリームです。

<END>は最後の送信データ・バイトとともにEOIがアサートされることを示します。

CONTAINERエンコーディング

WRITE CONTAINERトランザクションは下記の形式を取ります。

```
WRITE CONTAINER ExpressionList
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

WRITE CONTAINERトランザクションは、*ExpressionList*の各アイテムをVEEの特別なテキスト表現を使って書き込みます。

この表現では、書き込むデータ型に付随するVEEのすべての属性(形状、サイズ、名前など)が保持されます。すべてのWRITE CONTAINERデータは、READ CONTAINERを使って情報を一切失うことなく復元できます。

例えば、下記のトランザクションは

```
WRITE CONTAINER 1.2345
```

下記のように書き込みます。

```
(Real
(data 1.2345)
)
```

STATEエンコーディング

WRITE STATEトランザクションは下記の形式を取ります。

```
WRITE STATE [DownloadString]
```

*DownloadString*は任意指定の文字列で、対応する機器に対するダイレクトI/O構成でダウンロード文字列が指定されていない場合に、ダウンロード文字列を指定する役割を果たします。これについてはこのあとで詳しく説明します。

WRITE STATEトランザクションは、Direct I/Oオブジェクトがラーン・ストリングを機器にダウンロードするために用いられます。Direct I/Oオブジェクトの各インスタンスに1つずつのラーン・ストリングが存在します。このラーン・ストリングは、Direct I/Oオブジェクト・メニューでUploadをクリックすることでアップロードされます。最初にUploadを選択するまでは、ラーン・ストリングはヌル文字列になっています。

WRITE STATEの動作は、Direct I/O ConfigurationのConformanceとDownload Stringの設定に影響されます。これらの設定にアクセスするには、メニューでI/O ⇒

Instrument Managerを選択します。ConformanceがIEEE 488の場合、WRITE STATEトランザクションはDownload Stringを書き込んでからラーン・ストリングを書き込みます。ConformanceがIEEE 488.2の場合、IEEE 488.2の定義に従って、ラーン・ストリングはプレフィックスなしでダウンロードされます。WRITE STATEトランザクションの詳細については、*Controlling Instruments with VEE*を参照してください。

REGISTERエンコーディング

WRITE REGISTERは、VXI機器のA16メモリに値を書き込むために用いられます。

WRITE REGISTERトランザクションは下記の形式を取ります。

```
WRITE REG: SymbolicName ExpressionList INCR
-または-
WRITE REG: SymbolicName ExpressionList
```

ここで:

*SymbolicName*は、VXI機器の構成中に指定される名前です。この名前は、機器のレジスタ空間の特定のアドレスを指します。WRITE REGISTERトランザクションで使用できるデータ型は下記の通りです。

- BYTE - 8ビット符号なしバイト
- WORD16 - 16ビット2の補数整数
- WORD32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- WORD32*2 - Int32配列の隣接する要素にある2個の32ビット2の補数整数
- REAL64 - 64ビット

これらのデータ型もVXI機器の構成中に指定され、トランザクションでは指定されません。

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

INCRは、*SymbolicName*で指定されるレジスタ・アドレスを先頭に配列データをインクリメンタルに書き込むことを指定します。配列の最初の要素は開始アドレスに書き込まれ、2番目の要素はデータ型のバイト数に等しいオフセットを加えたアドレスに書き込まれ、同様にしてすべての配列要素が書き込まれます。トランザクションにINCRを指定しないと、SymbolicNameで指定される1つの位置に配列全体が書き込まれます。

MEMORYエンコーディング

WRITE MEMORYは、VXI機器のA24またはA32メモリに値を書き込むために用いられます。

WRITE MEMORYトランザクションは下記の形式を取ります。

```
WRITE MEM: SymbolicName ExpressionList INCR
-または-
WRITE MEM: SymbolicName ExpressionList
```

ここで:

*SymbolicName*は、VXI機器の構成中に指定される名前です。この名前は、機器の拡張メモリの特定のアドレスを指します。WRITE MEMORYトランザクションで使用できるデータ型は下記の通りです。

- BYTE - 8ビット符号なしバイト
- WORD16 - 16ビット2の補数整数
- WORD32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- WORD32*2 - Int32配列の隣接する要素にある2個の32ビット2の補数整数
- REAL64 - 64ビットIEEE 754浮動小数点数

これらのデータ型もVXI機器の構成中に指定され、トランザクションでは指定されません。

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

INCRは、*SymbolicName*で指定されるメモリ位置を先頭に配列データをインクリメンタルに書き込むことを指定します。配列の最初の要素は指定位置に書き込まれ、2番目の要素はデータ型のバイト数に等しいオフセットを加えた位置に書き込まれ、同様にしてすべての配列要素が書き込まれます。トランザクションにINCRを指定しないと、*SymbolicName*で指定される1つのメモリ位置に配列全体が書き込まれます。

IOCONTROLエンコーディング

WRITE IOCONTROLトランザクションは下記の形式を取ります。

```
WRITE IOCONTROL CTL ExpressionList
-または-
WRITE IOCONTROL PCTL ExpressionList
```

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

IOCONTROLエンコーディングは、GPIOインタフェースに対するDirect I/Oでのみ用いられます。

このトランザクションは、GPIOインタフェースの制御ラインを設定します。

```
WRITE IOCONTROL CTL a
```

VEEはaの値をIntegerに変換します。Integer値の下位*X*ビットがインタフェースの制御ラインにマッピングされます(*X*は制御ラインの数)。

例えば、HP 98622AのGPIOインタフェースには2本の制御ラインCTL0とCTL1があります。表A-12を参照してください。

表A-12. HP 98622AのGPIO制御ライン

書込み値	CTL1	CTL0
0	0	0
1	0	1
2	1	0
3	1	1

表A-12で、1は制御ラインがアサートされることを、0はラインがクリアされることを示します。このトランザクションは、GPIOインタフェースのコンピュータ側からのハンドシェーク・ラインを制御します。

```
WRITE IOCONTROL PCTL a
```

aの値が0以外の場合、PCTLラインがセットされます。値が0の場合、何の動作も行われません。周辺機器がハンドシェーク要件を満たすと、PCTLはインタフェースによって自動的にクリアされます。

READトランザクション

表A-13にReadエンコーディングおよびフォーマットを示します。

表A-13. READ**エンコーディングおよびフォーマット**

エンコーディング	フォーマット
TEXT	CHAR TOKEN STRING QUOTED STRING INT16 INT32 OCTAL HEX REAL32 REAL64 COMPLEX PCOMPLEX COORD TIME STAMP
BINARY	STR BYTE INT16 INT32 REAL32 REAL64 COMPLEX PCOMPLEX COORD
BINBLOCK	BYTE INT16 INT32 REAL32 REAL64 COMPLEX PCOMPLEX COORD
CONTAINER	使用不可
IOSTATUS	使用不可

表A-13. READ**エンコーディングおよびフォーマット**

エンコーディング	フォーマット
REGISTER[a]	BYTE WORD16 WORD32 REAL32
MEMORY[a]	BYTE WORD16 WORD32 WORD32*2 REAL32 REAL64

a. Direct I/OからVXIのみ

TEXTエンコーディング

READ TEXTトランザクションは、無関係なものを読み飛ばし、重要なものだけを選択的に読み取る機能を備えています。この機能はほとんどの場合うまくいきますが、場合によってはVEEが何を無関係と見なし、何を重要と見なすかを注意して考えなければならないこともあります。

VEEで書き込んだテキスト・ファイルを、書込みのときと同じフォーマットで読み取る場合には、問題はごくまれにしか生じません。最も問題が起きやすいのは、他のソフトウェア・アプリケーションのファイルをインポートしようとする場合です。

表A-14に示すREAD TEXTの動作は、一般的なものだけです。可能な動作をすべて理解するには、このあとの部分をすべて読んでください。

表A-14. READ TEXT **トランザクションのフォーマット**

フォーマット	説明
CHAR	**任意の**8ビット文字を読み取ります。
TOKEN	連続した文字のリストを、**トークン**と呼ばれる単位で読み取ります。トークンはユーザが指定したデリミタ文字で区切られます。例えば、通常の英語のテキストでは、ワードがトークン、スペースがデリミタです。
STRING	8ビット文字のリストを1つの単位として読み取ります。ほとんどの制御文字は読み捨てられます。文字列の終わりは、指定した数の文字が読み取られるか、改行文字が現れたときです。
QSTRING	IEEE 488.2の任意長文字列に適合する8ビット文字のリストを読み取ります。これは先頭と末尾が2重引用符文字(ASCIIの10進34)で定義されるものです。制御文字は捨てられません。エスケープ文字は対応する制御文字に展開されます。2重引用符文字(ASCIIの10進34)が読み取られると文字列は終了します。
INTEGER16	文字のリストを読み取り、16ビット整数の10進または非10進表現として解釈します。10進INTEGERの構成要素として認識される文字は0123456789-+だけです。プレフィックス0x(16進)と、IEEE 488.2で規定されている非10進数値フォーマット#H(16進)、#Q(8進)、#B(2進)が認識されます。
INTEGER32	文字のリストを読み取り、32ビット整数の10進または非10進表現として解釈します。10進INTEGERの構成要素として認識される文字は0123456789-+だけです。プレフィックス0x(16進)と、IEEE 488.2で規定されている非10進数値フォーマット#H(16進)、#Q(8進)、#B(2進)が認識されます。
OCTAL	文字のリストを読み取り、整数の8進表現として解釈します。OCTALの構成要素として認識される文字は01234567です。8進数を表すIEEE 488.2の非10進数値プレフィックス#Qも認識されます。

表A-14. `READ TEXT` **トランザクションのフォーマット**

フォーマット	説明
HEX	文字のリストを読み取り、整数の16進表現として解釈します。HEXの構成要素として認識される文字は0123456789abcdefABCDEFだけです。デフォルトのプレフィックスは0xという文字の組合わせで、これは数値の一部と見なされず、読み捨てられます。0xの他に、16進数を表すIEEE 488.2の非10進数値プレフィックス#Hも認識されます。
REAL32	文字のリストを読み取り、32ビットReal数(浮動小数点)の10進表現として解釈します。先頭の符号、符号付き指数、小数点など、一般的な表記法はすべて認識されます。REAL32の構成要素として認識される文字は0123456789-+.Eeです。 いくつかの文字はReal数のサフィックス・マルチプライアとして認識されます(516ページの表A-15を参照)。
REAL64	文字のリストを読み取り、64ビットReal数(浮動小数点)の10進表現として解釈します。先頭の符号、符号付き指数、小数点など、一般的な表記法はすべて認識されます。REALの構成要素として認識される文字は0123456789-+.Eeです。 いくつかの文字はReal数のサフィックス・マルチプライアとして認識されます(516ページの表A-15を参照)。
COMPLEX	2個のREAL64に相当するものを読み取り、複素数として解釈します。最初に読み取った数値が実数部、2番目に読み取った数値が虚数部になります。

表A-14. `READ TEXT` **トランザクションのフォーマット**

フォーマット	説明
`PCOMPLEX`	2個のREAL64に相当するものを読み取り、極座標形式の複素数として解釈します。一部の工学分野で「フェーザ記法」と呼ばれるものにあたります。最初に読み取った数値が大きさ、2番目に読み取った数値が角度になります。位相の測定単位をトランザクションで指定できます。
`COORD`	2個以上のREAL64に相当するものを読み取り、直交座標として解釈します。
`TIME STAMP`	日付と時刻を表すVEEで規定されたタイムスタンプ形式のどれかを読み取ります。

READ TEXTに関する一般的注記

READ TO END. `READ TEXT` フォーマットでは、指定した数の要素を読み取るか、EOFに達するまで読み取るかを選択できます。トランザクション中の*`NumElements`*は、1つの式またはカンマで区切った式のリストで、*`VarList`*の中の各変数の次元を指定します。

最初の式がアスタリスク(`*`)の場合、トランザクションはEOFに達するまでデータを読み取ります。このREAD TO END動作は下記のトランザクションでのみサポートされます。

- `From File`
- `From String`
- `From StdIn`
- `Execute Program`
- `To/From Named Pipe`
- `To/From Socket`
- `To/From Rocky Mountain Basic`

数値の代わりにアスタリスクを指定できるのは最初の次元だけです。

例えば、ファイルから読み取る下記のトランザクションを考えます。

```
READ TEXT a REAL ARRAY:*,10
```

このトランザクションはEOFに達するまで読み取り、10列の2次元配列を生成します。行の数はファイル中のデータの量によって決まります。読み取ったデータ要素

の総数は、既知の次元サイズの積(この例では10)で割り切れる必要があります。この条件が満たされない場合、エラーが発生します。

読み取る文字数. 下記のREAD TEXTフォーマットでは、DEFAULT NUM CHARSとMAX NUM CHARSのどちらかを選択できます。

```
STRING
INT16
INT32
OCTAL
HEX
REAL32
REAL64
```

このセクションでは、これらのフォーマットでのDEFAULT NUM CHARSとMAX NUM CHARSの効果について説明します。

DEFAULT NUM CHARSとMAX NUM CHARSの基本的な違いは下記の通りです。

- DEFAULT NUM CHARSを使うと、読取り対象の数値や文字列の一部でないと見なされる大半の文字が読み捨てられます。
- MAX NUM CHARSを使うと、指定した数の8ビット文字が**読み取られ**、それに基づいて指定した型の数値または文字列が生成されます。指定した数の文字が読み取られると、ただちに読取りが停止します。すべての文字が読み取られ、その結果がトランザクションで指定されたデータ型に変換されます。

DEFAULT NUM CHARSを指定した場合、すべての変数を満たすのに必要なだけの文字が読み取られます。指定したデータ型に不要な文字は読み捨てられます。

MAX NUM CHARSを指定した場合、指定したデータ型に必要な文字かどうかのチェックは行われません。不要な文字があった場合、そのまま読み取られ、あとでエラーを起こす可能性があります。

どちらの場合も、改行文字とEOFは文字列と数値の両方を終了します。数値フォーマットの場合、意味のある文字(数字)より前にあるホワイトスペースは読み捨てられます。意味のある文字を読み取ったあとでホワイトスペースや数字以外の文字が現れると、現在のREADが終了します。これは一般的な動作です。詳細については、このあとの例を参照してください。

この例は、INT32フォーマットでのDEFAULT NUM CHARSとMAX NUM CHARSの違いを示すためのものです。下記のデータを含むファイルを読み取るとします。

