NATIONAL INSTRUMENTS™
TestStand™

TestStand 入門

NATIONAL INSTRUMENTS™

2001 年 12 月版
製品番号 323152A-01

インターネットサポート
サポート電子メール：supportjapan@ni.com
電子メール：infojapan@ni.com
FTP サイト：ftp.ni.com
日本語ホームページ：http://www.ni.com/jp

電話サポート（日本）
Tel：03-5472-2981
Fax：03-5472-2977

海外オフィス
イスラエル 03 6393737、イタリア 02 413091、インド 91 80 535 5406、英国 01635 523545、
オーストラリア 03 9879 5166、オーストリア 0662 45 79 90 0、オランダ 0348 433466、
カナダ（オタワ）613 233 5949、カナダ（カルガリー）403 274 9391、カナダ（ケベック）514 694 8521、
カナダ（トロント）905 785 0085、カナダ（モントリオール）514 288 5722、韓国 02 3451 3400、
ギリシャ 30 1 42 96 427、シンガポール 2265886、スイス 056 200 51 51、スウェーデン 08 587 895 00、
スペイン 91 640 0085、スロベニア 386 3 425 4200、台湾 02 2528 7227、中国（上海）021 6555 7838,
中国（ShenZhen）0755 3904939、チェコ 02 2423 5774、デンマーク 45 76 26 00、ドイツ 089 741 31 30、
ニュージーランド 09 914 0488、ノルウェー 32 27 73 00、フィンランド 09 725 725 11、
フランス 01 48 14 24 24、ベルギー 02 757 00 20、ブラジル 011 3262 3599、ポーランド 0 22 528 94 06,
ポルトガル 351 210 311 210、香港 2645 3186、マレーシア 603 9596711、南アフリカ 11 805 8197、
メキシコ 001 800 010 0793、ロシア 095 238 7139

National Instruments Corporation
11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

日本ナショナルインスツルメンツ株式会社
〒 105-0011 東京都港区芝公園 2-4-1 秀和芝パークビル A 館 4F Tel：03-5472-2970

サポート情報の詳細については、付録 A「技術サポートのリソース」を参照してください。本書に対するご意見は、techpubs@ni.com まで電子メールでお送りください。

必ずお読みください

保証

限定的保証：National Instruments Corporation（以下「NI」という）のハードウェア製品は、NIがお客様に製品を出荷した日（以下「配送日」）から次の一定期間、素材及び製作技術上の欠陥に対して保証されています。すなわちIEEE 488に未対応のハードウェア製品については1年間、IEEE 488対応のハードウェア製品については2年間、ケーブルについては90日間の保証が適用されます。ソフトウェア製品の場合は、該当するNIのライセンス条項に基づき、お客様にライセンスが供与されます。配送日から90日間は、NIのソフトウェア製品（但しNIのハードウェア製品に正しくインストールされている場合）について、(a)付属のマニュアル文書に従い実質的に機能すること、および(b)ソフトウェア製品が記録されている媒体は、通常の利用やサービスにおいて素材及び製作技術上の欠陥を有しないこと、が保証されています。ライセンスが供与されたソフトウェア製品の交換については、当初の保証期間の残存期間または30日間のいずれか長い期間について保証されます。お客様が保証期間中の製品をNIに返却するには、事前にNIから返品確認（Return Material Authorization: RMA）番号を取得してください。また、修理・交換品をお客様からNIへ、NIからお客様あてに返送する送料は、お客様の負担になります。返却された製品を検査、試験した後、同製品には欠陥がないとNIが判断した場合、その旨をお客様に通知します。同製品の返送にかかる費用はお客様に負担いただき、試験にかかった費用については後日請求致します。製品の不具合が事故、乱用、誤用、お客様による不適切なキャリブレーションによって発生した場合や、お客様が当該NIソフトウェアと共に使用することが予定されていない第三者のソフトウェアと共に利用した場合、不適切なハードウェアまたはソフトウェアのキーを利用した場合、独断で保守または修理を行った場合、本書に定める限定的保証は無効となります。

救済方法：上記の限定的保証において、NIの唯一の義務（およびお客様の唯一の救済方法）は、NIの選択により、支払われた料金の返還、または欠陥製品の修理・交換に限定されます。ただし、NIが、当該製品に適用される保証期間内に、こうした欠陥について書面で通知を受け取った場合に限ります。お客様は、訴訟原因の発生から1年を超えて経過した後は、上記の限定的保証に基づく本救済方法を強制するために訴訟を提起することはできません。

返品および解約に関する方針：お客様は、不要な製品については、配送日から30日以内であれば、当該製品を返却することができます。この場合の送料はお客様にご負担いただきます。上記30日間満了後は不要な製品の返品は受け付けません。特殊機器または特殊なサービスが係わる場合、お客様は、進行中の関連作業全てに対して責任を負うものとします。ただし、お客様から書面による解約の通知を受領した場合、NIはただちに損害を軽減するための責任ある対策を講ずるものとします。製品の返却の際は、NIから返品確認番号を取得してください。お客様がNIに対して行った説明・表示等が虚偽または誤解を生じさせるものであった場合には、NIは注文を取り消すことがあります。

本書の内容については万全を期しており、技術的内容に関するチェックも入念に行っております。技術的な誤りまたは乱丁・落丁につきましては、お客様への事前の通告なく、NIにて次の版から修正する権利があるものとします。本書で誤りと思われる個所については、NIにご確認ください。NIは、本書およびその内容により、またはそれに関連して発生した損害に対して一切責任を負いません。

本書に規定する保証を唯一の保証とします。NIは、明示・暗示を問わず、ここに記載された以外の保証は行いません。特に、商品適合性の保証や特定用途に対する適合性についての保証は行いません。NIの過失または不注意により発生した損害に関するお客様の賠償請求権は、お客様が製品に支払われた金額を上限とします。NIは、データの消失、利益の逸失、製品の使用から生じた損失や、付随的または結果的に生じた損害に対して、その損害が発生する可能性を通知されていた場合でも、一切の責任を負いません。かかるNIの限定的責任は、訴訟方式、過失責任を含む契約上の責任または不法行為責任を問わず適用されます。NIに対する訴訟は、訴訟原因の発生から1年以内に提起する必要があります。NIは、NIが合理的に支配可能な範囲を超えた原因により発生した履行遅延に関しては一切の責任を負いません。所有者が、NIの指示通りインストール、操作、保守を実施しないことにより発生した損害、欠陥、誤作動、動作不良について、また、所有者による製品の改変、乱用、誤用、または不注意な行動、さらに停電、電源サージ、火災、洪水、事故、第三者の行為、その他の合理的に支配可能な範囲を超えた事象により発生する損害、欠陥、誤作動、動作不良については本書に定める保証の対象となりません。

著作権

商標

National Instrumentsの製品を医療用に使用することに関する警告

(1) National Instruments Coporation（以下「NI」という）の製品は、外科移植もしくはそれに関連する用途、または作動不良により人体に深刻な傷害を及ぼすことが合理的に予期される生命維持装置の重要なコンポーネントとしての用途に適した信頼性のレベルでのコンポーネントや試験を採用して設計されておりません。(2) 上記用途を含む、あらゆるアプリケーションにおいて、不利な要因によってソフトウェア製品の操作の信頼性が損なわれる可能性があります。これには、電力供給の変動、コンピュータハードウェアの誤作動、コンピュータ・オペレーティングシステム・ソフトウェアの適応性、アプリケーション開発に利用したコンパイラや開発ソフトウェアの適応性、インストールの間違い、ソフトウェアとハードウェアの互換性の問題、電子監視機器または制御機器の誤作動または故障、電気システム（ハードウェア及び/又はソフトウェア）の一時的な障害、予期せぬ使用または誤用、ユーザまたはアプリケーション設計者側のミスなどがありますが、これに限定されません(本書においてこのよ

うな不利な要因を総称して「システム故障」といいます)。システム故障が財産または人体に危害を及ぼす可能性（身体の損傷および死亡の危険を含む）があるアプリケーションにおいては、システム故障の危険があるため、単独の電気システム方式のみに依存すべきではありません。損害、人体への傷害、または死亡といった事態を避けるため、ユーザまたはアプリケーション設計者は、システム故障から保護するための合理的に慎重な対策を取る必要があります。これには、バックアップメカニズム、または非常停止メカニズムなどがありますが、これに限定されません。各エンドユーザのシステムはカスタマイズされており、NIの試験プラットフォームとは異なること、またユーザやアプリケーション設計者が、NIが評価したことのない方法や、予期しない方法でNI製品を他の製品と組み合わせて使用する可能性があることから、NI製品をシステムまたはアプリケーションに統合する場合は、ユーザまたはアプリケーション設計者が、最終的にNI製品の適合性(かかるシステムまたはアプリケーションの適切な設計、処理、安全レベルが含まれますが、これに限定されません。）の検証および確認における責任を負うものとします。

本書で使用する表記規則

本書では以下の表記規則を使用します。

→ →記号に沿って、入れ子のメニュー項目やダイアログボックスをたどっていくと、最終的に必要な操作を実行することができます。**ファイル→ページ設定→オプション**という順になっている場合、まず**ファイル**メニューをプルダウンし、次に**ページ設定**項目を選択して、最後のダイアログボックスから**オプション**を選択します。

 このアイコンは、注意すべき重要な情報があることを示しています。

 このアイコンは、人体への損傷、データの損失、システムのクラッシュなどを防止するための注意事項があることを示しています。

太字 太字のテキストは、メニュー項目やダイアログボックスなど、ソフトウェアでユーザが選択（クリック）する必要のある項目を表します。また、フロントパネル上のパラメータ名、制御器やボタン、ダイアログボックスまたはその一部、メニュー名、パレット名も表します。

下線 下線つきのテキストは、重要な事項を示します。

斜体 このフォントスタイルは変数を示します。または、ユーザが入力する必要がある語または値のプレースホルダを示します。

`monospace` このフォントのテキストは、キーボードから入力する必要のあるテキストや文字、コードの一部、プログラムサンプル、構文例を表します。また、ディスクドライブ名、パス名、ディレクトリ名、プログラム名、サブプログラム名、サブルーチン名、デバイス名、関数名、演算名、変数名、ファイル名と拡張子、引用するコードにも使います。ただし、日本語の文字の入力や表示は、前後の文と区別するため、「」で囲んでいる場合もあります。

`monospace` の太字 このフォントの太字テキストは、画面に自動印刷されるメッセージや応答を示します。また、他のサンプルとは異なるコードラインを強調する場合にも使用します。ただし、日本語の文字の入力や表示は、前後の文と区別するため、「」で囲んでいる場合もあります。

プラットフォーム このフォントのテキストは特定のプラットフォームを示し、そこに記載された説明はそのプラットフォームにのみ適用されます。

右クリック **(Macintosh)** 右クリックと同じ操作を実行するには、<Command> を押したままクリックします。

目次

第 4 章
シーケンスをデバッグする

第 5 章
変数およびプロパティを使用する

第 6 章
テストの作成とデバッグ

第 7 章
TestStand ランタイムオペレータインタフェースを使用する

表一覧

1

TestStand の概要

この章では、TestStand のインストール方法、および製品の概要について説明します。

TestStand は、フル装備のテストシステムを構築、カスタマイズ、および運用するための柔軟で強力なテストシステムフレームワークです。

TestStand をインストールする

テストアプリケーションの構築を開始する前に、コンピュータに TestStand をインストールする必要があります。TestStand セットアッププログラムは、ソフトウェアを 5 分ほどでインストールします。

最小システム条件

Windows 対応 TestStand を実行するには、以下のようなシステムが必要です。

- Windows NT 4.0 以降、または Windows 98/95 以降
- 266 MHz Pentium クラス以上のマイクロプロセッサを搭載したコンピュータ
- SVGA 解像度（またはそれ以上）のビデオアダプタ、最低 800 × 600 のビデオ解像度
- 64 MB の最小メモリ
- 100 MB の空きハードディスク容量（250 MB 推奨）
- Microsoft 対応マウス

TestStand をインストールする

次の手順に従って、TestStand をインストールしてください。

1. コンピュータとモニタに電源が入っていることと、Windows NT 4.0 以降または Windows 98/95 がインストールされていることを確認します。
2. 開いているすべての Windows アプリケーションを閉じ、オペレーティングシステムを Windows にしておきます。
3. CD-ROM ドライブにインストール CD を挿入します。
4. デスクトップのタスクバーから**実行**オプションを選択します。

5. コマンドラインボックスに *x*:\tssetup.exe（ここで *x* は使用するドライブ名です）と入力し、**OK** をクリックします。
6. ダイアログボックスに表示される指示に従ってください。

TestStand の機能をすべて活用していただくため、ナショナルインスツルメンツでは、完全な TestStand プログラムをインストールすることをお勧めします。オプションでインストールする場合は、希望するオプションを選んで画面の指示に従ってください。必要な場合は、後ほどセットアッププログラムを再度実行して追加のファイルをインストールすることも可能です。

注意　TestStand API を呼び出す LabVIEW VI が LabVIEW 5.1 より前のバージョンで保存されている場合は、TestStand 2.0 をインストールする前に、LabVIEW 5.1 以降のバージョンで一括コンパイルする必要があります。VI を一括コンパイルしないと、TestStand API を使用する ActiveX ダイアグラムノードをすべて手作業で入れ換えることが必要になります。TestStand 2.0 インストーラは、すでにインストールされている LabVIEW を検出するとメッセージボックスを表示します。

セットアッププログラムがインストールするファイル／プログラム

セットアッププログラムは、TestStand 開発環境およびその他のいくつかのファイルをシステムにインストールします。完全にインストールすると、TestStand の多くの機能を解説したサンプルファイルや本書で使用するチュートリアルプログラムもインストールされます。インストーラは、TestStand およびサブディレクトリの関連ファイルを、表 1-1 に示すとおりにハードディスクにインストールします。

表 1-1　TestStand のサブディレクトリ

ディレクトリ名	内容
AdapterSupport	LabVIEW および C/CVI 標準プロトタイプアダプタのサポートファイル
Api	LabWindows/CVI および MFC 用 TestStand ActiveX オートメーションサーバライブラリ
Bin	TestStand シーケンスエディタ実行可能ファイル、エンジン DLL、およびサポートファイル
Cfg	TestStand エンジンおよびシーケンスエディタオプション用構成ファイル
CodeTemplates	ステップタイプ用ソースコードテンプレート。このディレクトリには、NI および User サブディレクトリが含まれます。

表 1-1　TestStand のサブディレクトリ（続き）

ディレクトリ名	内容
Components	TestStand に付属のコンポーネントおよびユーザが開発したコンポーネント。このディレクトリには、コールバックファイル、コンバータ、アイコン、言語ファイル、プロセスモデルファイル、ステップタイプ、ソースファイル、およびユーティリティファイルが含まれます。また、NI および User サブディレクトリも含まれます。
Doc	ドキュメントファイル
Examples	シーケンスおよびテストのサンプル
OperatorInterfaces	ソースコードを含む LabVIEW、LabWindows/CVI、Microsoft Visual Basic、および Delphi オペレータインタフェースこのディレクトリには、NI および User サブディレクトリが含まれます。
Setup	TestStand インストーラ／アンインストーラ
Tutorial	本書のチュートリアルセッションで使用するシーケンスおよびコードモジュール

TestStand を習得する

TestStand の使用方法を習得するためには、以下のことを実行することをお勧めします。

1. TestStand に添付されている doc\readme.txt ファイルの熟読。
2. 本章の残りの部分を読んで、TestStand の概念および機能についての概要を習得。
3. 本書の 2 ～ 12 章までのチュートリアルセッションをすべて完了。
4. 『TestStand User Manual』の Chapter 1、「TestStand Architecture Overview」を読み、また他の章についても内容を把握。

TestStand を初めて使用する方は、本書のチュートリアルを完了することをお勧めします。『TestStand User Manual』では、一般に『TestStand 入門』の内容を把握されていることを前提としています。ただし、TestStand を習得または使用する際の疑問点については、他のマニュアルやオンラインヘルプも活用されることをお勧めします。

第 2 章、「シーケンスのロードと実行」では、TestStand のウィンドウ、メニュー、コマンド、およびダイアログボックスについて説明しています。『TestStand User Manual』には、TestStand の各ウィンドウおよびコンポーネントについての説明のみで構成された章があります。『TestStand 入門』を読み進める上でわからないことがありましたら、

それらの章を参照してください。各マニュアルの目次および索引に、そのマニュアルに掲載された重要な情報が一覧表示されています。

チュートリアルは、TestStand のシーケンスエディタの紹介に始まり、以降は TestStand でのシーケンスの構築方法を説明するセクションが続きます。チュートリアルの各ステップはその前のステップを基にしていますので、スキップすることなく順序どおりに学習することをお勧めします。

TestStand システムの概要

TestStand は、次のような主な特長を持つ強力なテストシステムフレームワークです。

- 即実行可能なテストシステムを実現する構成およびコンポーネント。
- 既製の構成およびコンポーネントを修正したり新しいコンポーネントを追加するさまざまな方法を用意。そのような拡張可能メカニズムにより、TestStand のテスト実行エンジンを変更することなく、個々のユーザのニーズに合ったテストシステムの作成が可能。ユーザによるカスタマイズを保持したまま新バージョンへのアップグレードが可能。
- 高度なシーケンス処理、実行、およびデバッグ機能と、ランタイム実行オペレータインタフェースとは別の強力なシーケンスエディタ。
- LabVIEW、LabWindows/CVI、Microsoft Visual Basic、および Delphi 用のソースコードによる 4 つの別々のランタイム実行オペレータインタフェース。
- 特定のアプリケーション開発環境（ADE）に依存しないシステム。さまざまな ADE でのテストモジュールの作成、および既存のモジュールや実行可能ファイルの呼び出しが可能です。ActiveX オートメーションサーバを制御可能なすべての言語でランタイム実行オペレータインタフェースを独自に作成することができます。
- テストシステム LabVIEW テストエグゼクティブツールキットバージョン 5.0 または LabWindows/CVI テストエグゼクティブツールキットバージョン 2.0 のシーケンスファイルを TestStand に変換。
- わかりやすい ActiveX アプリケーションプログラミングインタフェース（API）による、マルチスレッドテストシステムおよび他のシーケンスアプリケーションの構築。

本章の以降の部分では、TestStand の主要なソフトウェアコンポーネントについて説明します。

TestStand の主要なソフトウェアコンポーネント

このセクションでは、TestStand の主要なソフトウェアコンポーネントについて概要を説明します。

図 1-1 は、TestStand システムアーキテクチャの要素間の高レベルでの関係を示しています。

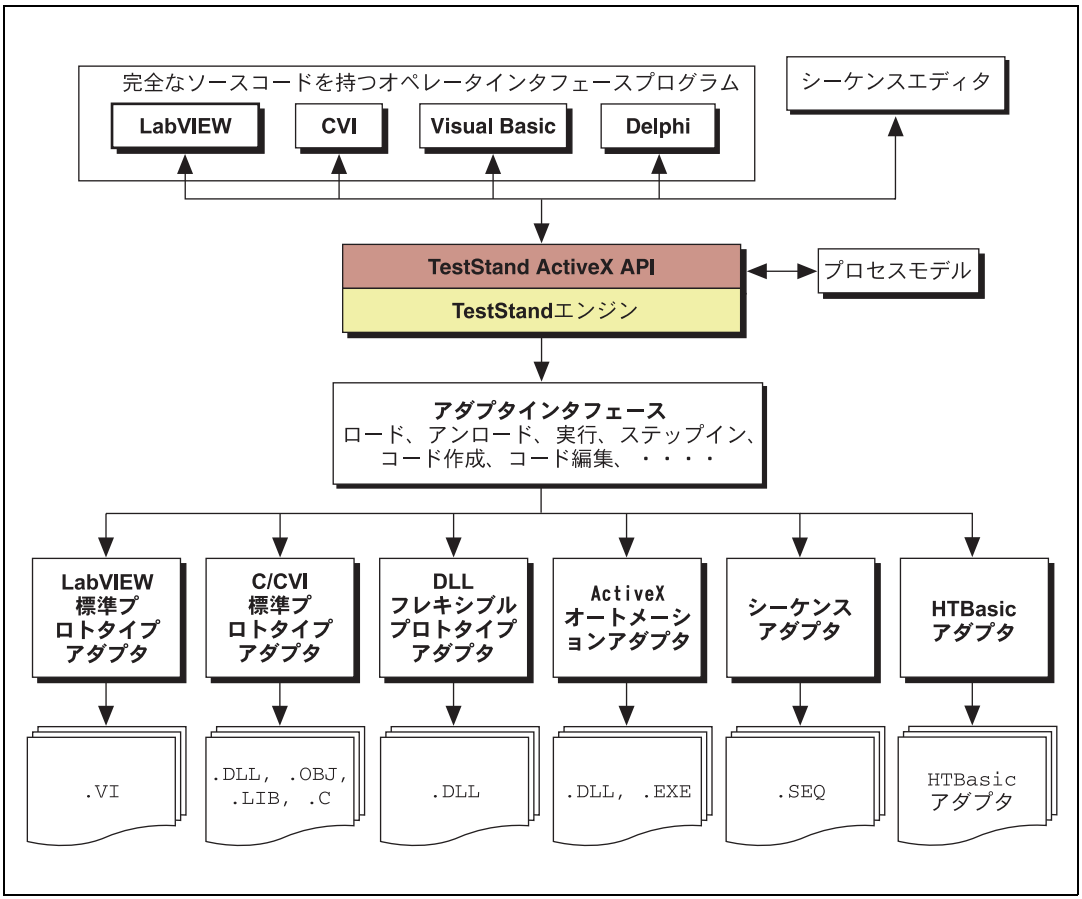

図 1-1　TestStand システムアーキテクチャ

図 1-1 に示すように、TestStand エンジンは、TestStand アーキテクチャにおいて中枢的な役割を果たします。TestStand エンジンは、シーケンスを実行します。シーケンスには、外部コードモジュールを呼び出すことができるステップが含まれています。標準アダプタインタフェースを持つモジュールアダプタを使用すると、TestStand で異なるタイプのコードモジュールをロードし実行することができます。TestStand のシーケン

スは、同じアダプタインタフェースを介してサブシーケンスを呼び出すことが可能です。TestStand は、プロセスモデルという特殊なタイプのシーケンスを使って、高レベルシーケンスのフローを制御します。TestStand エンジンは、TestStand シーケンスエディタとランタイムオペレータインタフェースが使用する ActiveX オートメーション API をエクスポートします。

TestStand シーケンスエディタ

TestStand シーケンスエディタは、シーケンスを作成、変更、およびデバッグするためのアプリケーションプログラムです。シーケンスエディタを使用すると、ステップタイプやプロセスモデルなど、TestStand の強力なすべての機能を簡単に使用できます。シーケンスエディタには、LabVIEW、LabWindows/CVI、Microsoft Visual C/C++ などのアプリケーション開発環境で使い慣れているデバッグツールが搭載されています。そのようなツールには、ブレークポイント、シングルステップ、関数コールのステップインまたはステップアウト、追跡、変数表示、ウォッチウィンドウなどがあります。

TestStand シーケンスエディタでは、複数の同時実行を開始することができます。同じシーケンスの複数のインスタンスを実行したり、異なるシーケンスを同時に実行することが可能です。各実行インスタンスには、それぞれ実行ウィンドウが開きます。追跡モードでは、実行ウィンドウに現在実行中のシーケンスのステップが表示されます。実行が中止されると、実行ウィンドウは次に実行するステップを表示し、シングルステップオプションを選べる状態になります。

TestStand ランタイムオペレータインタフェース

ご購入いただいた TestStand ソフトウェアには、4 つのランタイムオペレータインタフェースがソースと実行可能ファイルの 2 つの形式で含まれています。各ランタイムオペレータインタフェースは別々のアプリケーションプログラムです。各オペレータインタフェースは、基本的に開発された言語と ADE によって異なります。TestStand には、LabVIEW、LabWindows/CVI、Visual Basic、および Delphi で開発されたランタイムオペレータインタフェースが含まれています。

TestStand シーケンスエディタは生産ステーションで使用することもできますが、TestStand ランタイムオペレータインタフェースはそれほど複雑でなく完全にカスタマイズ可能です。シーケンスエディタ同様、ランタイムオペレータインタフェースを使用すると、複数の同時実行、ブレークポイントの設定、シングルステップなどが可能です。ただしシーケンスエディタと異なるところは、ランタイムオペレータインタフェースではシーケンスの変更ができず、またシーケンス変数、シーケンスパラメータ、ステップのプロパティなどが表示されません。

ランタイムオペレータインタフェースのカスタマイズ方法についての詳細は、『TestStand User Manual』の Chapter 16、「Run-Time Operator Interfaces」を参照してください。

TestStand テストシステムエンジン

TestStand テストシステムエンジンは、シーケンスの作成、編集、実行、およびデバッグのために ActiveX オートメーション API をエクスポートする DLL の集まりです。TestStand のシーケンスエディタとランタイムオペレータインタフェースは、エンジン API を使用します。エンジン API は、ActiveX オートメーションサーバへのアクセスをサポートするすべてのプログラミング環境から呼び出すことができます。したがって、テストモジュールからもエンジン API を呼び出すことができます。それには LabVIEW や LabWindows/CVI で作成したテストモジュールも含まれます。

エンジン API に関するドキュメントは、オンラインヘルプでご覧いただけます。オンラインヘルプには、シーケンスエディタの**ヘルプ**メニューまたは TestStand プログラムグループからアクセスすることができます。

モジュールアダプタ

TestStand シーケンスのほとんどのステップは、他のシーケンスやコードモジュールのコードを呼び出します。コードモジュールのコードを呼び出す際は、コードモジュールのタイプ、コードモジュールの呼び出し方法、およびパラメータをコードモジュールに渡す方法がわかっている必要があります。コードモジュールのタイプには、LabVIEW VI、DLL の C 関数、HTBasic ファイル、HTBasic サブルーチン、および LabWindows/CVI や他のコンパイラで作成したソース、オブジェクト、またはライブラリモジュールの C 関数などがあります。また、コードモジュールが使用するパラメータリストも必要です。

TestStand は、モジュールアダプタを使用してその情報を取得します。TestStand は現在、下記のような目的で次のモジュールアダプタを提供しています。

- **DLL フレキシブルプロトタイプアダプタ**—さまざまなパラメータタイプを持つ DLL の C 関数を呼び出します。
- **LabVIEW 標準プロトタイプアダプタ**—TestStand の標準 G パラメータリストを持つ LabVIEW VI を呼び出します。
- **C/CVI 標準プロトタイプアダプタ**—TestStand の標準 C パラメータリストを持つ C 関数を呼び出します。関数は、オブジェクトファイル、ライブラリファイル、または DLL のいずれかです。また、LabWindows/CVI 開発環境で現在使用しているプロジェクト内のソースファイルにあることもあります。

- **シーケンスアダプタ**—パラメータを持つサブシーケンスを呼び出します。
- **ActiveX オートメーションアダプタ**—メソッドを呼び出して ActiveX オブジェクトのプロパティにアクセスします。
- **HTBasic アダプタ**—パラメータなしの HTBasic サブルーチンを呼び出します。TestStand と HTBasic は、TestStand API を使ってデータを交換します。TestStand では、HTBasic バージョン 7.2 以降をサポートしています。

モジュールアダプタには、呼び出し規約とパラメータリスト以外にも重要な情報が含まれています。モジュールアダプタがアプリケーション開発環境（ADE）特定のものである場合、ADE の起動、その ADE での新しいコードモジュール用ソースコードの作成、およびその ADE での既存コードモジュール用ソースの表示は、そのアダプタが行うことができます。DLL フレキシブルプロトタイプアダプタは、DLL タイプライブラリにパラメータリストの情報についてクエリし、シーケンス開発者にその情報を提示することができます。

プロセスモデル

テストユニット（UUT）をテストするのに必要なのは、いくつかのテストの実行だけではありません。通常テストシステムは、テストを行うシーケンスを実行する前後に、一連の操作を実行する必要があります。共通の操作としては、UUT の識別、オペレータへの合否の通知、テストレポートの生成、および結果の記録などがあります。それらの操作によりテストのプロセスが定義されます。そのような操作の組み合わせおよびその実行フローのことをプロセスモデルといいます。

プロセスモデルを作成しておくと、別のテストシーケンスを作成する際に各シーケンスで同じ標準テスト操作を行う必要がないため、非常に便利です。プロセスモデルは変更可能なため、製造ラインや生産現場、または会社の体制や慣例などに合わせてテストプロセスを変更することができます。

TestStand では、プロセスモデルを定義するためのメカニズムを用意しています。プロセスモデルはシーケンスファイルの形式になっています。プロセスモデルは、他のシーケンスと同様に編集できます。TestStand には、3 つの完全機能型のプロセスモデルが含まれています。それは、シーケンシャルモデル、バッチモデル、およびパラレルモデルの 3 つです。シーケンシャルモデルは、一度に 1 つの UUT でテストシーケンスを実行するときに使用できます。パラレルモデルとバッチモデルを使用すると、複数の UUT で同じテストシーケンスを同時に実行することができます。

プロセスモデルは一連のエントリポイントを定義します。各エントリポイントは、プロセスモデルファイル内の 1 つのシーケンスです。プロセスモデル内に複数のエントリポイントを定義することで、テストステーションのオペレータはメインシーケンスをさまざまな方法で呼び出すことが可能になります。

たとえば、TestStand のデフォルトプロセスモデルである SequentialModel.seq には、UUT をテスト（Test UUTs）と一回実行（Single Pass）の 2 つのエントリポイントがあります。UUT をテスト（Test UUTs）エントリポイントは、UUT を繰り返し識別してテストするループを開始します。一回実行（Single Pass）エントリポイントは、1 つの UUT を識別せずにテストします。いずれのエントリポイントにも、結果の記録とレポートの生成のオプションがあります。そのエントリポイントは、実行エントリポイントといいます。実行エントリポイントは、シーケンスファイルウィンドウがアクティブになっていて、シーケンスファイルに MainSequence というシーケンスが含まれるときに、シーケンスエディタまたはオペレータインタフェースの**実行**メニューに表示されます。

バッチプロセスモデルは、複数の UUT を 1 つのまとまりとしてテストするテストソケットを制御するために使用します。たとえば、コモンキャリアに基板が取り付けられていることがあります。バッチモデルを使用すると、すべての基板のテストを同時に開始し終了することができます。バッチモデルにはまた、バッチ同期機能があります。たとえば、ステップはバッチ全体に適用されるため、ステップが各 UUT に対して一回ではなく各バッチに対して一回実行するように指定することができます。また、バッチモデルを使用すると、特定のステップやステップのグループが一度に複数の UUT に対して実行できないように指定したり、特定のステップがすべての UUT に対して同時に実行するように指定することも簡単にできます。さらに、バッチモデルでは、バッチ内の UUT のテスト結果を要約したバッチレポートを生成することもできます。

パラレルモデルは、複数の独立したテストソケットを制御するのに使用します。パラレルモデルを使用すると、いつでも任意のソケットでテストを開始したり停止したりすることができます。たとえば、ラジオをテストするのに 5 つのテストソケットがある場合があります。パラレルモデルを使用すると、他のソケットで他のラジオをテストしている間に、空いているソケットに新しいラジオをロードすることができます。

デフォルトプロセスモデルは SequentialModel.seq です。プロセスモデルを変更するには、**構成→ステーションオプション**を選択してモデルタブをクリックします。「ステーションモデル」リングから別のプロセスモデルを選んだり、**参照**ボタンをクリックしてモデルを指定することができます。また、「シーケンスファイルのプロパティ」ダイアログボックスで、シーケンスファイルで常に特定のプロセスモデルを使用するよう指定する

こともできます。プロセスモデルに関する詳細については、『TestStand User Manual』の Chapter 1、「TestStand Architecture Overview」、Chapter 3、「Configuring and Customizing TestStand」、および Chapter 14、「Process Models」を参照してください。

次の章に進み、TestStand のチュートリアルを開始してください。

2

シーケンスのロードと実行

本章では、TestStand シーケンスエディタでシーケンスをロードし実行する方法と、シーケンスエディタの各ウィンドウについて学習します。

TestStand の使用を開始する

TestStand の使用を開始するには、次の手順に従ってください。

1. TestStand を起動するには、**スタート**メニューから**プログラム→ National Instruments → TestStand → Sequence Editor** を選択します。シーケンスエディタにメインウィンドウが表示された後、図 2-1 に示すようなログインダイアログボックスが表示されます。

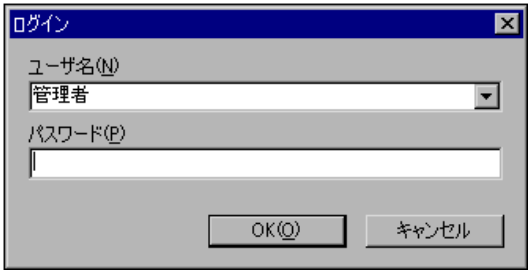

図 2-1 ログインダイアログボックス

2. デフォルトユーザ名が、図 2-1 に示すように管理者になっていない場合は、「ユーザ名」制御器をクリックしてポップアップリストから管理者を選択します。

3. 管理者ユーザログインのパスワードはデフォルトで空白になっていますので、パスワードを入力せずに **OK** ボタンをクリックします。
図 2-2 は、シーケンスエディタのメインウィンドウを示しています。

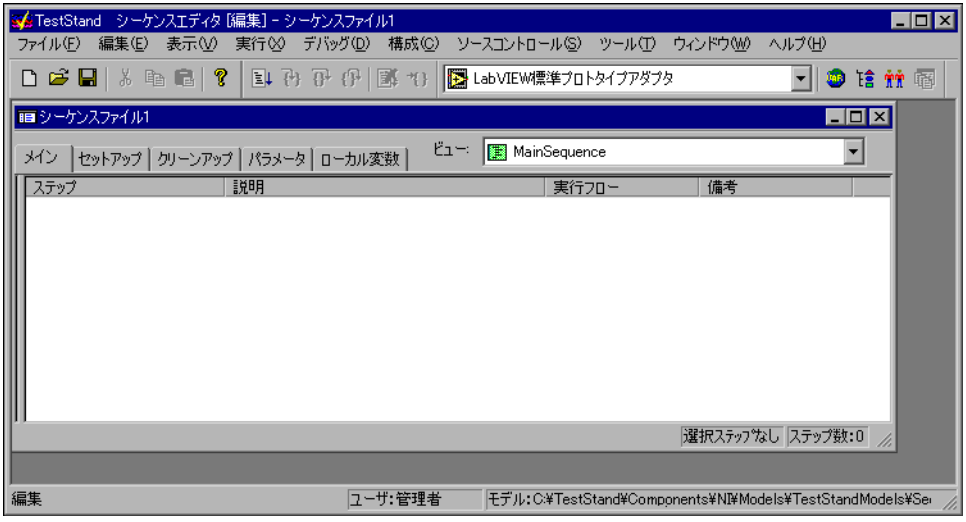

図 2-2　シーケンスエディタのメインウィンドウ

シーケンスエディタの概要

シーケンスエディタウィンドウは、メニューバー、ツールバー、開発ワークスペース、ステータスバーの 4 つの主な部分に分かれています。各部分の詳しい説明については、『TestStand User Manual』の Chapter 2、「Sequence Editor Concepts」を参照してください。

メニューバー

メニューバーに含まれるメニューは、**ファイル**、**編集**、**表示**、**実行**、**デバッグ**、**設定**、**ソース制御**、**ツール**、**ウィンドウ**、および**ヘルプ**です。シーケンスエディタのメニューを開いてみて、内容を確認してください。『TestStand User Manual』の Chapter 4、「Sequence Editor Menu Bar」に、各メニュー項目の詳しい説明があります。

ツールバー

ツールバーには、メニューバーの中で一般的に使用される項目へのショートカットが含まれています。図 2-3 に示すように、ツールバーには標準、デバッグ、環境の 3 つのセクションがあります。

図 2-3　シーケンスエディタツールバー

- **標準セクション**—シーケンスファイルの作成、ロード、保存、切り取り、および貼り付けのためのボタンが含まれています。
- **デバッグセクション**—シーケンスの実行、ステップイン、ステップオーバー、ステップアウト、一時停止、および実行の停止のためのボタンが含まれています。
- **環境セクション**—アダプタ選択リング、他の TestStand ステーションウィンドウを開くためのボタン、および開いているワークスペースをアクティブにするためのボタンが含まれています。

開発ワークスペース

開発ワークスペースは、シーケンスエディタの主要な領域です。シーケンスエディタはこの領域にウィンドウを表示します。

ステータスバー

ステータスバーは、シーケンスエディタの共通の情報を表示します。
図 2-4 に示すように、ステータスバーには選択項目ヘルプ、ログイン表示、およびモデル表示の 3 つのセクションがあります。

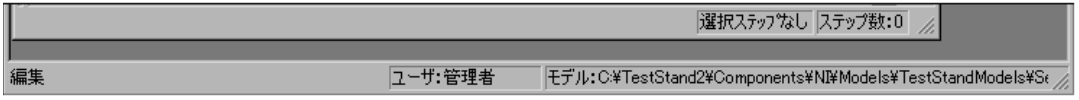

図 2-4　シーケンスエディタのステータスバー

- **選択項目ヘルプ**—現在選択されているメニュー項目に関する情報を表示します。
- **ログイン表示**—現在のユーザのユーザ名を表示します。
- **モデル表示**—プロセスモデルファイルのパス名を表示します。

TestStand のすべてのウィンドウは、TestStand のメニュー項目かオペレーティングシステムの通常のウィンドウ操作の方法で操作することができます。たとえば、画面上の TestStand ウィンドウの最大化、最小化、

位置変更や閉じる操作などは、Windows の標準ウィンドウ処理と同じ方法でできます。

シーケンスファイルをロードする

TestStand シーケンスエディタの機能を見るには、まず TestStand シーケンスエディタにシーケンスファイルをロードする必要があります。そのためには、次の手順に従ってください。