```
bird dog cat 12345 horse
```

READ TEXT INT32トランザクションでMAX NUM CHARSを使う場合、読み取る文字数をいくつにしたとしても、整数12345を取り出すのは不可能です。bird dog catという文字が必ず数字の前に読み取られ、これらはIntegerに変換できないため、エラーが発生するからです。

DEFAULT NUM CHARSを使えば、bird dog catは読み捨てられ、5のあとのホワイトスペースがデリミタと見なされるので、整数12345を取り出すことができます。

引用符付き文字列の効果. 引用符付き文字列があると、すべてのI/Oパスに対するREAD TEXT QSTRおよびREAD TEXT TOKENと、機器またはインタフェースI/Oに対するREAD TEXT STRINGの動作に影響します。ここで、引用符付き文字列とは、先頭と末尾に2重引用符文字があり、間に2重引用符(エスケープされたものは除く)が存在しない文字群のことです。2重引用符文字はASCIIの10進34です。上記のREADトランザクションで2重引用符文字があると、文字列とトークンに文字がグループ化される方法と、文字列内部の制御文字とエスケープ文字の処理方法が変わります。

この説明では、**制御文字**および**エスケープ文字**という用語を特別な意味に使用します。制御文字とは、ASCII文字の10進0～31に対応する1バイトのデータを表します。例えば、ラインフィードはASCIIの10進10であり、この説明では記号<LF>でラインフィード文字を表します。文字列\nはラインフィード文字を表す人間が読めるエスケープ文字であり、VEEによって認識されます。

一部のトランザクションでは、引用符付き文字列の扱いがI/Oパスによって異なります。READ TEXT QSTRは、機器I/O以外のI/Oパスの場合、引用符付き文字列を特別な方法で処理します。READ TEXT STRINGは、機器I/O以外のI/Oパスの場合、引用符付き文字列を認識しません。

機器I/Oに対してはREAD TEXT QSTRトランザクションが存在しません。その代わりに、READ TEXT STRINGが引用符付き文字列を認識して処理します。これは、引用符付き文字列がIEEE 488.2仕様で特別な意味を持つからです。READ TEXT TOKENは、機器を含むすべてのI/Oパスに対して引用符付き文字列を特別な方法で処理します。以下の説明では、I/OパスがファイルI/Oの場合を扱います。

文字列の先頭と末尾に2重引用符がない場合、READ TEXT STRINGトランザクションと、SPACE DELIMを指定したREAD TEXT TOKENトランザクションでは、ラインフィード以外の制御文字は読み捨てられます。STRINGとTOKENのどちらのトランザクションでも、ラインフィードがあるとREADが終了します。\n(改行)などのエスケープ文字シーケンスは、単に\とnの2文字として読み取られます。

2重引用符付き文字列の中では、READ TEXT QSTRとREAD TEXT TOKENは引用符に囲まれたすべての文字(制御文字を含む)を読み取り、入力変数に格納します。文字列内部にラインフィードがあっても、他の文字と同様に読み取られ、現在のREADは終了しません。エスケープ文字シーケンスは読み取られ、対応する1文字に変換されます。

グループ化の動作については、例で説明するのがわかりやすいでしょう。このセクションの残りの部分では、図A-26に示す内容を持つファイルからデータを読み取るものとします。

```
"This is in quotes."  This is not.
```

図A-26. 引用符付きのデータと引用符なしのデータ

図A-26のファイルを、From Fileで下記のトランザクションを使って読み取るとします。

```
READ TEXT x QSTR
READ TEXT y QSTR
```

ファイルを読み取ると、結果は下記のようになります。

```
x = This is in quotes.
y = This is not.
```

2重引用符はデリミタとして解釈され、入力変数には格納されません。

今度は、From Fileで下記のトランザクションを使って図A-26のファイルを読み取るとします。

```
READ TEXT x QSTR MAXFW:4
READ TEXT y QSTR
```

ファイルを読み取ると、結果は下記のようになります。

```
x = This
y = This is not.
```

ここでも、2重引用符はデリミタの役割を果たしています。最初のトランザクションは2重引用符から2重引用符までを読み取り、最初の4文字をxに格納します。これにより、ファイルの読取りポインタは2番目のThisの前に来ます。2番目のトランザクションでは前の例と同じ文字列が読み取られます。

次は、From Fileで下記のトランザクションを使って図A-26のファイルを読み取るとします。

```
READ TEXT x TOKEN
READ TEXT y QSTR
```

ファイルを読み取ると、結果は下記のようになります。

```
x = This is in quotes.
y = This is not.
```

ここでは、2重引用符のために最初のセンテンス全体が1つのトークンと見なされています。デフォルトのトークン・デリミタはホワイトスペースですが、引用符付き文字列は全体が1つのトークンとして扱われます。さらに、TOKENは2重引用符を読み捨てます。

CHARフォーマット

READ TEXT CHARトランザクションは下記の形式を取ります。

```
READ TEXT VarList CHAR:NumChar ARRAY:NumStr
```

*VarList*は、1個のText変数またはカンマで区切ったText変数のリストです。

*NumChar*は、*VarList*の各変数の各要素を満たすために読み取る8ビット文字の数を指定します。

*NumStr*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。

ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

CHARフォーマットは、一度に1文字だけ読み取る場合、あるいは入力データの**すべて**の文字を無視せずに読み取る場合に有用です。

下記のトランザクションは、2個の2次元Text配列を読み取ります。各配列の各要素には2個の文字が格納されます。

```
READ TEXT X,Y CHAR:2 ARRAY:2,2
```

上記のトランザクションで、下記の内容のファイルを読み取るとします。

```
<space>ABCDEFG"AB"<LF>'CD
```

READが終わると、変数XとYには下記の値が格納されます。

```
X [0 0] = <space>A
X [0 1] = BC
X [1 0] = DE
X [1 1] = FG

Y [0 0] = "A
Y [0 1] = B"
Y [1 0] = <LF>'
Y [1 1] = CD
```

記号<space>は、1文字のスペース(ASCIIの10進32)を表します。記号<LF>は、1文字のラインフィード(ASCIIの10進10)を表します。スペース、ラインフィード、2重引用符は、特別な扱いや解釈を受けることなく読み取られます。

TOKENフォーマット

READ TEXT TOKENトランザクションは下記の形式を取ります。

```
READ TEXT VarList TOKEN Delimiter ARRAY:NumElements
```

*VarList*は、1個のText変数またはカンマで区切ったText変数のリストです。

*Delimiter*は、トークンを終了する文字の組合わせ(デリミタ)を指定します。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

TOKENフォーマットでは、*Delimiter*として下記のどれかを選択することにより、トークンのデリミタ(境界)を指定できます。

- SPACE DELIM
- INCLUDE CHARS
- EXCLUDE CHARS

デリミタに関する以下の説明では、図A-27の内容を持つファイルを読み取る際に、デリミタの選択によってどのような違いが生じるかを示します。

```
A phrase.
"A phrase."
Tab follows.
XOXXOOXXXOOOXXXX
XAXXBCXXXDEF
```

図A-27. READ TOKEN**のデータ**

このファイルには文字Oはありますが、数字の0はありません。

図には示されていませんが、図A-27のファイルの最初の4行の末尾にはラインフィード文字があります。この図では、viなどのテキスト・エディタと同じようにファイルを表示しています。

SPACE DELIM. SPACE DELIMを使用すると、トークンはホワイトスペースで区切られます。ホワイトスペースには、スペース、タブ、改行、EOFが含まれます。これはおおむね英語のテキストのワードに対応します。SPACE DELIMを使用すると、テキストのパラグラフを含むファイルを読み取って、個々のワードを切り出すことができます。

2重引用符付き文字列は特別な扱いを受けます。2重引用符で囲まれた文字列は1個のトークンとして読み取られ、2重引用符は除去されます。2重引用符で囲まれた文字列の内部にある制御文字(ASCIIの10進0～31)は、出力変数に返されます。2重引用符で囲まれた文字列の内部にあるエスケープ文字(\nなど)は、対応する制御文字に変換されます。

2重引用符で囲まれた文字列に対するこの特別な処理は、SPACE DELIMトランザクションにのみ当てはまります。INCLUDE CHARSおよびEXCLUDE CHARSでは、2重引用符、エスケープ文字、制御文字は他の文字と同様に処理されます。

図A-27のデータをSPACE DELIMを使って下記のトランザクションで読み取るとします。

```
READ TEXT a TOKEN ARRAY:8
```

変数には下記の値が格納されます。

```
a[0] = A
a[1] = phrase.
a[2] = A phrase.
a[3] = Tab
a[4] = follows
a[5] = .
a[6] = XOXXOOXXXOOOXXXX
a[7] = XAXXBCXXXDEF
```

INCLUDE CHARS. INCLUDE CHARSを使うと、READが返すトークンに含まれるべき文字のリストを指定することができます。指定した文字だけがトークンとして返されます。指定したINCLUDE文字以外の文字があると、現在のトークンが終了します。終了文字はトークンに含まれず、捨てられます。

図A-27のデータをINCLUDE CHARSを使って下記のトランザクションで読み取るとします。

```
READ TEXT a TOKEN INCLUDE:"X" ARRAY:7
```

変数には下記の値が格納されます。

```
a[0] = X
a[1] = XX
a[2] = XXX
a[3] = XXXX
a[4] = X
a[5] = XX
a[6] = XXX
```

図A-27のデータをINCLUDE CHARSを使って下記のトランザクションで読み取るとします。

```
READ TEXT a TOKEN INCLUDE:"OXZ" ARRAY:4
```

変数には下記の値が格納されます。

```
a[0] = XOXXOOXXXOOOXXXX
a[1] = X
a[2] = XX
a[3] = XXX
```

INCLUDEリストの最初の文字は英文字のOであり、数字の0ではありません。

図A-28のデータを含むファイルを読み取るとします。

```
111 222 333 444 555
```

図A-28. READ TOKEN**のデータ**

図A-28のファイルを下記のトランザクションで読み取るとします。

```
READ TEXT x,y,z TOKEN INCLUDE:"1234567890"
```

Text変数x、y、zには下記の値が格納されます。

```
x = 111
y = 222
z = 333
```

もう1つの方法として、ARRAYで1より大きい数を指定し、配列に値を読み取ることもできます。例えば、図A-28のデータを下記のトランザクションで読み取るとします。

```
READ TEXT x TOKEN INCLUDE:"1234567890" ARRAY:3
```

Text変数xには下記の値が格納されます。

```
x[0] = 111
x[1] = 222
x[2] = 333
```

EXCLUDE CHARS. EXCLUDE CHARSを使うと、文字のリストを指定して、その中のどれかの文字が読み取られたときに現在のトークンが終了するようにできます。終了文字はトークンに含まれず、読み捨てられます。

図A-27のデータをEXCLUDEを使って下記のトランザクションで読み取るとします。

```
READ TEXT a TOKEN EXCLUDE:"X" ARRAY:8
```

変数には下記の値が格納されます。

```
a[0] = A phrase.<LF>"A phrase."<LF>Tab follows .<LF>
a[1] = O
a[2] = OO
a[3] = OOO
a[4] = <LF>
a[5] = A
a[6] = BC
a[7] = DEF<LF>
```

図A-29のデータが機器からVEEに送られたとします。

```
++1.23++4.98++0.45++2.34++0.01++23.45++12.2++
```

図A-29. READ TOKEN**のデータ**

図A-29のデータを下記のトランザクションで読み取るとします。

```
READ TEXT x TOKEN EXCLUDE:"+" ARRAY:7
```

変数xには下記の値が格納されます。

```
x[0] = ヌル文字列(空)
x[1] = 1.23
x[2] = 4.98
x[3] = 0.45
x[4] = 2.34
x[5] = 0.01
x[6] = 23.45
```

7個の数値があるにもかかわらず、6個しか読み取られていません。このトランザクションが終わった時点で、VEEは+で終わる7個のトークンを読み取っていますが、最初のものはデータが来る前に終了しています。

注記

EXCLUDE CHARSの動作は、VEE 5実行モード以降とVEE 4実行モード以前では異なっています。この違いについては33ページ「READ TEXTトランザクション」を参照してください。

STRINGフォーマット

READ TEXT STRINGトランザクションは下記の形式を取ります。

```
READ TEXT VarList STR ARRAY:NumElements
-または-
READ TEXT VarList STR MAXFW:NumChars ARRAY:NumElements
```

*VarList*は、1個のText変数またはカンマで区切ったText変数のリストです。

*NumChars*は、文字列を作成する際に読み取ることができる8ビット文字の最大数を指定します。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

このトランザクションは、入力文字をすべて読み取り、文字列を返します。先頭のスペースは削除されます。以下の説明は、GPIB、VXIなどの機器I/Oパスだけを対象とします。ファイルや名前付きパイプなど他のI/Oパスでは、引用符付き文字列は特

別な扱いを受けません。2重引用符付き文字列に対するREAD TEXT STRINGの動作の詳細については、504ページ「引用符付き文字列の効果」を参照してください。

制御文字とエスケープ文字の効果. この説明では、**制御文字**および**エスケープ文字**という用語を特別な意味に使用します。制御文字とは、ASCII文字の10進0～31に対応する1バイトのデータを表します。例えば、ラインフィードはASCIIの10進10であり、この説明では記号<LF>でラインフィード文字を表します。文字列\nはラインフィード文字を表す人間が読めるエスケープ文字であり、VEEによって認識されます。引用符で囲まれた文字列の中では、VEEはエスケープ文字を使って制御文字を表します。

制御文字とエスケープ文字の扱いは、2重引用符付き文字列の中にあるかどうかで異なります。

2重引用符付き文字列の外では、ラインフィード以外の制御文字は読み捨てられます。ラインフィードは現在の文字列を終了します。\nなどのエスケープ文字は、2個の独立した文字(\とn)として読み取られます。

2重引用符付き文字列の中では、制御文字とエスケープ文字は読み取られ、READが返す文字列に含められます。2重引用符付き文字列の中にラインフィードがあっても、現在の文字列は終了しません。\nなどのエスケープ文字は、対応する1文字(<LF>)として解釈され、返される文字列に制御文字として含められます。

下記の文字列データをREAD TEXT STRINGトランザクションで読み取る場合を考えます。

```
Simple string.
Random \n % $ * 'A'
"In quotes."
"In quotes
with control."
"In quotes\nwith escape."
```

下記のトランザクションで読み取るとします。

```
READ TEXT x STR ARRAY:5
```

変数xには下記の値が格納されます。

```
a[0] = Simple string.
a[1] = Random \n % $ * 'A'
a[2] = In quotes.
a[3] = In quotes<LF>with control.
a[4] = In quotes<LF>with escape.
```

同じ文字列データを下記のトランザクションで読み取るとします。

```
READ TEXT x STR MAXFW:16 ARRAY:5
```

変数xには下記の値が格納されます。

```
a[0] = Simple string.
a[1] = Random \n % $ *
a[2] = 'A'
a[3] = In quotes.
a[4] = In quotes<LF>with c
```

現在のREADが終了するのは、16文字を読み取ったか(a[1])、引用符の外で<LF>を読み取った場合(a[2])です。2重引用符付き文字列は、2重引用符から2重引用符まで読み取られ、引用符の中の最初の16文字が返されます(a[4])。

QUOTED STRINGフォーマット

READ TEXT QUOTED STRINGトランザクションは下記の形式を取ります。

```
READ TEXT VarList QSTR ARRAY:NumElements
-または-
READ TEXT VarList QSTR MAXFW:NumChars ARRAY:NumElements
```

*VarList*は、1個のText変数またはカンマで区切ったText変数のリストです。

*NumChars*は、文字列を作成する際に読み取ることができる8ビット文字の最大数を指定します。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

このトランザクションは、入力文字をすべて読み取り、文字列を返します。以下の説明は、機器I/O以外のパスを対象とします。機器I/OパスにはREAD TEXT QSTRトランザクションは実装されていません。2重引用符付き文字列に対するREAD TEXT STRINGの動作の詳細については、504ページ「引用符付き文字列の効果」を参照してください。

また、513ページ「制御文字とエスケープ文字の効果」も参照してください。

INT16およびINT32フォーマット

READ TEXT INT16およびREAD TEXT INT32トランザクションは下記の形式を取ります。