1. **ファイル→開く**を選択します。すると、図 2-5 に示すように Open ダイアログボックスが表示されます。

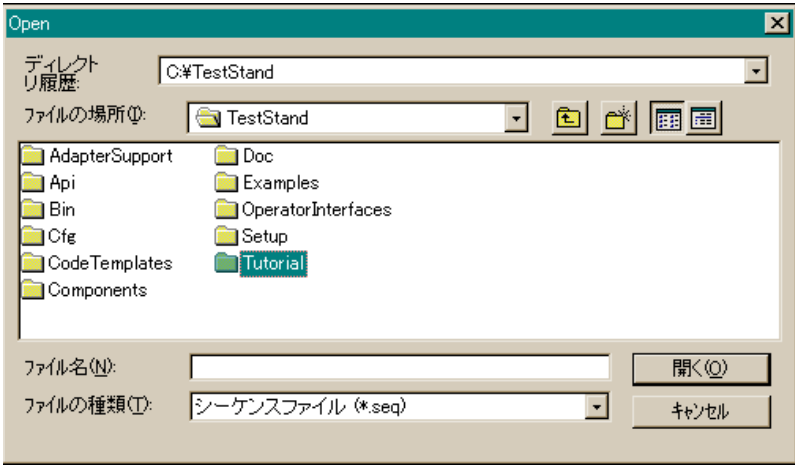

図 2-5　Open ダイアログボックス

2. `TestStand\Tutorial` サブディレクトリを開きます。
3. `Tutorial` サブディレクトリから `Sample1.seq` シーケンスファイルを選択して、**開く**ボタンをクリックします。

シーケンスファイルを開くと、図 2-6 に示すように、シーケンスエディタに新しいシーケンスファイルウィンドウが表示されます。

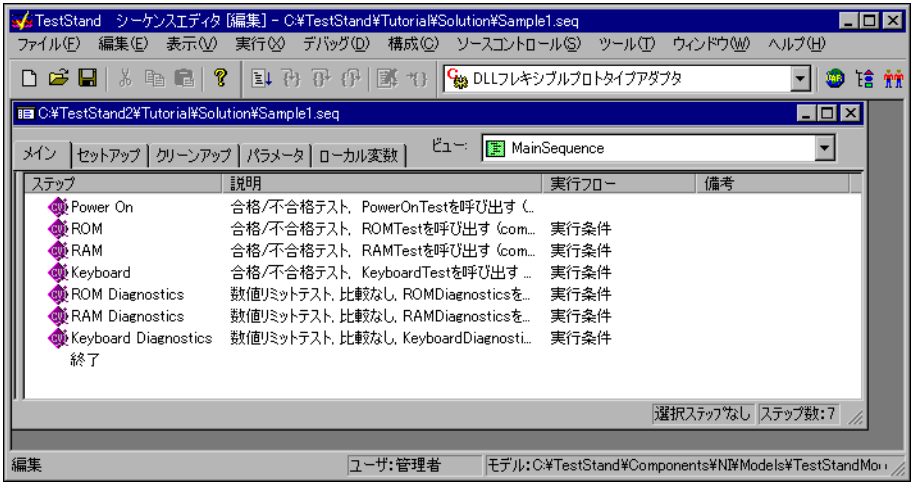

図 2-6　Sample1.seq シーケンスファイルウィンドウ

メモ　ご使用のコンピュータで以前に他の人がこのチュートリアルを使用している場合は、変更が加えられていないチュートリアルファイルを使用するために、TestStand を再インストールすることが必要な場合があります。

他の人が同じコンピュータでチュートリアルファイルを使用することがある場合は、**別名で保存**オプションを使用して異なるファイル名でファイルを保存するようにしてください。ファイルを保存する必要がある場合は、このマニュアルでお勧めのファイル名を指定しています。

`Sample1.seq` シーケンスファイルは、ご使用のコンピュータのシミュレーションテストで、さまざまなハードウェアコンポーネントを選択してテストを " 不合格 " にすることができます。シーケンスは、LabWindows/CVI で作成したダイナミックリンクライブラリ（`.dll`）の関数であるテストを実行します。

シーケンスファイルは、シーケンスエディタ内で個別のウィンドウとして表示されます。このウィンドウを*シーケンスファイルウィンドウ*といいます。複数のシーケンスファイルをシーケンスエディタにロードすると、各シーケンスファイルが個々のシーケンスファイルウィンドウに表示されます。

シーケンスエディタウィンドウの右上部分にある「ビュー」リングを使用して、ファイルの表示する部分を選択します。「ビュー」リングを使用すると、個々のシーケンス、ファイル内の全シーケンスのリス

ト、ファイル内のグローバル変数、またはファイルで使用するデータおよびステップタイプを表示することができます。

図 2-7 は、`Sample1.seq` シーケンスファイルの「ビュー」リングの内容を示しています。

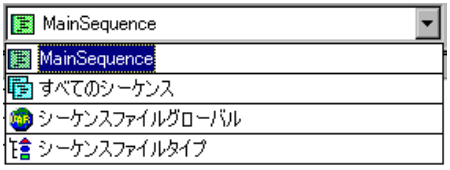

図 2-7　シーケンスファイルの「ビュー」リング

4. まだ選択されていない場合は、「ビュー」リングで `MainSequence` を選択します。

個々のシーケンスのビューにはメイン、セットアップ、クリーンアップ、パラメータ、およびローカルの 5 つのタブがあります。タブを選択して、シーケンスの表示したい部分を選ぶことができます。

メイン、セットアップ、クリーンアップの各タブには、シーケンスのステップグループの 1 つが表示されます。タブをクリックするとタブの内容を表示できます。各ステップグループに挿入するステップの目的は以下のとおりです。

- **メイン**—UUT をテストします。
- **セットアップ**—計測器、フィクスチャ、および UUT を初期化または構成します。
- **クリーンアップ**—計測器、フィクスチャ、および UUT を電圧低下または初期化解除します。

各タブの内容を確認して、終わったらメインステップグループに戻ります。

シーケンスについて

シーケンスは、一連のステップから構成されます。TestStand におけるステップは、計測器の初期化、複雑なテストの実行、シーケンスの実行フローに影響を及ぼすような決定など、さまざまなことを行います。ステップは、他のステップにジャンプしたり、サブシーケンスや外部コードモジュールを呼び出したり、または変数やプロパティの値を変更したりすることができます。

シーケンスのステップには、他のシーケンスを呼び出すものもあります。シーケンスはパラメータを持つこともできるため、値の受け渡しもすることができます。パラメータタブで、シーケンスのパラメータを定義します。

シーケンスは、ローカル変数をいくつでも持つことができます。ローカル変数を使用して、ステップが使用する値を保持します。また、ローカル変数を使用すると、カウントや中間値の保持、またはその他のあらゆる値の格納ができます。ローカルタブで、シーケンスのローカル変数を定義します。

シーケンスを実行する

ここでシーケンス実行の 2 つの方法について検討します。1 つはシーケンスを直接実行する方法で、もう 1 つはプロセスモデルを使用してシーケンスを実行する方法です。プロセスモデルとは、上位シーケンスフローを制御するための特殊なタイプのシーケンスです。

追跡オプションを設定する

シーケンスを実行する前に、以下の手順で、シーケンスエディタの追跡オプションが正しく構成されていることを確認します。

1. **構成→ステーションオプション**を選択します。ステーションオプションダイアログボックスが表示されます。
2. 実行タブのオプションが図 2-8 に示すように設定されていることを確認します。違っている設定をすべて変更します。

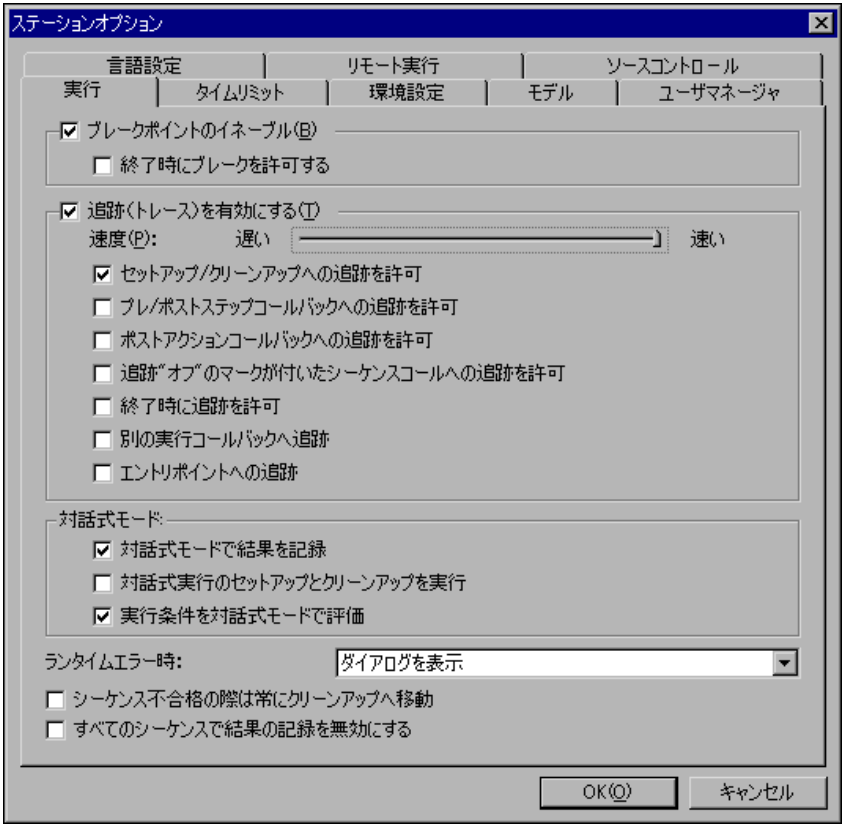

図 2-8　ステーションオプションダイアログボックスの実行タブ

3. **OK** をクリックしてダイアログボックスを閉じます。

シーケンスを直接実行する

シーケンスの実行を開始するには、シーケンスを直接実行するのが最も簡単です。

以下の手順に従って、`Sample1.seq` シーケンスファイルウィンドウの `MainSequence` を実行します。

1. まだ選択されていない場合は、シーケンスファイルウィンドウの「ビュー」リングで `MainSequence` を選択します。
2. **実行→ MainSequence を実行**を選択します。

   このように選択すると、シーケンスエディタは新しいウィンドウを開きます。このウィンドウを実行ウィンドウといいます。実行ウィンドウでは、実行中のステップ、変数やプロパティの値、および実行完了時のテストレポートを表示することができます。

図 2-9 は、実行ウィンドウの例を示します。

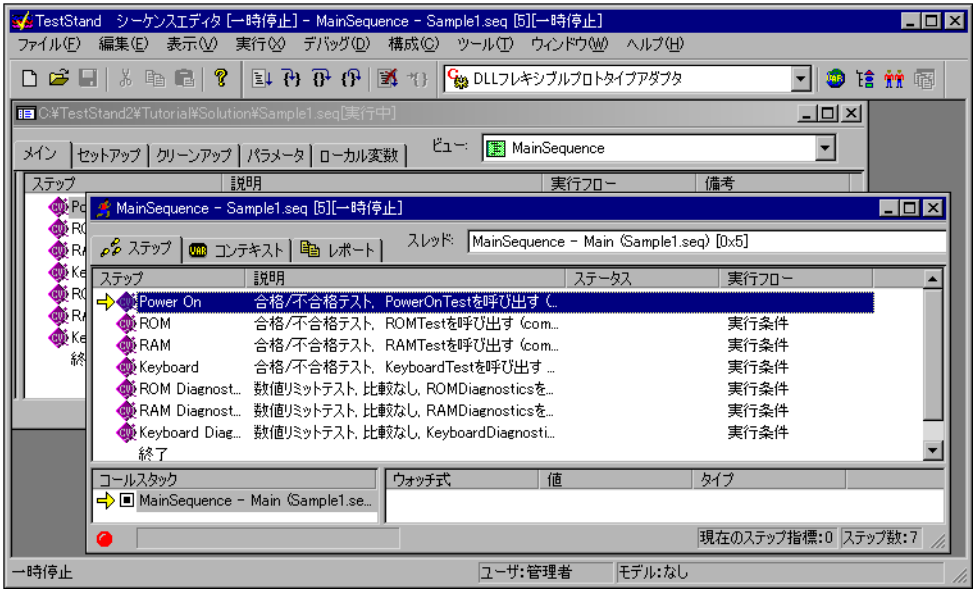

図 2-9　Sample1.seq の実行ウィンドウ

実行開始後、図 2-10 に示すように、実行ウィンドウの前面に Test Simulator ダイアログボックスが表示されます。

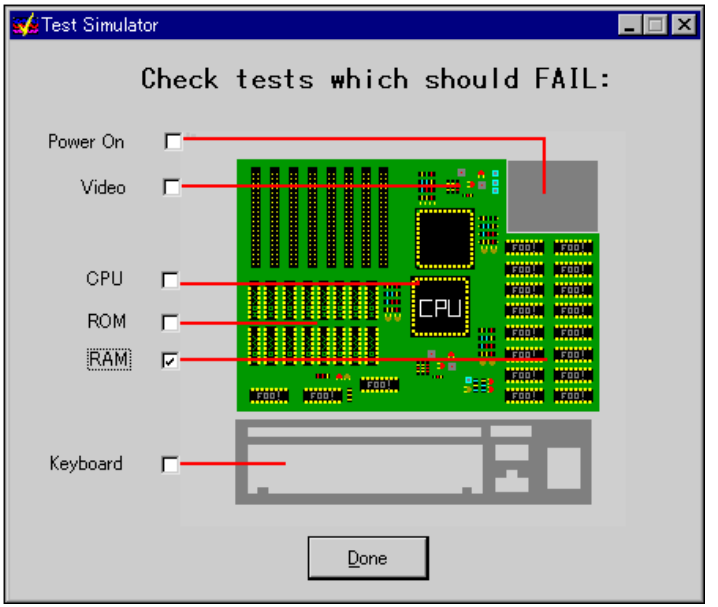

図 2-10　Test Simulator ダイアログボックス

実行するステップの 1 つによってこのダイアログボックスが表示されます。このダイアログボックスは、実行中に不合格にするコンピュータコンポーネントを入力するようプロンプトします。

3. RAM Test のチェックボックスをクリックして選択します。
4. **Done** ボタンをクリックします。実行ウィンドウが実行されたステップを追跡するのを確認してください。

   シーケンスエディタは、ステップタブで現在実行中のステップの左側に黄色のポインタアイコンを表示することによって、実行の進行状況を表示します。このポインタアイコンのことを実行ポインタといいます。RAM Test の状況コラムに不合格と表示されていることに注目してください。実行が完了すると、ウィンドウタイトルの状況セクションが (実行中) から (完了) に変わり、実行ウィンドウがグレーになります。

5. 実行が完了したら、**ファイル→閉じる**を選択するか、**ウィンドウ→実行終了ウィンドウを閉じる**を選択するか、あるいはウィンドウのタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。

シーケンスを再度実行するには、上記のステップ 1 から 5 をもう一度行います。ステップ 3 では、Test Simulator ダイアログボックスで Video と CPU 以外のいずれかのテストを選択してください。Video および CPU テストは、後の練習で行います。

メモ　高速のコンピュータを使用していて、TestStand の追跡機能の速度を遅くしたい場合は、ステーションオプションダイアログボックスの実行タブで速度スライダ制御器の値を変更します。

シーケンシャルプロセスモデルを使用してシーケンスを実行する

シーケンスの直接実行に加えて、プロセスモデルエントリポイントを使用してシーケンスを実行することもできます。第 1 章、「TestStand の概要」では、エントリポイントとはプロセスモデルシーケンスファイル内の単なるシーケンスであると説明しています。エントリポイントを実行すると、シーケンスファイルの `MainSequence` 呼び出しの前後に一連の操作が実行されます。プロセスモデルの共通の操作としては、UUT の識別、オペレータへの合否の通知、テストレポートの生成、および結果の記録などがあります。

次の手順に従い、`SequentialModel.seq` の一回 実行（`Single Pass`）実行エントリポイントを使用して、`Sample1.seq` シーケンスファイルの `MainSequence` を実行してください。

1. **構成→ステーションオプション**を選択して、`SequentialModel.seq` をデフォルトプロセスモデルとして設定します。モデルタブをクリックして、「ステーションモデル」リング制御器から `SequentialModel.seq` を選択します。
2. `Sample1.seq` シーケンスファイルがアクティブウィンドウになっていることを確認します。
3. **実行→一回実行**を選択します。
4. ここでもう一度、Test Simulator ダイアログボックスで Video と CPU 以外のいずれかのテストを選択します。Video および CPU テストは、後の練習で行います。
5. **Done** をクリックしてプロンプトを閉じます。

   メインシーケンスのステップが実行した後、一回実行エントリポイントによりテストレポートが生成されることにご注目ください。TestStand がレポートを生成する間、実行ウィンドウの下部に図 2-11 のようなステータス表示バーが表示されます。

図 2-11　レポート生成中のステータスバー

6. TestStand がレポートを生成した後、実行ウィンドウはレポートタブにレポートを表示します。テストレポートの内容を確認し、TestStand が実行した各ステップの結果が表示されていることを確かめてください。

   テストレポート機能については、本書の第 12 章、「レポートをカスタマイズする」でさらに詳しく説明します。

7. 実行が完了したら、**ファイル→閉じる**を選択するか、**ウィンドウ→実行終了ウィンドウを閉じる**を選択するか、あるいはウィンドウのタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。
8. **実行→ UUT をテスト**を選択します。

   メインシーケンスのステップを実行する前に、プロセスモデルシーケンスは UUT 情報ダイアログボックスを表示して、シリアル番号の入力を求めます。

9. 任意の番号を入力して **OK** ボタンをクリックします。
10. Test Simulator ダイアログボックスで不合格にするテストを選択します。
11. **OK** をクリックします。シーケンス実行中の実行ウィンドウの様子に注目してください。

    メインシーケンスのステップの完了後、プロセスモデルは UUT の結果を示すウィンドウを表示します。

12. **OK** ボタンをクリックして UUT の結果ウィンドウを閉じます。ここでプロセスモデルはレポートを生成しますが、実行を完了してレポートを表示する代わりに、UUT 情報ダイアログボックスを再度表示します。
13. シリアル番号を変えて、ステップ 9 から 12 を何度か繰り返してみてください。
14. **停止**ボタンをクリックして、ループを停止し実行を完了します。

    実行が完了すると、TestStand はすべての UUT のテストレポートを表示します。

15. テストレポートの内容を確認して、各 UUT の結果が記録されていることを確かめてください。
16. 実行が完了したら、**ファイル→閉じる**を選択するか、**ウィンドウ→実行終了ウィンドウを閉じる**を選択するか、あるいはウィンドウのタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。

バッチプロセスモデルを使用してシーケンスを実行する

バッチプロセスモデルは、複数の UUT に対して同時に同じテストシーケンスを実行します。バッチプロセスモデルを使用すると、すべての UUT のテストがグループとして同時に開始および終了します。バッチモデルにはまた、バッチ同期機能があります。たとえば、ステップが各 UUT に対

して一回ではなく各バッチに対して一回実行するように指定することができます。また、バッチモデルを使用すると、特定のステップやステップのグループが一度に複数の UUT に対して実行できないように指定したり、特定のステップがすべての UUT に対して同時に実行するように指定することもできます。さらに、バッチモデルでは、バッチ内の UUT のテスト結果を要約したバッチレポートを生成することもできます。

次の手順に従い、`BatchModel.seq` の一回 実行実行エントリポイントを使用して `BatchUUT.seq` シーケンスファイルを実行してください。

1. シーケンスファイル `<TestStand>\Tutorial\BatchUUT.seq` を開きます。

メモ　この練習では、デフォルトプロセスモデルを変更する必要はありません。このシーケンスファイルは、シーケンスエディタのデフォルトプロセスモデルに関わらず、常にバッチプロセスモデルを使用するよう構成されています。シーケンスファイルで常に特定のプロセスモデルを使用するよう指定するには、「シーケンスファイルのプロパティ」ダイアログボックスを使用します。

2. **構成→モデルオプション**を選択します。
3. 図 2-12 に合うように設定を変更して、**OK** ボタンをクリックします。
   テストソケットの数とは、バッチ内のテストする UUT の数です。少ないテストソケット数から始めた方が、実行の状況を解釈しやすいでしょう。

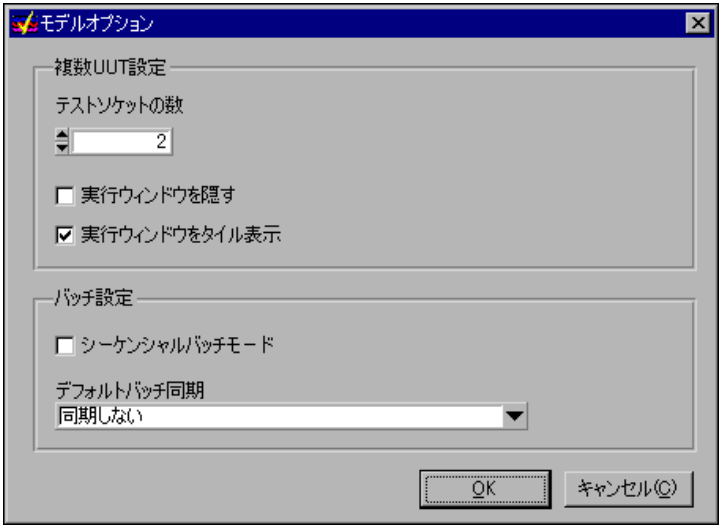

図 2-12　モデルオプションダイアログボックス

4. **実行→一回実行**を選択します。

   実行中、TestStand は各実行の追跡を示す実行ウィンドウを各テストソケットに対して表示します。このサンプルには、いくつかの同期セクションがあります。最初のセクションは、室内の温度変化をシミュレーションします。テストを実行するステップは、UUT につき一回でなく、各バッチにつき一回のみ実行されます。2 つめのセクションは、共通のリソースであるパルス発生器の共有をシミュレーションします。このセクションでは一度に一つの UUT のみステップを実行します。3 つめのセクションは周波数掃引をシミュレーションし、すべての UUT がパラレルで実行します。最後に、メッセージポップアップに周波数掃引の実行にかかる時間が表示されます。時間を報告するのは 1 つの UUT のみです。

   TestStand は、バッチおよび各 UUT のレポートを生成します。バッチレポートには、バッチ内のすべての UUT の結果の要約が表示されます。レポートが HTML 形式の場合、バッチレポートには各 UUT レポートへのハイパーリンクが含まれています。バッチレポート内のハイパーリンクをクリックして、特定の UUT の結果を表示することができます。

5. 実行が完了したら、**ファイル→閉じる**を選択するか、**ウィンドウ→実行終了ウィンドウを閉じる**を選択するか、あるいはウィンドウのタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。

6. **実行→ UUT をテスト**を選択します。

   メインシーケンスのステップを実行する前に、プロセスモデルシーケンスは UUT 情報ダイアログボックスを表示して、バッチのシリアル番号および各テストソケットの UUT シリアル番号の入力を求めます。特定のテストソケットを無効にすることもできます。

7. 任意のバッチシリアル番号および UUT シリアル番号を入力して、**Go** ボタンをクリックします。

8. **バッチレポート表示**ボタンをクリックします。

   メインシーケンスのステップの完了後、プロセスモデルは UUT の結果を示すウィンドウを表示します。このウィンドウでは、バッチレポートと個々の UUT レポートを表示することができます。

   ファイル全体と現在のバッチのみのオプションがあることにご注意ください。ファイル全体はテストしたすべてのバッチを含み、現在のバッチのみには最後にテストしたバッチのみが含まれます。

9. **現在のバッチのみ**を選択します。

   TestStand は、外部ビューワを起動してレポートを表示します。デフォルトで、HTML レポートは Microsoft Internet Explorer で表示され、テキストレポートは Microsoft Notepad で表示されます。外部ビューワとしてそれ以外のアプリケーションを使用するように構成することもできます。

10. 外部ビューワを閉じて結果ウィンドウに戻ります。
11. **次のバッチ**ボタンをクリックします。
12. 別のバッチで手順 7 から 11 を何度か繰り返してみてください。
13. **Stop** ボタンをクリックして実行を完了します。

    実行が完了したら、TestStand はすべてのバッチと UUT のテストレポートを内部ビューワに表示します。
14. レポートの内容を確認して、各バッチと UUT の結果が記録されていることを確かめてください。
15. 実行が完了したら、**ファイル→閉じる**を選択するか、**ウィンドウ→実行終了ウィンドウを閉じる**を選択するか、あるいはウィンドウのタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。

このチュートリアルセッションはこれで終わりです。次のセッションでは、シーケンスにステップを追加してステップのプロパティを編集する方法を説明します。

3

シーケンスのステップを編集する

本章では、シーケンスにステップを追加して、そのステップのプロパティを構成します。また、他のシーケンスを呼び出すサブシーケンスを追加します。

サンプルをセットアップする

第 2 章、「Loading and Running Sequences」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択し、`Teststand\Tutorial` ディレクトリから `Sample1.seq` ファイルを開きます。
3. 「ビュー」リングで `MainSequence` を選択して、シーケンスファイルウィンドウに `MainSequence` を表示します。

新規ステップを追加する

TestStand には、あらかじめ定義されたステップタイプがいくつか含まれています。ステップタイプは、そのタイプの各ステップの標準プロパティと動作のリストを定義します。ステップタイプは、モジュールアダプタを使用してコードモジュールを呼び出すこともあれば、モジュールアダプタをまったく使用しないこともあります。

TestStand で利用可能な定義済みのステップタイプには以下のようなものがあります。

- アクション
- 数値リミットテスト
- 複数値リミットテスト
- 文字列値テスト
- 合否テスト
- ラベル
- ジャンプ（Goto）
- ステートメント

- プロパティローダ
- メッセージポップアップ
- 実行可能ファイルのコール
- コールシーケンス
- 同期（ロック、セマフォ、ランデブー、キュー、ノーティフィケーション、ウェイト、スレッド優先順位、バッチ同期）
- データベース（Open Database、Open SQL Statement、Close Database、Close SQL Statement、Data Operation）
- IVI（Power Supply、DMM、Scope、Fgen、Tools）

上記の各ステップタイプの説明については、『TestStand User Manual』の Chapter 10、「Built-In Step Types」を参照してください。

この練習では、シーケンスにステップを追加して、DLL コードモジュールの関数を呼び出す要素のステップを構成します。次の手順に従って、シーケンスに合否テストを挿入します。

1. コードモジュールを呼び出すステップを挿入するには、その前にステップが使用するモジュールアダプタを指定する必要があります。モジュールアダプタを指定するには、図 3-1 に示すように、ツールバーのアダプタ選択リングを使用するか、**構成→アダプタ**を選択します。選択したアダプタは、モジュールアダプタを使用するステップタイプにのみ適用されます。

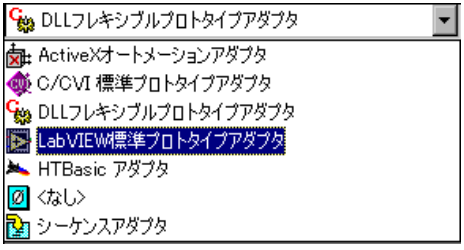

図 3-1　モジュールアダプタを選択する

シーケンスにステップを挿入すると、TestStand はシーケンスエディタのツールバーのリングで現在選択されているアダプタにそのステップを関連付けます。選択アダプタに <None> を選択してステップを挿入すると、挿入したステップはコードモジュールを呼び出しません。ステップのアイコンとして、アダプタのアイコンが表示されます。

アダプタ選択リングで C/CVI 標準プロトタイプアダプタを選択します。C/CVI 標準プロトタイプアダプタを使用すると、TestStand の標準 C パラメータリストを持つ C 関数を呼び出すことができます。呼び出す関数の形式は、ダイナミックリンクライブラリ（`.dll`）、

ソースファイル（.c）、オブジェクトファイル（.obj）、またはスタティックライブラリファイル（.lib）です。

2. シーケンスファイルウィンドウで RAM テストを右クリックして、図 3-2 に示すように、表示されたメニューから**ステップを挿入→テスト→合格／不合格テスト**を選択します。このようなメニューのことをコンテキストメニューといいます。

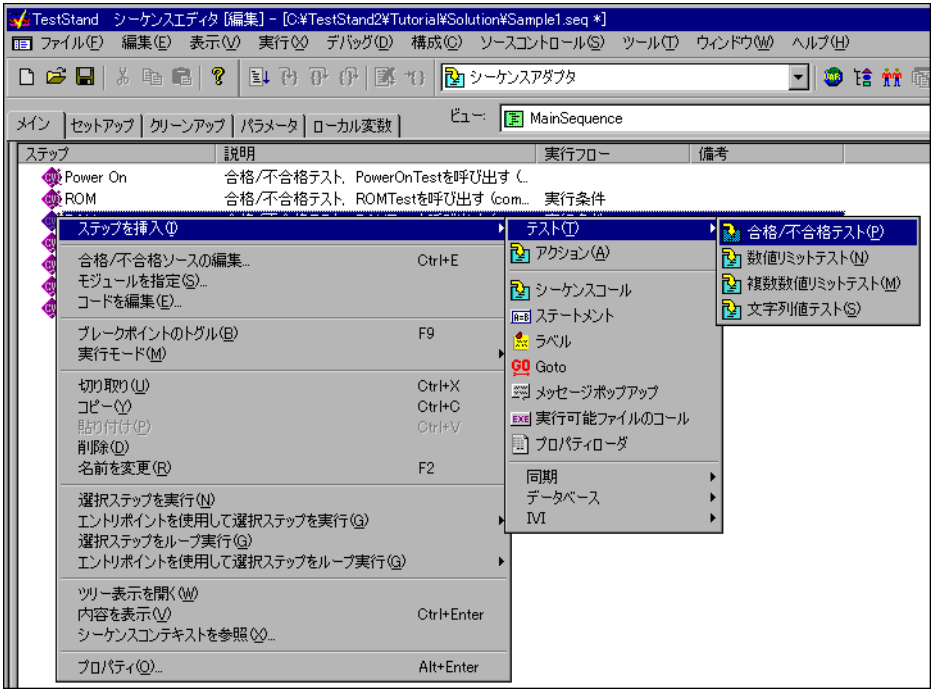

図 3-2　新規ステップを挿入する

すると、RAM ステップの後に新規の合格／不合格テストが挿入されます。

通常、合格／不合格テストステップは、独自の合格／不合格の決定を行うコードモジュールを呼び出すときに使用します。コードモジュールの実行後、合格／不合格テストステップはブール表現を確認してそのステップが合格したか不合格になったかを判定します。

3. デフォルトで、新規テストには合格／不合格テストという名前が付いています。ステップを挿入すると、そのステップの名前が選択されています。

4. `Video Test` と入力して <Enter> を押し、新規ステップの名前を変更します。後になってステップ名を変更する場合は、ステップ名を右ク

リックしてコンテキストメニューから**名前を変更**コマンドを選択します。

テストモジュールを指定する

シーケンスに新規ステップを追加したら、ステップが実行するテストモジュールを指定する必要があります。

1. 新規に作成した Video Test ステップを右クリックして、コンテキストメニューから**モジュールを指定**コマンドを選択します。

   選択すると、そのステップのコードモジュールと、コードモジュールを呼び出す際に渡すパラメータを指定することのできる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実際のタイトルは、ステップに関連付けられたモジュールアダプタによって異なります。

   ダイアログボックスに必要事項を入力すると、TestStand はその情報をステップのプロパティとして保存します。C/CVI 標準プロトタイプアダプタの場合、図 3-3 に示すように、C/CVI モジュールコールを編集ダイアログボックスが表示されます。
2. 「モジュールタイプ」リング制御器で Dynamic Link Library（`*.dll`）を選択します。これにより、テストのコードモジュールが DLL 内の関数を呼び出すように指定されます。
3. ステップが呼び出す DLL を選択するには、**参照**ボタンをクリックします。
4. `TestStand\Tutorial` ディレクトリから `computer.dll` ファイルを選択します。DLL を選択すると、TestStand は DLL のタイプライブラリの読み取りを試み、エクスポートされたすべての関数を関数名リング制御器にリスト表示します。
5. 制御器の右側の矢印をクリックして、関数名リング制御器から `VideoTest` 関数を選択します。`VideoTest` 関数を表示するには、スクロールバーの矢印をクリックします。この関数は、テストが合格か不合格かを示すブール値を返す簡単なルーチンです。図 3-3 は、完成したダイアログボックスを示します。

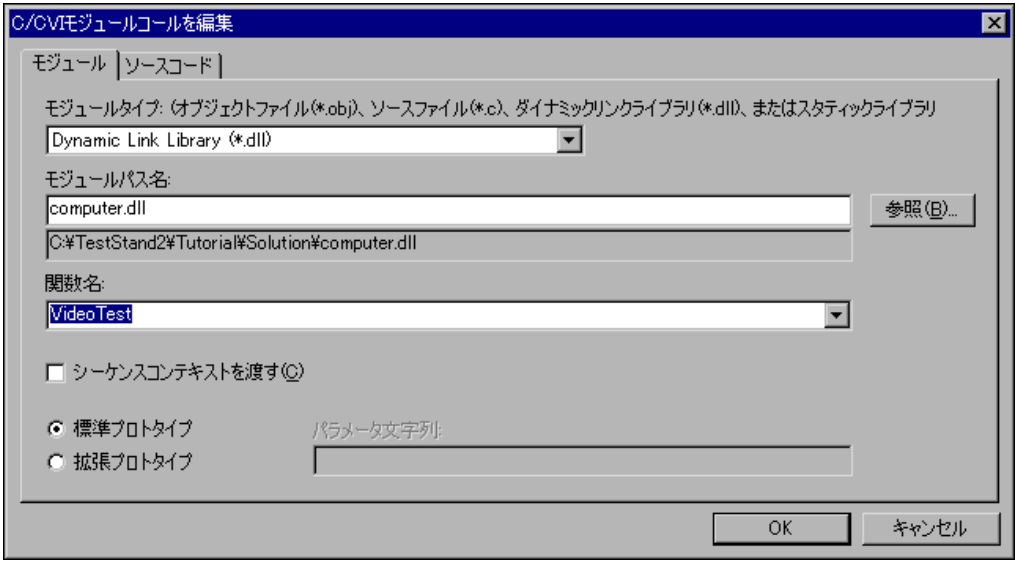

図 3-3　モジュールを指定ダイアログボックス

6. **OK** ボタンをクリックして C/CVI モジュールコールを編集ダイアログボックスを閉じ、シーケンスファイルウィンドウに戻ります。

ステップのプロパティを変更する

シーケンスの各ステップにはプロパティがあります。ステップのタイプにより、ステップのプロパティの内容が決まります。すべてのステップには、以下の事項を決定する共通のプロパティがあります。

- ステップをいつロードするか
- ステップをいつ実行するか
- テストの合否を判定するために使用される情報
- ステップをループで実行するか
- ステップの完了時にどのような条件付きアクションが発生するか

この練習では、これらの共通プロパティとその値の設定方法について学習します。

1. Video Test ステップを右クリックして、コンテキストメニューから**プロパティ**コマンドを選択します。

   すると、図 3-4 に示すように、ステップのプロパティダイアログボックスが表示されます。

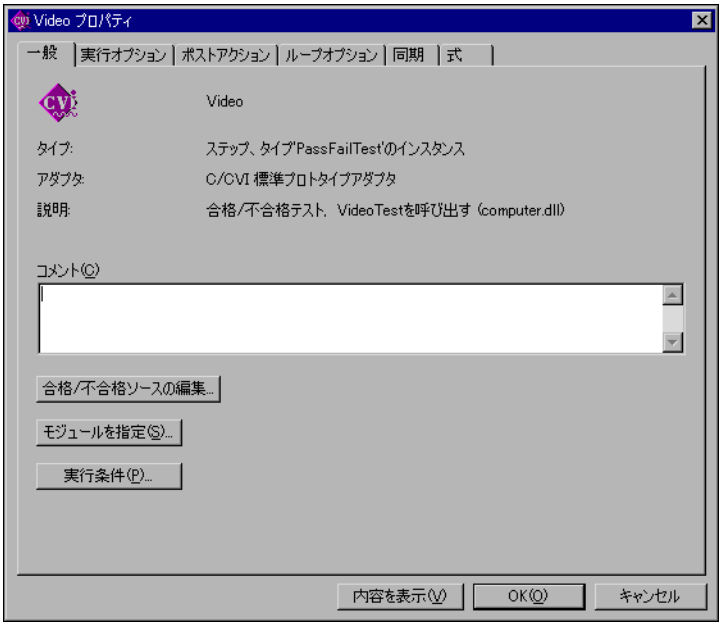

図 3-4　ステップのプロパティダイアログボックス

2. ダイアログボックスの**実行条件**ボタンをクリックすると、図 3-5 に示すような実行条件ダイアログボックスが表示されます。

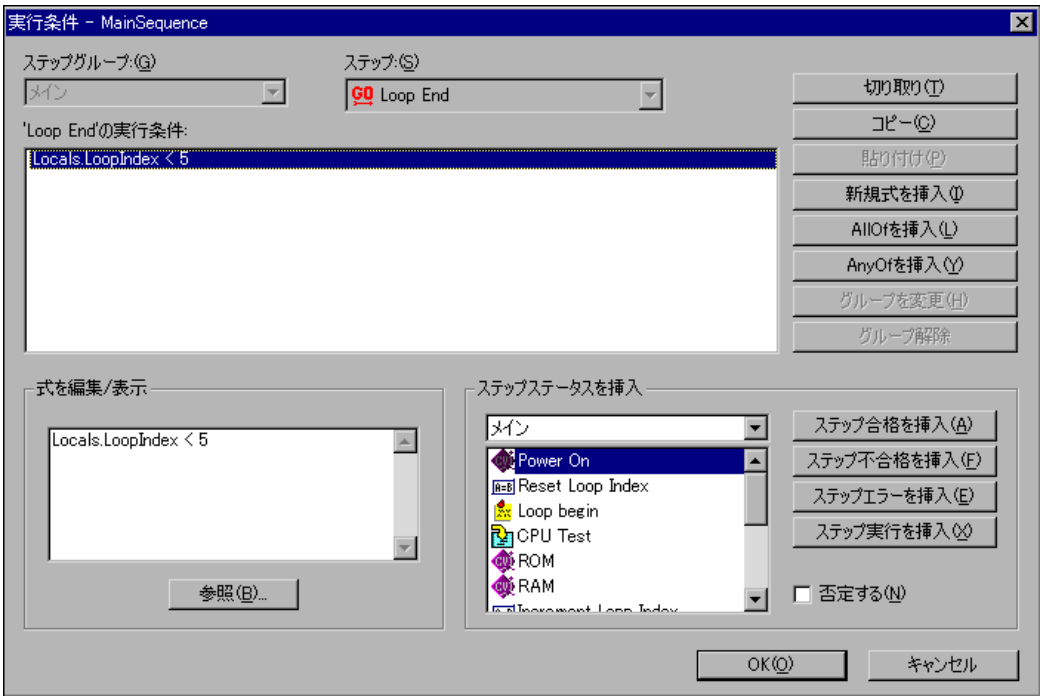

図 3-5　実行条件ダイアログボックス

実行条件は、TestStand がシーケンスの通常の実行フローの中でステップを実行するために true として評価されなければならない条件を指定します。たとえば、前のステップが合格したときのみステップを実行したい場合などです。