```
   READ TEXT VarList INT16(またはINT32) ARRAY:NumElements
   -または-
   READ TEXT VarList INT16(またはINT32) MAXFW:NumChars
ARRAY:NumElements
```

*VarList*は、1個のInteger変数またはカンマで区切ったInteger変数のリストです。

*NumChars*は、数値を作成する際に読み取ることができる8ビット文字の最大数を指定します。

*NumStr*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

READ TEXT INT16トランザクションは、入力文字を16ビットの2の補数整数として解釈します。この整数の有効範囲は32767～-32768です。範囲外の値は循環するため、オーバーフローは発生しません。例えば、32768は-32768と解釈されます。

READ TEXT INT32トランザクションは、入力文字を32ビットの2の補数整数として解釈します。この整数の有効範囲は2 147 483 647～-2 147 483 648です。範囲外の値は循環するため、オーバーフローは発生しません。例えば、2 147 483 648は-2 147 483 648と解釈されます。

数値を作成する前に、数値の一部と見なされない先頭の文字は捨てられます。数値の作成が始まったあとで、数値の一部と見なされない文字が来ると、その数値に対するREADが終了します。表A-15に、INT16またはINT32の一部と見なされる文字の組合わせを示します。

表A-15. INT16またはINT32の一部と見なされる文字

表記法	認識される文字
10進	有効な文字は±0123456789です。VEEのデータ入力フィールドの場合と異なり、先頭の0は8進数のプレフィックスとは見なされません。
VEEの16進	0xは16進数のプレフィックスと見なされます。プレフィックスのあとで有効な文字は0123456789aAbBcCdDeEfFです。
IEEE 488.2の2進	#bまたは#Bは2進数のプレフィックスと見なされます。プレフィックスのあとで有効な文字は0と1です。
IEEE 488.2の8進	#qまたは#Qは8進数のプレフィックスと見なされます。プレフィックスのあとで有効な文字は01234567です。
IEEE 488.2の16進	#hまたは#Hは16進数のプレフィックスと見なされます。プレフィックスのあとで有効な文字は0123456789aAbBcCdDeEfFです。

以下の表記はすべて、10進15の整数値と見なされます。

```
15
+15
015
0xF
0xf
#b1111
#Q17
#hF
```

OCTALフォーマット

`READ TEXT OCTAL`トランザクションは下記の形式を取ります。

```
READ TEXT VarList OCT ARRAY:NumElements
-または-
READ TEXT VarList OCT MAXFW:NumChars
```

ここで:

`ARRAY:`*`NumElements`*

*`VarList`*は、1個のInteger変数またはカンマで区切ったInteger変数のリストです。

*`NumChars`*は、数値を作成する際に読み取ることができる8ビット文字の数を指定します。

*`NumElements`*は、1つの式またはカンマで区切った式のリストであり、*`VarList`*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、`ARRAY`キーワードはトランザクションに用いられません。`ARRAY:1`は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

`READ TEXT OCTAL`トランザクションは、入力文字を32ビットの2の補数整数を表す8進数字として解釈します。この整数の有効範囲は10進の2 147 483 647～-2 147 483 648です。

トランザクションに`MAX NUM CHARS (MAXFW)`が指定されている場合、読み取った8進数には32ビットより多くのデータが含まれる場合があります。例えば、下記の8進データを読み取ることを考えます。

```
377237456214567243777
```

下記のトランザクションを使うとします。

```
READ TEXT x OCT MAXFW:21
```

VEEは8進データのすべての数字を読み取りますが、最後の11桁(`14567243777`)だけを使って`x`の値の数値を作成します。これは、1桁の数字が3ビットを表し、8進数はVEEのInteger(32ビット)に格納できなければならないからです。11桁の8進数は33ビットになるので、最上位ビットが捨てられて、VEEの整数に格納されます。オーバーフローは発生しません。

トランザクションにDEFAULT NUM CHARSが指定されている場合、*VarList*のすべての変数を満たす数が作成されるまで文字が読み取られます。ラインフィード文字によって数値の作成が途中で終了することはありません。例えば、下記のトランザクションを考えます。

```
READ TEXT x OCT ARRAY:4
```

このトランザクションは、下記の8進データの各行を同じ4個の8進数の組と解釈します。

```
0345 067 003<LF>0377<LF>
345 67 3 377<EOF>
345,67,3,377,45,67<EOF>
```

記号<LF>は1個のラインフィード文字(ASCIIの10進10)を表します。記号<EOF>はEOF条件を表します。

HEXフォーマット

READ TEXT HEXトランザクションは下記の形式を取ります。

```
READ TEXT VarList HEX ARRAY:NumElements
-または-
READ TEXT VarList HEX MAXFW:NumChars ARRAY:NumElements
```

*VarList*は、1個のInteger変数またはカンマで区切ったInteger変数のリストです。

*NumChars*は、数値を作成する際に読み取ることができる8ビット文字の数を指定します。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

READ TEXT HEXトランザクションは、入力文字を32ビットの2の補数整数を表す16進数字として解釈します。この整数の有効範囲は10進の2 147 483 647～-2 147 483 648です。

トランザクションにMAX NUM CHARS (MAXFW)が指定されている場合、読み取った16進数には32ビットより多くのデータが含まれる場合があります。例えば、下記の16進データを読み取ることを考えます。

```
ad2469Ff725BCdef37964    16進データ
```

下記のトランザクションを使うとします。

```
READ TEXT x HEX MAXFW:21
```

VEEは16進データのすべての数字を読み取りますが、最後の8桁(def37964)だけを使ってxの値の数値を作成します。これは、1桁の数字が4ビットを表し、16進数はVEEのInteger(32ビット)に格納できなければならないからです。8桁の16進数はちょうど32ビットになります。オーバーフローは発生しません。

同じ16進データを、異なるMAX NUM CHARSを使って、下記のトランザクションで読み取るとします。

```
READ TEXT x HEX MAXFW:3 ARRAY:7
```

この場合、トランザクションは同じデータを読み取り、それぞれ3桁の16進数字からなる7個のIntegerと解釈します。

トランザクションにDEFAULT NUM CHARSが指定されている場合、*VarList*のすべての変数を満たす数が作成されるまで文字が読み取られます。それぞれの数値は、**ちょうど**8桁の16進数字を読み取ります。ラインフィード文字によって数値の作成が途中で終了することはありません。

同じ16進データを、DEFAULT NUM CHARSを使って、下記のトランザクションで読み取るとします。

```
READ TEXT x HEX ARRAY:2
```

この場合、トランザクションは同じデータを読み取り、それぞれ8桁の16進数字からなる2個のIntegerと解釈します。最後の5個の数字(37946)は読み取られません。

REAL32およびREAL64フォーマット

READ TEXT REAL32およびREAD TEXT REAL64トランザクションは下記の形式を取ります。

```
READ TEXT VarList REAL32(またはREAL64) ARRAY:NumElements
-または-
READ TEXT VarList REAL32(またはREAL64) MAXFW:NumChars
ARRAY:NumElements
```

*VarList*は、1個のReal変数またはカンマで区切ったReal変数のリストです。

*NumChars*は、数値を作成する際に読み取ることができる8ビット文字の最大数を指定します。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

このトランザクションは10進数値を読み取り、VEEのReal32として解釈します。これは32ビットIEEE 754浮動小数点数です。この数値の範囲は下記の通りです。

```
-1.175 494 35E-38
-3.402 823 47E+38
～
3.402 823 47E+38
1.175 494 35E-38
```

このトランザクションは10進数値を読み取り、VEEのReal64として解釈します。これは64ビットIEEE 754浮動小数点数です。この数値の範囲は下記の通りです。

```
-1.797 693 134 862 315E+308
-2.225 073 858 507 202E-307
～
 2.225 073 858 507 202E-307
 1.797 693 134 862 315E+308
```

トランザクションにMAX NUM CHARS (MAXFW)が指定されている場合、読み取ったReal数には17桁より多くのデータが含まれる場合があります。例えば、下記の実数データを読み取ることを考えます。

```
1.234567890123456789    実数データ
```

下記のトランザクションを使うとします。

```
READ TEXT x REAL64 MAXFW:19
```

VEEは実数データのすべての数字を読み取りますが、仮数部の上位17桁だけを使ってxの値の数値を作成します。これは、Realの仮数部が54ビットで、10進16桁より多く、17桁より小さいからです。この結果、xの値は1.2345678901234567になります。

TextからRealへの変換では、最下位の桁まで一致する結果が得られることは保証されません。下位2ビットの比較は推奨されません。

同じ実数データを、異なるMAX NUM CHARSを使って、下記のトランザクションで読み取るとします。

```
READ TEXT x REAL64 MAXFW:6 ARRAY:3
```

この場合、トランザクションは同じデータを読み取り、それぞれ6桁の10進数字からなる3個のReal数と解釈します。最後の2個の文字は読み取られません。

トランザクションにDEFAULT NUM CHARSが指定されている場合、*VarList*のすべての変数を満たす数が作成されるまで文字が読み取られます。それぞれの数値は、最大17桁の10進数字を読み取ります。ラインフィード文字、ホワイトスペース、その他数値以外の文字があると、17桁を読み取る前でも数値の作成は終了します。

READ TEXT REAL64およびREAL32トランザクションは、先頭の符号、小数点、符号付き指数など、Real数の一般的な表記法を認識します。文字+-.0123456789Eeは、**すべての**READ TEXT REALトランザクションでReal数の有効な部分として認識されます。DEFAULT NUM CHARSがトランザクションに指定されている場合、表A-16に示すサフィックス文字も認識されます。サフィックス文字は、数値の最後の数字の直後に、ホワイトスペースを間におかずに存在する必要があります。

表A-16. `Real`**数のサフィックス**

サフィックス	乗数
P	10^{15}
T	10^{12}
G	10^{9}
M	10^{6}
kまたはK	10^{3}
m	10^{-3}
u	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}

下記のTextデータは、6個の実数を表します。

```
1001
+1001.
1001.0
1.001E3
+1.001E+03
1.001K
```

この実数テキスト・データを下記のトランザクションで読み取るとします。

```
READ TEXT x REAL64 ARRAY:6
```

この場合、Real変数xのすべての要素に値`1001`が入ります。

同じデータを下記のトランザクションで読み取るとします。

```
READ TEXT x REAL64 MAXFW:20 ARRAY:6
```

この場合、Real変数xの最初の5個の要素には値`1001`が、6番目の要素には値`1.001`が入ります。

COMPLEX、PCOMPLEX、COORDフォーマット

COMPLEX、PCOMPLEX、COORDは、同じ名前のVEEのマルチフィールド・データ型に対応します。これら3つのREADフォーマットの動作はよく似ています。このセクションで説明する動作は、特に記す場合を除いて3つのフォーマットすべてに当てはまります。

VEEデータ型のComplex、PComplex、Coordが複数のReal64数から構成されるのに対応して、COMPLEX、PCOMPLEX、COORDの各フォーマットはREAL64フォーマットの複合形式です。マルチフィールド・データ型の構成要素のReal値は、個々の値をREAL64フォーマットで入力した場合と同じ規則に従って読み取られます。

COMPLEXフォーマット. READ TEXT COMPLEXトランザクションは下記の形式を取ります。

```
READ TEXT VarList CPX ARRAY:NumElements
```

READ TEXT COMPLEXトランザクションは、2個のREALフォーマット数値に相当するものを読み取ります。最初に読み取った数値が実数部、2番目に読み取った数値が虚数部と解釈されます。

PCOMPLEXフォーマット. READ TEXT PCOMPLEXトランザクションは下記の形式を取ります。

```
READ TEXT VarList PCX:PUnit ARRAY:NumElements
```

*PUnit*は、PComplexの位相角の測定単位を指定します。

READ TEXT PCOMPLEXトランザクションは、2個のREALフォーマット数値に相当するものを読み取ります。最初に読み取った数値が大きさ、2番目に読み取った数値が位相と解釈されます。

COMPLEX、PCOMPLEX、COORDの各フォーマットを読み取るトランザクションで開き括弧が見つかった場合、閉じ括弧が存在するものと見なされます。

括弧を含む下記の内容を持つファイルを読み取る場合を考えます。

```
(1.23 , 3.45  (6.78 , 9.01) (1.23 , 4.56)
```

下記のトランザクションでデータを読み取るとします。

```
READ TEXT x,y CPX
```

変数xとyには下記のComplex値が格納されます。

```
x = (1.23 , 3.45)
y = (1.23 , 4.56)
```

このトランザクションは、閉じ括弧を探しながら6.78と9.01を読み飛ばします。データに括弧が全くなければ、yの値は(6.78 , 9.01)になります。

COORDフォーマット. READ TEXT COORDトランザクションは下記の形式を取ります。

```
READ TEXT VarList COORD:NumFields ARRAY:NumElements
```

*VarList*は、1個のCoord変数またはカンマで区切ったCoord変数のリストです。

*NumFields*は1個の変数または式で、各Coord値の直交座標系の次元の数を示します。値が2以上でないと、READがエラーになります。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

BINARYエンコーディング

READ BINARYトランザクションは下記の形式を取ります。

```
READ BINARY VarList DataType ARRAY:NumElements
```

*VarList*は、1個の変数またはカンマで区切った変数のリストです。

*DataType*は下記の定義済みフォーマットの1つです。これらは同じ名前のVEEデータ型に対応します。

- BYTE - 8ビット符号なしバイト
- INT16 - 16ビット2の補数整数
- INT32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- REAL64 - 64ビットIEEE 754浮動小数点数
- STRING - ヌル終止文字列
- COMPLEX - 2個のREAL64と同じ

- PCOMPLEX -2個のREAL64と同じ
- COORD - 2個以上のREAL64と同じ

注記

VEE 5以下の実行モードでは、整数値はすべてINT32データ型として、実数値はすべてRealデータ型(REAL64とも呼ぶ)として格納され、操作されます。したがって、INT16とREAL32のデータ型はI/O専用です。VEE 5以下の実行モードでは、機器I/Oの入力トランザクションで下記のデータ型変換が実行されます。

VEE 5およびそれ以前の実行モードでは、機器からのINT16値は**個々に**INT32値に変換されます。この変換では、INT16データが**符号付きの**データであると仮定されます。**符号なし**のINT32データが必要な場合、下記の式を持つFormulaオブジェクトにデータを通します。

```
BITAND(a, 0xFFFF)
```

VEE 5およびそれ以前の実行モードでは、機器からのREAL32値は**個々に**REAL64値に変換されます。

VEE 6実行モードではデータ型は保持されます。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。最初の式がアスタリスク(*)の場合、トランザクションはEOFに達するまでデータを読み取ります。このREAD TO END動作は下記のトランザクションでのみサポートされます。

- From File
- From String
- From StdIn
- Execute Program
- To/From Named Pipe
- To/From Rocky Mountain Basic

数値の代わりにアスタリスクを指定できるのは最初の次元だけです。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

例えば、ファイルから読み取る下記のトランザクションを考えます。

```
READ BINARY a REAL64 ARRAY:*,10
```

このトランザクションはEOFに達するまで読み取り、10列の2次元配列を生成します。行の数はファイル中のデータの量によって決まります。読み取ったデータ要素の総数は、既知の次元サイズの積(この例では10)で割り切れる必要があります。

READ BINARYトランザクションは、対応するWRITE BINARYトランザクションで生成されるのと**正確に**同じフォーマットのデータが入力されると想定します。BINARYエンコーディングのデータはサイズが非常に小さいという利点がありますが、VEE以外のアプリケーションと共有するには手間がかかります。

BINBLOCKエンコーディング

READ BINBLOCKトランザクションは下記の形式を取ります。