3. Video Test では、以下の手順に従って、Power On ステップが合格した場合のみステップが実行するように実行条件を定義してください。
   a. ダイアログボックスのステップステータスを表示セクションで、メインステップグループのステップ名リストから Power On ステップをクリックします。
   b. 実行条件リストに条件を追加するには、**ステップ合格を挿入**ボタンをクリックします。'Video Test' の実行条件テキストボックスには、合格 `Power On` という文字列が追加されました。これは、そのステップが Power On ステップ合格時にのみ実行することを示します。
   c. **OK** ボタンをクリックして実行条件ダイアログボックスを閉じ、ステップのプロパティダイアログボックスに戻ります。

4. 実行オプションタブをクリックして、図 3-6 に示すページを表示します。このタブには、TestStand がステップを実行する方法に影響するさまざまな設定が含まれています。

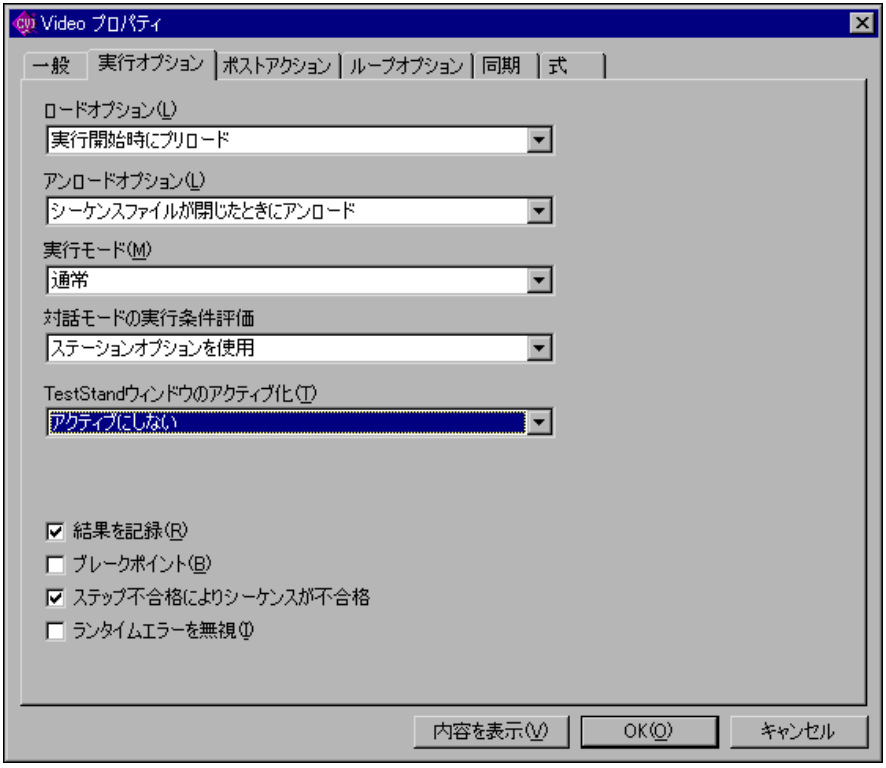

図 3-6　実行オプションタブ

ステップのプロパティダイアログボックスの実行オプションタブには、次の制御器が含まれています。

- **ロードオプション**—ステップの以下のロードオプション設定のいずれかに指定します。
  - **シーケンスファイルを開くときにプリロード**—そのステップを含むシーケンスを TestStand がメモリにロードするときに、ステップモジュールをロードします。
  - **実行開始時にプリロード**—そのステップを含むシーケンスファイル内のいずれかのシーケンスが実行を開始したときに、ステップモジュールをロードします。これがデフォルト設定です。
  - **ダイナミックにロード**—ステップが呼び出せるようになるまで、ステップモジュールをロードしません。

- **アンロードオプション**—ステップの以下のアンロードオプション設定のいずれかに指定します。
  - **実行条件が一致しないときにアンロード**—ステップの実行条件が `False` になったときにステップモジュールをアンロードします。
  - **ステップの実行後アンロード**—ステップの実行終了後にステップモジュールをアンロードします。
  - **シーケンスの実行後アンロード**—そのステップを含むシーケンスの実行終了後にステップモジュールをアンロードします。
  - **シーケンスファイルが閉じたときにアンロード**—そのステップを含むシーケンスファイルを TestStand がメモリからアンロードしたときに、ステップモジュールをアンロードします。これがデフォルト設定です。

メモ　シーケンスの一般プロパティにある「このシーケンスへの非再入実行を最適化する」がチェックされている場合、シーケンスファイルやシーケンスのステップのアンロードオプションが何に設定されているかにかかわらず、TestStand は実行が終了するまでシーケンスのコードモジュールをアンロードしません。

- **実行モード**—ステップの以下の実行モード値を設定します。
  - 強制的に合格
  - 強制的に不合格
  - スキップ
  - 通常
- **対話モードの実行条件評価**—ステップを対話式に実行したときに、TestStand がステップの実行条件を評価するかどうかを指定します。このリング制御器には以下のオプションが含まれています。
  - ステーションオプションを使用
  - 実行条件を評価
  - 実行条件を評価しない
- **TestStand ウィンドウのアクティブ化**—ステップ完了時に TestStand アプリケーションがウィンドウをアクティブにするかどうかを指定します。このリング制御器には以下のオプションが含まれています。
  - アクティブにしない
  - ステップが完了したときアクティブ
  - 始めアクティブの場合、ステップが完了したとき再びアクティブ

- **結果を記録**—ステップの Result プロパティの内容がシーケンスの結果リストに追加されるかどうかを指定します。結果の収集に関する詳細は、『TestStand User Manual』の Chapter 6、「Sequence Execution」の「Result Collection」のセクションを参照してください。
- **ブレークポイント**—実行前にこのステップでブレークするように指定します。また、コンテキストメニューから**ブレークポイントのトグル**を選択するか、シーケンスエディタのステップアイコンの左側をクリックして、ステップをブレークポイント状態に設定することもできます。
- **ステップ不合格によりシーケンスが不合格**—TestStand は各実行シーケンスの内部ステータス値を保持します。TestStand がステップの状況プロパティを不合格に設定し、そのステップに対して「ステップ不合格によりシーケンスが不合格」オプションをチェックしている場合、内部シーケンスステータス値は不合格になります。シーケンスが戻ったときにシーケンスの内部ステータスが不合格の場合、呼び出しているステップの状況は不合格に設定されます。アダプタがシーケンスアダプタの場合、これによって、シーケンスコールステップタイプまたはアクションステップタイプを使用するステップに影響します。シーケンスアダプタで、合格／不合格テスト、数値リミットテスト、および文字列値テストステップタイプを使用するステップは、ステップのステータスを上書きします。
- **ランタイムエラーを無視**—ステップがランタイムエラーをシーケンスに報告しないように設定します。ステップでランタイムエラーが発生すると、ステップは実行を停止し、TestStand はそのステップのステータスをエラーに設定します。このオプションを無効にすると、TestStand はシーケンスの内部ステータスもエラーに設定し、実行はそのシーケンスのクリーンアップステップグループに分岐します。このオプションをチェックすると、TestStand はシーケンスの内部ステータスをエラーに設定しません。その代わりに、TestStand はステップの `Error.Occurred` プロパティを `False` に設定し、次のステップでは実行は通常どおりに行われます。そのステップの `Result.Status` プロパティの値はエラーに設定されたままになります。

5. 図 3-7 に示すポストアクションタブをクリックします。ポストアクションタブでは、ステップの実行後に発生する動作を指定します。

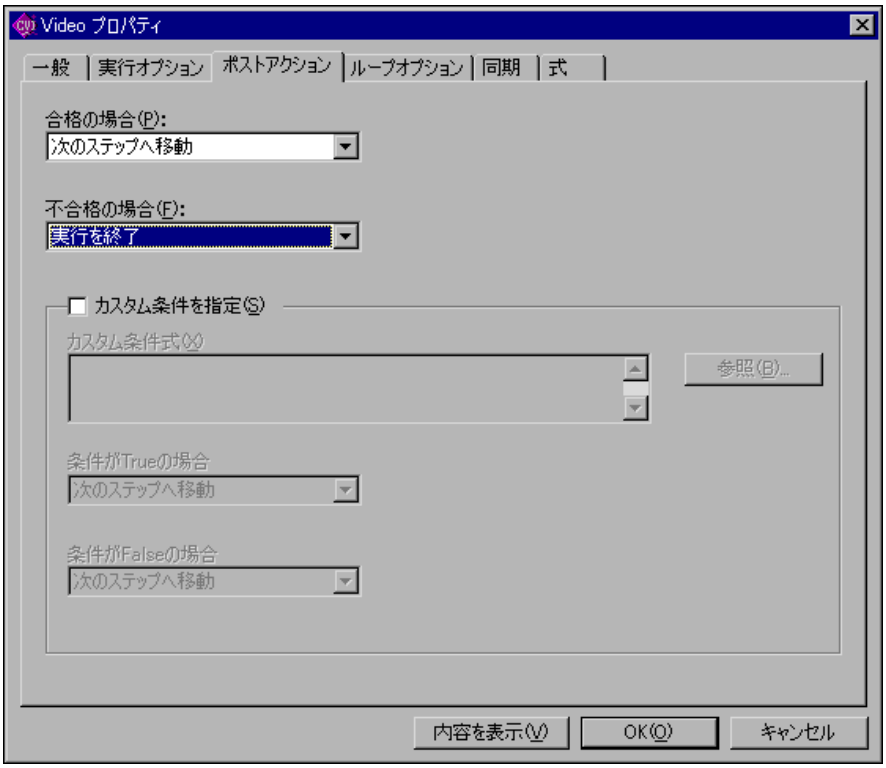

図 3-7　ポストアクションタブ

その動作は、ステップの合格／不合格のステータス、またはカスタム条件の式により条件付けることができます。たとえば、ステップが不合格になった場合に特定のステップに進むかコールバックシーケンスを呼び出すようにしたいことがあります。デフォルトで、合格の場合と不合格の場合の動作は「次のステップへ移動」になっています。

6. 不合格の場合のポストアクションを「実行を終了」に設定します。これにより、ステップが不合格になるとシーケンスの実行は終了します。

7. 図 3-8 に示すループオプションタブをクリックします。

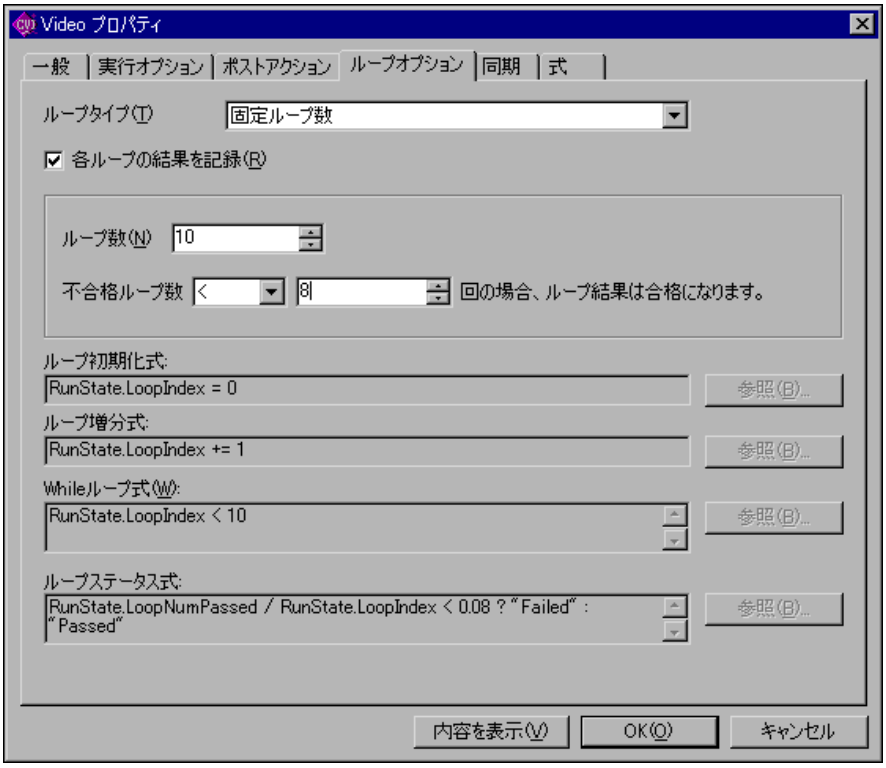

図 3-8　ループオプションタブ

ループオプションタブを使用して、ステップの実行時ループで反復的に実行するように個々のステップを構成することができます。「ループタイプ」リング制御器を使用して、そのステップのループのタイプを指定します。選択できる項目は以下のとおりです。

- **なし**—TestStand はそのステップでループを使用しません。これがデフォルト設定です。
- **固定ループ数**—TestStand はステップを特定の回数ループで実行して、ステップのステータスが合格になった反復の数の割合により、ステップの最終的な合格または不合格が決まります。
- **合格／不合格カウント**—ステップが特定の回数合格または不合格になるまで、あるいは最大ループ反復数が終了するまで、ステップをループで実行します。TestStand は、特定数の合格または不合格が発生したか、あるいはループ反復数が最大に達したかによって、ステップの最終的なステータスを決定します。

- **カスタム**—これに設定すると、ステップのループ動作をカスタマイズできます。ループ初期化式、ループ増分式、While ループ式、および最終的なループステータス式を入力します。

8. ループオプションタブで、次の制御器値を変更してください。

ループタイプ	固定ループ数
ループ数	10
合格ループ数	<80% の場合、ループ結果は不合格

上記の設定で、TestStand は Video Test ステップを 10 回実行し、合格が 10 回のうち 8 回より少ない場合ステップ全体のステータスは不合格になります。図 3-8 は、完成したダイアログボックスを示します。

9. 図 3-9 に示す同期タブをクリックします。

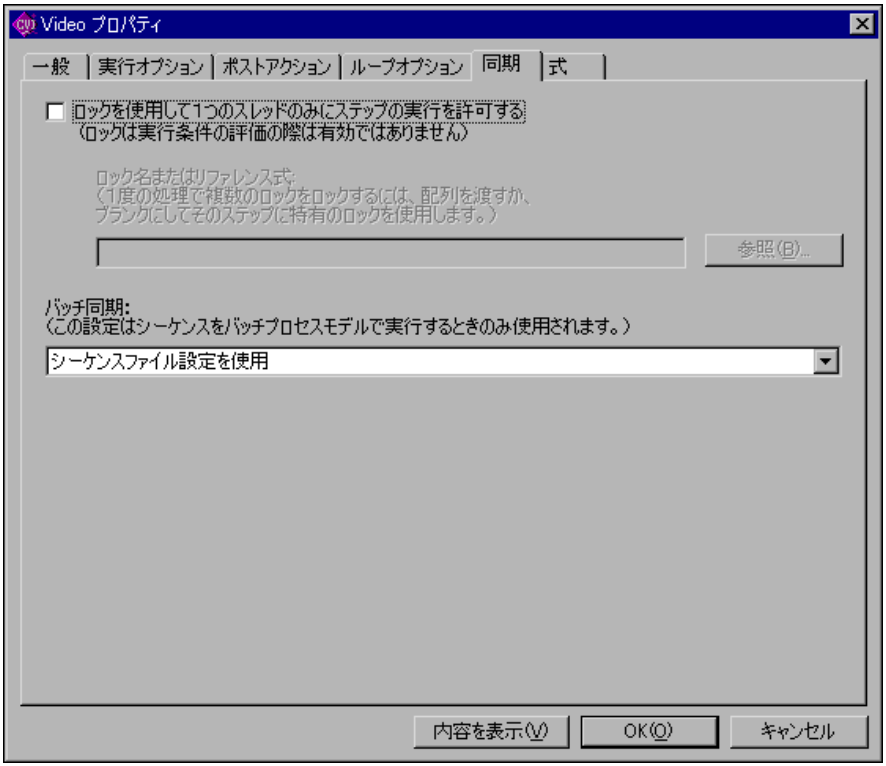

図 3-9　同期タブ

ステップのプロパティダイアログボックスの同期タブでは、ステップの実行の際に TestStand が実行する同期動作を指定します。ロックによってステップの実行を保護したり、バッチ実行でステップが他の

ステップと同期するように指定することができます。同期ステップタイプに関する詳細は、『TestStand User Manual』の Chapter 11、「Synchronization Step Types」の Batch Synchronization のセクションを参照してください。

- **ロックを使用して 1 つのスレッドのみにステップの実行を許可する**—ステップが実行前にロックを取得し実行完了後にロックを解放するよう指定します。
- **ロック名またはリファレンス式**—ステップが取得し解放するロックを指定します。文字列式を入力して既存のロックの名前を指定します。また、既存のロックオブジェクトへの ActiveX リファレンスに対して評価する式を入力することもできます。そのステップに対して固有のロックを使用する場合は、制御器を空白のままにしておいてください。
- **バッチ同期**—ステップが実行前に開始して実行完了後に終了するバッチ同期操作を指定します。

10. 図 3-10 に示す式タブをクリックします。

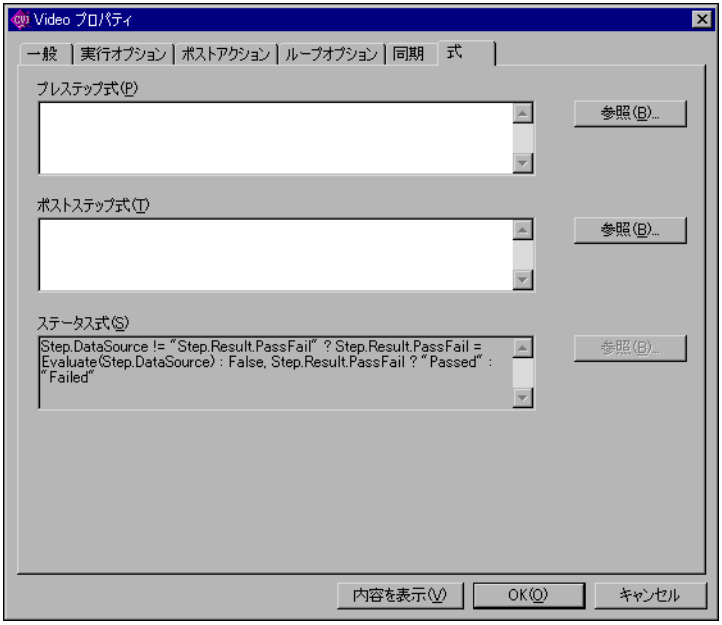

図 3-10　式タブ

式タブを使用して、TestStand がステップを呼び出す前と後に評価する式を入力します。TestStand の式については、本書の第 5 章、「変数およびプロパティを使用する」で説明しています。

11. **OK** をクリックしてステッププロパティダイアログボックスを閉じます。

    シーケンスファイルウィンドウで、Video Test の実行フローの列にポストアクションとループオプションが含まれていることに注意してください。

12. **ファイル→別名で保存**を選択して、シーケンスを `Sample2.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。

13. **実行→一回実行**を選択してシーケンスを実行します。Video Test が不合格になるように選択すると、シーケンスは Video Test をループで 10 回呼び出した後、直ちに実行を停止することに注意してください。

    TestStand がレポートを生成した後、video ステップが実行すると、各ループ反復の結果がレポートに記録されることを確認してください。

14. **ファイル→閉じる**を選択するか、ウィンドウタイトルバーの×アイコンをクリックして、実行ウィンドウを閉じます。

シーケンスからサブシーケンスを呼び出す

TestStand では、呼び出し側のシーケンスのシーケンスコールステップを使用して他のシーケンスを呼び出すことができます。呼び出されたシーケンスは、呼び出し側のシーケンスファイルまたは別のシーケンスファイル内にあります。この練習では、現在のシーケンスにシーケンスコールステップを追加します。

1. Power On ステップを右クリックして、コンテキストメニューから**ステップを挿入→シーケンスコール**を選択します。すると、Power On ステップの後にシーケンスコールステップが挿入されます。
2. ステップの名前を `CPU Test` に変更します。
3. 次の手順で、ステップが呼び出すシーケンスを指定します。
   a. CPU Test ステップを右クリックします。
   b. コンテキストメニューから**モジュールを指定**コマンドを選択します。すると「シーケンスコールを編集」ダイアログボックスが表示されます。
   c. ファイルパス名の右の**参照**ボタンをクリックします。

d. `TestStand\Tutorial` ディレクトリから `SubSequence1.seq` ファイルを選択します。図 3-11 は、完成したダイアログボックスを示します。

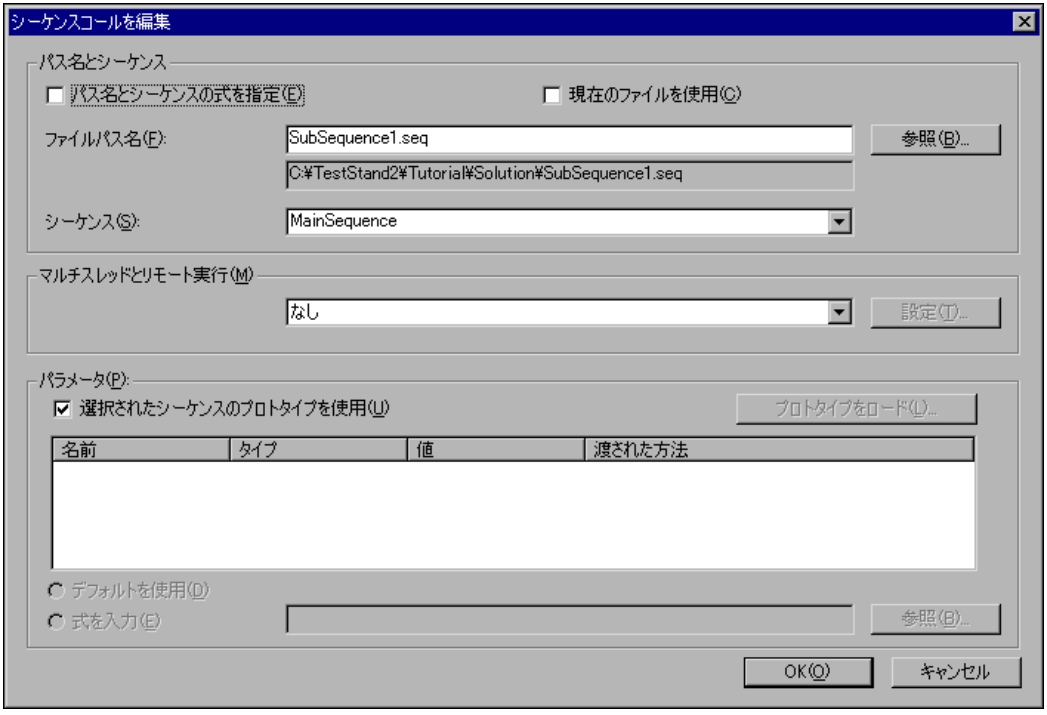

図 3-11　「シーケンスコールを編集」ダイアログボックス

e. **OK** ボタンをクリックしてダイアログボックスを閉じます。

4. **ファイル→保存**を選択して、シーケンスファイルへの変更を保存します。
5. CPU Test ステップをもう一度右クリックして、コンテキストメニューから**シーケンスを開く**コマンドを選択します。すると、シーケンスエディタは `SubSequence1.seq` シーケンスファイルを開いて `MainSequence` を表示します。

   他のすべてのシーケンスと同様、シーケンスファイル内のどのシーケンスでも実行できます。
6. **実行→ MainSequence を実行**を選択します。このシーケンスの実行の様子を確認してください。
7. 実行ウィンドウを閉じます。
8. `SubSequence1.seq` シーケンスファイルウィンドウを閉じます。
9. **実行→一回実行**を選択します。

10. 不合格にするテストを選択します。
11. **OK** をクリックします。

    シーケンスの実行後、テストレポートの内容を確認して、親シーケンスのステップとともに、サブシーケンスのステップの結果も記録されていることに注目してください。
12. 実行ウィンドウを閉じます。

シーケンスコールステップには、サブシーケンスを呼び出す別の方法があります。「シーケンスコールを編集」ダイアログボックスのマルチスレッドとリモート実行セクションで、呼び出し側のシーケンスのスレッドに並行してサブシーケンスをスレッドで呼び出すことができます。リング制御器を使用して、呼び出すシーケンスが現在の実行の中の新しいスレッドで実行するか、あるいは新しい実行として実行するかを指定することができます。さらに、リング制御器を使用すると、リモートホストでサーバとして動作する TestStand エンジンでサブシーケンスを呼び出すこともできます。設定ボタンをクリックすると、リング制御器で選択した項目に対して固有のダイアログボックスが表示されます。シーケンスコールの環境をさらに詳しく構成するには、このダイアログボックスを使用します。シーケンスの呼び出しに関する詳細は、『TestStand User Manual』の Chapter 13、「Module Adapters」を参照してください。

このチュートリアルセッションはこれで終わりです。次のセッションでは、TestStand の実行ツールおよびデバッグツールの使用方法について学習します。

4

シーケンスをデバッグする

本章では、TestStand のシーケンスデバッグ機能を使用して、実行中にシーケンスをシングルステップ処理します。デバッグするのは、第 6 章、「テストの作成とデバッグ」のテストのソースコードです。

サンプルをセットアップする

第 3 章、「シーケンスのステップを編集する」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択して、第 3 章、「シーケンスのステップを編集する」で作成した `<TestStand>\Tutorial\ Sample2.seq` ファイルを開きます。このファイルは、`<TestStand>\Tutorial\Solution` ディレクトリにもあります。
3. 「ビュー」リングで `MainSequence` を選択して、シーケンスファイルウィンドウに `MainSequence` を表示します。

ステップモード実行

TestStand でステップモード実行を行うには、次の手順に従ってください。

1. **実行→最初のステップでブレーク**を選択します。このコマンドは、TestStand が実行する最初のステップで実行を中断するために使用します。このコマンドを有効にすると、メニューでコマンド名の隣にチェックマークが付きます。
2. シーケンスウィンドウのクリーンアップタブをクリックします。

   シーケンスが正常に実行を完了したかランタイムエラーが発生したかにかかわらず、クリーンアップステップグループはメインステップグループの後に実行することを思い出してください。セットアップまたはメインステップによってランタイムエラーが発生した場合、実行フローは停止しクリーンアップステップグループにジャンプします。

3. 次の手順に従い、クリーンアップステップグループにメッセージポップアップを表示するステップを追加して、その動作を確認します。
   a. クリーンアップタブの空白のステップリスト上で右クリックして、コンテキストメニューから**ステップを挿入→メッセージポップアップ**を選択します。
   b. 新規のステップに `Cleanup Message` という名前を付けます。
   c. Cleanup Message ステップを右クリックして、コンテキストメニューから**メッセージの設定を編集**コマンドを選択します。すると、シーケンスエディタは「メッセージボックスステップを構成」ダイアログボックスを表示します。
   d. タイトル式文字列制御器に `"Cleanup Message"` というテキストを入力します。文字列は必ず二重引用符（"）で囲むようにしてください。これは、その式が実際に表示される式であることを示します。
   e. メッセージ式文字列制御器に、`"I am now in the Cleanup Step Group"`（二重引用符を含む）というテキストを入力します。
   f. タイムアウトボタン制御器からボタン `1` を選択します。これにより、待機時間制御器が入力可能になります。タイムアウトボタン選択リングでは、タイムアウト時間が経過した後にどのメッセージボックスボタンが自動的にアクティブになるかを指定します。
   g. ウェイト時間制御器に、`10` を入力します。シーケンスが実行されるとき、メッセージポップアップステップは、ユーザが 10 秒以内に入力を行わなかった場合に自動的に消える通知メッセージを表示します。これは、テスト中オペレータが常駐しない場合に、重要度の低いメッセージが表示されたときに便利です。

   図 4-1 は、すべて入力したダイアログボックスのテキストとボタンタブを示します。

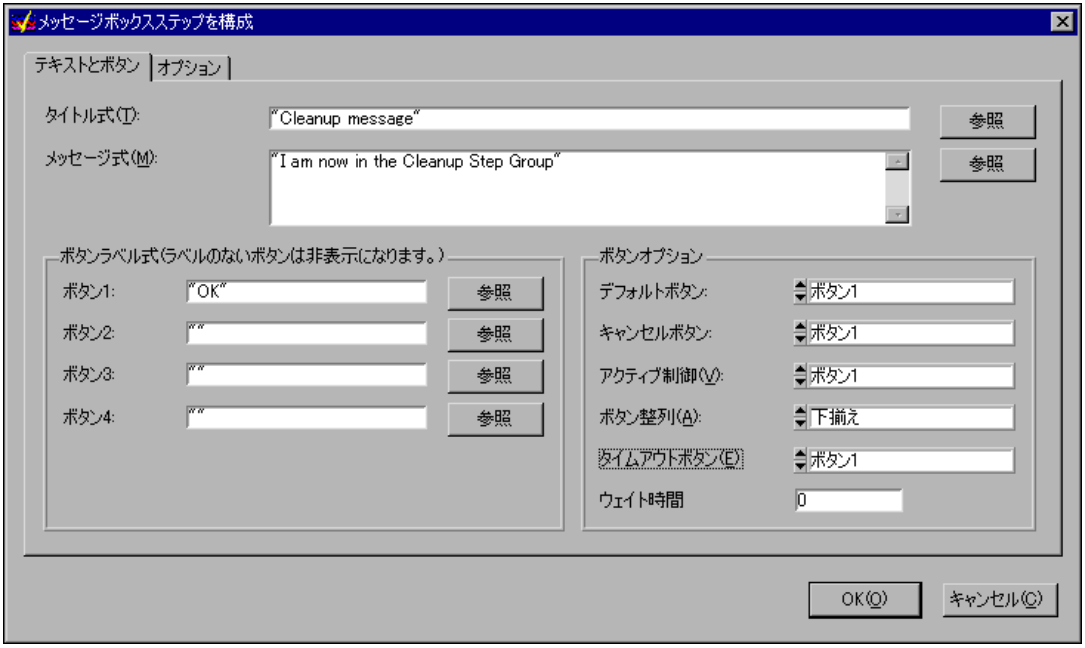

図 4-1　「メッセージボックスステップを構成」ダイアログボックス

「メッセージボックスステップを構成」ダイアログボックスのオプションタブでは、オペレータが入力するために応答テキストボックスを有効にしたり、座標を使ってメッセージポップアップの位置を決めたり、メッセージポップアップをアプリケーションに対してモーダルに設定したりすることができます。モーダルダイアログボックスとは、他のアプリケーションウィンドウを操作する前に閉じる必要のあるダイアログボックスのことをいいます。

h. **OK** ボタンをクリックしてダイアログボックスを閉じます。

4. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample3.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。

5. **実行→ MainSequence を実行**を選択して、シーケンスを直接実行します。

「最初のステップでブレーク」オプションを有効にしてあるため、シーケンスエディタは実行開始後、シーケンスの最初のステップで直ちに実行を一時停止します。図 4-2 は、シーケンスエディタの現在の状態を示しています。

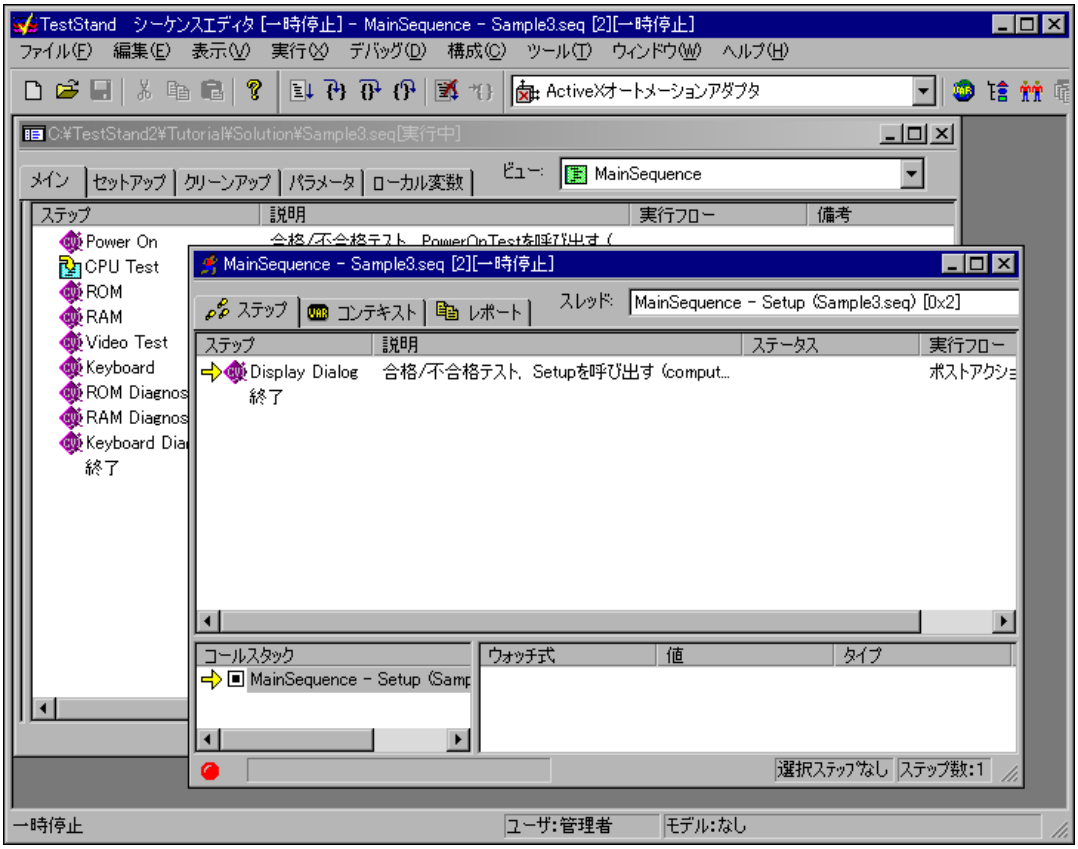

図 4-2　Sample3.seq の実行の一時停止

実行ウィンドウのタイトルには、(一時停止) のように、実行の状態が表示されます。実行が一時停止すると、実行ウィンドウのステップタブでは、実行が再開したときに実行するステップの隣に実行ポインタが表示されます。次に実行するステップは、セットアップステップグループの Display Dialog ステップです。

実行が一時停止状態になっているときは、**デバッグ**メニューの**ステップイン**、**ステップオーバー**、および**ステップアウト**コマンドを使用するか、図 4-3 に示すツールバーボタンを使用して、シーケンスをシングルステップ処理することができます。上記の各ステップタイプの説明については、『TestStand User Manual』の Chapter 6、「Sequence Execution」を参照してください。

図 4-3　シングルステップツールバーボタン

6. **ステップオーバー**ツールバーボタンをクリックして、Display Dialog ステップを実行します。このステップでは、Test Simulator ダイアログボックスが表示されます。
7. RAM テストを不合格にするよう選択し、**Done** をクリックします。

   ダイアログボックスを閉じると、セットアップステップグループの最後の「終了」でシーケンスの実行が一時停止します。
8. Sample3.seq のウィンドウをクリックするか、**ウィンドウ→Sample3.seq** を選択して、`Sample3.seq` をアクティブにします。
9. メインステップグループタブの CPU テストステップを右クリックして、コンテキストメニューから**ブレークポイントのトグル**コマンドを選択します。ステップ名の左に赤い停止サインのアイコンが表示されます。
10. ウィンドウ上をクリックするか**ウィンドウ**メニューからウィンドウ名を選択して、実行ウィンドウに戻ります。**デバッグ→再開**を選択して実行を再開します。すると、実行は CPU Test ステップで再び中断します。
11. **ステップイン**ツールバーボタンをクリックして、サブシーケンスの中に入ります。

図 4-4 は、サブシーケンスにステップインしたあとの実行ウィンドウのステップビューを示します。

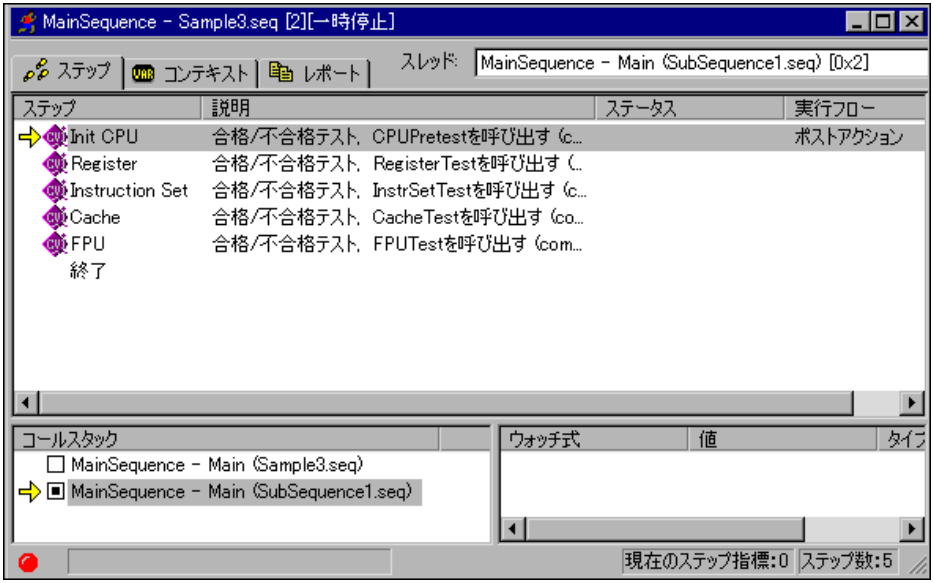

図 4-4　サブシーケンスで停止中のステップビュー

通常、ステップがサブシーケンスを呼び出すと、呼び出し側のステップを含むシーケンスはサブシーケンスが戻るのを待ちます。サブシーケンスの呼び出しは、呼び出し側のシーケンスの呼び出しにネストされています。ネストされたサブシーケンスが完了するのを待機している一連のアクティブシーケンスを、コールスタックといいます。コールスタックの最後のシーケンスが、最も深くネストされたシーケンス呼び出しです。