```
READ BINBLOCK VarList DataType ARRAY:NumElements
```

*VarList*は、1個の変数またはカンマで区切った変数のリストです。

*DataType*は下記の定義済みVEEデータ型のどれかです。

- BYTE - 8ビット符号なしバイト
- INT16 - 16ビット2の補数整数
- INT32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- REAL64 - 64ビットIEEE 754浮動小数点数
- COMPLEX - 2個のREALと同じ
- PCOMPLEX -2個のREALと同じ
- COORD - 2個以上のREALと同じ

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。列の数は、Binblockに含まれるチャネルの数と同じです。行の数は、1つのチャネルに含まれる測定値の数と同じです。数値の代わりにアスタリスクを指定できるのは最初の次元だけです。

最初の式がアスタリスク(*)の場合、トランザクションはEOFに達するまでデータを読み取ります。このREAD TO END動作は下記のトランザクションでのみサポートされます。

- From File
- From String
- From StdIn
- Execute Program
- To/From Named Pipe
- To/From Socket
- To/From Rocky Mountain Basic

トランザクションが1次元配列(1チャネル)を読み取るように構成されている場合、ただ1つの次元が行数を表し、この次元にアスタリスクを指定できます。

例えば、ファイルから読み取る下記のトランザクションを考えます。

```
READ BINBLOCK a REAL64 ARRAY:*,10
```

このトランザクションはEOFに達するまで読み取り、10列の2次元配列を生成します。1つの列が1つの機器チャネルを表します。行の数は各チャネルのデータの量によって決まります。Binblock内のデータ要素の総数は、列の数(この例では10)で割り切れる必要があります。

入力データのフォーマットについての情報はブロック・ヘッダに記録されているので、特に指定する必要はありません。READ BINBLOCKは、WRITE BINBLOCKトランザクションの項で説明したすべてのブロック・フォーマットを読み取ることができます。

下記のトランザクションは、オシロスコープから2つのトレースを読み取ります。これらのトレースはIEEE 488.2の固定長任意ブロック応答データとしてフォーマットされています。

```
READ BINBLOCK a,b REAL64
```

CONTAINERエンコーディング

READ CONTAINERトランザクションは下記の形式を取ります。

```
READ CONTAINER VarList
```

*VarList*は、1個の変数またはカンマで区切った変数のリストです。

READ CONTAINERトランザクションは、WRITE CONTAINERトランザクションで書き込まれた特別なテキスト表現のデータを読み取ります。コンテナ自体に情報が含まれるため、フォーマットなどの指定はREAD CONTAINERには必要ありません。

REGISTERエンコーディング

READ REGISTERは、VXI機器のA16メモリから値を読み取るために用いられます。

READ REGISTERトランザクションは下記の形式を取ります。

```
READ REG: SymbolicName ExpressionList INCR ARRAY:NumElements
-または-
READ REG: SymbolicName ExpressionList ARRAY:NumElements
```

ここで:

*SymbolicName*は、VXI機器の構成中に指定される名前です。この名前は、機器のレジスタ空間の特定のアドレスを指します。READ REGISTERトランザクションで使用できるデータ型は下記の通りです。

- BYTE - 8ビット符号なしバイト
- WORD16 - 16ビット2の補数整数
- WORD32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数

これらのデータ型もVXI機器の構成中に指定され、トランザクションでは指定されません。

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

INCRは、*SymbolicName*で指定されるアドレスを先頭に配列データをインクリメンタルにレジスタから読み取ることを指定します。配列の最初の要素は開始アドレスから読み取られ、2番目の要素はデータ型のバイト数に等しいオフセットを加えたアドレスから読み取られ、同様にしてすべての配列要素が読み取られます。トランザクションにINCRを指定しないと、*SymbolicName*で指定される1つの位置から配列全体が読み取られます。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

MEMORYエンコーディング

READ MEMORYは、VXI機器のA24またはA32メモリから値を読み取るために用いられます。

READ MEMORYトランザクションは下記の形式を取ります。

```
READ MEM: SymbolicName ExpressionList INCR ARRAY:NumElements
-または-
READ MEM: SymbolicName ExpressionList ARRAY:NumElements
```

ここで:

*SymbolicName*は、VXI機器の構成中に指定される名前です。この名前は、機器の拡張メモリの特定のアドレスを指します。READ MEMORYトランザクションで使用できるデータ型は下記の通りです。

- BYTE - 8ビット符号なしバイト
- WORD16 - 16ビット2の補数整数
- WORD32 - 32ビット2の補数整数
- REAL32 - 32ビットIEEE 754浮動小数点数
- WORD32*2 - Int32配列の隣接する要素にある2個の32ビット2の補数整数

REAL64 - 64ビットIEEE 754浮動小数点数

これらのデータ型もVXI機器の構成中に指定され、トランザクションでは指定されません。

*ExpressionList*は単一の式またはカンマで区切った式のリストです。

INCRは、*SymbolicName*で指定される位置を先頭に配列データをインクリメンタルにメモリ位置から読み取ることを指定します。配列の最初の要素は開始位置から読み取られ、2番目の要素はデータ型のバイト数に等しいオフセットを加えた位置から読み取られ、同様にしてすべての配列要素が読み取られます。トランザクションにINCRを指定しないと、*SymbolicName*で指定される1つのメモリ位置から配列全体が読み取られます。

*NumElements*は、1つの式またはカンマで区切った式のリストであり、*VarList*の各変数の次元を指定します。トランザクションがスカラを読み取るように構成されている場合、ARRAYキーワードはトランザクションに用いられません。ARRAY:1は1要素の1次元配列です。スカラと1要素の1次元配列とはVEEでは区別されます。

IOSTATUSエンコーディング

READ IOSTATUSトランザクションは下記の形式を取ります。

```
READ IOSTATUS STS Bits VarList
-または-
READ IOSTATUS DATA READY VarList
```

*VarList*は、1つのInteger変数またはカンマで区切ったInteger変数のリストです。

READ IOSTATUSトランザクションは、GPIOインタフェース用のDirect I/O、From StdIn、To/From Named Pipe、To/From Socket、To/From Rocky Mountain Basicで用いられます。

GPIO用のREAD IOSTATUSトランザクションは、インタフェースが用意している周辺機器ステータス・ビットを読み取ります。読み取られるビット数は、インタフェースのモデル番号に依存します。すべてのステータス・ビットの重み付き和が1個の整数値として返されます。

例えば、HP 98622AのGPIOインタフェースは、2本の周辺機器ステータス・ラインSTI0とSTI1をサポートします。表A-17に、このトランザクションのxの値を解釈する方法を示します。

```
READ IOSTATUS STS Bits a
```

表A-17. IOSTATUS**の値**

読み取った値	STI1	STI0
0	0	0
1	0	1
2	1	0
3	1	1

READ IOSTATUSトランザクションは、ステータス・ラインの瞬時値を読み取ります。ステータス・ラインにはラッチやバッファリングはありません。

READ IOSTATUSトランザクションは、読み取るデータが存在する場合には論理値の真(1)を返します。データがない場合、論理値の偽(0)が返されます。READ IOSTATUSトランザクションを使えば、データが来るまでREADによってプログラムの実行がブロックされるのを防ぐことができます。

その他のトランザクション

このセクションでは、VEEのその他のI/Oトランザクション、すなわちEXECUTEトランザクション、WAITトランザクション、SENDトランザクション、WRITE(POKE)トランザクション、READ(REQUEST)トランザクションについて説明します。

EXECUTEトランザクション

EXECUTEトランザクションは、特定のオブジェクトに対応するファイル、機器、インタフェースを制御する低レベルのコマンドを送信します。ファイル・ポインタの調整、バッファのクリア、ハードウェア・インタフェースの低レベル制御などに用いられます。表A-18にEXECUTEコマンドの一覧を示します。

表A-18. EXECUTEコマンドの一覧

コマンド	説明
To File、From File	
REWIND	読取りポインタ(From File)または書込みポインタ(To File)をファイルの先頭に位置づけます。ファイル内のデータは変更されません。
CLEAR	(To Fileのみ)ファイル内の既存のデータを消去し、書込みポインタをファイルの先頭に位置づけます。
CLOSE	ファイルを明示的にクローズします。複数のプロセスが同じファイルを読み書きする場合に便利です。
DELETE	ファイルを明示的に削除します。テンポラリ・ファイルを削除するのに便利です。

表A-18. EXECUTEコマンドの一覧

コマンド	説明
Interface Operations	
CLEAR	GPIBでは、DCL(Device Clear)を送信してすべての機器をクリアします。VXIでは、インタフェースをリセットし、リソース・マネージャを実行します。
TRIGGER	GPIBでは、リスン指定されたすべての機器にGET(Group Execute Trigger)を送ってトリガします。VXIでは、指定されたバックプレーン・トリガ・ラインまたは組込みコントローラの外部トリガをトリガします。
LOCAL	GPIBでは、REN(Remote Enable)ラインをリリースし、機器をローカル・モードにします。
REMOTE	GPIBでは、REN(Remote Enable)ラインをアサートします。
LOCAL LOCKOUT	GPIBでは、LLO(Local Lockout)メッセージを送信します。LLOが送信された時点でリモート・モードにある機器は、フロントパネル操作をロックします。
ABORT	IFC(Interface Clear)ラインをアサートすることにより、GPIBインタフェースをクリアします。
LOCK INTERFACE	リソースが共有されるマルチプロセス・システムで、1つのプロセスがクリティカル・セクションを実行する間リソースをロックし、他のプロセスが使用できないようにします。
UNLOCK INTERFACE	マルチプロセス・システムで1つのプロセスが共有リソースをロックしている場合、リソースのロックを解除して他のプロセスがアクセスできるようにします。

表A-18. EXECUTEコマンドの一覧

コマンド	説明
GPIBに対するDirect I/O	
CLEAR	Direct I/Oオブジェクトのアドレスにある機器にSDC(Selected Device Clear)を送信してクリアします。
TRIGGER	Direct I/Oオブジェクトのアドレスにある機器をリスン指定し、GET(Group Execute Trigger)を送信することによりトリガします。
LOCAL	Direct I/Oオブジェクトのアドレスにある機器をローカル・ステートにします。
REMOTE	Direct I/Oオブジェクトのアドレスにある機器をリモート・ステートにします。
GPIOに対するDirect I/O	
RESET	PRESETライン(周辺機器リセット)にパルスを送ることにより、Direct I/Oオブジェクトに対応するGPIOインタフェースをリセットします。
メッセージ・ベースVXIに対するDirect I/O	
CLEAR	ワード・シリアル・コマンドClear(0xffff)を送信することにより、Direct I/Oオブジェクトに対応するVXI機器をクリアします。
TRIGGER	ワード・シリアル・コマンドTrigger(0xedff)を送信することにより、Direct I/Oオブジェクトに対応するVXI機器をトリガします。
LOCAL	ワード・シリアル・コマンドClear Lock(0xefff)を送信することにより、Direct I/Oオブジェクトに対応するVXI機器をローカル・ステートにします。
REMOTE	ワード・シリアル・コマンドSet Lock(0xeeff)を送信することにより、Direct I/Oオブジェクトに対応するVXI機器をリモート・ステートにします。

表A-18. EXECUTEコマンドの一覧

コマンド	説明
シリアル・インタフェースに対するDirect I/O	
RESET	Direct I/Oオブジェクトに対応するシリアル・インタフェースをリセットします。
BREAK	Direct I/Oオブジェクトに対応するシリアル・インタフェースのData Outラインに下記のように信号を送出します。 400 msの間論理ハイ 60 msの間論理ロー
CLOSE	Direct I/Oオブジェクトに対応するシリアル・インタフェースとの接続をクローズします。次に接続を行うと接続が再確立されます。
Execute Program、To/From Named Pipe、To/From Rocky Mountain Basic	
CLOSE READ PIPE	(To/From)オブジェクトに対応する読取り名前付きパイプまたは(Execute Program)に対応するstdinパイプをクローズします。
CLOSE WRITE PIPE	(To/From)オブジェクトに対応する書込み名前付きパイプまたは(Execute Program)に対応するstdoutパイプをクローズします。
To/From Socket	
CLOSE	クライアント・ソケットとサーバ・ソケットとの間のコネクションをクローズします。コネクションを再確立するには、クライアントとサーバがバインド-受け入れおよび接続のプロトコルを繰り返す必要があります。

表A-18. EXECUTEコマンドの一覧

コマンド	説明
GPIB、VXI、シリアル、GPIOに対するDirect I/O、MultiInstrument Direct I/O、Interface Operations	
LOCK	リソースが共有されるマルチプロセス・システムで、1つのプロセスがクリティカル・セクションを実行する間リソースをロックし、他のプロセスが使用できないようにします。
UNLOCK	マルチプロセス・システムで1つのプロセスが共有リソースをロックしている場合、リソースのロックを解除して他のプロセスがアクセスできるようにします。

GPIBに関する詳細

GPIB機器に対するDirect I/OとInterface OperationsがEXECUTEコマンドは似ていますが違いがあります。

- Direct I/OのEXECUTEコマンドは、1台の機器を指定してコマンドを送ります。
- Interface OperationsのEXECUTEコマンドは、リスン指定されている複数の機器に影響する可能性があります。

表A-19から表A-24までは、Direct I/OとInterface OperationsのEXECUTEトランザクションで生じるバス動作の詳細を示します。

表A-19. EXECUTE ABORT **GPIBの動作**

Direct I/O	Interface Operations
使用不可	IFC (≧ 100 µsec) REN ATN

表A-20. EXECUTE CLEAR **GPIBの動作**

Direct I/O	Interface Operations
ATN MTA UNL LAG SDC	ATN DCL

表A-21. `EXECUTE TRIGGER` **GPIBの動作**

Direct I/O	Interface Operations
ATN	ATN
MTA	GET
UNL	
LAG	
GET	

表A-22. `EXECUTE LOCAL` **GPIBの動作**

Direct I/O	Interface Operations
ATN	REN
MTA	ATN
UNL	
LAG	
GTL	

表A-23. `EXECUTE REMOTE` **GPIBの動作**

Direct I/O	Interface Operations
REN	**REN**
ATN	**ATN**
MTA	
UNL	
LAG	

表A-24. `EXECUTE LOCAL LOCKOUT` **GPIBの動作**

Direct I/O	Interface Operations
使用不可	ATN LLO

VXIに関する詳細

VXI機器に対するDirect I/OとInterface Operationsが使用するEXECUTEコマンドは似ていますが違いがあります。メッセージ・ベースVXI機器に関する記述は、I-SCPIでサポートされるレジスタ・ベース機器にも該当します。

- Direct I/OのEXECUTEコマンドは、メッセージ・ベースのVXI機器を指定してワード・シリアル・コマンドを送信します。
- Interface OperationsのEXECUTEコマンドは、VXIインタフェースに直接影響し、インタフェースのサーバント・エリアにあるVXI機器に影響する可能性があります。

Interface Operationsオブジェクトの`EXECUTE TRIGGER`トランザクションは下記の形式を取ります。

```
EXECUTE TRIGGER TriggerType Expression TriggerMode
```

*TriggerType*は`EXECUTE TRIGGER`トランザクションが使用するトリガ・グループを指定します。下記のグループが使用できます。

- `TTL` - VXIバックプレーンの8本のTTLトリガ・ラインを指定します。
- `ECL` - VXIバックプレーンの4本のECLトリガ・ラインを指定します。
- `EXT` - 組込みVXIコントローラの外部トリガを指定します。

*Expression*は1個のInteger変数に評価され、指定した*TriggerType*のどのトリガ・ラインをトリガするかを示すビット・パターンを表します。値5(2進の`101`)は、TTLライン0と2をトリガすることを示します。255を指定すると8本のTTLラインすべてがトリガされます。*TriggerMode*はトリガ・ラインをアサートする方法を示します。

- PULSE - 短い時間(*TriggerType*に依存)の間ラインにパルスが送出されます。
- ON - トリガ・ラインをアサートし、そのままの状態に保持します。
- OFF - 以前のONトランザクションでアサートされたトリガ・ラインのアサートを解除します。

表A-25から表A-28までは、Direct I/OとInterface OperationsのEXECUTEトランザクションで生じるバス動作を示します。

表A-25. EXECUTE CLEAR **VXIの動作**

Direct I/O	Interface Operations
ワード・シリアル・コマンドClear(0xffff)	SYSRESETラインをパルスし、リソース・マネージャを再実行

表A-26. EXECUTE TRIGGER **VXIの動作**

Direct I/O	Interface Operations
ワード・シリアル・コマンド Trigger(0xedff)	バックプレーンのTTL/ECLトリガ・ライン、または組込みコントローラの外部トリガをトリガします。各トリガ・タイプごとにどのラインをトリガするかを指定できます。

表A-27. EXECUTE LOCAL **VXIの動作**

Direct I/O	Interface Operations
ワード・シリアル・コマンドSet Lock(0xeeff)	使用不可

表A-28. EXECUTE REMOTE **VXIの動作**

Direct I/O	Interface Operations
ワード・シリアル・コマンドClear Lock(0xefff)	使用不可

WAITトランザクション

WAITトランザクションには下記の4種類があります。

- WAIT INTERVAL
- WAIT SPOLL(GPIBおよびメッセージ・ベースのVXI機器に対するDirect I/Oのみ)
- WAIT REGISTER(VXI機器に対するDirect I/Oのみ)
- WAIT MEMORY(VXI機器に対するDirect I/Oのみ)

WAIT INTERVALトランザクションは、指定された秒数だけ待ってから、オブジェクトのオープン・ビューに記載された次のトランザクションを実行します。例えば、下記のトランザクションは10秒間待ちます。

```
WAIT INTERVAL:10
```

WAIT SPOLLトランザクションは下記の形式を取ります。

```
WAIT SPOLL Expression Sense
```

*Expression*は整数に評価される式です。この整数はビット・マスクとして用いられます。

*Sense*は下記のどちらかの値を取るフィールドです。

- ANY SET
- ALL CLEAR

WAIT SPOLLトランザクションは、対応する機器のシリアル・ポール応答バイトが特定の条件を満たすまで待ちます。テスト方法としては、シリアル・ポール応答と指定したマスクとのビットごとのANDを取り、結果のビットすべてのORを取って1個のテスト・ビットが生成されます。WAIT SPOLLのあとのトランザクションは、下記の条件のどれかが満たされたときに実行されます。

- トランザクションにANY (ANY SET)が指定され、テスト・ビットが真(1)のとき
- トランザクションにCLEAR (ALL CLEAR)が指定され、テスト・ビットが偽(0)のとき

下記のトランザクションは、WAIT SPOLLの使い方の例を示します。

```
WAIT SPOLL:256 ANY        どれかのビットがセットされるまで待つ
WAIT SPOLL:256 CLEAR      すべてのビットがクリアされるまで待つ
WAIT SPOLL:0x40 ANY       ビット6がセットされるまで待つ
WAIT SPOLL:0x40 CLEAR     ビット6がクリアされるまで待つ
```

WAIT REGISTERおよびWAIT MEMORYトランザクションは下記の形式を取ります。

```
WAIT REG:SymbolicName MASK:Expression Sense [Expression]
-または-
WAIT MEM:SymbolicName MASK:Expression Sense [Expression]
```

ここで:

*SymbolicName*は、VXI機器の構成中に指定される名前です。この名前は、機器のA16または拡張メモリの特定のアドレスを指します。

MASK:*Expression*は整数に評価される式です。この整数はビット・マスクとして用いられます。マスク値のバイト数は、*SymbolicName*の構成で指定されたデータ型に依存します。

*Sense*は下記のどれかの値を取るフィールドです。

- ANY SET
- ALL CLEAR
- *EQUAL

[*Expression*]は任意指定の比較値であり、整数に評価されます。この整数は*Sense*がEQUALの場合のみ用いられます。

WAIT REGISTERまたはMEMORYトランザクションは、対応するVXI機器の*SymbolicNames*で指定されるレジスタまたはメモリ位置から読み取った値が特定の条件を満たすまで待ちます。

読み取った値とMASK:*Expression*で指定されたビット・マスクとの論理ANDを取ることにより、テスト値が生成されます。テスト値のサイズは、指定されたレジスタまたはメモリ位置の構成で指定されたデータ型に依存します。WAIT SPOLLのあとのトランザクションは、下記の条件のどれかが満たされたときに実行されます。

- トランザクションにANY (ANY SET)が指定され、テスト値の少なくとも1ビットが真(1)のとき
- トランザクションにCLEAR (ALL CLEAR)が指定され、テスト値のすべてのビットが偽(0)のとき
- トランザクションにEQUALが指定され、[*Expression*]に指定された比較値とテスト値の対応するビット同士がすべて一致するとき

SENDトランザクション

SENDトランザクションは下記の形式を取ります。

```
SEND BusCmd
```

*BusCmd*は表A-29に示すバス・コマンドの1つです。

SENDトランザクションは、GPIBインタフェース経由で低レベルのバス・メッセージを送信するためにInterface Operationsオブジェクトで用いられます。これらのメッセージはIEEE 488.1で詳細に定義されています。

表A-29. SEND**のバス・コマンド**

コマンド	説明
COMMAND	ATNを真にし、指定されたデータ・バイト群を送信します。ATNが真の場合、データがバス・コマンドであることを示します。
DATA	ATNを偽にし、指定されたデータ・バイト群を送信します。ATNが偽の場合、データが機器依存の情報であることを示します。
TALK	指定された1次バス・アドレス(0～31)の機器をトーク指定します。
LISTEN	指定された1次バス・アドレス(0～31)の機器をリスン指定します。
SECONDARY	TALKまたはLISTENコマンドのあとで2次バス・アドレスを指定します。2次アドレスはカードケージ機器で主に用いられ、カードケージが1次アドレスで、各プラグイン・モジュールが2次アドレスに存在します。
UNLISTEN	すべての機器にリスンを停止させ、UNLを送信します。
UNTALK	すべての機器にトークを停止させます。UNTを送信します。
MY LISTEN ADDR	VEEが動作しているコンピュータをリスン指定し、MLAを送信します。
MY TALK ADDR	VEEが動作しているコンピュータをトーク指定し、MTAを送信します。
MESSAGE	マルチライン・バス・メッセージを送信します。詳細についてはIEEE 488.1を参照してください。マルチライン・メッセージは下記の通りです。 DCL Device Clear SDC Selected Device Clear GET Group Execute Trigger GTL Go To Local LLO Local Lockout SPE Serial Poll Enable SPD Serial Poll Disable TCT Take Control

WRITE(POKE)トランザクション

注記

WRITE(POKE)トランザクションは、Windows版VEEでのみサポートされます。

WRITE(POKE)トランザクションはWRITEトランザクションとよく似ていますが、To/From DDEオブジェクトでのみ使用できます。WRITE(POKE)の最大の違いは、アイテム名を指定することです。例を挙げます。

```
WRITE ITEM:"r2c3" TEXT a EOL
```

下記のエンコーディングが使用できます。

- TEXT
- BYTE
- CASE
- CONTAINER

上記のフォーマットの詳細については、WRITEトランザクションを参照してください。

READ(REQUEST)トランザクション

READ(REQUEST)トランザクションはREADトランザクションとよく似ていますが、To/From DDEオブジェクトでのみ使用できます。READ(REQUEST)の最大の違いは、アイテム名を指定することです。例を挙げます。