実行ウィンドウの左下のコールスタックペーンに、実行のコールスタックが表示されます。最も深くネストされているシーケンス呼び出しの左に、黄色の矢印アイコンが表示されます。図 4-4 のコールスタックは、`Sample3.seq` のメインシーケンスが `SubSequence1.seq` のメインシーケンスを呼び出していることを示します。

実行が中断しているときは、ラジオボタンをクリックすることによりコールスタックのシーケンス呼び出しを表示することができます。

12. コールスタックペーンのラジオボタンをクリックして、各シーケンスの呼び出しのステータスを表示します。
13. コールスタックの最も深くネストされたシーケンス呼び出しの下の部分に戻ります。

14. **ステップオーバー**ツールバーボタンを 2 回クリックして、サブシーケンス間のステップ処理を開始します。
15. シーケンスの最後に達する前に、**ステップアウト**ツールバーボタンを選択します。TestStand は現在のシーケンスの最後で実行を再開し、シーケンスコール後の次のステップの前またはブレークポイントに達するまでのいずれか早い方で、実行を中断します。
16. **ステップオーバー**ツールバーボタンを使用したシーケンスのシングルステップ処理を、実行完了まで続けます。最後に実行されるステップは、クリーンアップステップグループに追加したクリーンアップメッセージであることに注意してください。
17. 実行を完了する前に、**OK** をクリックしてクリーンアップメッセージを閉じます。実行が完了すると、実行ウィンドウはグレーになります。実行ウィンドウは閉じないでください。
18. **実行→再実行**を選択して、再実行します。シーケンスを再実行するには、実行ウィンドウがアクティブになっている必要があります。
19. 最初のステップで実行が中断した後で、**デバッグ→終了**を選択します。

    シーケンスの実行を終了しても、クリーンアップメッセージダイアログボックスが表示されることに注意してください。オペレータが実行を終了するか、ランタイムエラーによって実行が終了すると、実行は直ちにクリーンアップグループのステップに進みます。
20. **OK** をクリックしてクリーンアップメッセージを閉じます。
21. **実行→再実行**を選択して、もう一度実行します。
22. 最初のステップで実行が一時停止した後で、**デバッグ→中断**を選択します。シーケンスの実行は直ちに停止し、クリーンアップステップグループのステップは一切実行されません。
23. 実行ウィンドウを閉じます。
24. **ファイル→保存**を選択してシーケンスを保存します。
25. `Sample3.seq` ウィンドウを閉じます。

このチュートリアルセッションはこれで終わりです。次のセッションでは、TestStand の変数およびプロパティを作成し使用する方法を学習します。

5

変数およびプロパティを使用する

本章では、TestStand での変数およびプロパティの使用方法を説明し、変数およびプロパティの値をモニタするために役立つ TestStand の機能を紹介します。

TestStand では、変数をさまざまな範囲で指定して、シーケンスのステップ間や複数のシーケンス間でデータを共有することができます。シーケンスに対しローカルの変数、シーケンスファイルに対しグローバルの変数、およびテストステーションに対しグローバルの変数を定義することができます。それらの変数は以下のように使用してください。

- ローカル変数は、シーケンスの実行に関係するデータを保存するのに使用できます。各ステップおよびステップモジュールは、シーケンスローカル変数に直接アクセスすることができます。
- シーケンスファイルグローバル変数は、シーケンスファイル全体に関連するデータを保存するのに使用できます。各シーケンスおよびシーケンスフィル内のステップは、これらのグローバル変数に直接アクセスできます。
- ステーショングローバル変数には、いずれのシーケンス、ステップ、またはコードモジュールからでもアクセスできます。他の変数と異なり、ステーショングローバル変数は 1 つの TestStand セッションから次の TestStand セッションへ保存されます。通常、ステーショングローバル変数を使用して、統計を維持したり、テストステーションの構成を表します。

サンプルをセットアップする

第 4 章、「シーケンスをデバッグする」を終了していない場合は、シーケンスエディタ内のウィンドウをすべて閉じてからこのチュートリアルセッションを実行してください。

TestStand 変数を使用する

この練習では、ローカル変数の作成および使用方法について説明します。ここで習得する概念は、シーケンスファイルグローバル変数およびステーショングローバル変数に応用できます。

1. **ファイル→開く**を選択して、第 3 章、「シーケンスのステップを編集する」で作成した `<TestStand>\Tutorial\Sample2.seq` ファイルを開きます。このファイルは、`<TestStand>\Tutorial\Solution` ディレクトリにもあります。
2. 「ビュー」リングで `MainSequence` を選択して、シーケンスファイルウィンドウに `MainSequence` を表示します。
3. シーケンスウィンドウのローカル変数タブをクリックします。すると、`Sample2.seq` シーケンスファイルの `MainSequence` 用に現在定義されているすべてのローカル変数がビューに表示されます。新規シーケンスを作成する際、デフォルトで TestStand は 1 つのローカル変数 `ResultList` しか定義しません。TestStand は、この配列変数を使用して、このシーケンスで実行するステップの結果を保存します。このステップ結果の配列は、レポート生成に使用されます。
4. 右側のペーン内で右クリックして、図 5-1 に示すようにコンテキストメニューから**ローカル変数を挿入→数値**を選択します。このように選択すると、シーケンスエディタは新しい数値ローカル変数を挿入します。

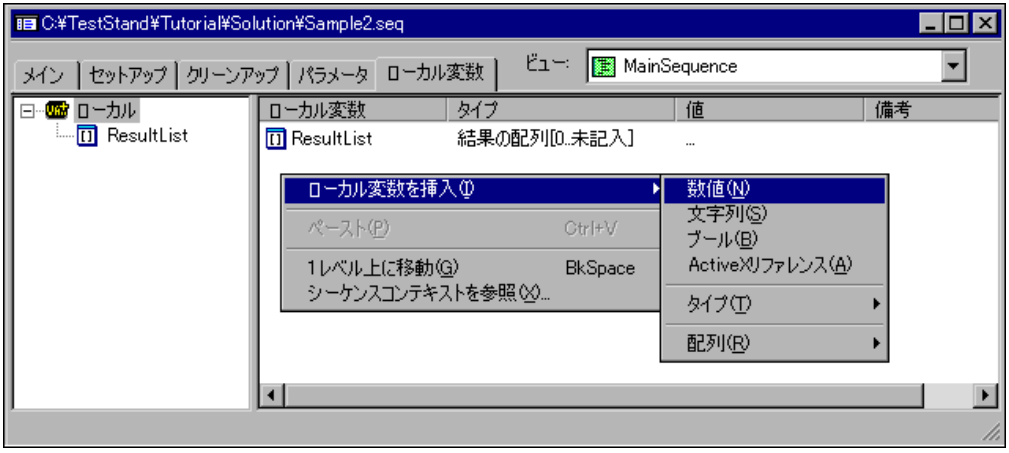

図 5-1　「ローカル変数を挿入」コンテキストメニューコマンド

5. 変数に `LoopIndex` という名前を付けます。

メモ　TestStand の変数名は、頭に数字を使用したりスペースを含むことはできません。

6. 以下の手順に従い、シーケンスにステップを追加して、`LoopIndex` ローカル変数の値に基づき一連のステップでループで実行するようにします。
   a. シーケンスファイルウィンドウのメインタブをクリックして、メインステップグループのステップを表示します。
   b. Power On テストを右クリックして、コンテキストメニューから**ステップを挿入→ステートメント**を選択します。
   c. 新しいステップに `Reset Loop Index` という名前を付けます。

      ステートメントステップを使用して、TestStand がそのステップを実行する際に評価する式を実行します。たとえば、ステートメントステップを使用して、シーケンスファイルのローカル変数の値を増分することができます。
   d. Reset Loop Index ステップを右クリックして、コンテキストメニューから**式を編集**を選択します。図 5-2 に示すように、「ステートメントステップを編集」ダイアログボックスが表示されます。

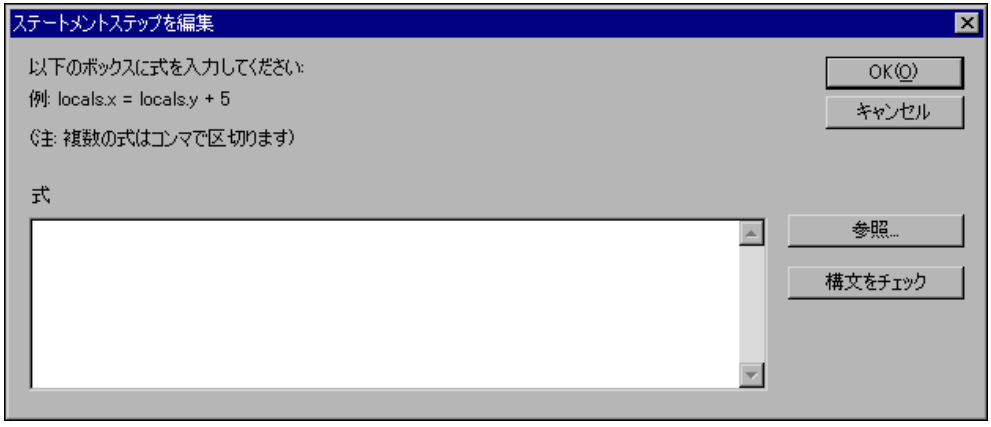

図 5-2　「ステートメントステップを編集」ダイアログボックス

e. **参照**ボタンをクリックして、図 5-3 に示すように式参照ダイアログボックスを表示します。

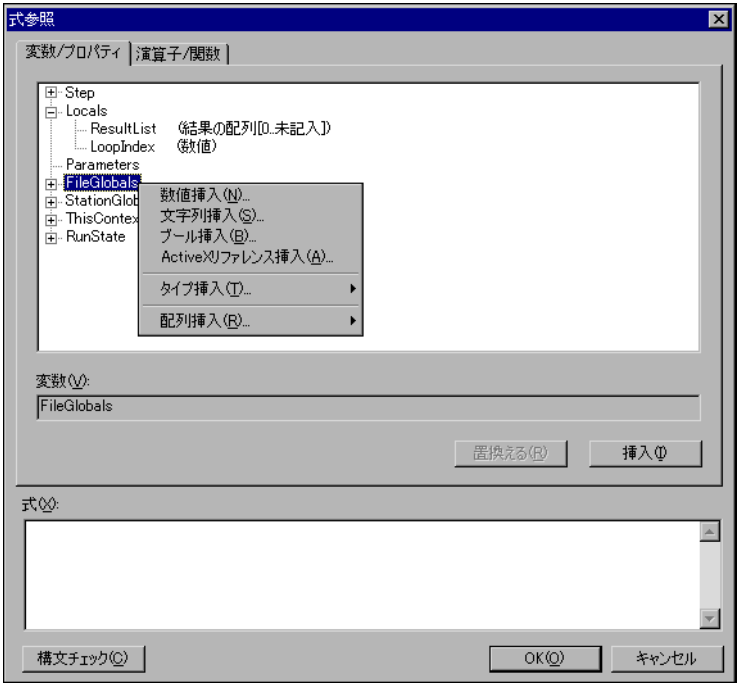

図 5-3　式参照ダイアログボックス

式参照ダイアログボックスを使用して、式を対話式に構築したり変数およびパラメータを作成したりします。式参照ダイアログボックスには、変数／プロパティと演算子／関数の 2 つのタブがあります。変数／プロパティタブのツリー表示から、変数やプロパティを選択できます。演算子／関数タブには、すべての定義済み演算子および関数のリストが含まれています。

式参照ダイアログボックスには、現在選択されている演算子または関数のヘルプテキストがあります。TestStand は、C、C++、Java、および Visual Basic で使用する、適用可能な式演算子や構文をすべてサポートします。

変数は、図 5-3 に示すコンテキストメニューを使用して、式参照ダイアログボックスから直接作成することができます。マウスポインタを式参照ダイアログボックスのプロパティ上に置くと、ヒントラベルが表示されます。変数を作成できるプロパティには、**右クリックして新しい変数を挿入**というヒントラベルが表示されます。

f. 名前をダブルクリックするか、項目の＋アイコンをクリックして、Locals 項目を展開します。ツリー表示の項目を展開すると、その項目の下のすべての項目がダイアログボックスに表示されます。ツリー表示の各項目は、TestStand のプロパティまたは変数です。

g. Locals プロパティの下の `LoopIndex` 変数を選択して**挿入**ボタンをクリックします。すると、式に `Locals.LoopIndex` が入力されます。

サブプロパティを参照するには、ピリオドを使用して、プロパティの名前とサブプロパティの名前を区切ります。たとえば、`Locals.LoopIndex` によって `Locals` プロパティの `LoopIndex` サブプロパティを参照します。

h. 演算子／関数タブをクリックして、左のペーンから代入カテゴリを選択します。

i. 右のペーンから代入演算子（=）を選択します。

j. **挿入**をクリックして、式に代入演算子を追加します。すると式表示器に `Locals.LoopIndex =` と表示されます。

k. 式制御器の等号のすぐ後にテキストカーソルを置き、数字の 0 を入力して、式が `Locals.LoopIndex = 0` となるようにします。

l. **OK** をクリックして「ステートメントステップを編集」ダイアログボックスに戻ります。

m. **構文チェック**ボタンをクリックして、式に無効な構文が含まれていないことを確認します。

n. **OK** ボタンをクリックして「ステートメントステップを編集」ダイアログボックスを閉じ、シーケンスエディタウィンドウに戻ります。

o. Reset Loop Index ステップを右クリックして、コンテキストメニューから**ステップを挿入→ラベル**を選択します。

p. 新しいステップに `Loop Begin` という名前を付けます。本セッションで後ほど説明しますが、ラベルステップは通常 Goto ステップのターゲットとして使用します。ラベルステップを使用すると、Goto ステップが参照するターゲットステップを変更することなしに、ラベルステップ周辺の他のステップの並べ替えや削除ができます。

7. ステートメントステップを追加して、以下のように、`LoopIndex` ローカル変数の値を増分します。

a. RAM テストを右クリックして、コンテキストメニューから**ステップを挿入→ステートメント**を選択します。

b. 新しいステップに `Increment Loop Index` という名前を付けます。

c. Increment Loop Index ステップを右クリックして、コンテキストメニューから**式を編集**を選択します。

d. **停止**ボタンをクリックして式参照ダイアログボックスを表示します。

e. 式参照ダイアログボックスを使用して、以下の式を作成するか、式制御器に直接入力します。

```
Locals.LoopIndex ++
```

増分演算子（++）は、演算子／関数タブの算術グループの中にあります。

f. **OK** を 2 回クリックして、式参照とステートメントステップを編集の両ダイアログボックスを閉じます。

8. ループストラクチャを完成させるには、シーケンスに次の手順で Goto ステップを追加します。

a. Increment Loop Index ステップを右クリックして、コンテキストメニューから**ステップを挿入→ Goto** を選択します。

b. ステップに `Loop End` という名前を付けます。

c. Loop End ステップを右クリックして、コンテキストメニューから**移動先を編集**を選択します。

d. 移動先制御器の隣の矢印をクリックして、Loop Begin を選択します。

e. **OK** をクリックしてダイアログボックスを閉じます。

9. ループストラクチャを完成させるには、`LoopIndex` 変数の値が特定の値より低い場合にのみ実行するように、Loop End ステップの実行条件を設定します。

a. Loop End ステップを右クリックして、コンテキストメニューから**プロパティ**を選択します。

b. **実行条件**ボタンをクリックして実行条件ダイアログボックスを開きます。

c. **新規式を挿入**をクリックします。

d. ダイアログボックスの「式を編集／表示」セクションの**参照**ボタンをクリックして、式参照ダイアログボックスを開きます。

e. 式参照ダイアログボックスを使用して、次の式を作成します。

```
Locals.LoopIndex < 5
```

小なり演算子（<）は、演算子／関数タブの比較グループの中にあります。

f. **OK** をクリックして式参照ダイアログボックスを閉じます。

図 5-4 は、完成した実行条件ダイアログボックスを示します。

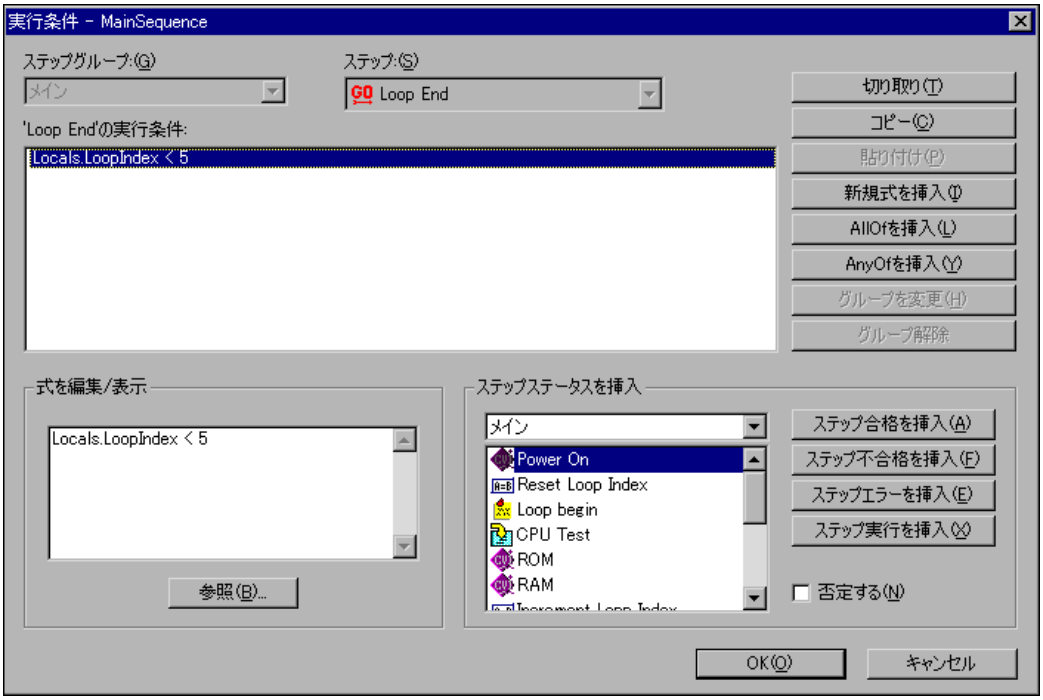

図 5-4　Loop End ステップの実行条件

10. **OK** を 2 回クリックして実行条件ダイアログボックスとステップのプロパティダイアログボックスの両方を閉じます。
11. **ファイル→別名で保存**を選択します。シーケンスを `Sample4.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。
12. **実行**メニューをクリックして、**最初のステップでブレーク**がチェックされているかどうか確認します。
13. **最初のステップでブレーク**がチェックされている場合は、チェックを外します。
14. **実行→一回実行**を選択してシーケンスを実行します。
15. Test Simulator ダイアログボックスの **Done** をクリックします。
16. シーケンスの実行後、テストレポートの内容を確認し、ループ内のステップ（CPU Test、ROM Test、RAM Test）が 5 回実行されたことに注目してください。
17. 実行ウィンドウを閉じます。

コンテキストタブを使用する

この練習では、実行ウィンドウのウォッチ式ペーンを使って、TestStand がシーケンスを実行する間の `LoopIndex` 変数の値を確認します。

1. Loop End goto ステップを右クリックして、コンテキストメニューから**ブレークポイントのトグル**を選択し、ステップにブレークポイントを設定します。シーケンスウィンドウで、Loop End ステップの隣にブレークポイントを示す赤い停止サインのアイコンが表示されます。
2. **実行→一回実行**を選択してシーケンスを実行します。
3. Test Simulator ダイアログボックスの **Done** をクリックします。実行は Loop End ステップで中断します。
4. シーケンスウィンドウのコンテキストタブをクリックします。
5. 左側上のツリー表示の中の `Locals` プロパティを展開します。
6. `Locals` プロパティの中の `LoopIndex` プロパティをクリックして、図 5-5 に示すように数値が 1 であることを確認してください。

図 5-5　コンテキストタブ

コンテキストタブには、コールスタックセクションで現在選択されているシーケンス呼び出しのシーケンスコンテキストが表示されます。シーケンスコンテキストには、選択したシーケンス呼び出しの中のステップによってアクセス可能なすべての変数およびプロパティが含まれています。コンテキストタブを使用して、それらの変数およびプロパティの値を確認したり変更したりします。

シーケンスのステップを実行する前に、TestStand はシーケンスのランタイムコピーを作成します。これにより同じシーケンスのパラレル実行が可能になり、各実行で他の実行の変数やプロパティが変更され

ることはありません。実行が完了すると、TestStand はシーケンスのランタイムコピーを破棄します。

TestStand は、アクティブなシーケンスごとにシーケンスコンテキスト上を保持します。シーケンスコンテキストは、シーケンスの実行状態を表します。シーケンスコンテキストの内容は、現在実行中のシーケンスおよびステップによって異なります。プロパティおよび変数のオリジナルとランタイムコピーのどちらもシーケンスコンテキストからアクセスできます。

シーケンスコンテキストを使用して、式の中やステップモジュールからの TestStand ActiveX API の呼び出しを介して変数やステップのプロパティにアクセスすることができます。式に関する詳細については、本章の「ウォッチ式ペーンを使用する」セクションを参照してください。TestStand API に関する詳細は、『TestStand Programmer Help』を参照してください。

表 5-1 は、シーケンスコンテキストの最上位プロパティとその内容のリストです。シーケンスコンテキストに関する詳細は、『TestStand User Manual』の Chapter 8、「Sequence Context and Expressions」を参照してください。

表 5-1　シーケンスコンテキストの最上位プロパティ

シーケンスコンテキストのサブプロパティ	説明
`Step`	現在のシーケンス呼び出しの実行中のステップのプロパティが含まれます。`Step` プロパティは、ステップの実行中のみ存在します。ブレークポイントなど、実行がステップとステップの間にあるときこのプロパティは存在しません。
`Locals`	現在のシーケンス呼び出しのシーケンスローカル変数が含まれます。
`Parameters`	現在のシーケンス呼び出しのシーケンスパラメータが含まれます。
`FileGlobals`	現在の実行のシーケンスファイルグローバル変数が含まれます。
`StationGlobals`	エンジン呼び出しのステーショングローバル変数が含まれます。TestStand はステーショングローバル変数のコピーを 1 つメモリに保持しています。
`ThisContext`	現在のシーケンスコンテキストへのリファレンスが含まれます。通常このプロパティは、シーケンスコンテキスト全体を引数としてサブシーケンスやステップモジュールに渡すのに使用します。
`RunState`	現在のステップ、現在のシーケンス、呼び出し側のシーケンスなど、シーケンス呼び出しの実行の状態を記述するプロパティが含まれます。

7. **デバッグ→再開**を選択して、実行が再開し Loop End goto ステップで再び中断することを確認してください。

8. コンテキストタブを再度クリックして、LoopIndex の値が 2 になっていることに注意してください。実行を一時停止状態のままにしておきます。

ウォッチ式ペーンを使用する

この練習では、ウォッチ式ペーンで変数 LoopIndex の値をモニタします。ウォッチ式ペーンは、図 5-6 に示すように実行ウィンドウの右下部分にあります。ウォッチ式ペーンには、ユーザが入力するウォッチ式の値が表示されます。実行がブレークポイントで中断されると、ウォッチ式ペーンの値が更新されます。追跡が有効になっている場合は、各ステップの実行後にも値が更新されます。

通常、ウォッチ式を使用して、シーケンスを追跡したりシングルステップする際に、変数やプロパティの値をモニタします。個々の変数やプロパティをコンテキストタブからウォッチ式ペーンにドラッグすることができます。

変数 LoopIndex のウォッチ式を作成するには、以下の手順に従ってください。

1. コンテキストタブのツリー表示の LoopIndex プロパティをクリックして、マウスボタンを押したまま、ツリー表示からウォッチ式ペーンにドラッグします。カーソルがウォッチ式ペーンの上に来たときにマウスボタンを放します。

   図 5-6 に示すように、ウォッチ式の値はすでに 2 になっていることに注目してください。

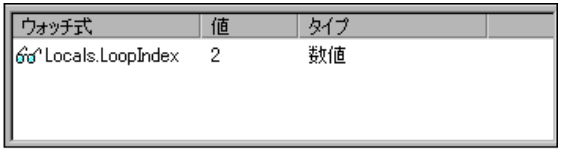

図 5-6　更新されたウォッチウィンドウセクション

2. ここで、**デバッグ→再開**を選択し、実行が Goto ステップで再度中断すると、ウォッチ式の値は 2 から 3 に変わることを確認してください。
3. Loop End ステップアイコンの左をクリックして、ブレークポイントを削除します。
4. **デバッグ→再開**を選択して実行を完了します。

5. 実行ウィンドウを閉じます。
6. **ファイル→保存**を選択してシーケンスを保存すると、`Sample4.seq` シーケンスファイルに対して加えた変更が保存されます。

ウォッチ式ペーンでは、より複雑な式を作成することができます。新規の式を追加するには、ウォッチ式ペーンを右クリックして、コンテキストメニューから**ウォッチを追加**を選択します。既存のウォッチ式を編集するには、式を右クリックしてコンテキストメニューから**式を編集**を選択します。どちらの項目を選択した場合も、式を作成するための式参照ダイアログボックスが表示されます。

メモ　ウォッチ式をコピーして次の実行のウォッチ式ペーンに貼り付けることができます。実行完了後、実行表示ウィンドウを閉じる前に**実行→再実行**を選択すると、以降の実行でウォッチ式が自動的に保持されます。

ウォッチ式ペーンの詳細については、『TestStand User Manual』の Chapter 6、「Sequence Execution」を参照してください。TestStand での変数やプロパティの使用に関する詳細については、『TestStand User Manual』の Chapter 5、「Sequence Files」、Chapter 7、「Station Global Variables」、および Chapter 8, 「Sequence Context and Expressions」を参照してください。TestStand の変数やプロパティの使用に関しては、『TestStand Programmer Help』の下記のトピックも参照いただけます。「Object Relationships」、「Sequence Context」、「Property Paths」、「Using Property Paths」、「Finding a Property Path」、「Viewing Step Properties」、「Commonly Used Properties」。

このチュートリアルセッションはこれで終わりです。次のセッションでは、LabVIEW および LabWindows/CVI 開発環境でテストを作成しデバッグする方法を学習します。

6

テストの作成とデバッグ

本章では、LabVIEW 標準プロトタイプアダプタ、C/CVI 標準プロトタイプアダプタ、および DLL フレキシブルプロトタイプアダプタを使用して呼び出したコードモジュールの作成およびデバッグ方法について説明します。コードモジュールの作成やデバッグには、LabVIEW や LabWindows を使用します。

LabVIEW または LabWindows/CVI を使用していない場合は、LabVIEW 標準プロトタイプアダプタおよび C/CVI 標準プロトタイプアダプタを使用する練習はスキップしてもかまいません。コードモジュールが C スタイルの DLL の場合は、アプリケーション開発環境が何かにかかわらず、DLL フレキシブルプロトタイプアダプタを使用する練習は役に立つでしょう。

他のアプリケーション開発環境（ADE）で作成されたコードモジュールも作成しデバッグすることができます。ただし、それについては本書では説明しません。他の ADE を使用したコードの作成やデバッグに関する情報は、National Instruments Developers Zone（英語）をご覧ください。

LabVIEW 標準プロトタイプアダプタを使用して LabVIEW VI をデバッグする

この練習では、TestStand で使用可能な LabVIEW テストモジュールを作成する方法、および TestStand シーケンスエディタから仮想計測器（VI）にステップインしてデバッグする方法について説明します。このチュートリアルセッションでは、LabVIEW 開発環境の使用経験があることを前提としています。LabVIEW を使用していなくて、LabWindows/CVI を使用しているか C スタイルの DLL を作成する場合は、このセクションをスキップして本章の「C/CVI 標準プロトタイプアダプタを使用して LabWindows/CVI DLL をデバッグする」のセクションに進んでけっこうです。

メモ TestStand に対応する正しいバージョンの LabVIEW を使用していることを確認してください。詳細については、`TestStand\Doc` ディレクトリの `readme.txt` ファイルを参照してください。

サンプルをセットアップする

第 5 章、「変数およびプロパティを使用する」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択して、第 5 章、「変数およびプロパティを使用する」で作成した <TestStand>\Tutorial\ Sample4.seq ファイルを開きます。このファイルは、<TestStand>\Tutorial\Solution ディレクトリにもあります。

仮想計測器コードモジュールを作成する

この練習では、TestStand のシーケンスから呼び出す LabVIEW VI を作成します。

1. 次の手順で、LabVIEW 標準プロトタイプアダプタが正しく構成されていることを確認してください。
   a. **構成→アダプタ**を選択すると、図 6-1 に示すようなアダプタ構成ダイアログボックスが表示されます。

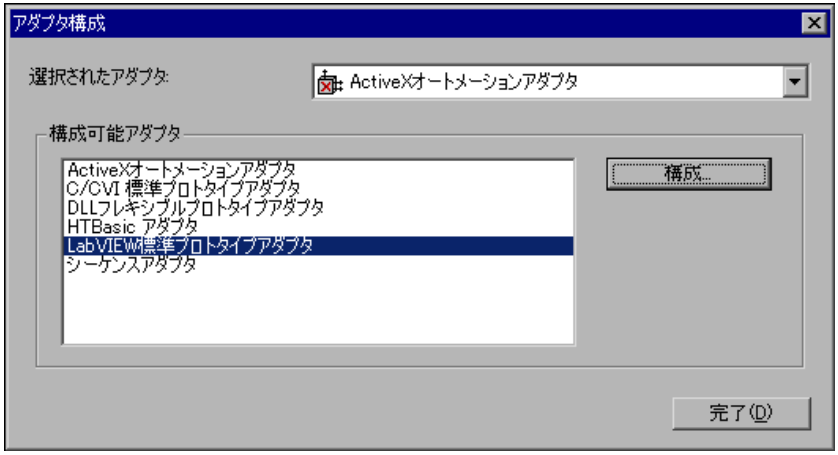

図 6-1　アダプタ構成

   b. 構成可能アダプタセクションで、LabVIEW 標準プロトタイプアダプタを選択します。
   c. **構成**ボタンをクリックします。

d. 図 6-2 のように、「使用する LabVIEW ActiveX サーバを選択」制御器が LabVIEW になっていることを確認します。

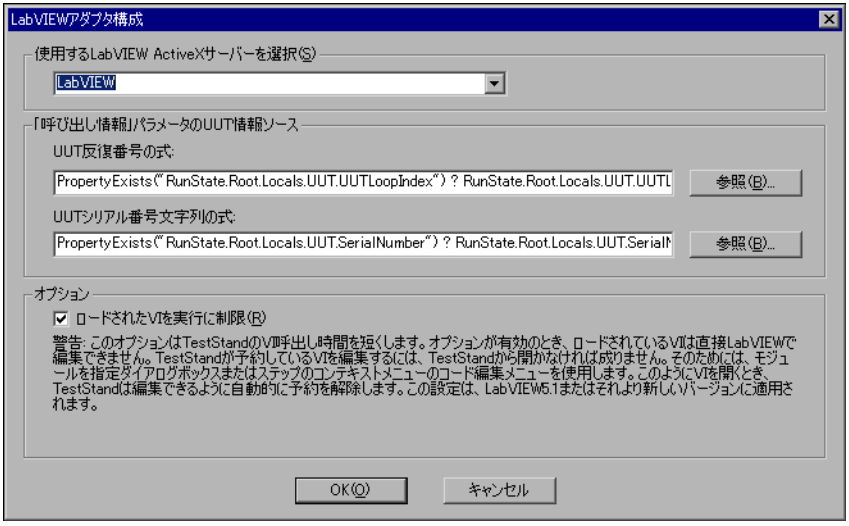

図 6-2　LabVIEW アダプタ構成

e. **OK** をクリックして、次に**完了**をクリックして、LabVIEW アダプタ構成ダイアログボックスとアダプタ構成ダイアログボックスを閉じます。

2. アダプタ選択リング制御器で、LabVIEW 標準プロトタイプアダプタを選択します。
3. メインステップグループの `Power On` テストを右クリックして、コンテキストメニューから **ステップを挿入→テスト→数値リミットテスト**を選択します。
4. 新しいステップに `Clock Frequency Test` という名前を付けます。
5. Clock Frequency Test を右クリックして、コンテキストメニューから**モジュールを指定**を選択します。

6. 「呼び出されたら VI のフロントパネルを表示」チェックボックス制御器にチェックを付けます。LabVIEW ステップモジュール情報は、図 6-3 に示すようになっているはずです。

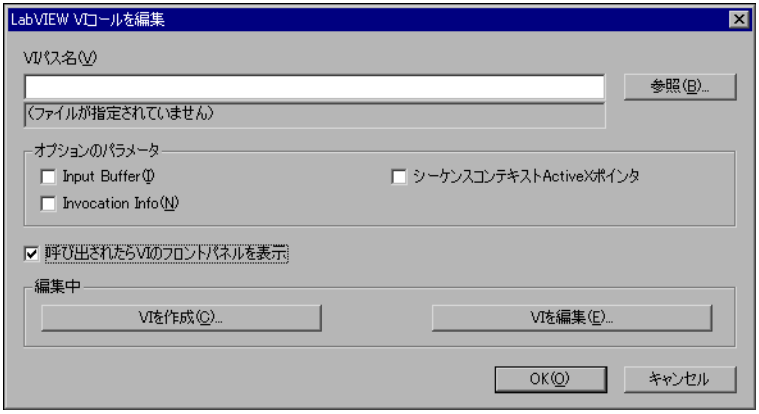

図 6-3　LabVIEW ステップモジュール情報

モジュールアダプタを使用すると、ソースコードテンプレートを使用してステップモジュールのソースコードシェルを生成することができます。テンプレートファイルはステップタイプおよびモジュールアダプタにより異なります。特定のアダプタとステップタイプの組み合わせに対しては、複数のソースコードテンプレートを使用できる場合があります。

ソースコードテンプレートをサポートする各モジュールアダプタに対し、モジュールを指定ダイアログボックスにはソースコード作成用のボタンが含まれています。ステップタイプに対して複数のテンプレートがある場合は、使用可能なテンプレートのリストから 1 つ選ぶようにプロンプトされます。それ以外の場合は、アダプタは 1 つのテンプレートしか使用しません。

7. 「LabVIEW VI コールを編集」ダイアログボックスの **VI を作成**ボタンをクリックします。すると、TestStand はステップのコードモジュールのパス名を選択するようプロンプトします。
8. `TestStand\Tutorial` ディレクトリを開きます。ファイル名制御器に `Clock Frequency.vi` という名前を入力します。このチュートリアルセッションを他の人が以前に行っていると、この VI はすでにあるかもしれません。
9. **OK** をクリックして、「ステップのコードモジュールのパス名」ダイアログボックスを閉じます。

TestStand は、数値リミットテストステップと LabVIEW 標準プロトタイプアダプタに関連付けられたコードテンプレートを使用して、`Clock Frequency.vi` という名前の新規の VI を作成します。次に TestStand は、図 6-4 に示すように、LabVIEW で新規の VI を開きます。

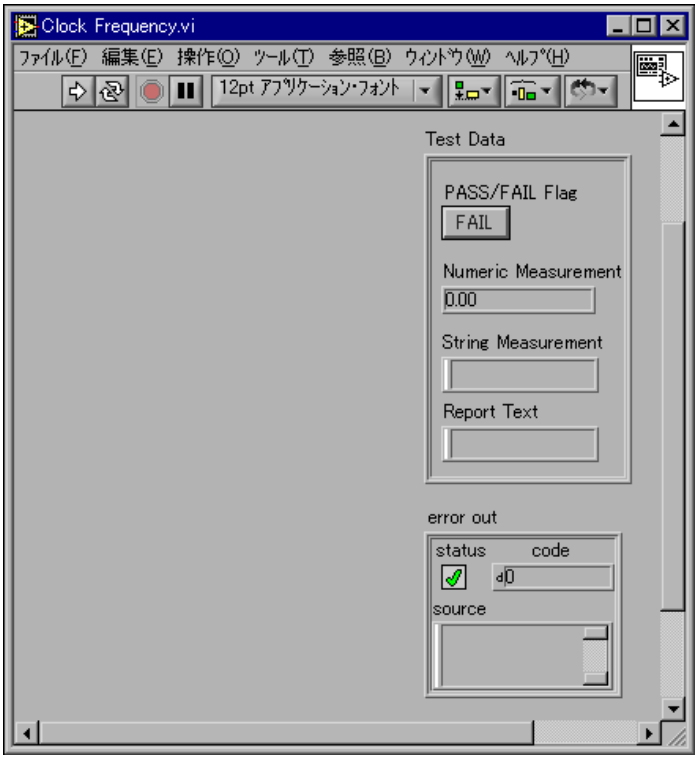

図 6-4　LabVIEW の新規の Clock Frequency VI

`Clock Frequency.vi` のフロントパネルには、テストデータクラスタとエラー出力クラスタの 2 つの表示器があります。LabVIEW 標準プロトタイプアダプタは、それらの特殊なデータクラスタを使用して TestStand とテスト VI 間で共通のデータの受け渡しを行います。第 10 章、「コードモジュールで ActiveX を使用する」で説明しますが、シーケンスコンテキストと TestStand API を使用して、シーケンスのすべての変数およびプロパティへのアクセスや、シーケンスの実行もできます。モジュールアダプタでデータの受け渡しが可能な特殊なデータタイプは他にもありますが、テストデータクラスタとエラー出力クラスタは必須制御器です。それらの 2 つのクラスタ内の要素と、アダプタによるそれらの使用方法を以下に示します。