```
READ ITEM:"r2c3" TEXT a EOL
```

READ(REQUEST)トランザクションは、Windows版VEEでのみサポートされます。

下記のエンコーディングが使用できます。

- TEXT
- CONTAINER

上記のフォーマットの詳細については、READトランザクションを参照してください。

B

トラブルシューティングの技法

トラブルシューティングの技法

この付録では、機器制御のトラブルシューティングの方法と、一般的な状況および可能な回復手段について説明します。表B-1は、機器制御のトラブルシューティングに関する説明です。

表B-1. 機器制御のトラブルシューティング

問題	対策/原因
機器が全く応答しない。	下記の条件が満たされる必要があります。 ・機器が立ち上がっており、適切なケーブルでインタフェースに接続されていること。適切なI/Oライブラリがインストールされていること。 ・To/From VXIplug&playオブジェクトの場合: 機器に対応する適切なVXI*plug&play*ドライバ・ファイルがインストールされ、構成されていること。また、各機器に対するAdvanced Instrument Propertiesダイアログ・ボックスで、正しいVXI*plug&play*アドレス文字列が指定されていること。各機器のアドレスが一意であること。 ・Direct I/O、Panel Driver、Component Driverの各オブジェクトの場合: インタフェースの論理ユニットと機器のアドレスが、Instrument Propertiesダイアログ・ボックスのAddressフィールドの設定と一致していること。各機器のアドレスが一意であること。また、Advanced Instrument Propertiesダイアログ・ボックスのLive Modeフィールドが**ON**に設定されていること。 ・機器に対して正しく動作するように、ユーザまたはシステム管理者によってVEEが構成されていること。通常これはVEEのインストール時に行われます。インストール・ガイドを参照してください。 ・UNIXシステムの場合、正しいドライバとインタフェース・カードに合わせてUNIXカーネルが構成されていること

表B-1. 機器制御のトラブルシューティング

問題	対策/原因
機器アドレスがわからない。	GPIOおよびシリアル・インタフェースの場合、機器アドレスはインタフェースの論理ユニットと一致します。GPIBの機器アドレスは、ハードウェア・スイッチまたはフロントパネル・コマンドによって設定されます。詳細については機器のプログラミング・マニュアルを参照してください。VXIデバイスの論理アドレスは、カード外部のスイッチで設定されます(スイッチにアクセスするには通常カード・ケージからカードを外す必要があります)。アドレスの構成方法の詳細については、第3章「機器の構成」を参照してください。
インタフェースの論理ユニットがわからない。	インタフェースの論理ユニットは、HP I/Oライブラリに付属するI/O Configユーティリティで構成する必要があります。詳細については、*Installing the Agilent I/O Libraries (VEE for Windows)*または*Installing the Agilent I/O Libraries (VEE for HP-UX)*を参照してください。推奨される論理ユニット設定については、213ページ「推奨されるI/O論理ユニット」を参照してください。

表B-2は、VEEの一般的なトラブルシューティングに関する説明です。

表B-2. VEEのトラブルシューティング

問題	考えられる原因	解決法の例
VEE 6.0より前のバージョンで作成したプログラムをVEE 6実行モードで実行すると、予期したとおりに動作しない。		17ページ「VEE実行モードの使用」に考えられる解決法が記載されています。
UserObjectが予期したとおりに動作しない。	コンテキスト境界を非同期データが横切っている(UserObject内部のオブジェクトのXEQピンに接続しているなど)。	考えられる解決法1: 非同期の依存性をUserObject外部に移します。 考えられる解決法2: Show Execution FlowまたはShow Data Flowをオンにして、プログラムの実行順序を調べます。
オブジェクトの機能を変更したい。		オブジェクト・メニューの機能を使って、制御入力端子を追加したり、プロパティを編集したりします。
UserObject内部の反復子から1個の値しか出力されない。	UserObjectは出力を1回しかアクティブにしないため。	反復子をUserObject外部に移します。
反復子が1回しか動作しない。	反復サブスレッドがデータ出力ピンでなくシーケンス出力ピンに接続されている。	データ出力ピンから反復サブスレッドを開始します。
For Countが動作しない。	For Countの値が0か負。	値を変更します。負の値が必要な場合、出力の符号を反転するか、For Rangeを使います。

表B-2. VEEのトラブルシューティング

問題	考えられる原因	解決法の例
For RangeまたはFor Log Rangeが動作しない。	ステップ・サイズの符号が間違っている。FromがThruより小さい場合、Stepは正でなければなりません。ThruがFromより小さい場合、Stepは負でなければなりません。	Stepを変更します。
sh:*name* - not foundというUNIXのメッセージが表示される。	実行可能ファイルの名前を間違って入力した。	もう一度veetestと入力してください。実行可能ファイルのフル・パスを指定しなければならない可能性もあります。
Error: cannot open displayというUNIXのエラー・メッセージが表示される。	DISPLAY環境変数が正しく設定されていないか、表示用に指定されたマシン上でパーミッションが正しく設定されていない。	環境変数DISPLAYを設定(およびエクスポート)します。一般的には*hostname:0.0*に設定します。リモート・マシンに表示する場合、リモート・マシン上で*xhost*を使ってパーミッションを設定します。
VEEがハングしている。ポインタが砂時計になったまま。	考えられる原因1: Auto Line Routingがオンになっているときにオブジェクトを移動したため、ラインの再ルーティングが行われている。 考えられる原因2: VEEが画面またはプログラムをプリントしている。 考えられる原因3: 大きなオブジェクトまたは多数のオブジェクトをCutした。VEEがオブジェクトをPasteバッファに保存している。	しばらく待ってください。数分たってもポインタが十字線に戻らない場合、**CTRL+C**(または、VEE 6.0を起動したターミナル・ウィンドウのintr設定)を押すか、VEEウィンドウをクローズするか、VEEプロセスをkillしてください。
プログラムをOpenできない、オブジェクトをCutできない、ラインを削除できない(機能が淡色表示になっている)。	プログラムが実行中。	Stopを押してプログラムを停止してから、もう一度動作を実行してください。
Pasteができない(機能が淡色表示になっている)。	Pasteバッファが空。	もう一度オブジェクトをCut、Copy、またはCloneしてください。

表B-2. VEEのトラブルシューティング

問題	考えられる原因	解決法の例
Cut、Create UserObject、Add to Panelができない(機能が淡色表示されている)。	オブジェクトが選択されていない。	オブジェクトを選択してから、もう一度動作を実行してください。
UserObjectの出力に、最後に生成されたデータ要素しか出てこない。	UserObjectの出力端子バッファにはデータが蓄積されないため。このバッファには最後に受信したデータ要素だけが格納されます。	生成されたデータをすべてCollectorで収集し、配列にします。このデータを出力端子に送ります。
ライン・ドローイング・モードから抜けられない。		ダブルクリックするか**Esc**を押すとライン・ドローイング・モードが終了します。
プログラムをOpenするとParse Errorオブジェクトが表示される。		Parse Errorオブジェクトを新しいオブジェクトに置き換えます。
文字が正しく表示されない。	USASCII以外のキーボードを使っている。	回復手段については5ページ「VEEの構成」を参照してください。
VEEの外部のカラーが変化する(VEEを使っているときはVEEのカラーは正常)。	カラー・マップ・プレーンが使い尽くされている。	回復手段については5ページ「VEEの構成」を参照してください。

C

機器I/Oのデータ型変換

機器I/Oのデータ型変換

数値データを扱う機器I/Oトランザクションでは、下記の規則に基づいてVEEが自動データ型変換を実行します(このデータ型変換は完全に自動的です。通常はユーザが意識する必要はありません)。この変換が行われるのは、VEE 5およびそれ以前の実行モードだけです。

- 入力トランザクション(READ)の場合、機器からのInt16値またはByte値は、符号拡張を保ったままInt32値に変換されます。また、機器からのReal32値は64ビットのReal値に変換されます。

- 出力トランザクション(WRITE)の場合、Int32値またはReal値は、下記のように機器にとって適切な出力フォーマットに変換されます。

  - 機器がReal32フォーマットをサポートする場合、64ビットReal値はReal32値に変換されて機器に出力されます。Real値がReal32値の範囲を超えている場合、エラーが発生します。

  - 機器がInt16フォーマットをサポートする場合、Int32値はInt16値に切りつめられて機器に出力されます。Real値はまずInt32値に変換され、切りつめられてから出力されます。ただし、Real値がInt32の範囲を超えている場合は、エラーが発生します。

  - 機器がByteフォーマットをサポートする場合、Int32値がByte値に切りつめられて機器に出力されます。Real値はまずInt32値に変換され、切りつめられてから出力されます。ただし、Real値がInt32の範囲を超えている場合は、エラーが発生します。

D

高速なプログラミングの鍵

高速なプログラミングの鍵

この付録では、VEEプログラムの性能を改善するための指針を紹介します。プログラムの性能を上げるための一般的なヒントについては、**VEEオンライン・ヘルプ**のHow Do IにあるImproving the Performance of a VEE Programを参照してください。

注記

VEE 6.0より前のバージョンでプログラムを開発した場合、コンパイラを使うようにプログラムを変換する方法について17ページ「VEE実行モードの使用」を参照してください。

下記の方法を使うと、コンパイラ使用時に最高の速度を実現するために役立ちます(File ⇒ Default PreferencesでExecution ModeがVEE 4、VEE 5、VEE 6に設定されている場合)。

■ Profilerを使用する

Profiler(View ⇒ Profiler)を使うことにより、必要以上に時間を使っているルーチンを見つけることができます。Profilerは下記の手順で実行します。

1. Start Profilingをクリックし、プログラムを実行します。
2. プログラムの実行が終了したら、Refreshをクリックして結果を見ます。
3. Stop Profilingをクリックしてプロファイラを停止します。Clearをクリックして現在の結果の表示をクリアします。

■ ライン・カラーを調べる

VEEが実行前にデータ型を決定することができた場合、ラインがカラーで表示されます。カラーの付いた(黒以外の)ラインが多いほど、プログラムは高速に動作します。

■ 端子の拘束条件を追加する

UserFunctionはさまざまな場所から呼ばれる可能性があるため、プログラムが実行されるまで入力データ型を決定することができません。UserFunctionを高速に実行させるには、可能な限りデータ入力端子に端子拘束条件を追加します。

■ 宣言済み変数を使う

グローバル変数を使う場合、できるだけDeclare Variable(Dataメニューに存在)を使って変数の型と形状を宣言し、実行前にVEEが変数の型を判定できるようにします。そうすれば、変数のスコープを設定することもできます。

■ Autoscale制御入力をなくす

グラフィカル表示のAutoscale制御入力は、多くのプログラムで必要以上に実行される傾向があります。ポイントをプロットするたびにオートスケールを実行させるのではなく、表示の更新が終わってから実行するようにすれば、プログラムの動作が高速になります。Automatic Scalingプロパティ (Scalesタブを参照)を使ってAutoscale制御入力をなくせば、さらに実行速度を改善できます。

■ すべてのデータをまとめて送る

グラフィカル表示でAutomatic Scalingプロパティがオンになっている場合(Scalesタブを参照)、すべてのデータをまとめて表示オブジェクトに送るようにすると、プログラムの動作が高速になります。この場合、表示の自動再スケーリングは1回しか行われません。プログラムで一度に1データ・ポイントずつ表示オブジェクトに送ると、そのたびに再スケーリングが自動的に行われ、プログラムの動作速度が低下します。この場合、Collectorオブジェクトを使って配列を作成し、配列を表示オブジェクトに送るようにします。

■ 表示は1回だけ実行する

ループの最終出力を表示オブジェクトに表示する場合、生成されるデータをループの反復のたびに表示するのでなければ(例えば、AlphaNumericオブジェクトがLogging AlphaNumericでない場合は)、ループの中で毎回実行するのは避けます。反復子のシーケンス出力ピンを表示オブジェクトのシーケンス入力ピンに接続し、最後の1回だけ表示が実行されるようにします。

■ デバッグ機能をオフにする

プログラムが正常に動作することがわかったら、デバッグ機能をオフにしてプログラムを実行します。File ⇒ Default Preferencesを選択し、Debugグループで Disable Debug Featuresを選択します。

-rオプションを使うか、VEE RunTimeを使う方法もあります。これらのモードではデバッグ命令が生成されないので、プログラムの動作が多少高速になります。ただし、デバッグ機能、例えば一時停止、ステップ実行、Breakpoints、Line Probe、Show Data Flow、Show Execution Flowなどはいっさい使用できません。

E

ASCIIコード表

ASCIIコード表

この付録には、ASCII 7ビット・コードのリファレンス表を記載します。

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
NUL	0000000	000	00	0	
SOH	0000001	001	01	1	GTL
STX	0000010	002	02	2	
ETX	0000011	003	03	3	
EOT	0000100	004	04	4	SDC
ENQ	0000101	005	05	5	PPC
ACK	0000110	006	06	6	
BEL	0000111	007	07	7	
BS	0001000	010	08	8	GET
HT	0001001	011	09	9	TCT
LF	0001010	012	0A	10	
VT	0001011	013	0B	11	
FF	0001100	014	0C	12	
CR	0001101	015	0D	13	
SO	0001110	016	0E	14	
SI	0001111	017	0F	15	
DLE	0010000	020	10	16	
DC1	0010001	021	11	17	LLO
DC2	0010010	022	12	18	

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
DC3	0010011	023	13	19	
DC4	0010100	024	14	20	DCL
NAK	0010101	025	15	21	PPU
SYN	0010110	026	16	22	
ETB	0010111	027	17	23	
CAN	0011000	030	18	24	SPE
EM	0011001	031	19	25	SPD
SUB	0011010	032	1A	26	
ESC	0011011	033	1B	27	
FS	0011100	034	1C	28	
GS	0011101	035	1D	29	
RS	0011110	036	1E	30	
US	0011111	037	1F	31	
スペース	0100000	040	20	32	listen addr 0
!	0100001	041	21	33	listen addr 1
"	0100010	042	22	34	listen addr 2
#	0100011	043	23	35	listen addr 3
$	0100100	044	24	36	listen addr 4
%	0100101	045	25	37	listen addr 5
&	0100110	046	26	38	listen addr 6
'	0100111	047	27	39	listen addr 7
(	0101000	050	28	40	listen addr 8
)	0101001	051	29	41	listen addr 9

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
*	0101010	052	2A	42	listen addr 10
+	0101011	053	2B	43	listen addr 11
,	0101100	054	2C	44	listen addr 12
-	0101101	055	2D	45	listen addr 13
.	0101110	056	2E	46	listen addr 14
/	0101111	057	2F	47	listen addr 15
0	0110000	060	30	48	listen addr 16
1	0110001	061	31	49	listen addr 17
2	0110010	062	32	50	listen addr 18
3	0110011	063	33	51	listen addr 19
4	0110100	064	34	52	listen addr 20
5	0110101	065	35	53	listen addr 21
6	0110110	066	36	54	listen addr 22
7	0110111	067	37	55	listen addr 23
8	0111000	070	38	56	listen addr 24
9	0111001	071	39	57	listen addr 25
:	0111010	072	3A	58	listen addr 26
;	0111011	073	3B	59	listen addr 27
<	0111100	074	3C	60	listen addr 28
=	0111101	075	3D	61	listen addr 29
>	0111110	076	3E	62	listen addr 30
?	0111111	077	3F	63	UNL
@	1000000	100	40	64	talk addr 0

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
A	1000001	101	41	65	talk addr 1
B	1000010	102	42	66	talk addr 2
C	1000011	103	43	67	talk addr 3
D	1000100	104	44	68	talk addr 4
E	1000101	105	45	69	talk addr 5
F	1000110	106	46	70	talk addr 6
G	1000111	107	47	71	talk addr 7
H	1001000	110	48	72	talk addr 8
I	1001001	111	49	73	talk addr 9
J	1001010	112	4A	74	talk addr 10
K	1001011	113	4B	75	talk addr 11
L	1001100	114	4C	76	talk addr 12
M	1001101	115	4D	77	talk addr 13
N	1001110	116	4E	78	talk addr 14
O	1001111	117	4F	79	talk addr 15
P	1010000	120	50	80	talk addr 16
Q	1010001	121	51	81	talk addr 17
R	1010010	122	52	82	talk addr 18
S	1010011	123	53	83	talk addr 19
T	1010100	124	54	84	talk addr 20
U	1010101	125	55	85	talk addr 21
V	1010110	126	56	86	talk addr 22
W	1010111	127	57	87	talk addr 23

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
X	1011000	130	58	88	talk addr 24
Y	1011001	131	59	89	talk addr 25
Z	1011010	132	5A	90	talk addr 26
[	1011011	133	5B	91	talk addr 27
\	1011100	134	5C	92	talk addr 28
]	1011101	135	5D	93	talk addr 29
^	1011110	136	5E	94	talk addr 30
_	1011111	137	5F	95	UNT
`	1100000	140	60	96	secondary addr 0
a	1100001	141	61	97	secondary addr 1
b	1100010	142	62	98	secondary addr 2
c	1100011	143	63	99	secondary addr 3
d	1100100	144	64	100	secondary addr 4
e	1100101	145	65	101	secondary addr 5
f	1100110	146	66	102	secondary addr 6
g	1100111	147	67	103	secondary addr 7
h	1101000	150	68	104	secondary addr 8
i	1101001	151	69	105	secondary addr 9
j	1101010	152	6A	106	secondary addr 10
k	1101011	153	6B	107	secondary addr 11
l	1101100	154	6C	108	secondary addr 12
m	1101101	155	6D	109	secondary addr 13
n	1101110	156	6E	110	secondary addr 14

表E-1. ASCII 7ビット・コード

	2進	8進	16進	10進	GPIBメッセージ
o	1101111	157	6F	111	secondary addr 15
p	1110000	160	70	112	secondary addr 16
q	1110001	161	71	113	secondary addr 17
r	1110010	162	72	114	secondary addr 18
s	1110011	163	73	115	secondary addr 19
t	1110100	164	74	116	secondary addr 20
u	1110101	165	75	117	secondary addr 21
v	1110110	166	76	118	secondary addr 22
w	1110111	167	77	119	secondary addr 23
x	1111000	170	78	120	secondary addr 24
y	1111001	171	79	121	secondary addr 25
z	1111010	172	7A	122	secondary addr 26
{	1111011	173	7B	123	secondary addr 27
|	1111100	174	7C	124	secondary addr 28
}	1111101	175	7D	125	secondary addr 29
~	1111110	176	7E	126	secondary addr 30
[del]	1111111	177	7F	127	

F

UNIX版VEEとWindows版VEEの違い

UNIX版VEEとWindows版VEEの違い

一般に、あるプラットフォームのVEEで作成されたプログラムは、サポートされる他のすべてのプラットフォームでも動作します。問題が生じる可能性があるのは、特定のプラットフォームに固有の機能、例えばPCのDLLやUNIXの名前付きパイプなどをプログラムで使用した場合だけです。この付録では、UNIX版とPC版のVEEの違いについて説明します。

Execute Program. `Execute Program`オブジェクトには、UNIXプラットフォーム用のものとPCプラットフォーム用のものが存在します。どのプラットフォームで動作しているかを判断するには、組込み関数の`whichOS()`、`whichPlatform()` (`Function & Object Browser`に存在)を使います。これにより、どの`Execute Program`オブジェクトを使用するかをプログラムで決定できます。

DLLと共有ライブラリ. コンパイル関数からDLLと共有ライブラリを作成する場合の違いは下記の通りです。

- 共有ライブラリでは、SICL、VISA、TERMIOを使ってI/Oを行います。DLLでは、SICLまたはVISAを使います。システム・リソースの衝突を避けるため、ソース・コードで使用するライブラリ・コマンドが、コンパイル関数が動作するプラットフォームとインタフェース・システムをサポートすることを確認してください。
- 共有ライブラリでは X11 グラフィックスを使用するのに対して、DLL では Microsoft WindowsのGDI呼出しを使用します。共有ライブラリはXウィンドウ・リリース6のライブラリとリンクします。コンパイル関数はXウィンドウで動作しますが、VEEがXウィンドウのヒューマン・インタフェースで動作することはありません。

データ・ファイル. UNIXプラットフォームとPCプラットフォームではバイト順序が逆なので、バイナリ・ファイルの互換性はありません。これに対して、`To File`オブジェクトで書き込まれたASCIIデータ・ファイルは他のプラットフォームの`From File`オブジェクトで読み取ることができます。また、VEEのプログラム・ファイルもASCIIで記録されているので互換性があります。ASCIIデータ・ファイルを他のプラットフォームで使用する場合、UNIXファイルではラインフィード文字が改行に用いられるのに対して、MS Windowsではキャリッジ・リターン/ラインフィードのシーケンスが用いられることに注意してください。

G

Callable VEEについて

Callable VEEについて

場合によっては、他の言語で構築したアプリケーションでVEEのUserFunctionを使用したいことがあります。VEEでリモート関数を使うことで別のVEEのUserFunctionにアクセスできるように、Callable VEEを使えば、Cのプログラムや、Cのルーチンにアクセスできる任意の言語からUserFunctionを呼び出すことができます。

VEEの以前のバージョンでは、他の開発環境からVEEコードを実行する方法が2つ用意されていました。VEE RPC APIとCallable VEE ActiveXコントロールです。VEE 6では、Callable VEE ActiveXコントロールの代わりにCallable VEEオートメーション・サーバを用意して、Visual Basicなどのプログラミング環境からVEEコードに容易にアクセスできるようにしました。

Callable VEEオートメーション・サーバについて知るには、Help ⇒ Contents and Indexをクリックし、What's New in Agilent VEE 6.0をオープンして、Agilent VEE 6.0 New Featuresをダブルクリックします。Callable VEE ActiveX Automation Serverのトピックまでスクロールします。

この付録には、VEE RPC APIに関する情報を記載します。

注記

UserFunctionを実行するには、サーバ・システム上でVEEがアクセス可能でなければなりません。UserFunctionだけを独立して実行することはできません。UserFunctionはVEEがロードして実行できるライブラリに組み込まれている必要があります。

VEE RPC APIの使用

VEE RPC APIをサポートするのに必要なツールは、VEEに付属しています。

■ VEEをインストールしたディレクトリのlibサブディレクトリに、libvapi.lib (HP-UXではlibvapi.a)という名前のCライブラリがあります。このライブラリをCプログラムにリンクします。

このライブラリは、2つのAPI(Application Program Interface)をサポートします。1つ(VEE RPC)はCプログラムとVEEとの間のRPC(Remote Procedure Call)の設定と制御を行います。このAPIの関数のプロトタイプはveeRPC.hに記述されています。これらの関数は下記の動作を実行します。

- ❑ VEEサーバのロードとアンロード
- ❑ VEEライブラリのロードとアンロード
- ❑ VEEライブラリのUserFunctionリストの取得
- ❑ UserFunctionの呼出しとデータの受取り
- ❑ 関連するステータスなどの管理

2番目のAPI(VEE DATA)は、CとVEEとの間でデータ型の変換を行います。このAPIの関数のプロトタイプはveeData.hに記述されています。

注記

Borlandのコンパイラを使用した場合、libvapi.libライブラリはプログラムにリンクできません。

■ VEE サービス・マネージャ veesm.exe(HP-UX では veesm) は、VEE の他の実行ファイルといっしょにVEEのインストール・ディレクトリにあります。このプログラムの役割は、ターゲットVEEとそのUserFunctionを実行させ、リモート・クライアントからVEEをサーバとして起動できるようにすることです。

HP-UXシステムの場合、veesmはinetデーモン・プロセスから自動的に起動されます。PCの場合、veesm.exeを手動で実行するか、WindowsのStartupグループに入れてPCの立上げ時に自動的に起動されるようにします。

VEE RPC APIの使い方に関するサンプル・プログラムが、CallableVEE\RPC_APIディレクトリにあります。名前はcallVEE.cおよびcallVEE.veeです。

VEE RPC APIについて

VEE RPC APIは、Cクライアント・プログラムとVEEサーバとの間のコネクションの設定、維持、クローズを処理します。

VEE RPC APIのルーチンは、3つのハンドルを動作の際に使用します。