Test Data

- **PASS/FAIL Flag**—テストの合否をこのブール表示器に表示します。
- **Numeric Measurement**—テスト VI が返す数値測定値。
- **String Measurement**—テスト関数が返す文字列値。
- **Report Text**—レポートに表示する出力メッセージ。

エラー出力

- **Status**—このブール表示器は、エラーが発生すると `True` になります。
- **Code**—エラーが発生すると、0 以外の値になることがあります。
- **Source**—エラーが発生すると、記述的な文字列になることがあります。

これらのストラクチャに関する詳細は、『TestStand User Manual』の Chapter 13、「Module Adapters」を参照してください。

10. 図 6-5 に示すように、次の LabVIEW 制御器をフロントパネルに追加します。
    - 周波数測定というラベルのついた数値制御器
    - 「追加のレポートテキスト」というラベルのついた文字列制御器
    - 「戻る」というラベルのついたダイアログボタン

図 6-5　完成した Clock Frequency.vi のフロントパネル

11. VI のブロックダイアグラムを図 6-6 のように配線します。

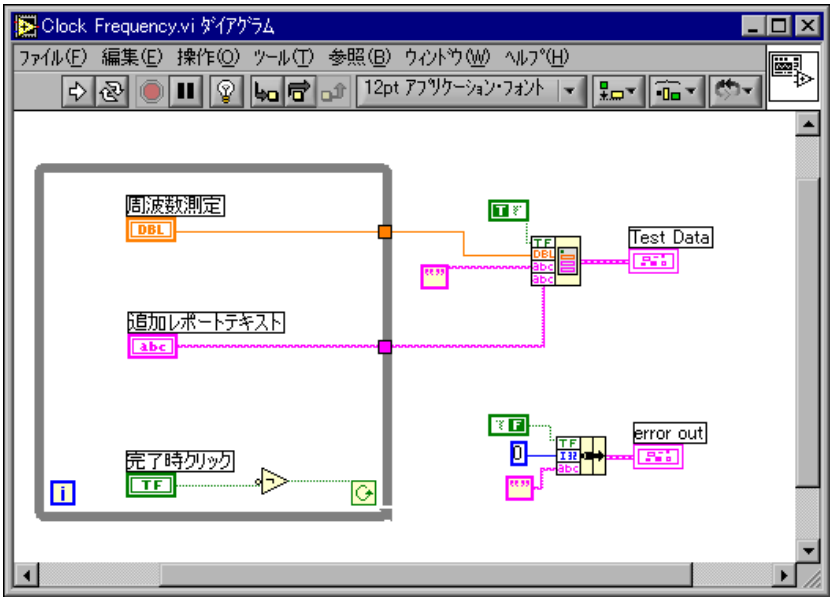

図 6-6　Clock Frequency.vi のブロックダイアグラム

VI を実行すると、ユーザが周波数制御器と「追加のレポートテキスト」制御器に値を入力して **Return** ボタンをクリックするまで、VI はループで実行を続けます。

12. VI の構築が終了したら、LabVIEW で**ファイル→保存**を選択して VI を保存します。
13. VI のダイアグラムとフロントパネルを閉じます。
14. シーケンスエディタに戻り、**OK** をクリックして「LabVIEW VI コールを編集」ダイアログボックスを閉じます。
15. Clock Frequency Test を右クリックして、コンテキストメニューから**リミットの編集**コマンドを選択します。「数値リミットテストの編集」ダイアログボックスが表示されます。
16. 図 6-7 に示すように、比較タイプ制御器を `LT (<)` に、値を `100` に設定します。
17. このステップはマザーボードのクロック周波数測定をシミュレーションしますので、単位を `MHz` に変更します。単位制御器の隣のドロップダウンリングを使用して、単位には `Hertz` を、単位の接頭辞には `Mega` を選択します。すると単位制御器には、`megahertz` という値が表示されます。短縮形の名前を使用するには、各リングで短縮名を選択します。これにより単位制御器には、`MHz` という値が表示されます。指定した単位は、レポートと結果のデータベースに表示されま

す。単位と単位の接頭辞は表示およびドキュメント用のみのもので、測定値をスケールしたりリミットの比較に影響を与えることはありません。

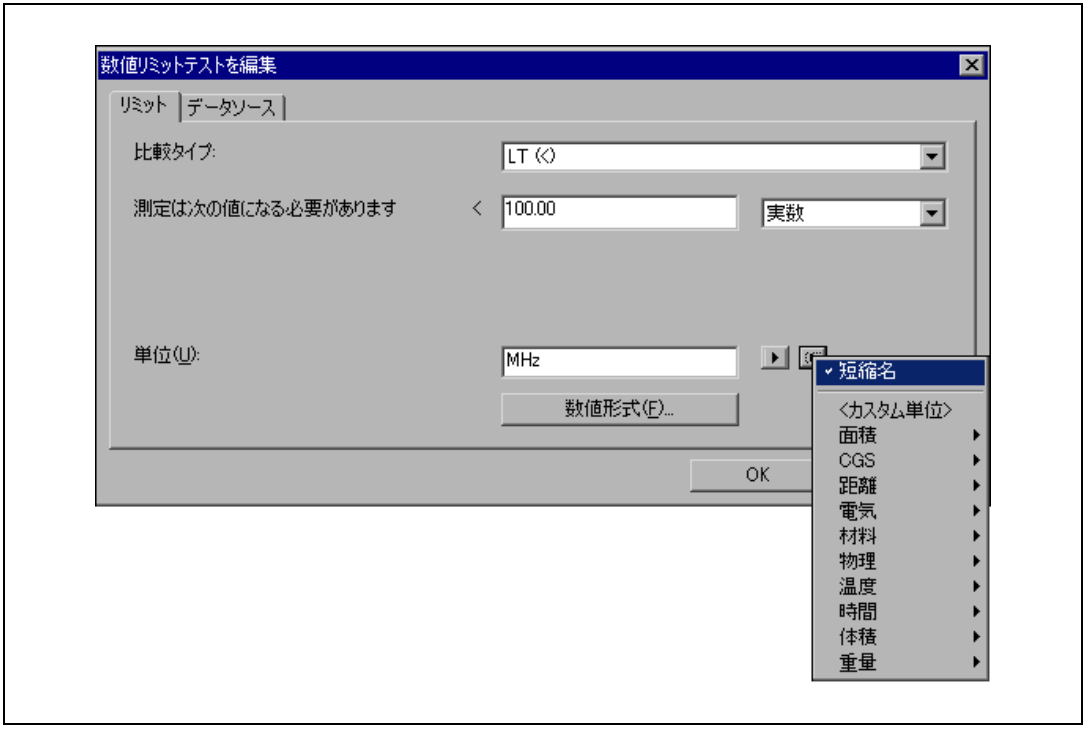

図 6-7　「数値リミットテストを編集」ダイアログボックス

18. **数値形式**ボタンをクリックして数値形式ダイアログボックスを開きます。このダイアログボックスの制御器の値を、図 6-8 に示すように設定します。これらの設定により、ステップの測定とリミットの値が指定されます。形式は、「数値リミットテストを編集」ダイアログボックス、ステップの説明、およびテストレポートに表示されるリミット値に適用されます。

    この設定で、TestStand は VI が返す数値測定値を定数 100 と比較します。比較が `True` の場合、ステップは合格になります。それ以外の場合は不合格です。

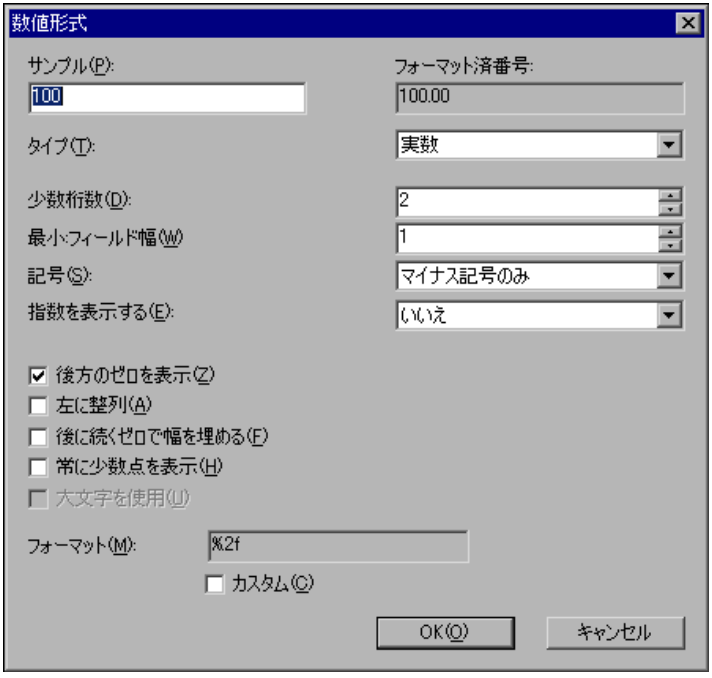

図 6-8　数値形式ダイアログボックス

19. **OK** を 2 回クリックして、数値形式ダイアログボックスと数値リミットテストを編集ダイアログボックスを閉じます。
20. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample5.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。
21. **実行→一回実行**を選択してシーケンスを実行します。TestStand が Clock Frequency Test ステップを実行すると、VI のフロントパネルが表示されて VI が実行します。
22. 周波数測定制御器に `20` という数値を入力します。
23. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
24. **Return** コマンドボタンをクリックして、VI からシーケンスの実行に戻ります。
25. シーケンスの実行が完了したら、テストレポートの内容を確認します。Clock Frequency ステップのステータス、測定値、およびレポートテキスト値を確認してください。
26. 実行ウィンドウを閉じます。

仮想計測器コードモジュールをデバッグする

TestStand では、シーケンスをデバッグできるだけでなく、デバッグ可能な LabVIEW VI にステップインすることもできます。この練習では、シーケンスエディタでシーケンスを実行しながら LabVIEW テスト VI をデバッグする方法を学習します。

1. ステップ名を右クリックして**ブレークポイントのトグル**を選択するか、ステップのアイコンの左側をクリックして、Clock Frequency Test ステップにブレークポイントを設定します。
2. **実行→一回実行**を選択してシーケンスを実行します。
3. Test Simulator プロンプトで **Done** をクリックします。すると実行は Clock Frequency Test ステップで一時停止します。
4. **ステップイン**ツールバーボタンをクリックすると、LabVIEW で `Clock Frequency.vi` のフロントパネルが表示されます。VI テストは一時停止状態になっています。
5. LabVIEW で、実行ツールバーボタンをクリックして VI を実行します。
6. LabVIEW ウィンドウで**ウィンドウ→ダイアグラムを表示**を選択して、VI のブロックダイアグラムを表示します。
7. 図 6-9 に示すように、**実行のハイライト**ツールバーボタンをクリックして、VI 内の実行フローをハイライトします。より詳細なデバッグを行うには、VI 内にブレークポイントやプローブを設定することができます。

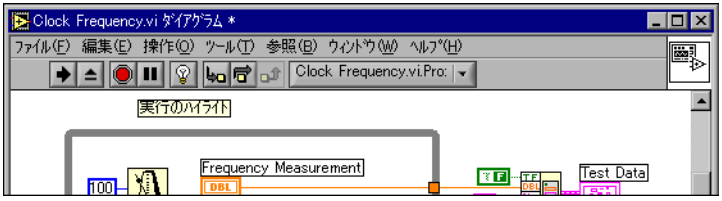

図 6-9　LabVIEW 実行のハイライトモード

8. **実行のハイライト**ツールバーボタンを再度クリックして、実行のハイライトをオフにします。
9. **ウィンドウ→パネルを表示**を選択して、フロントパネルに戻ります。
10. 周波数測定制御器に `200` という数値を入力します。
11. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
12. **Return** コマンドボタンをクリックして VI を停止します。
13. **実行ボタンの右隣の発呼者へ戻る**ツールバーボタンをクリックして、テストデータ制御器とエラー出力表示器に値を持つシーケンスの実行に戻ります。

Clock Frequency Test ステップの実行後、TestStand は Reset Loop Index ステップでシーケンスの実行を中断します。Clock Frequency Test ステップのステータスが意図したとおり不合格になっていることを確認してください。

メモ　発呼者に戻らないと、TestStand はシーケンスの実行を続行できません。発呼者に戻らずに、VI のデバッグからシーケンスエディタに切り替えてしまうという誤りがよくあります。そのような場合、シーケンスの実行はハングしたようになります。

14. **デバッグ→再開**を選択して実行を完了します。
15. `Sample5.seq` 保存して、実行ウィンドウとシーケンスファイルウィンドウを閉じます。

ダイアログボックスの表示など、ステップの中でタスクを実行する際は、現在の TestStand の実行状況を監視する必要があります。実行が終了しているときは、VI 内で実行しているタスクを中止する必要があります。この機能は、このサンプルのソリューション、`<TestStand>\Tutorial\Solution\Clock Frequency.vi` に実装されています。

DLL フレキシブルプロトタイプアダプタを使用して LabVIEW DLL 関数をデバッグする

この練習では、DLL フレキシブルプロトタイプアダプタを使用して呼び出す LabVIEW DLL テストモジュールの作成方法を学習します。また、LabVIEW で開発した TestStand オペレータインタフェースを使用してモジュールをデバッグする方法も説明します。この練習を行うには、LabVIEW バージョン 6*i* 以降が必要で、LabVIEW 開発環境の使用経験があることを前提としています。

LabVIEW を使用していなくて、LabWindows/CVI を使用しているか C スタイルの DLL を作成する場合は、このセクションをスキップして本章の「C/CVI 標準プロトタイプアダプタを使用して LabWindows/CVI DLL をデバッグする」のセクションに進んでけっこうです。

仮想計測器コードを作成する

この練習では、LabVIEW VI を完成させ、それを使用して、TestStand のシーケンスから呼び出す DLL コードモジュールを構築します。この DLL 関数は、数値と TestStand に渡す追加のレポートテキストを入力するようプロンプトします。

1. スタートメニューから**プログラム→ National Instruments → LabVIEW 6 → LabVIEW** を選択して、LabVIEW 開発環境を起動します。
2. LabVIEW で、`<TestStand>\Tutorial\Clock Frequency Function.vi` を開きます。この VI の未完成のダイアグラムに切り替えます。

   この練習には、現在の TestStand の実行状態を監視するコードが含まれています。ダイアログボックスの表示など、ステップの中でタスクを実行する際は、実行の状況を監視する必要があります。実行が終了または停止されているときは、VI 内で実行しているタスクを中止する必要があります。この機能は、TestStand 関数パレットにある次の VI を使用して、LabVIEW で実行されます。`InitializeTerminationMonitor.vi`、`GetMonitorStatus.vi`、および `CloseTerminationMonitor.vi`。
3. VI のフロントパネルに切り替えて、図 6-10 のように完成させます。右側のシーケンスコンテキスト制御器および表示器は、コードモジュールへのデータの受け渡しに使用します。LabVIEW 標準プロトタイプアダプタと異なり、DLL フレキシブルプロトタイプアダプタを使用する際はコードモジュールのパラメータを定義する必要があります。

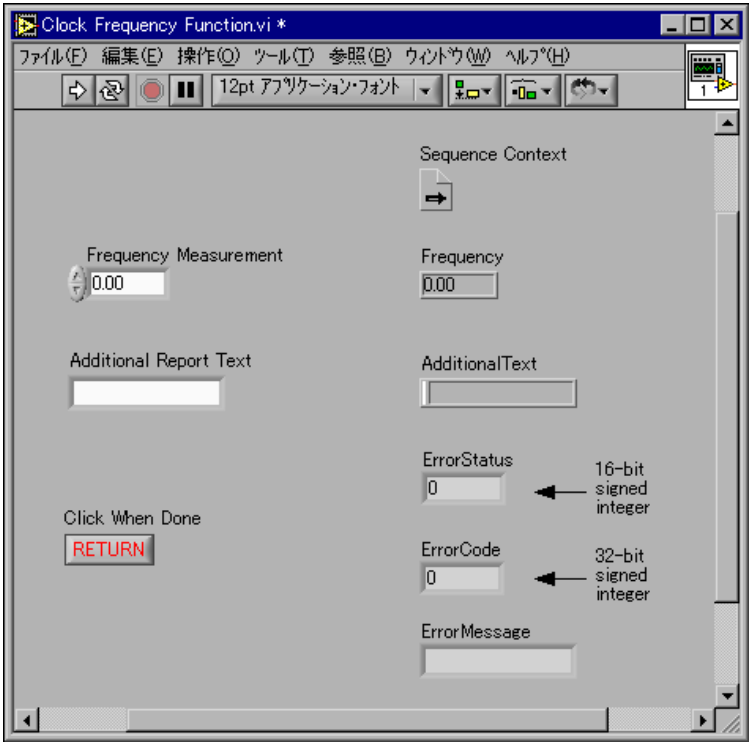

図 6-10　Clock Frequency 関数のフロントパネル

4. LabVIEW で**ファイル→別名で保存**を選択して、VI を `Clock Frequency Function.vi` という名前で保存します。
5. フロントパネルの右部分にある制御器と 5 つの表示器を VI のコネクタペーンに配線する必要があります。フロントパネルの右上コーナーにある VI のアイコンを右クリックして、**コネクタを表示**を選択します。配線ツールを使用して、図 6-10 に示すように制御器と各表示器に端子を割り当てます。コネクタの配列と順序は重要ではありません。
6. VI のダイアグラムを図 6-11 のように完成させます。オペレータが Return ボタンをクリックしたり、VI 内でエラーが発生したり、現在の TestStand の実行が終了または停止すると、While ループが停止することに注意してください。周波数測定、追加のレポートテキスト、およびエラー情報は、DLL 関数のパラメータとして TestStand に返されます。

メモ　関数パラメータのうちの 1 つがブールの場合、現在のところ LabVIEW は DLL タイプライブラリを作成しません。タイプライブラリを使用すると、TestStand で DLL 関数のパラメータ情報を取得できます。関数の呼び出しを簡易化するため、図 6-11 に示すようにブールのエラーステータスを 16 ビット整数に変換します。この整数をエラーステータス関数パラメータとして返します。

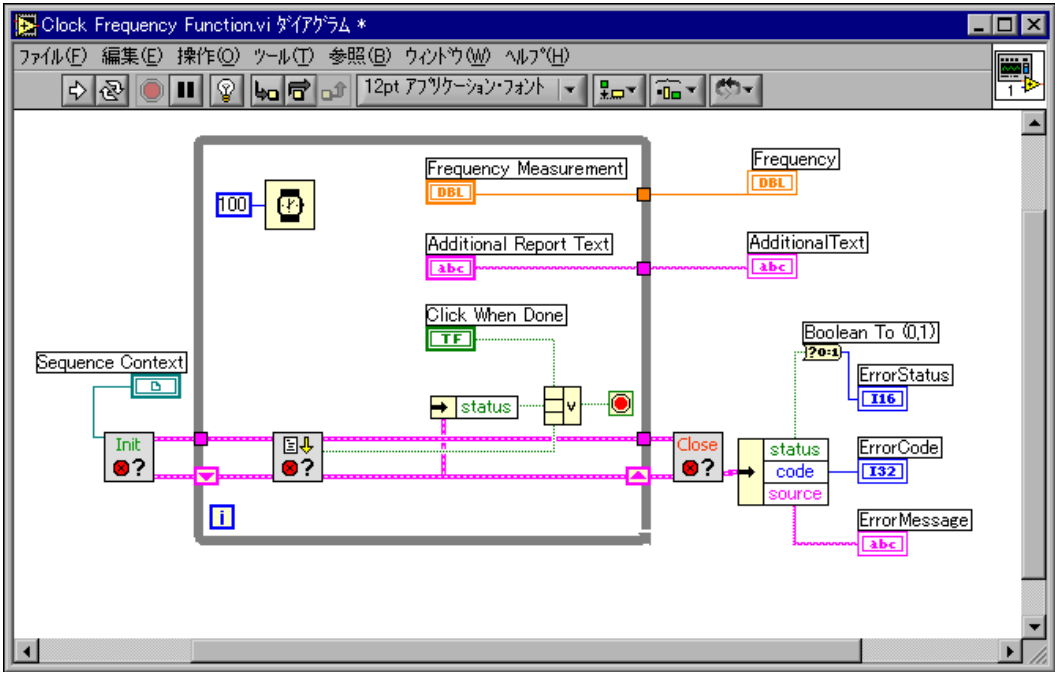

図 6-11　Clock Frequency 関数ダイアグラム

7. VI を保存します。
8. VI が呼び出されたときにフロントパネルを開くように VI 設定を構成します。ダイアグラムウィンドウの右上コーナーにある VI のアイコンを右クリックして、**VI プロパティ**を選択します。
9. VI プロパティダイアログボックスで、図 6-12 に示すように、「カテゴリ」リング制御器から「ウィンドウの外観」を選択します。**カスタマイズする**ボタンをクリックします。

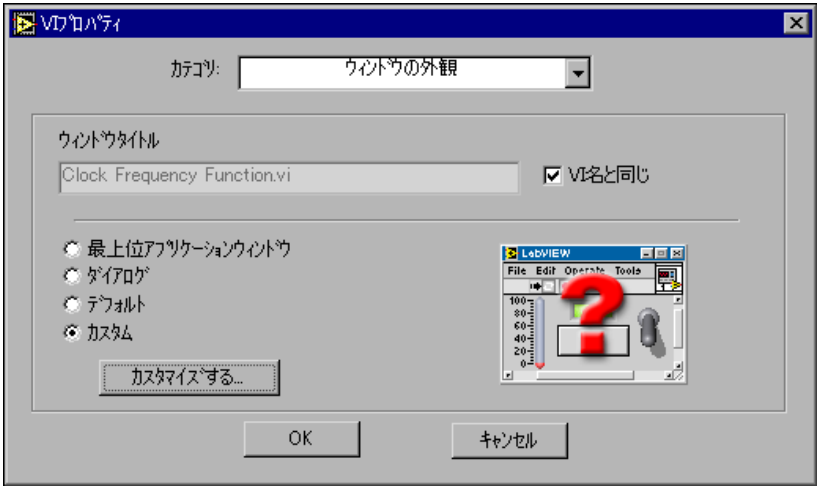

図 6-12　VI プロパティダイアログボックス

10. 「ウィンドウの外観をカスタマイズする」ダイアログボックスで、**呼び出されたらフロントパネルを表示する**と**元々閉じていたら閉じる**のオプションにチェックを付けます。**OK** を 2 回クリックして、ダイアグラムウィンドウに戻ります。
11. VI を保存します。

LabVIEW DLL コードモジュールを構築する

`Clock Frequency Function.vi` の構築が完了しましたので、それを DLL 関数に組み込む必要があります。それをする前に、VI を閉じます。

1. LabVIEW では、新規 VI を開いて**ツール→アプリケーションまたは共有ライブラリ (DLL) を作成**を選択します。
2. 「アプリケーションまたは共有ライブラリ（DLL）を作成」ダイアログボックスの**ロード**ボタンをクリックして、ビルドスクリプト `<TestStand>\Tutorial\Clock Frequency.bld` をロードします。ターゲットタブのビルドターゲット制御器が共有ライブラリ（DLL）に設定されていることを確認します。
3. ソースファイルタブを選択します。エクスポートされた VI は `Clock Frequency Function.vi` になっているはずです。このエントリをハイライトして**VI プロトタイプを定義**ボタンをクリックします。
4. 関数パラメータを追加して構成する必要があります。"+" マークの付いた追加ボタンを、VI コネクタペーンに配線した制御器と表示器それぞれについて 1 回ずつ、全部で 6 回クリックします。パラメータ制御器には、図 6-13 のように 9 つのパラメータが表示されているはずです。パラメータはこの順序になってなくてもかまいません。

LabVIEW では、この VI の 2 つの文字列出力パラメータ用に len および len2 パラメータが必要です。これらのパラメータにより、DLL 関数から最大文字列長が返されるよう LabVIEW に指示されます。

図 6-13　「VI プロトタイプを定義」ダイアログボックス

パラメータ制御器にこれらの 9 つの項目が表示されていない場合は、制御器と表示器が VI のコネクタペーンに正しく配線されていません。続行する前に VI のコネクタペーンに制御器と表示器を配線し直してください。

5. 上下矢印を使って、図 6-13 のパラメータと順序が一致するようにパラメータを並べ替えます。パラメータは、シーケンスから呼び出されたのと同じ順序になっていなければなりません。
6. **標準呼び出し規約**ラジオボタンを選択します。
7. **OK** をクリックして「VI プロトタイプを定義」ダイアログボックスを閉じます。
8. 「アプリケーションまたは共有ライブラリ（DLL）を作成」ダイアログボックスで**作成**ボタンをクリックします。作成が完了したら、すべての LabVIEW ダイアログボックスを閉じてシーケンスエディタに戻ります。

メモ　DLL 作成の際に LabVIEW がファイル許可エラー（エラー 8）を返した場合は、シーケンスエディタに戻って**ファイル→すべてのモジュールの解放**を選択します。すると TestStand は、DLL、VI、およびアダプタがロードした他のすべてのモジュールを含む、すべてのステップのコードモジュールを解放します。LabVIEW に戻って DLL を再構築します。

LabVIEW での DLL の作成に関する詳細については、LabVIEW のドキュメントを参照してください。

LabVIEW DLL 関数を呼び出す

LabVIEW DLL 関数が作成されましたので、この関数を呼び出すステップを作成します。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択して、第 5 章、「変数およびプロパティを使用する」で作成した `<TestStand>\Tutorial\ Sample4.seq` ファイルを開きます。このファイルは、`<TestStand>\Tutorial\Solution` ディレクトリにもあります。
3. アダプタ選択リング制御器で、DLL フレキシブルプロトタイプアダプタを選択します。
4. この練習または前の練習を終了している場合は、Clock Frequency Test という名前のステップがすでに存在する可能性があります。そのステップを削除します。
5. メインステップグループの Power On Test を右クリックして、コンテキストメニューから **ステップを挿入→テスト→数値リミットテスト**を選択します。
6. 新しいステップに Clock Frequency Test という名前を付けます。
7. **ファイル→別名で保存**を選択して、シーケンスを `Sample6.seq` という名前で `<TestStand>\Tutorial` ディレクトリに保存します。
8. Clock Frequency Test ステップを右クリックして、コンテキストメニューからモジュールを指定コマンドを選択します。シーケンスエディタは「DLL コールを編集」ダイアログボックスを表示します。
9. モジュールタブで、DLL パス名制御器の隣の**参照**をクリックします。
10. 作成した `ClockFrequency.dll` を選択します。
11. 呼び出し規約制御器が標準コール（`stdcall`）に設定されていることを確認してください。
12. 関数名リング制御器で `ClockFrequencyFunction` を選択します。TestStand は DLL タイプライブラリからパラメータを読み取って、図 6-14 に示すように関数プロトタイプを表示します。

メモ　その関数では DLL にパラメータ情報がないという内容のメッセージが表示されることがあります。関数パラメータのうちの 1 つがブールの場合、現在のところ LabVIEW は DLL タイプライブラリを作成しません。LabVIEW VI を変更して DLL を再構築するか各パラメータを手作業で入力します。パラメータを手作業で入力する場合は、DLL を構築したときに LabVIEW で作成した関数プロトタイプに一致していることを確認してください。

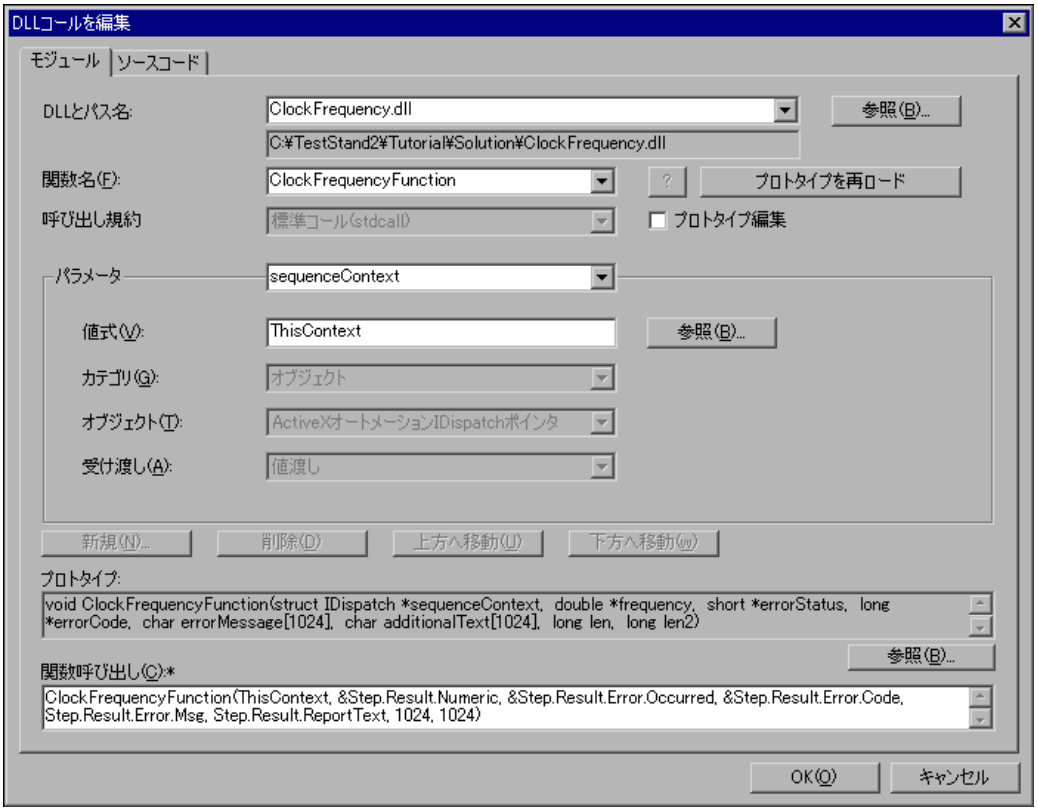

図 6-14　「DLL コールを編集」ダイアログボックス

13. 8 つの関数パラメータに対して下記のとおりに値の式を入力します。

パラメータ名	値式
sequenceContext	ThisContext
Frequency	Step.Result.Numeric
errorStatus	Step.Result.Error.Ocurred
errorCode	Step.Result.Error.Code
errorMessage	Step.Result.Error.Msg
additionalText	Step.Result.ReportText
len	1024
len2	1024

メモ　プロトタイプ編集制御器がチェックされていない場合は、新規パラメータを追加したり既存のパラメータを編集することができません。この制御器は、OK ボタンをクリックして「DLL コールを編集」ダイアログボックスを終了すると、自動的にチェックされます。

パラメータとその式の追加が終了したら、関数呼び出しは以下の値と一致している必要があります。

```
ClockFrequencyFunction(ThisContext、
    &Step.Result.Numeric、
    &Step.Result.Error.Occurred、
    &Step.Result.Error.Code、
    Step.Result.Error.Msg、Step.Result.ReportText、
    1024、1024)
```

14. **OK** をクリックして「DLL コールを編集」ダイアログボックスを閉じます。
15. メインのシーケンスエディタウィンドウで、**ファイル→保存**を選択してシーケンスファイルの変更を保存します。
16. シーケンスエディタウィンドウで、Clock Frequency Test ステップを右クリックしてコンテキストメニューから**プロパティ**を選択します。実行オプションタブで、TestStand ウィンドウアクティブ化制御器から**始めアクティブの場合、ステップが完了したとき再びアクティブ**を選択します。
17. **OK** ボタンをクリックして Clock Frequency Test プロパティダイアログボックスを閉じます。
18. シーケンスファイルウィンドウで Clock Frequency Test ステップを右クリックして、コンテキストメニューから**リミットの編集**を選択

します。「数値リミットテストの編集」ダイアログボックスが表示されます。

19. 図 6-15 に示すように、比較タイプ制御器を `LT (<)` に、値を `100` に設定します。
20. このステップはマザーボードのクロック周波数測定をシミュレーションしますので、単位を `MHz` に変更します。単位制御器の隣のドロップダウンリングを使用して、単位には `Hertz` を、単位の接頭辞には `Mega` を選択します。すると単位制御器には、`megahertz` という値が表示されます。短縮形の名前を使用するには、各リングで短縮名を選択します。これにより単位制御器には、`MHz` という値が表示されます。指定した単位は、レポートと結果のデータベースに表示されます。単位と単位の接頭辞は表示およびドキュメント用のみのもので、測定値をスケールしたりリミットの比較に影響を与えることはありません。

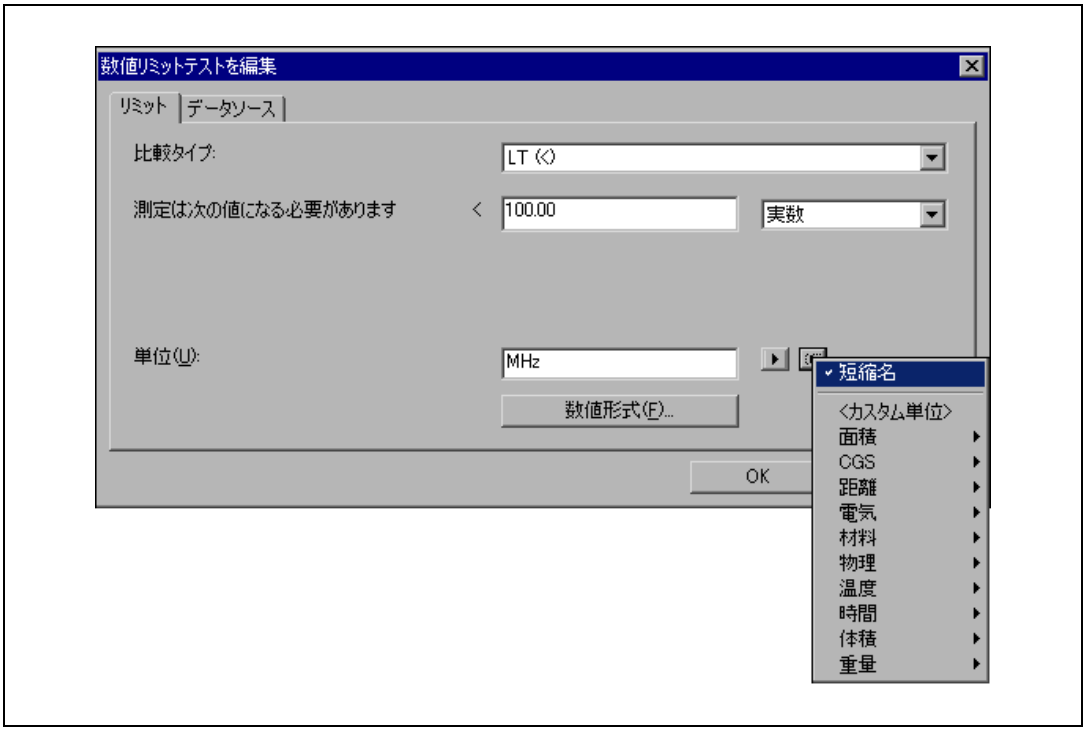

図 6-15　「数値リミットテストを編集」ダイアログボックス

21. **数値形式**ボタンをクリックして数値形式ダイアログボックスを開きます。このダイアログボックスの制御器の値を、図 6-16 に示すように設定します。これらの設定により、ステップの測定とリミットの値が指定されます。形式は、「数値リミットテストを編集」ダイアログ

ボックス、ステップの説明、およびテストレポートに表示されるリミット値に適用されます。

この設定で、TestStand は VI が返す数値測定値を定数 100 と比較します。比較が `True` の場合、ステップは合格になります。それ以外の場合は不合格です。

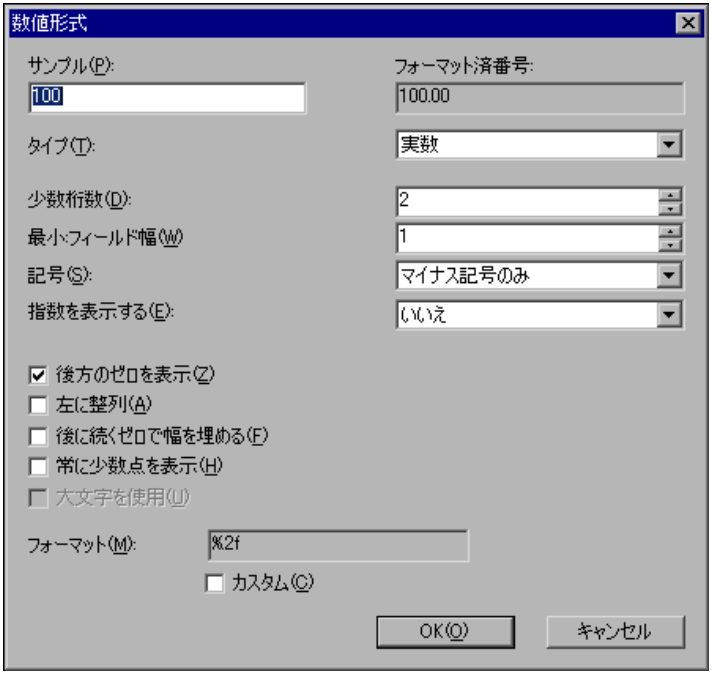

図 6-16　数値形式ダイアログボックス

22. **OK** を 2 回クリックして、数値形式ダイアログボックスと数値リミットテストを編集ダイアログボックスを閉じます。
23. **ファイル→保存**を選択して、`Sample6.seq` を保存します。
24. **実行→一回実行**を選択して、シーケンスを実行します。
25. Test Simulator ダイアログボックスの **Done** をクリックします。TestStand が Clock Frequency Test ステップを実行すると、コードモジュールはユーザインタフェースパネルを表示して入力を待ちます。
26. 周波数測定制御器に 20 という数値を入力します。
27. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
28. **Return** ボタンをクリックしてシーケンスの実行を続行します。