```
VRPC_SERVICE;  // VEEサーバのハンドル
VRPC_LIBRARY;  // VEE UserFunctionライブラリのハンドル
VRPC_FUNCTION; // VEE UserFunctionのハンドル
```

API呼出しの分類については、以下の部分で説明します。

サーバの始動と停止

最も基本的なAPI関数は、VEEサーバの始動と停止のための2つの関数です。VEEサーバをロードするには、下記の関数を使います。

```
VRPC_SERVICE vrpcCreateService( char *hostName,
                                char *display,
                                char *geometry,
                                double aTimeoutInSeconds,
                                unsigned long flags );
```

この関数は、hostNameで指定されたホスト上でVEEサーバを始動します。hostNameはテキスト形式(mycomputer@lvld.hp.comなど)でも数値形式(15.11.55.105)でもかまいません。この関数はサーバ・ハンドルを返します。エラーが発生した場合はNULL(実効的には0)が返ります。この場合、詳細なエラー情報を得るには、次のセクションで説明するveeGetErrorNumber()関数とveeGetErrorString()関数を使います。

display引数は、ネットワーク環境のXウィンドウ・システムの場合に、リモート・ディスプレイのネットワーク・アドレスをテキスト形式(babylon:0.0)または数値形式(15.11.55.101:0.0)で指定します。

geometry 引数は、VEE ウィンドウのサイズと配置を指定します。例えば、800x500+0+0を指定すると、画面左下隅に800×500のVEEウィンドウが作成されます。

aTimeoutInSeconds引数は、サービスを始動する際の待ち時間を秒単位で指定します。この値は、vrpcSetTimeout()で変更しない限り、このセッションの以後のすべての呼出しで用いられます。

flags引数は通常は用いられませんが、この引数をVEERPC_CREATE_NEWに設定すると、同じサーバ上ですでにVEEが動作していても、新しいVEEのコピーが始動されます。

VEEサーバを停止するには、下記の関数を使います。

```
VRPC_SERVICE vrpcDeleteService( VRPC_SERVICE aService );
```

引数は、サーバを始動したときに返されたサーバ・ハンドルです。成功するとNULLポインタが、エラーの場合はNULLでないポインタが返されます。

ライブラリのロードとアンロード

サーバを始動したら、次にはVEEのリモート・コピーにライブラリをロードします。このためには下記の関数を使います。

```
VRPC_LIBRARY vrpcLoadLibrary( VRPC_SERVICE aService,
                              char *LibraryPath );
```

この関数の引数は、サーバ・ハンドルと、UserFunction ライブラリのパス名(LibraryPath)であり、戻り値はライブラリ・ハンドルです。エラーが発生すると、NULLが返されます。

ロードしたライブラリに対して、通常の実行モードまたはデバッグ用の実行モードを指定するには、下記の関数を使います。

```
void vrpcSetExecutionMode( VRPC_LIBRARY aLibrary,
                           unsigned long executionMode );
```

この関数には、ライブラリのハンドルと、実行モード・フラグ(executionMode)を指定します。executionModeをVRPC_DEBUG_EXECUTIONに設定すると、ターゲット・システム上でUserFunctionがシングル・ステップ実行されます。デフォルトの実行モードに戻すには、VRPC_NORMAL_EXECUTIONをexecutionModeに指定します。

ライブラリをアンロードするには、下記の関数を使います。

```
VRPC_LIBRARY vrpcUnLoadLibrary( VRPC_LIBRARY aLibrary );
```

引数はライブラリ・ハンドルです。

UserFunctionの選択

サーバに接続し、ライブラリをロードしたら、UserFunctionのハンドルを取得します。

関数ハンドルを取得するには下記の関数を使います。

```
VRPC_FUNCTION vrpcFindFunction( VRPC_LIBRARY aLibrary,
                                char *aFunctionName );
```

引数は、ライブラリ・ハンドルと、UserFunction名を指定する文字列です。戻り値は関数ハンドルであり、エラーが発生した場合はNULLが返されます。

関数に関する情報を得るには、下記の関数を使います。

```
struct VRPC_FUNC_INFO*
          vrpcFunctionInfo( VRPC_FUNCTION aFunction );
```

この関数はデータ構造を返し、エラーが発生した場合はNULLを返します。データ構造の形式は下記の通りです。

```
typedef struct VRPC_FUNC_INFO
{
   char *functionName;           // 関数名
   long numArguments;            // 関数の入力ピン数
   enum veeType *argumentTypes; // 引数型のリスト
   veeShape *argumentShapes;     // 引数形状のリスト
   long numResults;              // 関数の出力ピンêî
   enum veeType *resultTypes;    // 出力型のリスト
   veeShape *resultShapes;       // 出力形状のリスト
               };
```

NULL以外が返された場合、このデータのためのメモリがCプログラムのプロセス空間に確保されているので、解放するには下記の関数を使います。

```
struct VRPC_FUNC_INFO*
   vrpcFreeFunctionInfo(struct VRPC_FUNC_INFO *funcinfo);
```

どんな関数がライブラリにあるかを知るには、下記の関数を使います。

```
char** vrpcGetFunctionNames( VRPC_LIBRARY aLibrary,
                             long *numberOfFunctions );
```

この関数の引数はライブラリ・ハンドルです。戻り値はヌル終端文字列の配列であり、関数名を直接記載したリストになっています。numberOfFunctions引数にはライブラリ中の関数の数が返されます。エラーが発生した場合はNULLポインタが返されます。文字列配列はCプログラムのプロセス空間に存在します。

UserFunctionの呼出し

これでUserFunctionを呼び出すことができるようになりました。

1つの関数を呼び出して値を受け取るには、下記の関数を使います。

```
VDC* vrpcCallAndReceive( VRPC_FUNCTION aFunction,
                         VDC *arguments );
```

この関数は、呼び出した関数が終了するか、タイムアウトが発生するまで待ちます。引数は、関数ハンドルと、VEEデータ・コンテナ(VDC)の入力配列です。VDCの処理はVEE DATA APIで行います。これについては、578ページ「VEE DATA APIについて」で説明します。

また、ノンブロッキング・モードでUserFunctionを呼び出すには、下記の関数を使います。

```
long vrpcCall( VRPC_FUNCTION aFunction,
               VDC *arguments );
```

この関数は、動作したかどうかにかかわらず、ただちに戻ります。成功した場合は0、エラーが発生した場合はエラー・コードを返します。

ほとんどのUserFunctionはいつか戻るので、戻り値を取得するには下記の関数を使います。

```
VDC* vrpcReceive( VRPC_FUNCTION aFunction,
                  unsigned long waitMode );
```

引数は関数ハンドルと`waitMode`フラグです。waitModeフラグの値は下記の3つのうちのどれかです。

- `VRPC_NO_WAITING` 結果のあるなしにかかわらず呼出しはただちに戻ります。
- `VRPC_WAIT_SLEEPING`　タイムアウトまでデータを待ちます(サーバはスリープ状態)。
- `VRPC_WAIT_SPINNING`　タイムアウトまでデータを待ちます(サーバはビジー状態)

エラーが発生すると、NULLが返されます。

その他の関数

このセクションでは、VEE RPC APIの他のユーティリティ関数を紹介します。

- 下記の関数は、タイムアウトを変更するために使います。サーバ・ハンドルとタイムアウト秒数を指定します。成功した場合は0、エラーが発生した場合はエラー・コードが返されます。

```
long vrpcSetTimeout( VRPC_SERVICE aService,
                     double aTimeoutInSeconds );
```

■ 下記の関数は、データ受信に関するCクライアントのデフォルト動作を設定するために使います。

```
long vrpcSetBehavior( VRPC_SERVICE aService,
                      unsigned long flags );
```

引数はサーバ・ハンドルとフラグで、0またはエラー・コードが返されます。フラグは下記の通りです。

VRPC_WAIT_SLEEPING タイムアウトまでデータを待ちます(クライアントはスリープ状態)。
VRPC_WAIT_SPINNING タイムアウトまでデータを待ちます(クライアントはビジー状態)

フラグにVRPC_BUFFER_EXPANDをORすると、サーバから返されたデータが大きい場合、Cクライアントは大きいバッファを割当て、そのまま保持します。

■ リモートveesmのリビジョン番号を問い合わせるには、下記の関数を使います。

```
long vrpcGetServerVersion( VRPC_SERVICE aService );
```

引数はサーバ・ハンドルで、リビジョン・コードまたは0(エラーの場合)が返されます。

VEE RPC APIのエラー・コード

VEEサーバとのコネクションが確立できない場合、下記のエラー・コードが返されます。

エラー・コード	意味
850: eUnknownHost	ホスト名またはIPアドレスが解決できません。
851: eNoServiceManager	サーバ・ホスト上にveesmが見つかりません。
861: eServiceManagerTO	サービス・マネージャがタイムアウトしました。
863: eServiceNotFound	VEEサービスが見つかりません。
864: eServiceNotStarted	VEEサービスを始動できません。
866: eConnectRefused	veesmまたはinetdへのコネクションが拒否されました。
868: eFailedSecurity	UNIXのセキュリティ・チェックに通りませんでした。

下に示すのは、VEEサーバとのコネクションが確立された後に発生する致命的エラーです(コネクションは終了しています)。

エラー・コード	意味
852: eHostDown	VEEサーバ・ホストがダウンしています。
853: eConnectTimedOut	コネクションがタイムアウトしました。
855: eConnectBroken	コネクションが切れました。

下記のエラーは、サービス内部の致命的でないステートから発生します。

エラー・コード	意味
865: eSomeInternalError	致命的でない内部エラーが発生しました。
869: eVeeServiceError	UserFunction内部にエラーがあります。
870: eWouldBlock	ノンブロッキングRPCで戻りました。
871: eDebugTermination	デバッグ・セッション中にユーザが停止を指示しました。

下記のエラーは、RPC関数呼出しから返されます。

エラー・コード	意味
851: eInvalidArgument	無効な引数があります。

VEE DATA APIについて

すでに説明したように、UserFunctionを呼び出したり値を受け取ったりするには、VEEデータ・コンテナ(VDC)フォーマットのデータの処理が必要になります。VDCとは、VEEの内部動作に用いられるデータ構造の組です。CプログラムでVEEと通信するには、VDCと通常のCデータ型との間の変換が必要です。VEE DATA APIにはこの機能(および他のいくつかの機能)が用意されています。

データ型、形状、マッピング

VDCの基本的な型は、veeData.hヘッダ・ファイルに下記のように記載されています。

```
enum veeType
{
   VEE_TYPE_ANY=0,     // 制約なしのデフォルト
   VEE_NOT_DEFINED1,   // スペース確保用
   VEE_LONG,           // 32ビット符号付き整数(VEEには16ビット整数はない)
   VEE_NOT_DEFINED2,   // スペース確保用
   VEE_DOUBLE,         // IEEE 754の64ビット浮動小数点数
   VEE_COMPLEX,        // 複素数: 直交座標形式の2個のdouble
   VEE_PCOMPLEX,       // 複素数: 極座標形式の2個のdouble
   VEE_STRING,         // 8ビットASCIIのヌル終止文字列
   VEE_NIL,            // 関数呼出しから返される空コンテナ
   VEE_NOT_DEFINED3,   // スペース確保用
   VEE_COORD,          // XY、XYZ等のデータを表す2個以上のdouble
   VEE_ENUM,           // 順序付きの文字列のリスト
   VEE_RECORD,         // VEEレコード - データを構造化
   VEE_NOT_DEFINED4,   // スペース確保用
   VEE_WAVEFORM,       // 時間マッピング付きのVEE_DOUBLEの1次元配列
   VEE_SPECTRUM        // 時間マッピング付きのVEE_PCOMPLEXの1次元配列
};
```

veeData.hファイルには、VEEデータ型との変換用のCデータ型が便宜のために定義されています。

```
typedef short int16;
typedef long int32;
typedef struct {double rval, ival;} veeComplex;
typedef struct {double mag, phase;} veePComplex;
typedef struct {double xval, yval;} vee2DCoord;
typedef struct {double xval, yval, zval;} vee3DCoord;
typedef void veeDataContainer;
typedef veeDataContainer* VDC;
```

データ型には下記の次元数(numDims)を指定できます。

```
enum veeShape
{
   VEE_SHAPE_SCALAR,   // 1個のデータ要素
   VEE_SHAPE_ARRAY1D,  // 1次元配列
   VEE_SHAPE_ARRAY2D,  // 2次元配列
   VEE_SHAPE_ARRAY3D,  // 3次元配列
   VEE_SHAPE_ARRAY,    // 4～10次元の配列
   VEE_SHAPE_ANY       // 未定義形状のためのプレースホルダ
};
```

配列にはマッピングを持たせることができます。通常の配列にはマッピングはありませんが、VEE_WAVEFORMとVEE_SPECTRUMの2つのデータ型にはマッピングがあり、配列要素が時間間隔に対応します。マッピングは下記のように指定されます。

```
enum veeMapType
{
   VEE_MAPPING_NONE,    // マッピングなし
   VEE_MAPPING_LINEAR,  // リニア・マッピング
   VEE_MAPPING_LOG      // 対数マッピング
};
```

一般的には、マッピングを指定する必要はありません。

スカラ・データの処理

CデータからVDCスカラを作成するには、下記の関数を使います。

```
VDC vdcCreateLongScalar( int32 aLong );

VDC vdcCreateDoubleScalar( double aReal );

VDC vdcCreateStringScalar( char *aString );

VDC vdcCreateComplexScalar( double realPart,
                            double imaginaryPart );

VDC vdcCreatePComplexScalar( double magnitude,
                             double phase );
VDC vdcCreate2DCoordScalar( double xval,
                            double yval );


VDC vdcCreate3DCoordScalar( double xval,
                            double yval,
                            double zval );
```