29. シーケンスの実行が完了したら、テストレポートの内容を確認します。Clock Frequency Test ステップのステータス、測定値、およびレポートテキスト値を確認してください。
30. 実行ウィンドウを閉じます。

DLL 関数をデバッグする

現在のところ、LabVIEW DLL 関数をデバッグするには、LabVIEW 開発環境の中から DLL を呼び出す必要があります。そのためには、LabVIEW オペレータインタフェースを使用します。

1. 以前に DLL 関数内に組み込んだ `<TestStand>\Tutorial\Clock Frequency Function.vi` のファイルを開きます。
2. この VI のダイアグラムにブレークポイントを設定します。
3. **スタート→プログラム→ National Instruments → TestStand → Operator Interfaces → LabVIEW** を選択するか、`<TestStand>\OperatorInterfaces\NI\LV TestStand - Runtime Operator Interface.vi` を開いて実行することにより、LabVIEW 開発環境で LabVIEW オペレータインタフェースを開いて実行します。

メモ　TestStand に付属されている LabVIEW オペレータインタフェースの VI は、LabVIEW バージョン 5.1.1 で作成されています。それより新しいバージョンの LabVIEW を使用している場合は、VI を一括コンパイルして VI のロード時間を削減する必要があります。

4. ログインした後、オペレータインタフェース内で `<TestStand>\Tutorial\Sample6.seq` を開き、一回実行ボタンをクリックして実行します。

シーケンスのステップが DLL 関数を呼び出すと、LabVIEW は `Clock Frequency Function.vi` で設定したブレークポイントで停止します。標準の LabVIEW デバッグテクニックを使用して VI を実行することができます。VI を実行したら、オペレータインタフェースの実行表示ウィンドウに戻り、レポートでステップの結果を確認します。

C/CVI 標準プロトタイプアダプタを使用して LabWindows/CVI DLL をデバッグする

この練習では、LabWindows/CVI 標準プロトタイプアダプタを使用して呼び出す LabWindows/CVI DLL コードモジュールの作成方法を学習します。また、シーケンスエディタから LabWindows/CVI コードにステップインしてモジュールをデバッグする方法も説明します。このチュー

トリアルセッションでは、LabWindows/CVI 開発環境の使用経験があることを前提としています。LabWindows/CVI を使用していない方で C スタイル DLL を作成する場合は、このセクションをスキップし、「DLL フレキシブルプロトタイプアダプタを使用して LabWindows/CVI DLL をデバッグする」に進んでください。

メモ　TestStand に対応する正しいバージョンの LabWindows/CVI を使用していることを確認してください。詳細については、`TestStand\Doc` ディレクトリの `readme.txt` ファイルを参照してください。

サンプルをセットアップする

第 5 章、「変数およびプロパティを使用する」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択して、第 5 章、「変数およびプロパティを使用する」で作成した `<TestStand>\Tutorial\ Sample4.seq` ファイルを開きます。このファイルは、`<TestStand>\ Tutorial\Solution` ディレクトリにもあります。

C/CVI コードモジュールテストを作成する

この練習では、C/CVI 標準プロトタイプアダプタを使用して呼び出した LabWindows/CVI コードモジュールを作成します。このコードモジュールは、数値を入力するようオペレータにプロンプトし、そのデータを TestStand に返します。

1. LabWindows/CVI 標準プロトタイプアダプタが、以下のとおり LabWindows/CVI の外部インスタンスでコードモジュールを実行するように正しく構成されていることを確認します。
   a. **構成→アダプタ**を選択すると、図 6-17 に示すようなアダプタ構成ダイアログボックスが表示されます。

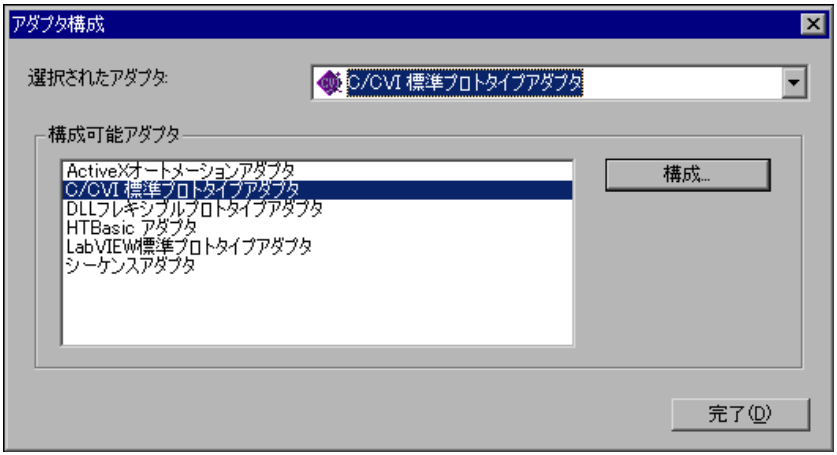

図 6-17　アダプタ構成

   b. 構成可能アダプタセクションで、LabWindows/CVI 標準プロトタイプアダプタを選択します。
   c. **構成**ボタンをクリックします。すると C/CVI 標準アダプタ構成ダイアログボックスが表示されます。

d. 図 6-18 に示すように、「CVI の外部インスタンス内でステップを実行」がチェックされていることと、実行サーバを含む LabWindows/CVI プロジェクトのパス名が `TestStand\AdapterSupport\CVI` ディレクトリの `tscvirun.prj` ファイルであることを確認してください。

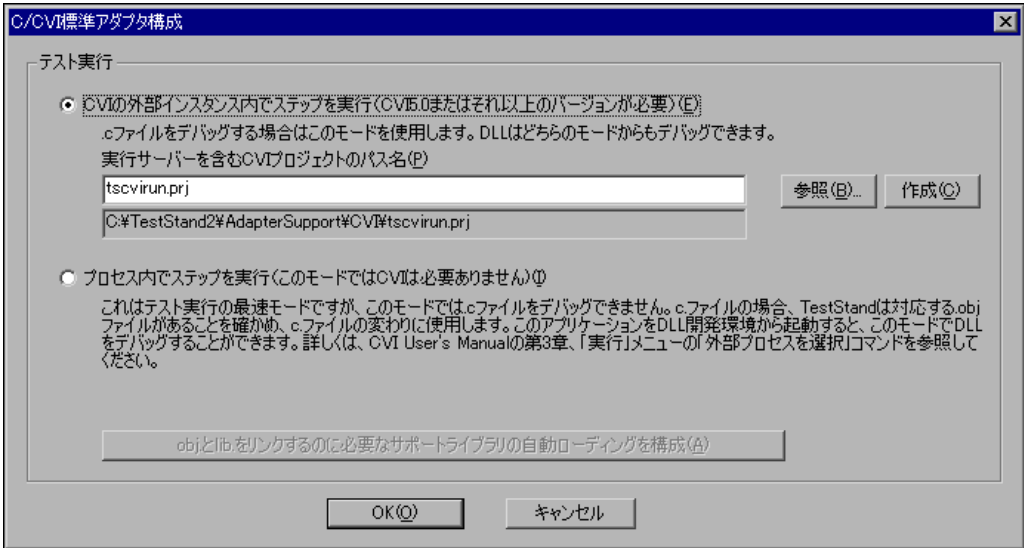

図 6-18　C/CVI 標準アダプタ構成

e. **OK** をクリックして C/CVI 標準アダプタ構成ダイアログボックスを閉じます。TestStand は、テストが実行する場所を変更するとすべてのモジュールがアンロードされることを警告します。

f. **OK** をクリックして警告ダイアログボックスを閉じます。

g. **OK** をクリックしてアダプタ構成ダイアログボックスを閉じます。

2. アダプタ選択リング制御器で C/CVI 標準プロトタイプアダプタが選択されていることを確認します。

3. メインステップグループの Power On Test を右クリックして、コンテキストメニューから **ステップを挿入→テスト→数値リミットテスト**を選択します。

4. そのステップに `Clock Frequency Test` という名前を付けます。

5. Clock Frequency Test を右クリックして、コンテキストメニューから**モジュールを指定**を選択します。すると、シーケンスエディタは「C/CVI モジュールコールを編集」ダイアログボックスを表示します。

6. 「モジュールタイプ」リング制御器で、Dynamic Link Library（`*.dll`）を選択します。
7. モジュールパス名制御器に `frequency.dll` と入力します。
8. 「C/CVI モジュールコールを編集」ダイアログボックスの関数名制御器に、`GetFrequency` と入力します。
9. 「シーケンスコンテキストを渡す」チェックボックスにチェックを付けます。

   図 6-19 は、すべてを入力したモジュールタブを示します。

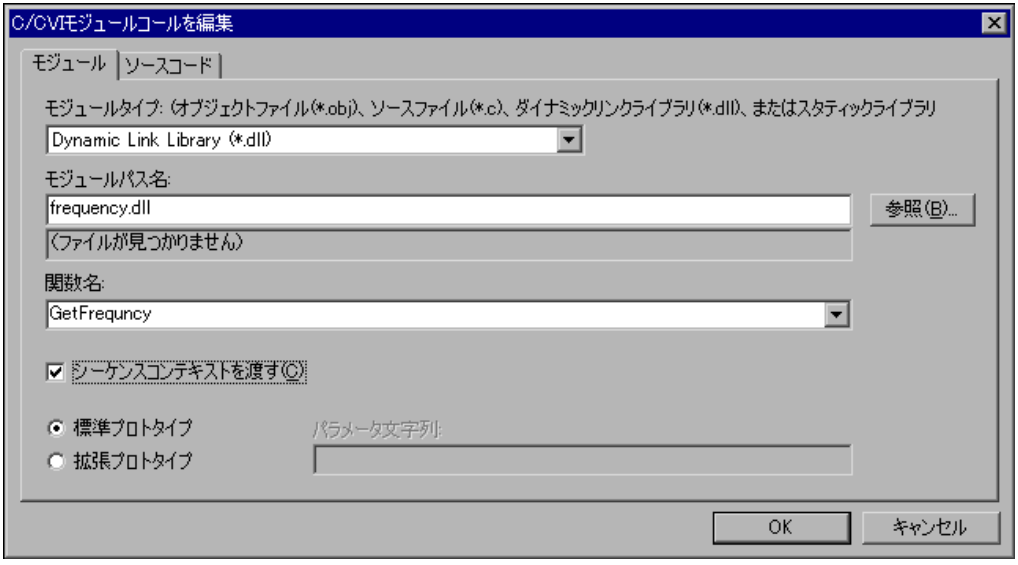

図 6-19　CVI モジュールコールを編集—モジュールタブ

10. ソースコードタブをクリックします。ソースコードタブを使用して、ステップが呼び出す関数のソースコードを生成したり編集したりすることができます。
11. 「関数を含むソースファイルのパス名」制御器の右にある**参照**ボタンをクリックして、`TestStand\Tutorial` ディレクトリの `frequency.c` ファイルを選択します。
12. **OK** ボタンをクリックして、「ソースファイルのパス名を選択」ダイアログボックスを閉じます。
13. 「開く CVI プロジェクトのパス名」制御器の右にある**参照**ボタンをクリックして、`TestStand\Tutorial` ディレクトリの `frequency.prj` ファイルを選択します。

14. **OK** ボタンをクリックして、「CVI プロジェクトファイルのパス名を選択」ダイアログボックスを閉じます。

    図 6-20 は、すべてを入力した「C/CVI モジュールコールを編集」ダイアログボックスのソースコードタブを示します。

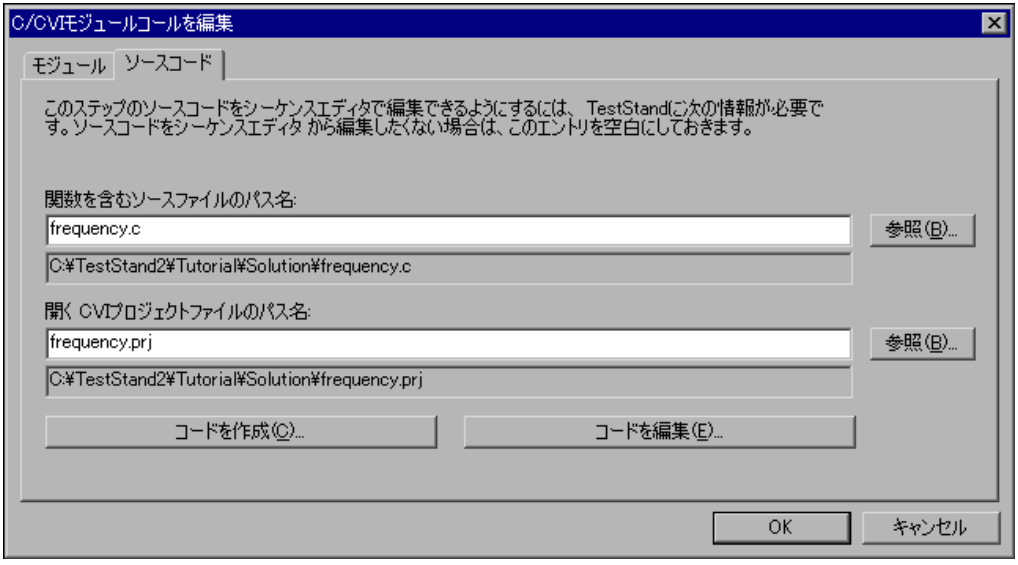

図 6-20　CVI モジュールコールを編集—ソースコードタブ

モジュールアダプタは、あらかじめ定義されたテンプレートを使用してステップモジュールのソースコードシェルを生成することができます。使用可能なテンプレートは、ステップタイプと各モジュールアダプタにより異なります。ソースコードテンプレートをサポートする各モジュールアダプタに対し、「モジュールを指定」ダイアログボックスにはソースコード作成用のコマンドボタンがあります。ステップタイプと特定のアダプタに複数のテンプレートが関連している場合、アダプタはテンプレートを選択するようオペレータにプロンプトし、それ以外の場合はデフォルトテンプレートからコードモジュールを作成します。

15. **コードを作成**ボタンをクリックします。

    それにより TestStand は以下の動作を実行します。

    1. LabWindows/CVI の外部インスタンスを起動。
    2. ない場合は新規のプロジェクトファイルを作成。
    3. LabWindows/CVI でプロジェクトファイルをロードおよび保存。
    4. ない場合は新規ソースファイルを作成して保存。

5. テンプレート関数とともにソースファイルを生成して保存。
6. LabWindows/CVI が関数名をハイライト。

16. 図 6-21 は、`frequency.c` ソースファイルで生成された関数を示します。

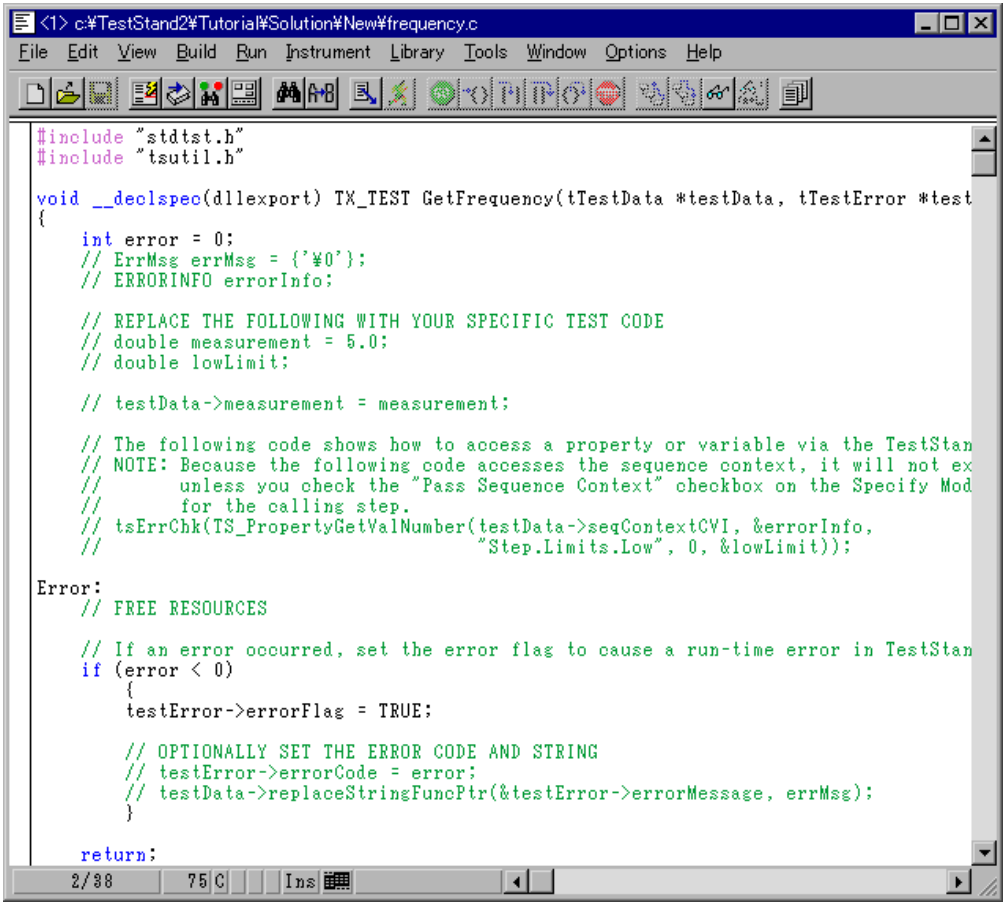

図 6-21　「コードを作成」コマンドから生成された結果

メモ　このチュートリアルセッションを他の人が以前に完了している場合、ソースファイル内に `GetFrequency` 関数がすでに存在することがあります。LabWindows/CVI が既存の関数を入れ換えるようプロンプトした場合は、**入れ換え**ボタンをクリックしてこのチュートリアルセッションを続行します。

GetFrequency プロトタイプ関数には、tTestData と tTestError の 2 つのパラメータがあります。LabWindows/CVI 標準プロトタイプアダプタは、これらのパラメータを使用して TestStand とコードモジュールの間で共通のデータを受け渡します。それらの 2 つのストラクチャ内の要素と、アダプタによるそれらの使用方法を以下に示します。

tTestData

- result：テストの合否を示すためにテスト関数により設定されます。
- measurement：テスト関数が返す数値測定値。
- inBuffer：文字列パラメータをテスト関数に渡すのに使用します。
- outBuffer：レポートに表示する出力メッセージ。
- modPath：テスト関数を含むモジュールのディレクトリパス。
- modFile：テスト関数を含むモジュールのファイル名。
- hook：保留（使用されていません）。
- hookSize：保留（使用されていません）。
- mallocFuncPtr：malloc への関数ポインタを含みます。これは、inBuffer、outBuffer、および errorMessage の各フィールドに割り当てるすべてのバッファに対してメモリを確保するためにコードモジュールが使用するものです。
- freeFuncPtr：free への関数ポインタを含みます。これは、inBuffer、outBuffer、および errorMessage の各フィールドが示すすべてのバッファを解放するためにコードモジュールが使用するものです。
- seqContextDisp：シーケンスコンテキストへのディスパッチポインタ。
- seqContextCVI：シーケンスコンテキストの CVI ActiveX ハンドル。
- stringMeasurement：テスト関数が返す文字列値。
- structVersion：ストラクチャのバージョン番号。
- replaceStringFuncPtr：ReplaceString への関数ポインタを含みます。これは、inBuffer、outBuffer、および errorMessage の各フィールドが示すすべてのバッファの値を変更するためにコードモジュールが使用するものです。

tErrorData

- errorFlag：エラーが発生するとテスト関数はこれを True に設定します。
- errorLocation：保留（使用されていません）。

- errorCode：エラーが発生すると、テスト関数はこれを 0 以外の値に設定することがあります。
- errorMessage：エラーが発生すると、テスト関数はこれを記述的な文字列に設定することがあります。

これらのストラクチャに関する詳細は、『TestStand User Manual』の Chapter 12、「Module Adapters」を参照してください。

17. 以下のように、Frequency.c コードを更新して、frequency.uir ユーザインタフェースのリソースファイルを使用して周波数数値制御器およびレポートテキスト文字列制御器に値を入力するようオペレータにプロンプトします。
    a. Frequency.c ウィンドウを開いたままで、**Windows → Project** を選択してプロジェクトウィンドウに戻ります。
    b. プロジェクトウィンドウから Frequency.uir ファイルを開きます。
    c. **Return** ボタンを右クリックして、コンテキストメニューから **Generate Control Callback** を選択します。複数の .c ファイルが開いていると、LabWindows/CVI はコールバックを挿入するターゲットファイルを選択するようプロンプトします。ターゲットファイルに frequency.c を選択します。
    d. これで Frequency.c に ReturnCallback 関数が作成されました。ReturnCallback 関数はユーザインタフェースを終了します。終了したら UIR ファイルを閉じます。変更は行いません。
    e. GetFrequency および ReturnCallback 関数のソースコードを下記のとおりに更新します。ソースファイルの他の関数は変更の必要はありません。変更された行は太字で示しています。

```
void __declspec(dllexport) TX_TEST
    GetFrequency(tTestData   *testData, tTestError
    *testError)
{
    int error = 0;
    int panelHandle, panel, control;
    char stringBuffer[512];
    panelHandle = LoadPanelEx (0, "frequency.uir",
        PANEL, __CVIUserHInst);
    if (panelHandle < 0)
    {
    error = panelHandle;
    goto Error;
    }
    DisplayPanel (panelHandle);
    //Automatically closes user display dialog when
    //TestStand execution terminates or aborts.
```

```
    TS CancelDialogIfExecutionStops
      (panelHandle,testData->seqContextCVI);
    RunUserInterface();
    // Assign values from UIR to return data structure
    GetCtrlVal (panelHandle, PANEL_FREQUENCY,
      &testData->measurement);
    GetCtrlVal (panelHandle, PANEL_ADDITIONAL_REPORT,
      stringBuffer);
    testData->replaceStringFuncPtr(&testData->  outBuff
      er, stringBuffer);
Error:
    // FREE RESOURCES
    DiscardPanel(panelHandle);
    // If an error occurred, set the error flag to
    // cause a run-time error in TestStand.
    if (error < 0)
    {
    testError->errorFlag = TRUE;
    // OPTIONALLY SET THE ERROR CODE AND STRING
        testError->errorCode = error;
        testData->replaceStringFuncPtr(&testError->erro
          rMessage, "A run-time error occurred.");
    }
    return;
}
intCVICALLBACK ReturnCallback (int panelHandle, int
    control, int event, void*callbackData, int
    eventData1, int eventData2
{
    switch(event)
        {
        case EVENT_COMMIT:
        QuitUserInterface(0);
        break;
    }
return 0;
}
```

18. **Build → Compile File** を選択して `Frequency.c` ソースコードをコンパイルし、正しく変更されていることを確認します。

メモ　ダイアログボックスの表示など、ステップモジュール内でタスクを実行する際は、ステップモジュールが呼び出されている TestStand の実行の状況をモニタしてください。実行が終了または中止した場合、ステップモジュール内で実行しているタスクを中止する必要があります。実行が停止したり中止したときに自動的に閉じるようにするダイアログボックスを表示する `TS_CancelDialogIfExecutionStops()` 関数をステップモジュールで使用します。この関数では、`RunUserInterface` への呼び出しによってプログラムがダイアログボックスを制御していることを前提としています。

19. コンパイルが正しく行われたらソースコードを保存します。
20. プロジェクトウィンドウで **Build → Create Dynamic Link Library** を選択して、DLL を再構築します。DLL がすでに存在する場合は、既存のコピーを上書きします。

メモ　DLL 作成の際に LabWindows/CVI がファイル許可エラーを返した場合は、シーケンスエディタに戻ってファイルメニューから「すべてのモジュールの解放」を選択します。すると TestStand は、DLL、VI、およびアダプタがロードした他のすべてのモジュールを含む、すべてのステップのコードモジュールを解放します。LabWindows/CVI に戻って DLL を再構築します。

21. `frequency.prj` DLL プロジェクトを含む LabWindows/CVI の外部インスタンスを閉じます。
22. TestStand シーケンスエディタで、**OK** ボタンをクリックして「C/CVI モジュールコールを編集」ダイアログボックスを閉じます。
23. シーケンスエディタウィンドウで、Clock Frequency Test ステップを右クリックしてコンテキストメニューから**リミットの編集**を選択します。

24. 図 6-22 に示すように、「数値リミットテストを編集」ダイアログボックスで、比較タイプを LT (<)、つまり小なりに、測定値を 100 に設定します。

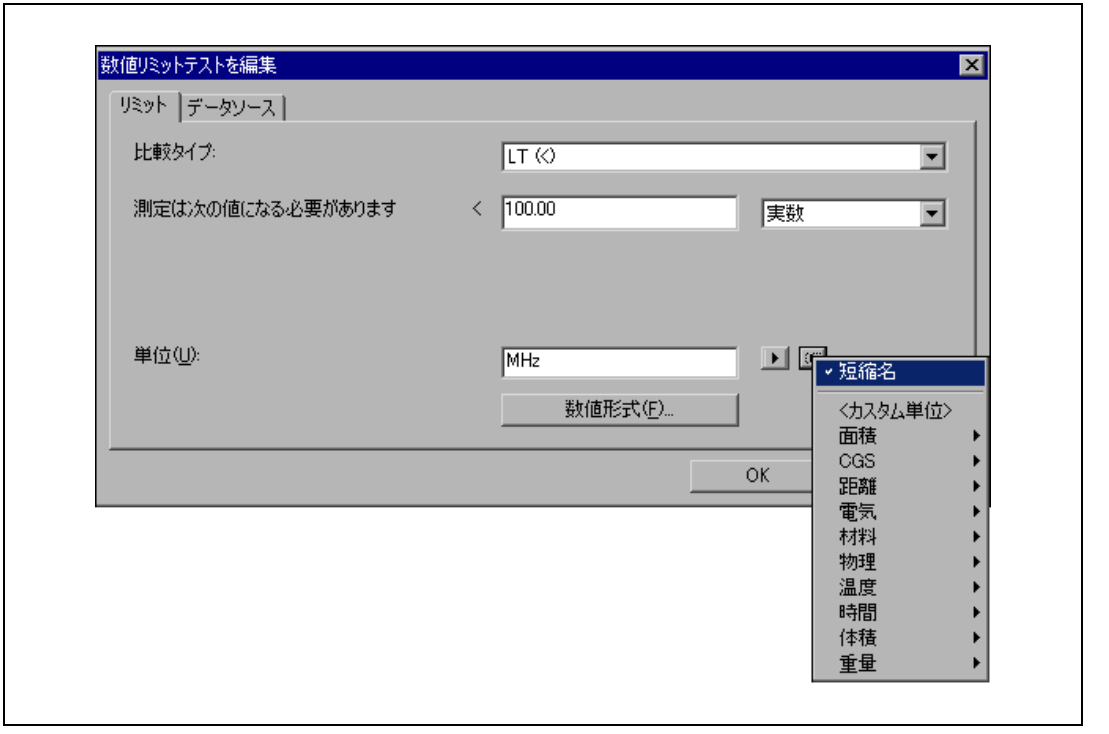

図 6-22　「数値リミットテストを編集」ダイアログボックス

25. このステップはマザーボードのクロック周波数測定をシミュレーションしますので、単位を MHz に変更します。単位制御器の隣のドロップダウンリングを使用して、単位には Hertz を、単位の接頭辞には Mega を選択します。すると単位制御器には、megahertz という値が表示されます。短縮形の名前を使用するには、各リングで短縮名を選択します。これにより単位制御器には、MHz という値が表示されます。指定した単位は、レポートと結果のデータベースに表示されます。単位と単位の接頭辞は表示およびドキュメント用のみのもので、測定値をスケールしたりリミットの比較に影響を与えることはありません。
26. **数値形式**ボタンをクリックして数値形式ダイアログボックスを開きます。このダイアログボックスの制御器の値を、図 6-23 に示すように設定します。これらの設定により、ステップの測定とリミットの値が指定されます。形式は、「数値リミットテストを編集」ダイアログ

ボックス、ステップの説明、およびテストレポートに表示されるリミット値に適用されます。

この設定で、TestStand は VI が返す数値測定値を定数 100 と比較します。比較が `True` の場合、ステップは合格になります。それ以外の場合は不合格です。

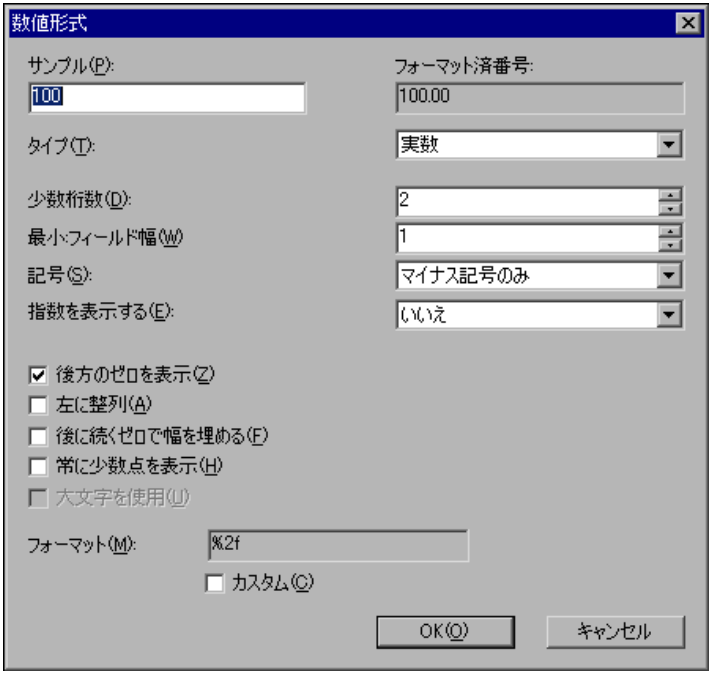

図 6-23　数値形式ダイアログボックス

27. **OK** を 2 回クリックして、数値形式ダイアログボックスと数値リミットテストを編集ダイアログボックスを閉じます。
28. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample7.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。
29. **実行→一回実行**を選択してシーケンスを実行します。TestStand は、LabWindows/CVI の新規外部インスタンスを起動して、ステップを実行します。
30. Test Simulator ダイアログボックスの **Done** をクリックします。

    TestStand が Clock Frequency Test ステップを実行すると、コードモジュールはユーザインタフェースパネルを表示して入力を待ちます。
31. 周波数測定制御器に `20` という数値を入力します。

32. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
33. **Return** ボタンをクリックしてシーケンスの実行を続行します。
34. シーケンスの実行が完了したら、テストレポートの内容を確認します。Clock Frequency ステップのステータス、測定値、およびレポートテキスト値を確認してください。
35. 実行ウィンドウを閉じます。

CVI コードモジュールをデバッグする

TestStand では、シーケンスをデバッグできるだけでなく、デバッグ可能な LabWindows/CVI コードモジュールに直接ステップインすることもできます。この練習では、シーケンスエディタでシーケンスを実行しながら LabWindows/CVI コードモジュールをデバッグする方法を学習します。

1. ステップ名を右クリックして**ブレークポイントのトグル**を選択し、Clock Frequency Test ステップにブレークポイントを設定します。シーケンスウィンドウでステップ名の左に停止サインのアイコンが表示されている場合は、ブレークポイントが有効になっていることを示します。
2. **実行→一回実行**を選択してシーケンスを実行します。
3. Test Simulator プロンプトで **Done** をクリックします。すると実行は Clock Frequency Test ステップで一時停止します。
4. **ステップイン**ツールバーボタンをクリックします。すると、図 6-24 に示すように、LabWindows/CVI の外部インスタンスが起動され、`GetFrequency` 関数でブレークポイント状態に入ります。

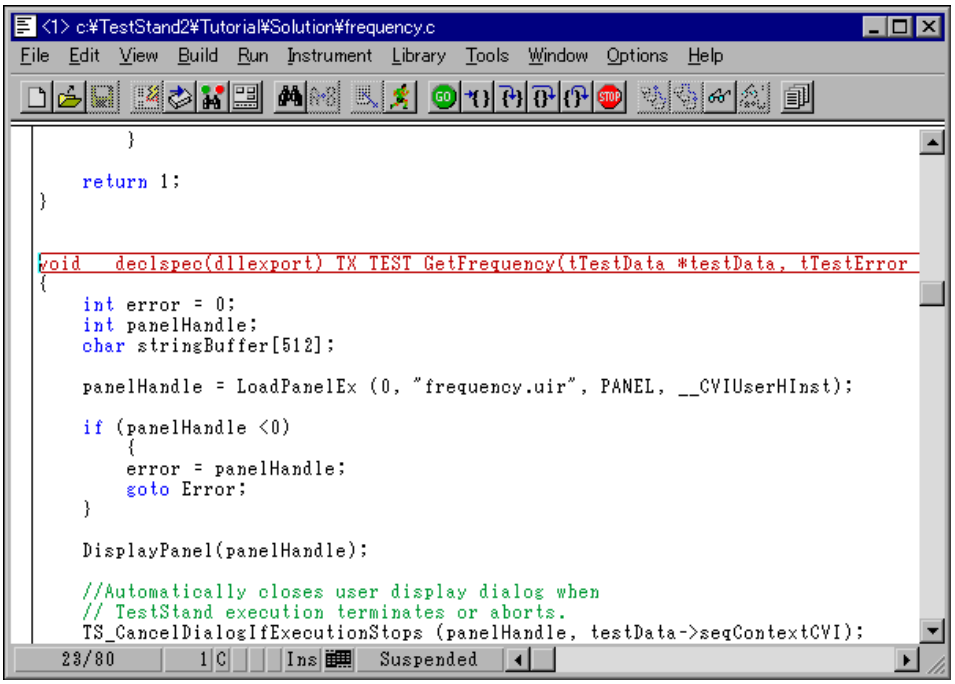

図 6-24　GetFrequency 関数にステップインする

5. LabWindows/CVI のメニューから **Run → Step Over** コマンドを使用し、コードモジュールをステップ処理します。

メモ　デバッグする間 LabWindows/CVI デバッグウィンドウは前面に表示されたままになります。Clock Frequency ダイアログボックスに切り替えて、制御器に値を入力します。

6. ユーザインタフェースリソースが表示されたら、Frequency Measurement 制御器に 200 を入力します。
7. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
8. **Return** ボタンをクリックします。
9. **Run → Finish Function** を選択してシーケンスの実行に戻り、関数を終了します。
10. Clock Frequency Test ステップの実行後、TestStand は Reset Loop Index ステップでシーケンスの実行を中断します。Clock Frequency Test ステップのステータスが意図したとおり不合格になっていることを確認してください。
11. **デバッグ→再開**を選択してシーケンスの実行を完了します。
12. 実行ウィンドウとシーケンスファイルウィンドウを閉じます。

13. TestStand がシーケンスファイルを保存するようプロンプトされたら、シーケンスを `Sample7.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。

DLL フレキシブルプロトタイプアダプタを使用して LabWindows/CVI DLL をデバッグする

この練習では、LabWindows/CVI DLL コードモジュールを作成して、DLL フレキシブルプロトタイプアダプタを使用してそれを呼び出します。また、LabWindows/CVI 開発環境からシーケンスエディタを起動して、このモジュールをデバッグする方法も説明します。

この練習では LabWindows/CVI を使用してコードモジュールの作成やデバッグを行いますが、DLL フレキシブルプロトタイプアダプタおよびデバッグテクニックに関する情報は、他のアプリケーション開発環境で作成した C スタイルの DLL を呼び出す際にも適用可能です。LabWindows/CVI を使用していなくて C スタイルの DLL を作成しない場合は、このセクションをスキップして、第 7 章、「ランタイムオペレータインタフェースを使用する」に進んでください。

サンプルをセットアップする

第 5 章、「変数およびプロパティを使用する」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタのすべてのウィンドウを閉じます。
2. **ファイル→開く**を選択して、第 5 章、「テストの作成とデバッグ」で作成した `<TestStand>\Tutorial\ Sample2.seq` ファイルを開きます。このファイルは、`<TestStand>\ Tutorial\Solution` ディレクトリにもあります。

LabWindows/CVI コードモジュールを作成する

この練習では、数値を入力するようオペレータにプロンプトしてそのデータを TestStand に返す LabWindows/CVI コードモジュールを作成します。

1. アダプタ選択リング制御器で DLL フレキシブルプロトタイプアダプタが選択されていることを確認します。
2. メインステップグループの Power On Test を右クリックして、コンテキストメニューから **ステップを挿入→テスト→数値リミットテスト**を選択します。
3. そのステップに Clock Frequency Test という名前を付けます。

4. Clock Frequency Test ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。図 6-25 に示すように、「DLL コールを編集」ダイアログボックスが表示されます。

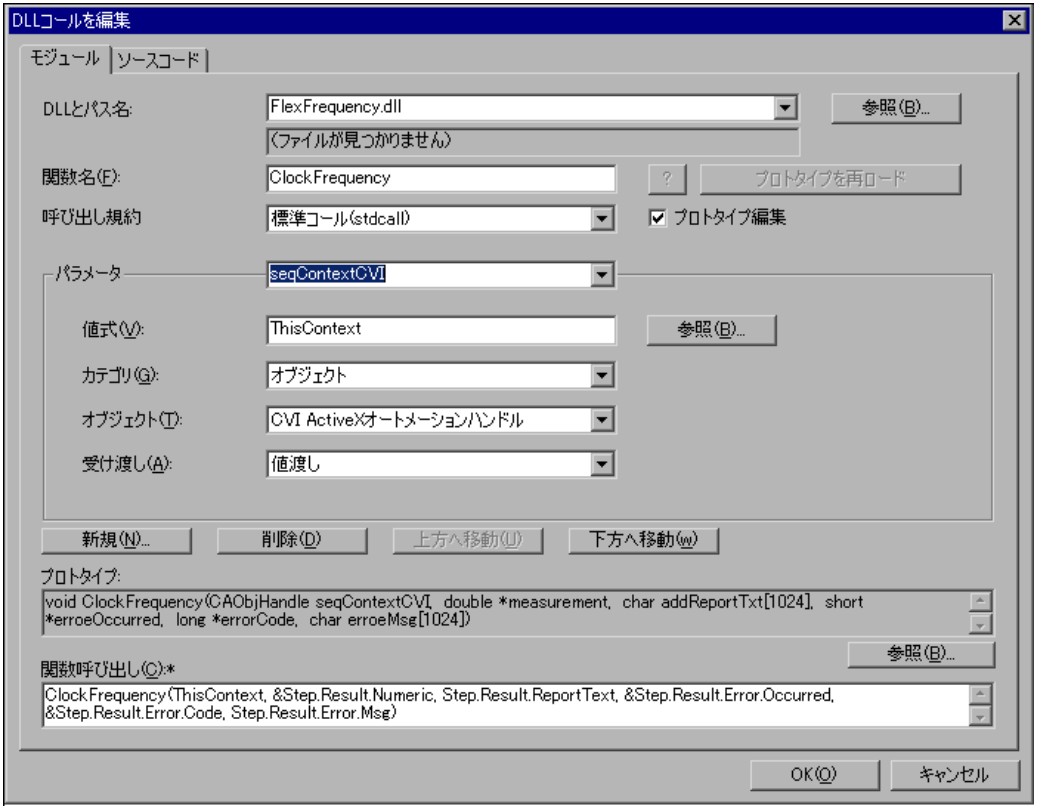

図 6-25　LabWindows/CVI コードモジュールの「DLL コールを編集」ダイアログボックス

5. モジュールタブで、DLL パス名制御器に `FlexFrequency.dll` と入力します。
6. 関数名制御器に `ClockFrequency` と入力します。
7. 呼び出し規約制御器が標準コール（`stdcall`）に設定されていることを確認してください。
8. C/CVI 標準プロトタイプアダプタと異なり、DLL フレキシブルプロトタイプアダプタを使用する際はコードモジュールのパラメータを定義する必要があります。希望のパラメータを入力すると、TestStand が C コードを生成します。**新規**ボタンをクリックして、関数の最初のパラメータの追加を開始します。

9. arg1 パラメータに `seqContextCVI` という名前を付けます。引数の名前は重要ではありません。引数に付ける名前を選択してください。C コード内のパラメータ名として使用されるため、パラメータ名にスペースを使用しないでください。
10. 「カテゴリ」リング制御器で**オブジェクト**を選択します。
11. 「オブジェクトタイプ」リング制御器で **CVI ActiveX Automation Handle** を選択します。
12. 値式制御器に `ThisContext` という式を入力します。
13. 受け渡し制御器を値に設定します。
14. DLL 関数の他の 5 つのパラメータを同様に追加します。各パラメータを追加するには、まず**新規**ボタンをクリックしてください。表 6-1 の値を使用して、各パラメータの制御器を設定します。

メモ　プロトタイプ編集制御器がチェックされていない場合は、新規パラメータを追加したり既存のパラメータを編集することができません。この制御器は、**OK** ボタンをクリックして「DLL コールを編集」ダイアログボックスを終了すると、自動的にチェックされます。

表 6-1　パラメータ制御器値の表

制御器	値
パラメータ名 : 値式 : 結果アクション : カテゴリ : データタイプ : 受け渡し :	measurement Step.Result.Numeric アクションなし 数値 64 ビット実数（倍精度） 参照（ポインタ）渡し
パラメータ名 : 値式 : カテゴリ : 受け渡し : 要素数 :	addReportTxt Step.Result.ReportText 文字列 C 文字列バッファ 1024

表 6-1　パラメータ制御器値の表（続き）

制御器	値
パラメータ名 : 値式 : 結果アクション : カテゴリ : データタイプ : 受け渡し :	errorOccurred Step.Result.Error.Occurred アクションなし 数値 符号付き 16 ビット整数 参照（ポインタ）渡し
パラメータ名 : 値式 : 結果アクション : カテゴリ : データタイプ : 受け渡し :	errorCode Step.Result.Error.Code アクションなし 数値 符号付き 32 ビット整数 参照（ポインタ）渡し
パラメータ名 : 値式 : カテゴリ : 受け渡し : 要素数 :	errorMsg Step.Result.Error.Msg 文字列 C 文字列バッファ 1024

パラメータとその設定の追加が終了したら、自動生成されるプロトタイプ制御器の値は以下の値と一致している必要があります。

```
void ClockFrequency(CAObjHandle seqContextCVI, double
    *measurement, char addReportTxt[1024], short
    *errorOccurred, long *errorCode, char
    errorMsg[1024])
```

さらに、関数呼び出し制御器の値は、以下に示す値と一致する必要があります。

```
ClockFrequency(ThisContext, &Step.Result.Numeric,
    Step.Result.ReportText,
    &Step.Result.Error.Occurred,
    &Step.Result.Error.Code,
    Step.Result.Error.Msg)
```

15. 「DLL コールを編集」ダイアログボックスのソースコードタブを選択して、「関数を含むソースファイルのパス名」に `FlexFrequency.c` と入力します。この制御器のすぐ下にメッセージ（ファイルが指定されていません）が表示されているはずです。ファイルが指定されてい

る場合は、他の人がこの練習を以前に終了しているか、あるいは TestStand 検索ディレクトリの 1 つとして <TestStand>/Tutorial/Solutions ディレクトリが追加されているかのいずれかです。その場合は、別のファイル名を選択します。

16. **コードを作成**ボタンをクリックします。
17. ソースファイルのパス名を選択します。<TestStand>/Tutorial ディレクトリを参照して **OK** ボタンをクリックすると、図 6-26 に示すような「コードテンプレートを選択」ダイアログボックスが表示されます。

図 6-26　「コードテンプレートを選択」ダイアログボックス

18. 最初のオプションの TestStand numeric limit template - NumericLimit_Template を選択して、**OK** ボタンをクリックします。
19. テンプレートは、TestStand に付属されている既存の .c ファイルに基づいています。この .c ファイル内の関数には、先ほど指定したものとは異なるデフォルトパラメータがあります。TestStand は図 6-27 のような「プロトタイプの競合」ダイアログボックスを表示して、パラメータの違いを処理するようプロンプトします。

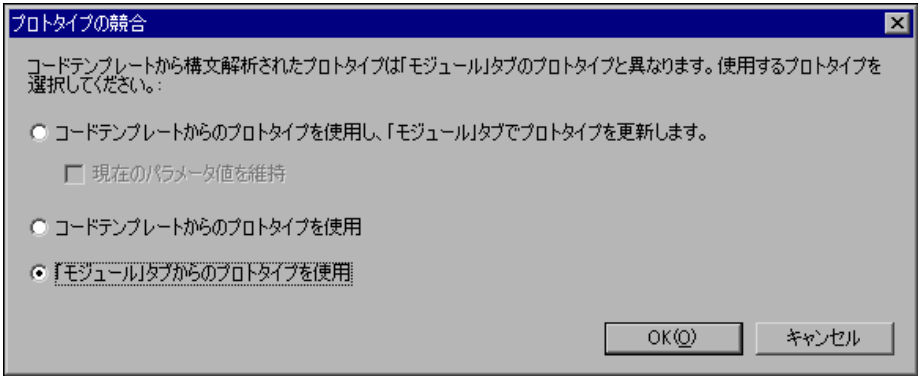

図 6-27　「プロトタイプの競合」ダイアログボックス

20. **「モジュール」タブからのプロトタイプを使用**を選択して、**OK** ボタンをクリックします。
21. TestStand は、`*.c` 拡張子を持つファイルを開くように登録されているシステム内のアプリケーションを使って `.c` ファイルを開くか、あるいはメモ帳を起動して作成した `.c` ファイルを表示するようプロンプトします。`.c` ファイルが表示されたら、シーケンスエディタに戻って **OK** ボタンをクリックし、「DLL コールを編集」ダイアログボックスを閉じます。
22. シーケンスファイルウィンドウで Clock Frequency Test を右クリックして、コンテキストメニューからリミットの編集を選択します。「数値リミットテストの編集」ダイアログボックスが表示されます。
23. 図 6-28 に示すように、比較タイプ制御器を `LT (<)` に、値を `100` に設定します。
24. このステップはマザーボードのクロック周波数測定をシミュレーションしますので、単位を `MHz` に変更します。単位制御器の隣のドロップダウンリングを使用して、単位には `Hertz` を、単位の接頭辞には `Mega` を選択します。すると単位制御器には、`megahertz` という値が表示されます。短縮形の名前を使用するには、各リングで短縮名を選択します。これにより単位制御器には、`MHz` という値が表示されます。指定した単位は、レポートと結果のデータベースに表示されます。単位と単位の接頭辞は表示およびドキュメント用のみのもので、測定値をスケールしたりリミットの比較に影響を与えることはありません。

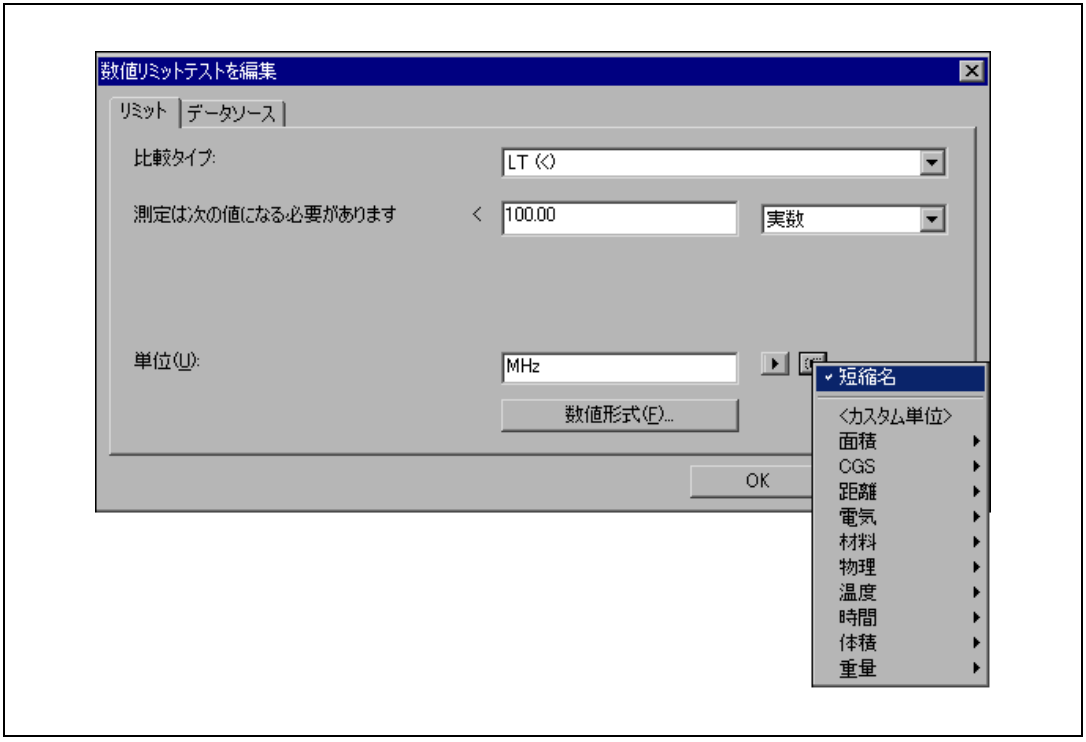

図 6-28　「数値リミットテストを編集」ダイアログボックス

25. **数値形式**ボタンをクリックして数値形式ダイアログボックスを開きます。このダイアログボックスの制御器の値を、図 6-29 に示すように設定します。これらの設定により、ステップの測定とリミットの値が指定されます。形式は、「数値リミットテストを編集」ダイアログボックス、ステップの説明、およびテストレポートに表示されるリミット値に適用されます。

    この設定で、TestStand は VI が返す数値測定値を定数 100 と比較します。比較が `True` の場合、ステップは合格になります。それ以外の場合は不合格です。

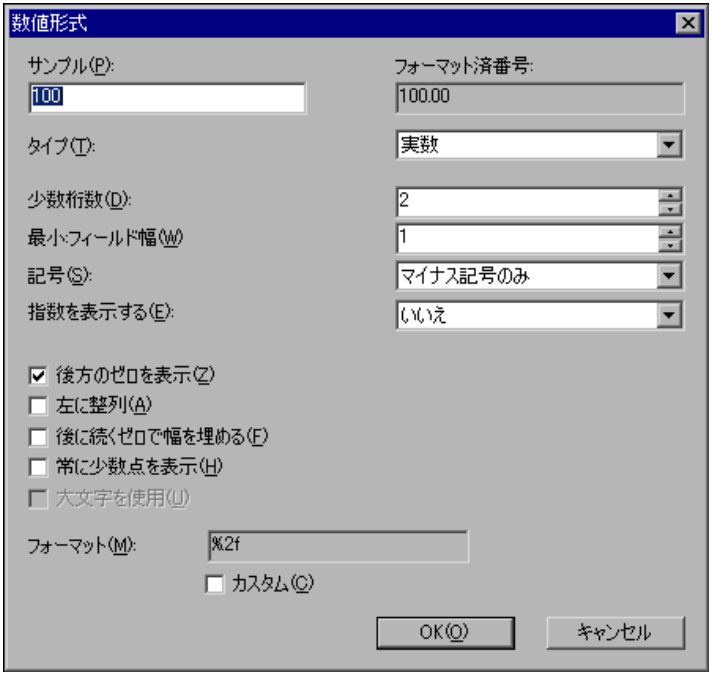

図 6-29　数値形式ダイアログボックス

26. **OK** を 2 回クリックして、数値形式ダイアログボックスと「数値リミットテストを編集」ダイアログボックスを閉じます。
27. **ファイル→別名で保存**を選択して、シーケンスを `Sample8.seq` という名前で `<TestStand>\Tutorial` ディレクトリに保存します。

LabWindows/CVI DLL を構築する

1. ここで、DLL モジュールを構築するための LabWindows/CVI プロジェクトを作成します。**スタート→プログラム→ National Instruments → Measurement Studio → CVI IDE** を選択して、LabWindows/CVI を起動します。
2. **File → New → Project** を選択して、新規のプロジェクトを開きます。LabWindows/CVI でプロジェクトがすでにロードされている場合は、現在のプロジェクトをアンロードするかどうか尋ねるプロンプトが表示されます。**Yes** ボタンをクリックします。
LabWindows/CVI は、現在のプロジェクトのオプションを新規のプロジェクトに移行するかどうか尋ねるプロンプトを表示します。Transfer Project Options ダイアログボックスのすべてのオプションのチェックを外して、**OK** ボタンをクリックします。

3. **Edit → Add Files to Project → All Files (*.*)** を選択して、Add Files to Project ダイアログボックスを開きます。下記の各ファイルを選び、Add ボタンを使って Selected Files 制御器に追加します。
   - `TestStand\Tutorial\FlexFrequency.c`
   - `TestStand\Tutorial\Frequency.uir`
   - `TestStand\API\CVI\tsapicvi.fp`
   - `TestStand\API\CVI\tsutil.fp`

   すべてのファイルを追加したら、**OK** ボタンをクリックしてファイル名が表示されているプロジェクトウィンドウに戻ります。
4. **File → Save** を選択して、プロジェクトを `FlexFrequency.prj` という名前で `<TestStand>\Tutorial` ディレクトリに保存します。
5. プロジェクトウィンドウでファイル名をダブルクリックして、`FlexFequency.c` ファイルを開きます。
6. 以下のように、`FlexFrequency.c` コードを更新して、`frequency.uir` ユーザインタフェースのリソースファイルを使用して周波数数値制御器およびレポートテキスト文字列制御器に値を入力するようオペレータにプロンプトします。
   a. `FlexFrequency.c` ウィンドウを開いたままで、**Windows → Project** を選択してプロジェクトウィンドウに戻ります。
   b. プロジェクトウィンドウから `frequency.uir` ファイルを開きます。
   c. **Return** ボタンを右クリックして、コンテキストメニューから **Generate Control Callback** を選択します。何らかの理由で複数の `.c` ファイルが開いていると、LabWindows/CVI はコールバックを挿入するターゲットファイルを選択するようプロンプトします。ターゲットとして `FlexFrequency.c` を選択します。
   d. これで、`FlexFrequency.c` にユーザインタフェースを終了する ReturnCallback 関数が作成されました。終了したら `.uir` ファイルを閉じます。変更は行いません。
   e. `ClockFrequency` および `ReturnCallback` 関数のソースコードを更新します。ソースファイルの他の関数は変更の必要はありません。変更された行は太字で示しています。

```
void __declspec(dllexport) __stdcall
    ClockFrequency(CAObjHandle seqContextCVI, double
    *measurement, char addReportTxt[1024], short
    *errorOccurred, long *errorCode, char
    errorMsg[1024])
{
    int error = 0;
    int panelHandle, panel, control;
```

```
    panelHandle = LoadPanelEx (0, "frequency.uir",
    PANEL, __CVIUserHInst);

    if (panelHandle < 0)
        {
        error = panelHandle;
        goto Error;
        }

    DisplayPanel (panelHandle);

    // Automatically closes user display dialog when
    // TestStand execution terminates or aborts.
    TS_CancelDialogIfExecutionStops (panelHandle,
    seqContextCVI);

    RunUserInterface();

    // Assign values from UIR to return data
    GetCtrlVal (panelHandle, PANEL_FREQUENCY,
    measurement);
    GetCtrlVal (panelHandle,
    PANEL_ADDITIONAL_REPORT, addReportTxt);


Error:
    // FREE RESOURCES
    DiscardPanel(panelHandle);

    // If an error occurred, set the error flag to cause
    // a run-time error in TestStand.
    if (error < 0)
    {
        *errorOccurred = TRUE;

    // OPTIONALLY SET THE ERROR CODE AND STRING
    *errorCode = error;
    strcpy(errorMsg, "A run-time error occurred.");
}
return;
}
```

```
int CVICALLBACK ReturnCallback (int panelHandle, int
    control, int event,void *callbackData, int
    eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            QuitUserInterface (0);
        break;
    }
    return 0;
}
```

7. プロジェクトで DLL を構築するために、プロジェクトウィンドウで **Build → Target Type → Dynamic Link Library** を選択します。
8. **Build → Create Debuggable Dynamic Link Library** を選択して DLL を構築します。**Build → Create Release Dynamic Link Library** というオプションしかない場合は、DLL を構築する前に **Build → Configuration → Debug** を選択します。DLL が正しく作成されると、LabWindows/CVI はそれを通知します。
9. これで、DLL 関数を呼び出すシーケンスを実行できる状態になりました。シーケンスエディタに戻り、**実行→一回実行**を選択してシーケンスを実行します。
10. Test Simulator ダイアログボックスの **Done** をクリックします。TestStand が Clock Frequency Test ステップを実行すると、コードモジュールはユーザインタフェースパネルを表示して入力を待ちます。
11. 周波数測定制御器に 20 という数値を入力します。
12. 「追加のレポートテキスト」制御器に任意のテキストを入力します。
13. **Return** ボタンをクリックしてシーケンスの実行を続行します。
14. シーケンスの実行が完了したら、テストレポートの内容を確認します。Clock Frequency ステップのステータス、測定値、およびレポートテキスト値を確認してください。
15. 実行ウィンドウを閉じます。

DLL 関数をデバッグする

DLL フレキシブルプロトタイプアダプタで呼び出された DLL をデバッグするには、DLL を作成したアプリケーション開発環境のデバッグ機能を使用する必要があります。この練習では、LabWindows/CVI の機能を使

用して、DLL フレキシブルプロトタイプアダプタを使って呼び出した LabWindows/CVI 関数をデバッグする方法を説明します。

1. シーケンスエディタを終了します。
2. LabWindows/CVI で、**File → Open → Project** を選択します。`<TestStand>\Tutorial\` を選択して `FlexFequency.prj` を開きます。
3. プロジェクトウィンドウで、**Run → Select External Process** を選択します。**Browse** ボタンをクリックして `<TestStand>\bin\SeqEdit.exe` を選択します。
4. **OK** をクリックして、LabWindows/CVI プロジェクトウィンドウに戻ります。**Run → Run SeqEdit.exe** を選択して、LabWindows/CVI からシーケンスエディタを起動します。変更を保存するようプロンプトされたら、**Yes** をクリックします。LabWindows/CVI は、シーケンスエディタを外部プロセスとして開始してデバッグのためにそのプロセスで実行します。
5. シーケンスエディタを起動してログインしたら、シーケンスファイル、`<TestStand>\Tutorial\Sample8.seq` を開きます。このファイルは、`<TestStand>\ Tutorial\Solution` ディレクトリにもあります。
6. Clock Frequency Test ステップにブレークポイントを設定します。

メモ　ステップのアイコンの左側をクリックするか、カーソルを特定の行の上に置いて F9 を押し、ブレークポイントを設定します。

7. **実行→一回実行**を選択してシーケンスを実行します。
8. Test Simulator ダイアログボックスの **Done** をクリックします。TestStand が Clock Frequency Test ステップで停止したら、F8 キーを押して DLL 関数にステップインします。また、関数にステップインする方法として、**デバッグ→ステップイン**を選択するか、シーケンスエディタツールバーのステップインボタンを使用することもできます。

   図 6-30 に示すように、実行は `FlexFrequency.c` の関数定義の最初の行で停止します。LabWindows/CVI デバッグウィンドウで、LabWindows/CVI ステップツール、ウォッチウィンドウ、および変数ウィンドウを使用してコードをデバッグすることができます。

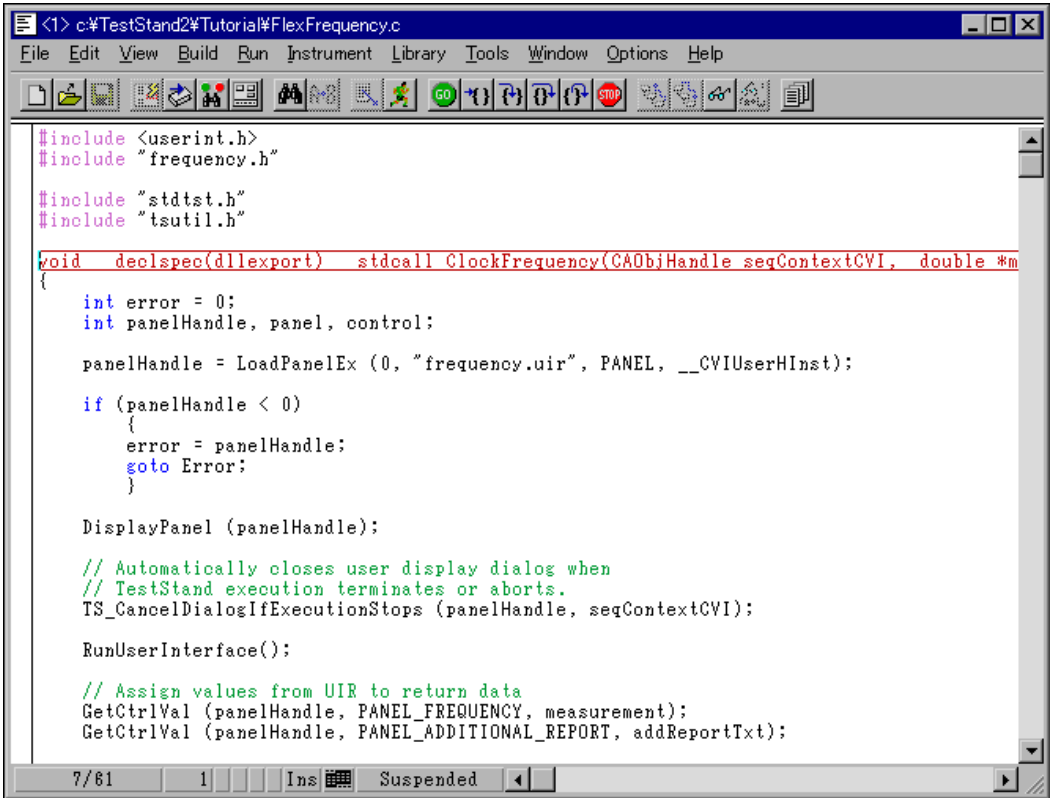

図 6-30　ClockFrequency 関数をデバッグする

9. `ClockFrequency` 関数は `RunUserInterface` 関数を呼び出し、`RunUserInterface` 関数は Clock Frequency ダイアログボックスを表示します。デバッグ中 LabWindows/CVI デバッグウィンドウは前面に表示されたままであるため、制御器に値を入力するには Clock Frequency ダイアログボックスに切り替えることが必要な場合があります。
10. Clock Frequency ダイアログボックスの測定値制御器に 20 という数値を入力します。
11. Clock Frequency ダイアログボックスの「追加のレポートテキスト」制御器に任意のテキストを入力します。
12. **Return** ボタンをクリックして続行します。
13. LabWindows/CVI は、DLL のソースファイルで設定したすべてのブレークポイントを受け入れます。Clock Frequency ステップが実行すると、シーケンスの実行は、ソースコードで設定した最初のブレークポイントで停止します。

14. シーケンスの実行に戻って実行を完了するまでに、DLL 関数のステップ処理を終了してください。このデバッグテクニックは、C/CVI 標準プロトタイプアダプタを使用するステップにも適用できます。
15. シーケンス実行ウィンドウを閉じて、シーケンスエディタを終了します。

メモ　外部プロセスを起動して添付する他のアプリケーション開発環境でも、同様のテクニックを使用することができます。デバッグに関する詳細については、National Instruments の Developer Zone（ni.com）を参照してください。

このチュートリアルセッションはこれで終わりです。次のセッションでは、TestStand オペレータインタフェースの使用方法について学習します。

7

TestStand ランタイムオペレータインタフェースを使用する

本章では、LabWindows/CVI オペレータインタフェースの使用方法を学習します。本章で説明する機能は、LabVIEW、Visual Basic、および Delphi の各オペレータインタフェースにも適用されます。ランタイムオペレータインタフェースのカスタマイズ方法についての詳細は、『TestStand User Manual』の Chapter 16、「Run-Time Operator Interfaces」を参照してください。

TestStand には、ランタイムオペレータインタフェースがソースと実行可能ファイルの 2 つの形式で含まれています。各ランタイムオペレータインタフェースは別々のアプリケーションプログラムです。各オペレータインタフェースは、基本的に開発された言語とアプリケーション開発環境（ADE）によって異なります。TestStand には、LabVIEW、LabWindows/CVI、Visual Basic、および Delphi で開発されたランタイムオペレータインタフェースが含まれています。TestStand ランタイムオペレータインタフェースは、シーケンスエディタほど複雑でなく、カスタマイズにも完全対応しています。

シーケンスをロードする

以下の手順に従って、ランタイムオペレータインタフェースでシーケンスをロードします。

メモ このチュートリアルセッションでは、TestStand のいずれのオペレータインタフェースも使用できます。LabVIEW オペレータインタフェースを使用していて、5.1 よりも新しいバージョンの LabVIEW を使用している場合、まず TestStand インストールディレクトリで VI を一括コンパイルする必要があります。このセッションのスクリーンショットは、LabWindows/CVI オペレータインタフェースを表示しています。

1. **スタート→プログラム→ National Instruments → TestStand → Operator Interfaces → LabWindows-CVI** を選択して、Windows タスクバーから LabWindows/CVI オペレータインタフェースを起動します。

   オペレータインタフェースのメインウィンドウが表示された後、ログインダイアログボックスが表示されます。

2. ユーザ名には管理者を選択し、パスワードは空白のままで、**OK** ボタンをクリックします。

   ログインすると、オペレータインタフェースが図 7-1 のように表示されます。

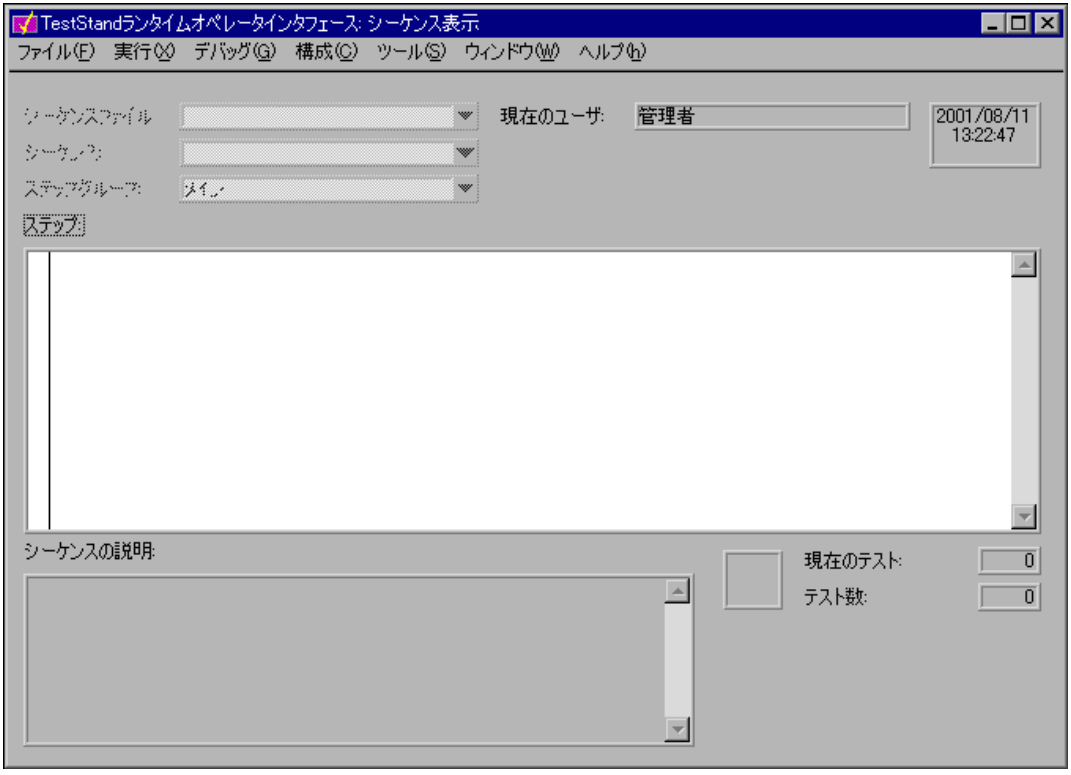

図 7-1　LabWindows/CVI オペレータインタフェース

3. マウスを使って各メニューを開いてみて、オペレータインタフェースのメニューにはシーケンスエディタと同じコマンドが多く含まれていることを確認してください。

   シーケンスエディタ同様、ランタイムオペレータインタフェースを使用すると、複数の同時実行、ブレークポイントの設定、シングルステップなどが可能です。ただしシーケンスエディタと異なるところは、ランタイムオペレータインタフェースではシーケンスの変更ができず、またシーケンス変数、シーケンスパラメータ、ステップのプロパティなどが表示されません。

4. この練習では、LabWindows/CVI のコードモジュールのデバッグは行いません。以下の手順で、LabWindows/CVI 標準プロトタイプアダプタが、オペレータインタフェースと同じプロセスでコードモジュールを実行するように正しく構成されていることを確認してください。
   a. **構成→アダプタ**を選択します。するとアダプタ構成ダイアログボックスが表示されます。
   b. アダプタセクションで、C/CVI 標準プロトタイプアダプタを選択します。
   c. **構成**ボタンをクリックします。すると CVI 標準アダプタ構成ダイアログボックスが表示されます。
   d. 「インプロセス内でステップを実行」オプションを有効にします。
   e. **OK** をクリックし、次に**完了**をクリックして構成ダイアログボックスを閉じます。
5. **ファイル→シーケンスファイルを開く**を選択して、第 3 章、「シーケンスのステップを編集する」で作成した `<TestStand>\Tutorial\Sample2.seq` ファイルを開きます。このファイルは、`<TestStand>\ Tutorial\Solution` ディレクトリにもあります。

シーケンスファイルを開くと、図 7-2 のようなオペレータインタフェースウィンドウが表示されます。

図 7-2　オペレータインタフェースで開いたシーケンス

シーケンスファイル、シーケンス、およびステップグループの各リング制御器は、ステップリストボックスに表示されるステップを指定します。

6. ステップグループ制御器の右の矢印をクリックしてステップを表示し、セットアップステップグループを選択します。
7. 再びメインを選択して、メインステップグループに戻ります。

シーケンスの実行とデバッグ

以下の手順に従って、ランタイムオペレータインタフェースでシーケンスを実行しデバッグします。

1. ステップをクリックしてから**デバッグ→ブレークポイントのトグル**を選択し、CPU Test ステップにブレークポイントを設定します。ステップ名の左に "B" という文字が表示されます。
2. **実行→一回実行**を選択してシーケンスを実行します。

   実行を開始すると、オペレータインタフェースは実行を別のウィンドウに表示します。
3. Test Simulator ダイアログボックスが表示されたら、**Done** をクリックします。すると、図 7-3 に示すように、CPU Test ステップのブレークポイントで実行が一時停止します。

図 7-3　オペレータインタフェースの実行の一時停止

4. **デバッグ→ステップイン**を選択します。実行ウィンドウが変わって、`SubSequence1.seq` シーケンスファイルの `MainSequence` シーケンスのステップを表示します。
5. **デバッグ→ステップオーバー**を選択してシングルステップ処理を数回行います。
6. **デバッグ→再開**を選択して実行を完了します。

   ソースコードモジュールのシングルステップおよびステップインは、すべてオペレータインタフェースアプリケーションで使用できます。
7. **ファイル→実行を終了**を選択して実行ウィンドウを閉じます。

複数の実行を処理する

次の手順に従って、新規シーケンスファイルを開き、複数の実行を開始します。

1. **ファイル→シーケンスファイルを開く**を選択して、`<TestStand>\Tutorial` ディレクトリから `LoopForever.seq` シーケンスファイルを開きます。このシーケンスには、連続的にループで実行して最初のステップに戻る一連の空のステップが含まれています。
2. **実行→ “MainSequence” の実行**を選択して実行を開始します。

   新規実行ウィンドウが表示されたら、実行ウィンドウの位置を移動してシーケンス表示ウィンドウが見えるようにします。
3. シーケンス表示ウィンドウで**実行→ “MainSequence” 実行**を再度選択して、2 度目の実行を開始します。

   メニューに **“MainSequence” 実行**コマンドがない場合は、タブをクリックして正しいウィンドウを選択していることを確認してください。
4. 全部で 4 つの実行を行います。開始した実行はそれぞれオペレータインタフェースのメインプロセスで実行します。
5. 開始した実行をすべて終了するには、**デバッグ→すべての実行を停止**を選択します。**すべての実行を停止**コマンドは、いずれのオペレータインタフェースウィンドウからでも使用できます。
6. **ウィンドウ→完了した実行ウィンドウを閉じる**を選択して実行ウィンドウを閉じます。
7. **ファイル→終了**を選択して、オペレータインタフェースのメインウィンドウを閉じます。

このチュートリアルセッションはこれで終わりです。次のセッションでは、コールバックの使用方法について学習します。

8

コールバックを使用する

本章では、コールバックを使用して、TestStand 内でシーケンスの実行をカスタマイズする方法を学習します。コールバックとは、シリアル番号の照会やレポートの記録など、一般的なタスクを処理するのに使用するシーケンスです。

TestStand に含まれるデフォルトコールバックシーケンスはすべてソースの形式で提供されていますので、編集や入れ換えなどを行って特定のアプリケーション用に TestStand をカスタマイズすることができます。このチュートリアルセッションでは、TestStand のデフォルトコールバックの 1 つをユーザ独自のものと入れ換えます。

サンプルをセットアップする

第 7 章、「コールバックを使用する」を終了していない場合は、次の手順で TestStand シーケンスエディタをセットアップしてからこのチュートリアルセッションを開始してください。

1. シーケンスエディタが開いていない場合は、シーケンスエディタを起動します。
2. シーケンスエディタのすべてのウィンドウを閉じます。

プロセスモデルコールバックを無効にする

TestStand プロセスモデルには、UUT のテストの前と後に TestStand が実行する操作を定義するシーケンスが含まれています。プロセスモデル内のシーケンスを呼び出す場合は、そのモデル内のエントリポイントシーケンスを実行することができます。本書の第 2 章、「シーケンスのロードと実行」で説明するとおり、各モデルはデフォルトのモデルエントリポイントである `UUT をテストと一回実行`を使用します。プロセスモデルには工夫された仕掛けがあり、それによって、プロセスモデルを直接編集することなく、それを使用する各メインシーケンスのプロセスモデルの動作をカスタマイズすることが可能になっています。それらの仕掛けはシーケンスの形式になっていて、モデルコールバックといいます。

たとえば、TestStand プロセスモデルは、各 UUT のテストレポートを生成する `TestReport` コールバックを定義します。さまざまなタイプのテスト結果を処理できるため、通常はプロセスモデルファイル内の

TestReport コールバックで十分です。ただし、特定のクライアントシーケンスファイルで別の TestReport コールバックを定義することによって、デフォルトの TestReport コールバックを無効にすることができます。すべてのシーケンスのプロセスモデルの動作を変更するには、プロセスモデルを修正するか、あるいはモデル全体を入れ換えます。

プロセスモデルの実行エントリポイントは、コールバックを使用して、クライアントシーケンスファイルのメインシーケンスを呼び出します。各クライアントシーケンスファイルは、シーケンスを MainSequence という名前で定義する必要があります。プロセスモデルには MainSequence コールバックが含まれますが、これは単なるプレースホルダです。クライアントシーケンスファイルの MainSequence は、モデルファイルの MainSequence プレースホルダに優先します。

図 8-1 は、デフォルトの TestStand プロセスモデルである `SequentialModel.seq` が呼び出すコールバックと、TestStand が「UUT をテスト」実行エントリポイント内でコールバックを実行する順序を示します。