```
VDC vdcCreateCoordScalar( int16 aFieldCount,
                          double *values );
```

これらの関数はすべてVDCへのポインタを返し、エラーが発生した場合はNULLを返します。

VEE_WAVEFORM型とVEE_SPECTRUM型は定義上常に1次元配列なので、スカラとしては存在しません。

VDCの値を変更するためには、下記のルーチンが用意されています。

```
int32 vdcSetLongScalar( VDC aVD,
                        int32 aLong );

int32 vdcSetDoubleScalar( VDC aVD,
                          double aReal );

int32 vdcSetStringScalar( VDC aVD,
                          char *aStr );

int32 vdcSetComplexScalar( VDC aVD,
                           double realPart,
                           double imaginaryPart );

int32 vdcSetPComplexScalar( VDC aVD,
                            double magnitude,
                            double phase );

int32 vdcSet2DCoordScalar( VDC aVD,
                           double xval,
                           double yval );

int32 vdcSet3DCoordScalar( VDC aVD,
                           double xval,
                           double yval,
                           double zval );

int32 vdcSetCoordScalar( VDC aVD,
                         int16 aFieldCount,
                         double* values );
```

上と同じく、これらの関数は0またはエラー・コードを返します。

作成した、または関数から返されたスカラVDCからCのデータ型を得るために、下記のルーチンが用意されています。

```
int32 vdcGetLongScalarValue( VDC aVD,
                             int32 *aLong );

int32 vdcGetDoubleScalarValue( VDC aVD,
                               double *aReal );

char* vdcGetStringScalarValue( VDC aVD );

int32 vdcGetComplexScalarValue( VDC aVD,
                                veeComplex *aComplex );

int32 vdcGetPComplexScalarValue( VDC aVD,
                                 veePComplex *aPComplex );

int32 vdcGet2DCoordScalarValue( VDC aVD,
                            vee2DCoord *aCoord );

int32 vdcGet3DCoordScalarValue( VDC aVD,
                            vee3DCoord *aCoord );

double* vdcGetCoordScalarValue( VDC aVD,
                            int16 *aFieldCount );
```

一般に、これらの関数は最初の引数のVDCからデータを取り出し、2番目の引数であるCの変数(このセクションの初めに記載されたどれかの型)に格納します。エラーがなければ0、エラーが発生した場合はエラー・コードが返されます。

例外として、`vdcGetStringScalarValue()`関数は戻り値として文字列(エラーが発生した場合はNULL)を直接返し、`vdcGetCoordScalarValue()`関数はN次元座標データの配列へのポインタを返します(Nは引数に返されます)。

最後に、座標型に関して、座標次元の数を問い合わせたり、必要なら座標次元の数を変更したりすることができます。

```
int16 vdcNumCoordDims( VDC aVD );
int32 vdcCoordSetNumCoordDims( VDC, int16 );
```

配列データの処理

これらの関数は、VEEの型の配列VDCを作成します。指定した値はVDCにコピーされます。呼出し側のメモリが用いられることはありません。エラーが発生した場合、ヌル・ポインタが返されます。VDC配列を作成するには以下の関数群を使います。

- 下記の関数は、VEE_LONG型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数valuesが指すデータの配列は、指定したのと同じサイズでなければなりません。引数の型int32は、veeData.hでlong型に等しく定義されています。

```
VDC vdcCreateLong1DArray( int32 numPts,
                          int32 *values );
```

- 下記の関数は、VEE_STRING型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数stringsはポインタの配列を指し、それぞれのポインタがヌル終止文字列を指します。配列内の文字列の数は、指定したサイズと同じでなければなりません。引数の型int32は、veeData.hでlong型に等しく定義されています。

```
VDC vdcCreateString1DArray( int32 numPts,
                            char **strings );
```

- 下記の関数は、VEE_DOUBLE型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数valuesはデータの配列を指します。配列内のdoubleの数は、指定したサイズと同じでなければなりません。引数の型int32は、veeData.hでlong型に等しく定義されています。

```
VDC vdcCreateDouble1DArray( int32 numPts,
                            double *values );
```

- 下記の関数は、VEE_COMPLEX型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数の型int32は、veeData.hでlong型に等しく定義されています。引数valuesはveeComplex型の構造体の配列を指します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double rval, ival;} veeComplex;

VDC vdcCreateComplex1DArray( int32 numPts,
                             veeComplex *values );
```

■ 下記の関数は、VEE_PCOMPLEX 型の VDC を返します。割り当てるサイズは引数numPtsで決まります。引数の型int32は、veeData.hでlong型に等しく定義されています。引数valuesはveePComplex型の構造体の配列を指します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double mag, phase;} veePComplex;

VDC vdcCreatePComplex1DArray( int32 numPts,
                              veePComplex *values );
```

■ 下記の関数は、VEE_COORD型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数の型int32は、veeData.hでlong型に等しく定義されています。引数valuesはvee2DCoord型の構造体の配列を指します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double xval, yval;} vee2DCoord;

VDC vdcCreate2DCoord1DArray( int32 numPts,
                             vee2DCoord *values );
```

■ 下記の関数は、VEE_COORD型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数の型int32は、veeData.hでlong型に等しく定義されています。引数valuesはvee3DCoordの構造体の配列を指します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double xval, yval, zval;} vee3DCoord;

VDC vdcCreate3DCoord1DArray( int32 numPts,
                             vee3DCoord *values );
```

■ 下記の関数は、VEE_COORD型のVDCを返します。割り当てるサイズは引数numPtsで決まります。引数aFieldCountは、座標のフィールド数を指定します。引数の型int32は、veeData.hでlong型に等しく定義されています。引数valuesはdouble型の配列を指します。この配列の長さは、numPtsとaFieldCountの積に等しくなければなりません。

```
VDC vdcCreateCoord1DArray( int32 numPts,
                           int16 aFieldCount,
                           double *values );
```

■ 下記の関数は、VEE_WAVEFORM型のVDCを返します。サンプル数は引数numPtsに等しくなります。波形の開始時間と終了時間は、引数fromとthruで指定されます。引数mapTypeはveeData.hで定義されてたVMT型で、用いられるマッピングのタイプを示します。詳細については、579ページ「データ型、形状、マッピング」を参照してください。

引数dataが指すdoubleの配列は、引数numPtsと同じサイズでなければなりません。引数の型int32は、veeData.hでlong型に等しく定義されています。

```
VDC vdcCreateWaveform( int32 numPts,
                       double from,
                       double thru,
                       VMT mapType,
                       double *data );
```

■ 下記の関数は、VEE_SPECTRUM型のVDCを返します。サンプル数は引数numPtsに等しくなります。スペクトルの開始周波数と終了周波数は、引数fromとthruで指定されます。引数mapTypeはveeData.hで定義されてたVMT型で、用いられるマッピングのタイプを示します。詳細については、579ページ「データ型、形状、マッピング」を参照してください。

引数dataが指すveePComplex型の配列は、引数numPtsと同じサイズでなければなりません。veePComplex型はveeData.hで定義された構造体です。

```
typedef struct {double mag, phase;} veePComplex.
```

構造体の配列はコピーされます。引数の型int32は、veeData.hでlong型に等しく定義されています。

```
VDC vdcCreateSpectrum( int32 numPts,
                       double from,
                       double thru,
                       VMT mapType,
                       veePComplex *data);
```

上記の関数群は、配列サイズ、配列を表現するために必要な付加データ(VEE_WAVEFORM型やVEE_SPECTRUM型のマッピング・データなど)、および配列データを指定して、VDC(エラーが発生した場合はNULL)を得るものです。

VDCからC配列に戻すには下記の関数を使います。

- 下記の関数は、int32型の配列へのポインタを返します。引数aVDはVEE_LONG型で、配列でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
int32* vdcGetLong1DArray( VDC aVD,
                          int32 *numPts );
```

- 下記の関数は、double型の配列へのポインタを返します。引数aVDはVEE_DOUBLE型でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
double* vdcGetDouble1DArray( VDC aVD,
                             int32 *numPts );
```

- 下記の関数はポインタの配列へのポインタを返し、配列内のポインタのそれぞれがヌル終止文字列を指します。引数aVDはVEE_STRING型でなければなりません。参照渡しの引数numPtsには文字列の数が返されます。

```
char** vdcGetString1DArray( VDC aVD,
                            int32 *numPts );
```

- 下記の関数は、veeComplex型の構造体の配列へのポインタを返します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double rval, ival;} veeComplex;
```

引数aVDはVEE_COMPLEX型でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
veeComplex* vdcGetComplex1DArray( VDC aVD,
                                  int32 *numPts );
```

- 下記の関数は、veePComplex型の構造体の配列へのポインタを返します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double mag, phase;} veePComplex;
```

引数aVDはVEE_PCOMPLEX型でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
veePComplex* vdcGetPComplex1DArray( VDC aVD,
                                    int32 *numPts );
```

- 下記の関数は、vee2DCoord型の構造体の配列へのポインタを返します。この構造体はveeData.hで下記のように定義されています。

```
typedef  struct {double xval, yval;} vee2DCoord;
```

引数aVDはVEE_COORD型でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
vee2DCoord* vdcGet2DCoord1DArray( VDC aVD,
                                  int32 *numPts );
```

- 下記の関数は、vee3DCoord型の構造体の配列へのポインタを返します。この構造体はveeData.hで下記のように定義されています。

```
typedef  struct {double xval, yval, zval;} vee3DCoord;
```

引数aVDはVEE_COORD型でなければなりません。参照渡しの引数numPtsには配列の長さが返されます。

```
vee3DCoord* vdcGet3DCoord1DArray( VDC aVD,
                                  int32 *numPts );
```

- 下記の関数は、double型の配列へのポインタを返します。引数aVDはVEE_COORD型でなければなりません。参照渡しの引数numPtsには配列内の座標データの個

数が返されます。参照渡しの引数aFieldCountには、各座標データのフィールド数が返されます。返された配列の長さは、numPtsとaFieldCountの積になります。

```
double* vdcGetCoord1DArray( VDC aVD,
                            int32 *numPts,
                            int16 *aFieldCount );
```

- 下記の関数は、double型の配列へのポインタを返します。引数aVDはVEE_WAVEFORM型でなければなりません。参照渡しの引数numPts、from、thru、mapTypeには、順番に配列の長さ、開始時間、終了時間、マッピングのタイプが返されます。

```
double* vdcGetWaveform( VDC aVD,
                        int32 *numPts,
                        double *from,
                        double *thru,
                        VMT *mapType );
```

- 下記の関数は、veePComplex型の構造体の配列へのポインタを返します。この構造体はveeData.hで下記のように定義されています。

```
typedef struct {double mag, phase;} veePComplex;
```

引数aVDはVEE_WAVEFORM型でなければなりません。参照渡しの引数numPts、from、thru、mapTypeには、順番に構造体配列の長さ、開始周波数、終了周波数、マッピングのタイプが返されます。

```
veePComplex* vdcGetSpectrum( VDC aVD,
                             int32 *numPts,
                             double *from,
                             double *thru,
                             VMT *mapType );
```

これらの関数は、VDCを受け取って、データの配列へのポインタを直接返し、配列のサイズ(およびその他の関連情報)を引数に返します。

配列を作成したら、下記の関数を使って配列のチェック、問合せ、操作が可能です。

```
int32 vdcSetNumDims( VDC, int16 );

int16 vdcGetNumDims( VDC );

int32 vdcSetDimSizes( VDC, int32* );

int32 *vdcGetDimSizes( VDC );

int32 vdcCurNumElements( VDC );
```

Enum型

VEEのEnum型は、すでに説明したように、順序付きの文字列のリストであり、下記のルーチンで操作が可能です。

- 下記の関数は、指定した数の文字列-順序数ペアを持つ空のVEE_ENUM構造体を作成します。エラーが発生した場合はNULL VDCを返します。

```
VDC vdcCreateEnumScalar( int16 numberOfPairs );
```

- 下記の関数は、指定されたVEE ENUM構造体にEnumペアを登録し、更新された構造体を返し、0またはエラー・コードを返します。

```
int32 vdcEnumAddEnumPair( VDC aVD,
                          char* aString,
                          int32 aValue );
```

- 下記の関数は、引数の順序数で指定される Enum ペアを削除します。0 またはエラー・コードが返されます。

```
int32 vdcEnumDeleteEnumPairWithOrdinal( VDC aVD,
                                        int32 anOrd );
```

- 下記の関数は、他のvdcEnumルーチンが使用する順序数を設定します。0またはエラー・コードが返されます。

```
int32 vdcSetEnumScalar( VDC aVD,
                        int32 anOrdinal );
```

- 下記の関数は、vdcSetEnumScalar()で設定された順序数に対してVEE_ENUM構造体に文字列を登録します。

```
int32 vdcEnumDeleteEnumPairWithStr( VDC aVD,
                                    char* aString );
```

■ 下記の関数は、vdcSetEnumScalar()で設定された現在の順序数を返します。

```
int32 vdcGetEnumOrdinal( VDC aVD );
```

■ 下記の関数は、現在の順序数に対応する文字列を返し、エラーが発生した場合はNULLを返します。

```
char* vdcGetEnumString( VDC aVD );
```

マッピング関数　VEE DATA APIには、配列のマッピングを操作するための下記の関数が用意されています。

```
int32 vdcAtDimPutLowerLimit( VDC aVD,
                             int16 aDim,
                             double aValue );
   // マッピングの下限を指定

int32 vdcAtDimPutUpperLimit( VDC aVD,
                             int16 aDim,
                             double aValue );
   // マッピングの上限を指定

int32 vdcAtDimPutRange( VDC aVD,
                        int16 aDim,
                        double lowerLimit,
                        double upperLimit );
   // "vdcAtDimPutLowerLimit"と
         "vdcAtDimPutUpperLimit"の組合わせ

int32 vdcAtDimPutMapping( VDC aVD,
                          int16 aDim,
                          VMT aMapping);
   // 上記の制限値の間でマッピングを設定

int32 vdcMakeMappingsSame( VDC VD1,
                           VDC VD2 );
   // 2個のコンテナを同じマッピングに設定

int32 vdcUnMap( VDC aVD );
   // コンテナからマッピング情報を削除
```

その他の関数　他のVEE DATA API関数として、下記のものがあります。

- VDCの型を取得します。エラーが発生した場合は`VEE_NOTDEFINED1`を返します。

```
enum veeType vdcType( VDC aVD );
```

- VDCのコピーを作成します。エラーが発生した場合はNULLを返します。

```
VDC vdcCopy( VDC oldVD );
```

■ コンテナを破棄し、メモリを解放します。エラーが発生した場合はNULLを返します。

```
VDC vdcFree( VDC aVD );
```

■ 最新のエラーのエラー番号/メッセージを取得します。

```
int16 veeGetErrorNumber( void );
char *veeGetErrorString( void );
```

■ エラー番号を0にリセットします。

```
void veeClearErrorNumber(void)
```

副作用としてエラー・コードを設定する可能性があるCまたはVEEの関数を呼び出し、その後で`veeGetErrorNumber()`を使ってエラー・コードを取得する場合には、関数を呼び出す前に上記の関数でエラー番号を0にリセットしておきます。あらかじめ`veeClearErrorNumber()`を呼び出しておけば、`veeGetErrorNumber()`で返されたエラー・コードが、`veeClearErrorNumber()`と`veeGetErrorNumber()`の呼出しの間で実行された関数で設定されたものであることが確実になります。

索引

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

W

X

あ

か

さ

た

な

は