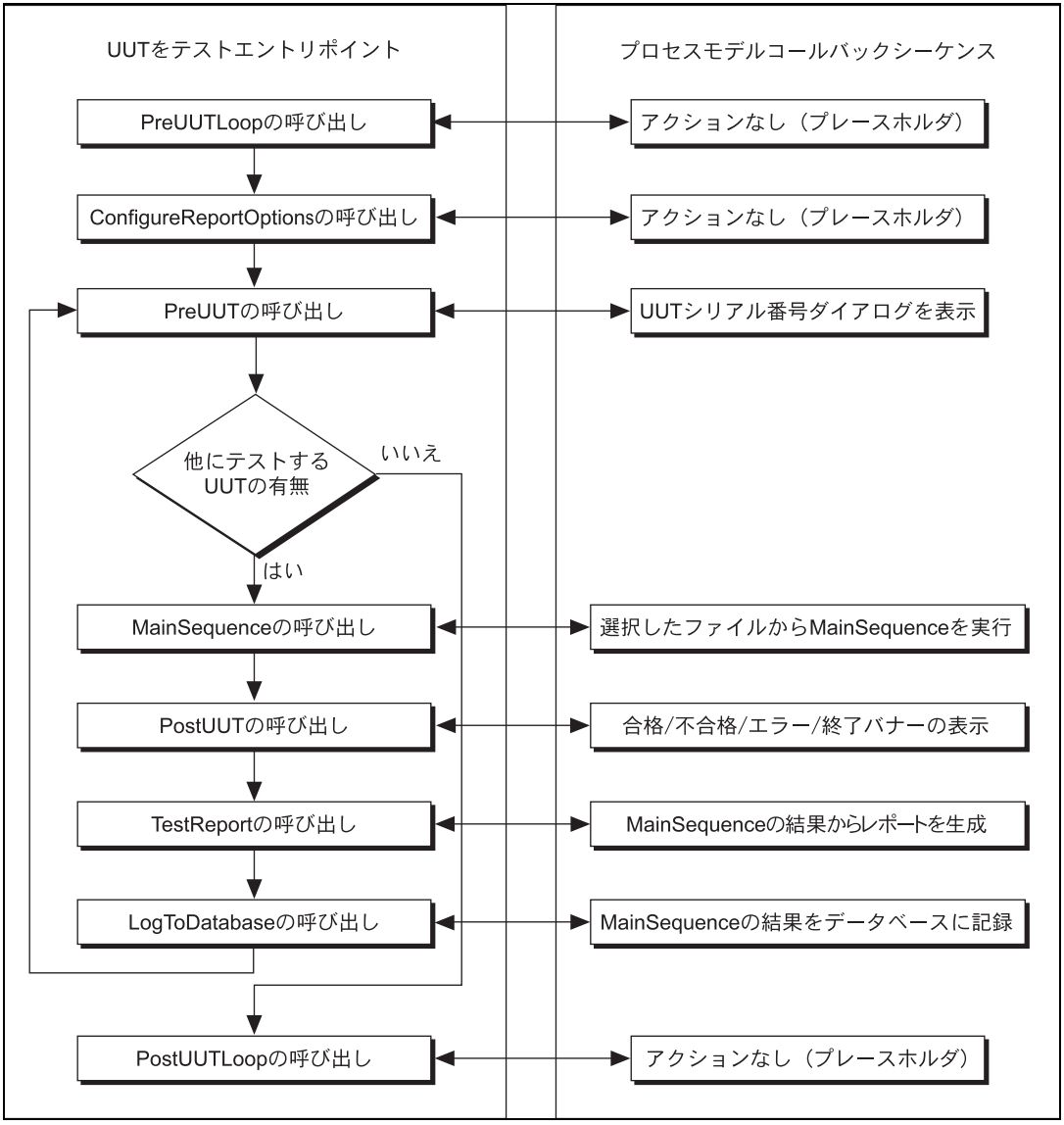

図 8-1　TestStand シーケンシャルモデルコールバック

次の手順にしたがって、メッセージポップアップのプロンプトを表示する PreUUTLoop コールバックを追加します。

1. <TestStand>\Components\NI\Models\TestStandModels\SequentialModel.seq を開きます。このファイルは、<TestStand>\ Tutorial\Solution ディレクトリにもあります。このファイルは、TestStand がシーケンス実行のために使用するデフォルトプロセスモデルです。
2. ビュー選択リングで TestUUTs シーケンスを選択します。このシーケンスは、**実行→ UUT をテスト**を選択したときに TestStand が実行する UUT をテストエントリポイントです。PreUUTLoop Callback、PreUUT Callback、MainSequence Callback、および PostUUTLoop Callback など、UUT をテストシーケンスが呼び出すコールバックシーケンスに注目してください（図 8-2 参照）。

図 8-2　「UUT をテスト」シーケンス

3. PreUUT Callback ステップを右クリックして、コンテキストメニューから**シーケンスを開く**を選択します。PreUUT Callback シーケンスには、Identify UUT と Set Serial Number の 2 つのステップがあります。
4. IdentifyUUT ステップを右クリックして、コンテキストメニューから**選択ステップを実行**を選択します。すると、見慣れた外観のダイアログボックスが表示されます。

   このステップは、UUT をテストエントリポイントを使ってシーケンスを実行したときに UUT 情報ダイアログボックスを表示します。バーコードからの読み取りなど、TestStand が UUT シリアル番号を取得する方法を変更する場合は、このコールバックを独自のものと入れ換えます。
5. UUT 情報ダイアログボックスの **OK** をクリックします。
6. 実行ウィンドウを閉じます。
7. ビュー選択リングで `TestUUTs` をもう一度選択します。
8. PreUUTLoop Callback ステップを右クリックして、コンテキストメニューから**シーケンスを開く**を選択します。このコールバックは空であることに注目してください。空のシーケンスはプレースホルダであるため、UUT ループの前に実行するステップを追加する場合に、このコールバックで作成することができます。
9. `SequentialModel.seq` シーケンスファイルウィンドウを閉じます。変更を保存するようプロンプトされたら、保存しないで閉じます。

   PreUUTLoop Callback ステップの代わりに、独自のコールバックシーケンスを有効にします。
10. 第 5 章、「変数およびプロパティを使用する」で作成した `<TestStand>\Tutorial\Sample4.seq` ファイルを開きます。このファイルは、`<TestStand>\ Tutorial\Solution` ディレクトリにもあります。

11. **編集→シーケンスファイルコールバック**を選択して、図 8-3 に示すような `Sample4.seq` コールバックを表示します。

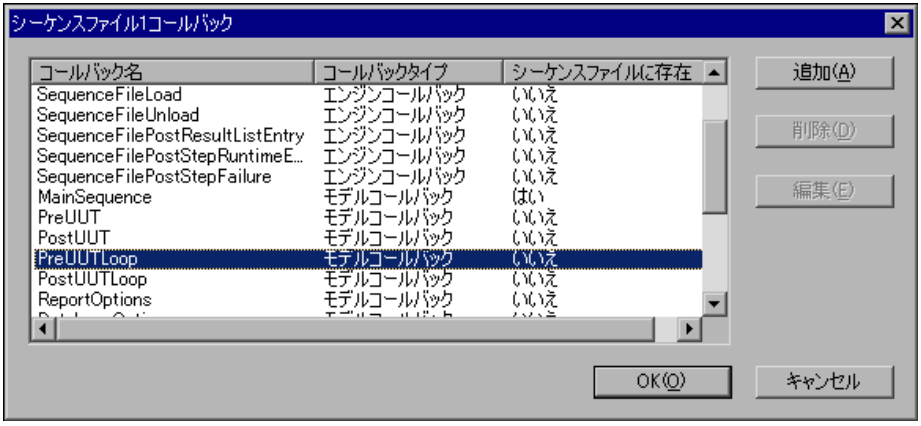

図 8-3　コールバックをシーケンスに追加する

12. `PreUUTLoop` コールバックの名前を選択します。
13. **追加**ボタンをクリックします。

    「シーケンスファイルに存在」コラムの値が「いいえ」から「はい」に変わったことに注意してください。**追加**ボタンをクリックすると、シーケンスエディタはシーケンスファイルに新しい空のコールバックシーケンスを作成します。ここでモデルエントリポイントを使って実行を開始すると、TestStand は `SequentialModel.seq` プロセスモデルのシーケンスではなく、シーケンスファイル内のコールバックを呼び出します。
14. **OK** をクリックしてコールバックダイアログボックスを閉じます。
15. シーケンスファイルウィンドウのビュー選択リングですべてのシーケンスを選択します。シーケンスファイルは、`MainSequence` と `PreUUTLoop` の 2 つのシーケンスが含まれていることを確認してください。
16. `PreUUTLoop` シーケンスを右クリックして、コンテキストメニューから**内容を表示**を選択します。
17. メインタブの空白のステップリスト上で右クリックして、コンテキストメニューから**ステップを挿入→メッセージポップアップ**を選択します。
18. 新規のステップに `Pre UUT Message` という名前を付けます。
19. Pre UUT Message ステップを右クリックして、コンテキストメニューから**メッセージの設定を編集**を選択します。

20. タイトル式制御器に、リテラル文字列 "Pre UUT Loop Callback Message" を入力します。式フィールドに入力するリテラル文字列は、二重引用符 (") で囲む必要があります。
21. メッセージ式に、リテラル文字列 "Pre UUT Loop Callback Message" を入力します。
22. **OK** をクリックして「メッセージボックスステップを構成」ダイアログボックスを閉じます。
23. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを Sample9.seq という名前で TestStand\Tutorial ディレクトリに保存します。
24. **実行→ UUT をテスト**を選択してシーケンスを実行します。最初に表示されるプロンプトが Pre UUT Loop Callback Message ダイアログボックスであることに注目してください。
25. **OK** をクリックしてダイアログボックスを閉じます。すると、SequentialModel.seq プロセスモデルの PreUUT Callback シーケンスからの UUT 情報ダイアログボックスが表示されます。
26. シリアル番号を入力して **OK** をクリックします。
27. シーケンスの反復を数回実行してください。
28. UUT 情報ダイアログボックスの**停止**ボタンをクリックします。

    実行の最初で、Pre UUT Loop Callback Message ダイアログボックスのみが一度表示されることに注意してください。その理由は図 8-1 に示すとおり、PreUUTLoop Callback はループの前に実行し、PreUUT Callback はループ内で実行されるためです。
29. シーケンスエディタのすべてのウィンドウを閉じます。

この例のような変更は、ParallelModel.seq および BatchModel.seq の各 TestStand プロセスモデルでも行うことができます。これらのモデルについては、第 2 章、「シーケンスのロードと実行」で説明しています。プロセスモデル、コールバック、およびコールバックの変更に関する詳細については、『TestStand User Manual』の Chapter 1、「TestStand Architecture Overview」、Chapter 3、「Configuring and Customizing TestStand」、および Chapter 14、「Process Models」を参照してください。

このチュートリアルセッションはこれで終わりです。次のセッションでは、TestStand でのユーザの追加およびユーザ特権の構成方法について学習します。

9

ユーザの追加と特権の設定

本章では、TestStand ユーザマネージャの使用方法と、新規ユーザの追加およびユーザ特権の変更方法について説明します。

サンプルをセットアップする

第 8 章、「コールバックを使用する」を終了していない場合は、シーケンスエディタ内のウィンドウをすべて閉じてからこのチュートリアルセッションを実行してください。

ユーザマネージャを使用する

TestStand シーケンスエディタには、ユーザの追加や削除、および各ユーザの特権を管理をするためのユーザマネージャが含まれています。

メモ TestStand ユーザマネージャは、テストステーションの使用に関する方針や手順を実装するのに役立ちます。これはセキュリティシステムではないため、オペレーティングシステムやサードパーティアプリケーションに常駐したり制御したりするものではありません。テストステーション用のコンピュータの悪用を防止するには、オペレーティングシステムで提供されているシステムレベルのセキュリティ機能を使用する必要があります。

この練習では、現在のユーザの表示方法と新規ユーザの追加方法を学習します。

1. **表示→ユーザマネージャ**を選択して、ユーザマネージャウィンドウを起動します。左側のペーンには、このステーションで構成されているすべてのユーザが表示されます。管理者ツリーノードを展開して、このユーザに関連付けられたプロパティの階層を表示します。
   `Privileges` ノードには、シーケンスの実行やデバッグ、新規ユーザの追加など、ユーザが実行できるすべてのアクションの設定が含まれています。
   ツリー表示でノードをハイライトすると、ノード以下の対応するプロパティの値が右側のリストペーンに表示されます。管理者ユーザの privileges のプロパティ値はすべて `True` であることに注目してください。

特権は、階層グループに分けられています。各 previleges グループには、GrantAll という名前のブール形式のサブプロパティがあります。privilege のプロパティを True に設定すると、ユーザに特権が与えられます。別の方法として、privileges グループの GrantAll プロパティを設定すると、個々の privilege のプロパティ値にかかわらず、ユーザが privilege グループ内のすべての特権を与えられるかどうかを指定することができます。

メモ　プロパティ User.Privileges.GrantAll は、すべての previlege グループに適用されます。このプロパティが True に設定されている場合、ユーザはすべての特権を与えられます。各 previlege グループ内での privilege 設定を優先する場合は、このプロパティを False に設定する必要があります。

特権を許可するにはいくつかの方法があります。次の例は、特権を許可する 1 つの方法を示します。ユーザは、以下の条件が満たされたときに実行を終了する特権を持つことができます。

- User.Privileges.GrantAll は、True に設定されています。これにより、他のすべての特権も許可されます。
- User.Privileges.GrantAll は False に、User.Privileges.Operate.GrantAll は True に設定されています。これにより、Operate グループの他の特権も許可されます。
- User.Privileges.GrantAll と User.Privileges.Operate.GrantAll は False に、User.Privileges.Operate.Terminate は True に設定されています。この設定により許可されるのは、実行を停止するユーザ特権のみです。

図 9-1 は、管理者ユーザのプロパティを展開した状態のツリー表示を示します。

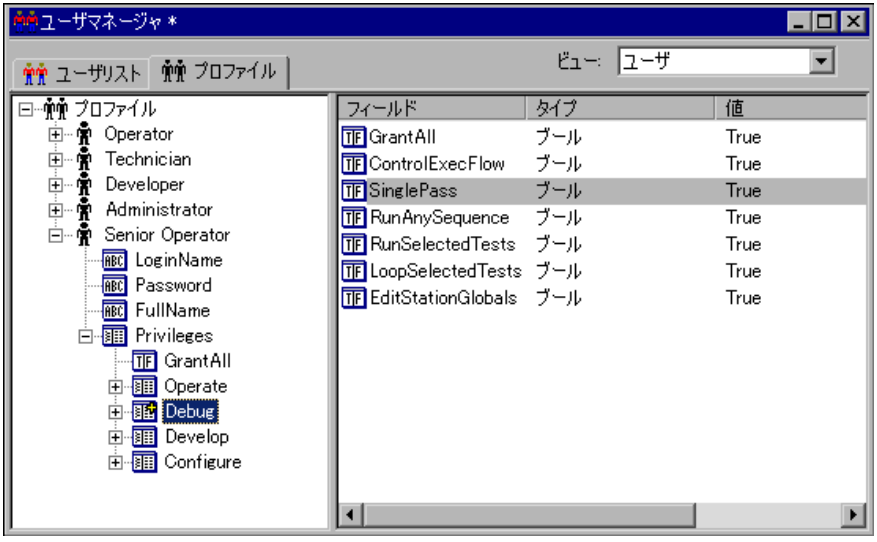

図 9-1　ユーザマネージャウィンドウ

2. 次の手順で新規ユーザを追加します。
   a. ツリー表示のユーザリストノードをクリックして、右側のペーンのリストに管理者ユーザを表示します。
   b. 右側のペーンを右クリックして、コンテキストメニューから**ユーザを挿入**を選択し、新規ユーザダイアログボックスを表示します。
   c. ユーザ名とフルネームの各制御器に名前を入力します。
   d. パスワードおよびパスワード確認の各制御器にパスワードを入力します。

図 9-2 は、すべてを入力した新規ユーザダイアログボックスの例を示します。

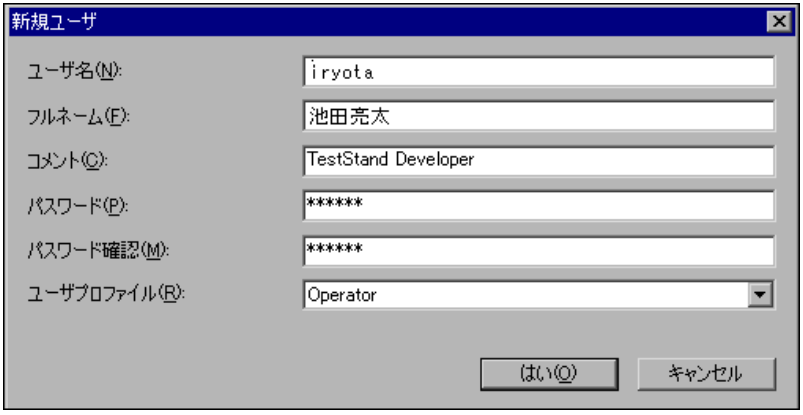

図 9-2　新規ユーザダイアログボックス

e. ユーザプロファイル制御器では、`Operator` を選択します。

ユーザプロファイルにより、新規ユーザに与えられる特権の初期設定が定義されます。デフォルトで、`Operator` プロファイルによりシーケンスの実行、終了、および中止の特権がユーザに許可されますが、シーケンスの作成やデバッグの特権は与えられません。TestStand では、`Operator`、`Technician`、`Developer`、および `Administrator` の 4 つのユーザプロファイルがデフォルトで用意されています。

f. **OK** をクリックして新規ユーザダイアログボックスを閉じます。

新規ユーザの追加のほかに、ユーザマネージャではデフォルトプロファイルを変更したり、テストステーションに適した特権の設定を定義する新規プロファイルを作成したりすることができます。

3. 次の手順で新規ユーザを作成します。

a. ユーザマネージャウィンドウのプロファイルタブをクリックします。リストペーンに、4 つのデフォルトプロファイルが表示されているはずです。

b. ツリー表示のプロファイルノードをクリックして、右側のリストペーンに現在定義されているプロファイルを表示します。

c. 右側のペーンで `Operator` プロファイルを右クリックして、コンテキストメニューから**コピー**を選択します。

d. 右側のペーンを右クリックして**貼り付け**を選択します。

e. 新規プロファイルに `Senior Operator` という名前を付けます。新規プロファイルは、`Operator` プロファイルとまったく同じです。

メモ　Senior Operator プロファイルがすでにある場合は、貼り付けすることによって名前の最後に下線と固有の番号が付けられます。

プロファイルの値を変更しても、ユーザリストにすでに存在するユーザの特権には影響ありません。ユーザを作成した後は、特権を個々に変更する必要があります。既存のユーザについては、ユーザプロファイルを変更しても特権を変更することはできません。

4. 以下の手順で、この新規プロファイルのデフォルト特権設定を変更します。
   a. ツリー表示で新しく作成した Senior Operator ノードを選択し、展開して特権設定を表示します。
   b. 図 9-3 に示すように、ツリー表示で Debug ノードを選択します。

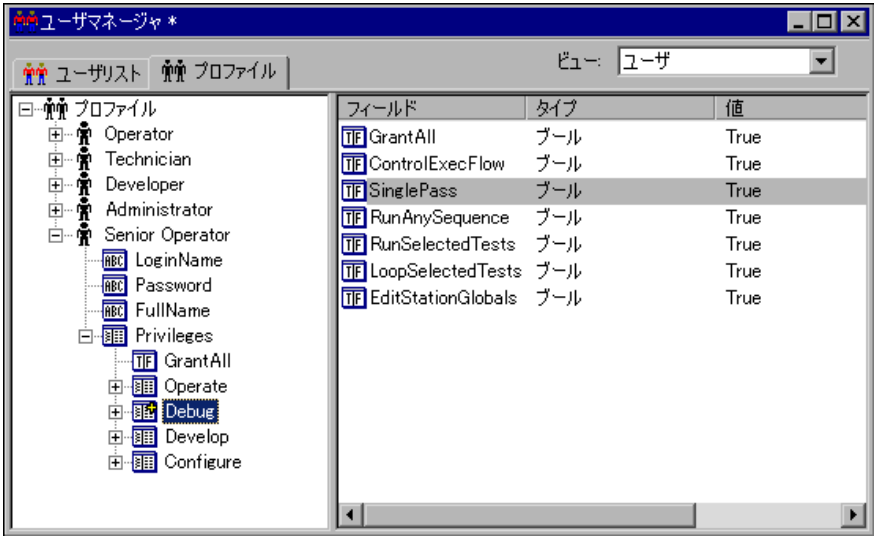

図 9-3　新規プロファイルの特権の構成

Debug はコンテナプロパティで、ブールのサブプロパティが含まれています。Debug の中のサブプロパティの値はすべて False です。

右側のペーンの各特権の値を設定するには、各項目を右クリックしてコンテキストメニューから**プロパティ**を選択します。

privilege グループ内の特権を無効にしてグループ内のすべての特権をユーザに許可するには、グループ内にある GrantAll プロパティの値を True に設定します。

以下の手順で、Senior Operator プロファイルに対して Debug グループの中の SinglePass プロパティを True に設定します。

a. 右側のペーンで Single Pass の名前をダブルクリックします。ブールプロパティダイアログボックスが表示されます。
b. 値を True に変更します。
c. **OK** をクリックしてダイアログボックスを閉じます。

5. 以下の手順で、Senior Operator プロファイルを使って新規ユーザを追加します。
   a. ユーザリストタブをクリックします。
   b. 右側のペーンを右クリックして、コンテキストメニューから**ユーザを挿入**を選択します。
   c. 前の手順と同様に、新規ユーザダイアログボックスに情報を入力しますが、今回は別のユーザ名にします。
   d. Senior Operator プロファイルを選択します。
   e. **OK** をクリックして新規ユーザダイアログボックスを閉じます。
6. **ファイル→ログイン**を選択します。追加したばかりの 2 人の新規ユーザが、administrator ユーザ名の下に表示されます。
7. Senior Operator プロファイルで作成したユーザを選択します。
8. 正しいパスワードを入力します。
9. **OK** をクリックします。
10. TestStand\Tutorial ディレクトリから Sample1.seq を開きます。
11. **実行**メニューを開きます。**一回実行**メニューは選択可能になっていますが、モデルエントリポイントなしでシーケンスを実行する特権を失ったため、**MainSequence を実行**コマンドはグレー表示になっていることに注意してください。
12. シーケンスビューで右クリックして新規ステップを挿入しようとしても、特権が変わっているため、挿入コマンドもやはりグレー表示になっていることを確認してください。
13. シーケンスエディタのすべてのウィンドウを閉じます。
14. **ファイル→ログイン**を選択します。
15. 管理者としてログインします。管理者のパスワードは空白です。

TestStand API を使用して、ユーザをプログラム的に追加したり削除したりすることができます。付属されているサンプル、<TestStand>\Example\CreateDeleteUsers\CreateDeleteUsers.seq で、ユーザの追加および削除の方法を示しています。

このチュートリアルセッションはこれで終わりです。次のセッションでは、コードモジュールで TestStand ActiveX API を使用する方法について学習します。

10

コードモジュールで ActiveX を使用する

本章では、TestStand のコードモジュール内で ActiveX を使用する方法について説明します。LabVIEW または LabWindows/CVI を使用していない場合は、この章をスキップして第 11 章、「上級開発機能」に進んでください。

TestStand では、さまざまな場所にデータ値を保存することができます。そのような場所を変数およびプロパティといいます。

第 5 章、「変数およびプロパティを使用する」で説明したように、変数とはある一定のコンテキストで自由に作成することのできるプロパティです。変数は、シーケンスファイルに対してグローバルであったり、特定のシーケンスに対してローカルであったりすることができます。また、ステーショングローバル変数を持つこともできます。ステーショングローバル変数の値は、異なる実行間、およびシーケンスエディタやランタイムオペレータインタフェースの別の呼び出し間でも一貫しています。TestStand エンジンは、ステーショングローバル変数の値をランタイムコンピュータ上のファイルに保持しています。

シーケンスの各ステップにはプロパティがあります。たとえば、ステップには整数エラーコードプロパティなどがあります。ステップのタイプにより、ステップのプロパティの設定が決まります。たとえば、数値リミットテストステップには比較タイプおよび上限下限値のプロパティが含まれます。

TestStand 変数を使用すると、データ表記に互換性がなくても、別のプログラミング言語で作成したテスト間でデータを共有できます。変数およびプロパティに保存した値をコードモジュールに渡すことができます。さらに、TestStand ActiveX API を使用すると、コードモジュールから直接変数やプロパティの値にアクセスすることが可能です。シーケンスを実行する際、TestStand はすべてのグローバル変数、ローカル変数、およびアクティブシーケンスのステッププロパティへのリファレンスを含むシーケンスコンテキストを保持しています。シーケンスコンテキストの内容は、現在実行中のシーケンスおよびステップによって異なります。シーケンスコンテキストのオブジェクトリファレンスをコードモジュールに渡すと、TestStand ActiveX API を使用してシーケンスコンテキストの変数およびプロパティにアクセスすることができます。

このセッションでは、ActiveX API を使って TestStand とデータを共有する LabVIEW および LabWindows/CVI 開発環境で、コードモジュールを作成する方法について説明します。

LabVIEW テスト仮想計測器で ActiveX を使用する

この練習では、LabVIEW VI を呼び出す 2 つのステップを含む新規のシーケンスを作成します。最初のステップでは、データの配列を生成してそれを TestStand の変数に保存します。次のステップでは、変数に保存されたデータをプロットします。このチュートリアルセッションでは、LabVIEW 開発環境の使用経験があることを前提としています。LabVIEW を使用していなくて、LabWindows/CVI を使用しているかまたは C スタイルの DLL を作成する場合は、このセクションをスキップして本章の「LabWindows/CVI コードモジュールで ActiveX を使用する」のセクションに進んでけっこうです。

メモ　TestStand に対応する正しいバージョンの LabVIEW を使用していることを確認してください。詳細については、`TestStand\Doc` ディレクトリの `readme.txt` ファイルを参照してください。

サンプルをセットアップする

このチュートリアルセッションを完了するためには、シーケンスエディタ内のウィンドウをすべて閉じます。

シーケンスおよび仮想計測テストを作成する

この練習では、シーケンスエディタで新規のシーケンスを作成します。

1. メニューから**ファイル→新規シーケンスファイル**を選択して新規シーケンスを開きます。
2. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample10.seq` という名前で `<TestStand>\Tutorial` ディレクトリに保存します。ここでシーケンスファイルを保存すると、コードモジュールの絶対パスではなく相対パスを指定することができます。
3. シーケンスファイルウィンドウのローカル変数タブをクリックします。

4. 図 10-1 に示すように、右側のペーンで右クリックして数値配列変数を挿入します。

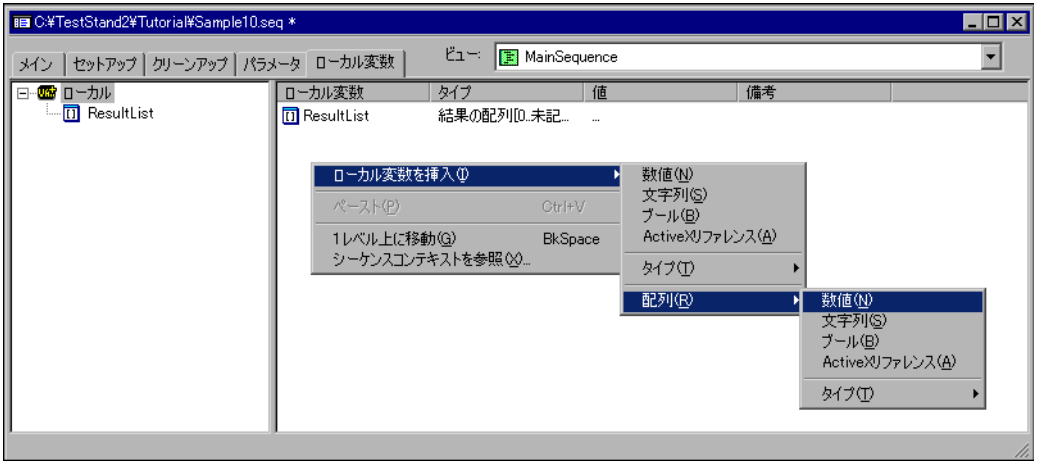

図 10-1　数値のローカル変数配列を挿入

するとシーケンスエディタは、図 10-2 に示すような配列範囲ダイアログボックスを表示します。

図 10-2　配列範囲ダイアログボックス

5. 図 10-2 に示すように、上限制御器に `1023` という値を入力します。
6. **OK** をクリックしてダイアログボックスを閉じます。
7. 新規変数を右クリックして、コンテキストメニューから**名前変更**を選択します。
8. 変数に `Arraydata` という名前を付けます。

9. シーケンスファイルウィンドウのメインタブをクリックします。アダプタ選択リング制御器で、LabVIEW 標準プロトタイプアダプタを選択します。
10. ステップリストを右クリックして、**ステップを挿入→アクション**を選択し新規アクションステップを挿入します。
11. ステップに `Get Array Data From LabVIEW` という名前を付けます。
12. Get Array Data From LabVIEW ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。
13. 「LabVIEW VI コールを編集」ダイアログボックスの**参照**ボタンをクリックして、`TestStand\Tutorial` ディレクトリの `GenerateWaveform.vi` を選択します。
14. 「LabVIEW VI コールを編集」ダイアログボックスで、「シーケンスコンテキスト ActiveX ポインタ」と「呼び出されたら VI のフロントパネルを表示」の各オプションにチェックを付けます。
15. VI の指定ができましたので、「LabVIEW VI コールを編集」ダイアログボックスで **VI を編集**ボタンをクリックして、LabVIEW で VI を開きます。

図 10-3 は、`GenerateWaveform.vi` のフロントパネルを示します。この VI は、フロントパネルからの入力に基づいて波形を生成します。

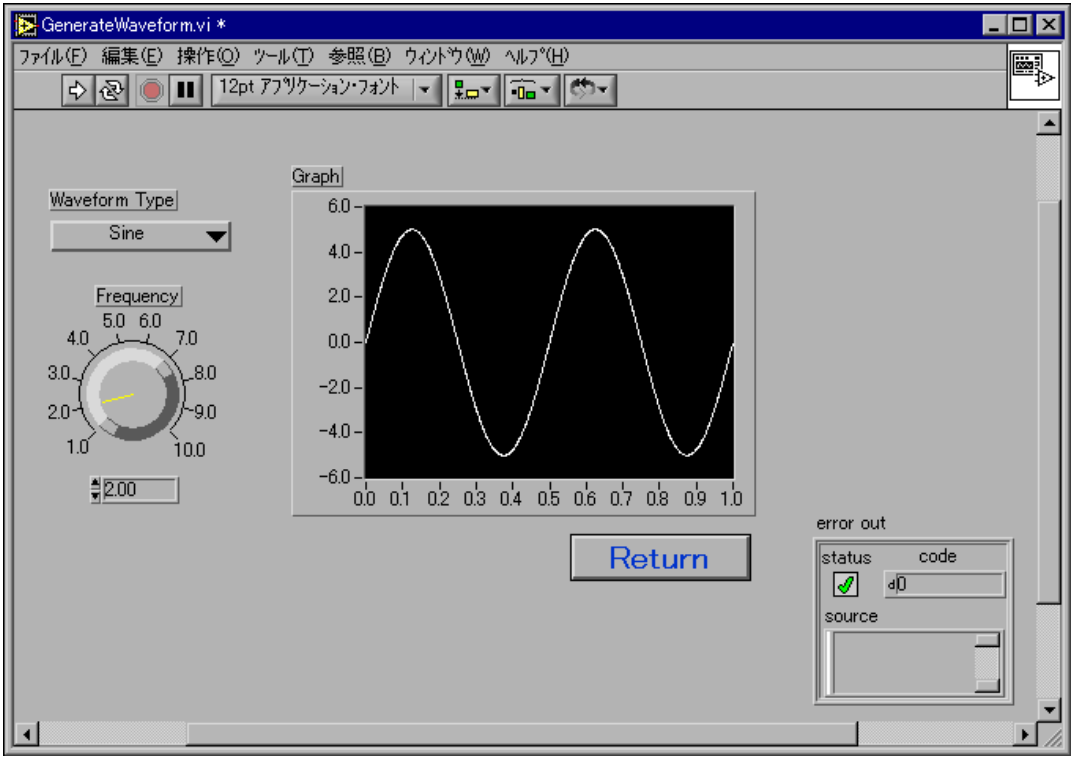

図 10-3　GenerateWaveform.vi フロントパネル

16. LabVIEW で**ファイル→別名で保存**を選択して、VI ファイルを `GenerateTestStandWaveform.vi` という名前で `TestStand\Tutorial` ディレクトリに保存します。

    このように保存するのは、他の人が次回このチュートリアルを行えるよう元の VI を残しておくためです。後ほど、この新規 VI を示すように TestStand のステップの指定のモジュールを変更します。

    TestStand をインストールする際、`LabVIEW\User.lib` ディレクトリに `TestStand.LLB` が追加されます。この VI ライブラリには、TestStand LabVIEW 標準プロトタイプ制御器と、TestStand ActiveX オートメーションサーバを呼び出す際に役立つ追加の VI が含まれています。

図 10-4 は、TestStand 制御器パレットを示します。

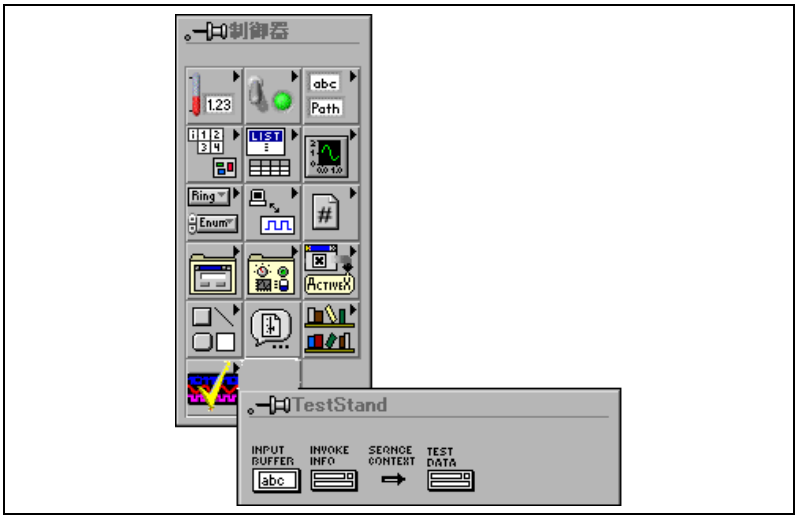

図 10-4　TestStand 制御器パレット

LabVIEW 標準プロトタイプアダプタが呼び出すすべてのテスト VI には、Test Data 制御器と error out 制御器がなくてはなりません。テスト VI にはまた、Sequence Context、Input Buffer、および Invocation Info の各制御器が含まれている場合があります。

17. LabVIEW 標準プロトタイプアダプタが TestStand から VI を呼び出せるように、以下の手順でフロントパネルに制御器を追加します。
    a. TestStand 制御器パレットから Sequence Context 制御器と Test Data 制御器を VI フロントパネルにドラッグします。

b. 図 10-5 に示すように、配線ツールを使って 2 つの新規制御器をフロントパネルウィンドウの右上コーナーにある端子コネクタに割り当てます。Sequence Context を左上位置に配線します。Test Data クラスタを右上位置に配線します。接続の位置は、厳密に守らなければならないわけではありません。ただし慣例により、制御器はコネクタペーンの左側に、表示器は右側に配線するようにしてください。制御器を端子に配線しないと、TestStand はデータを制御器に渡すことができません。

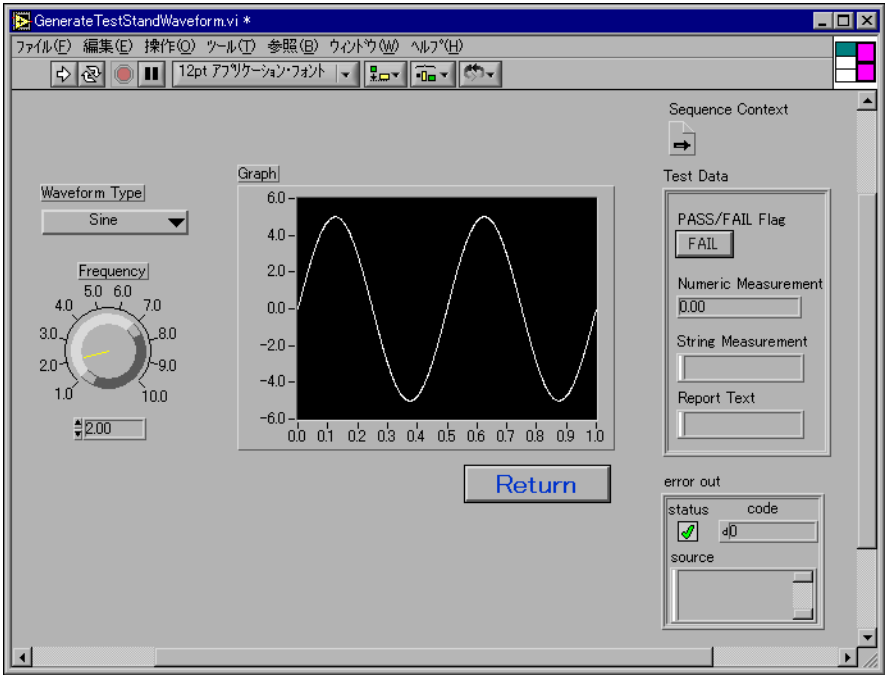

図 10-5　GenerateTestStandWaveform.vi フロントパネル

c. LabVIEW 標準プロトタイプアダプタでは、VI 端子を介して配列データを直接テスト VI に渡すことはできません。その代わりに、テスト VI は TestStand API を使用して TestStand エンジンに配列データを提供する必要があります。

18. 次の手順で、TestStand 関数パレット内の関数を使用して、シーケンスコンテキスト refnum によって配列データを TestStand 変数に割り当てます。
    a. 図 10-6 に示すように、テスト VI のブロックダイアグラムを開きます。

メモ　`InitializeTerminationMonitor.vi`、`GetMonitorStatus.vi`、および `CloseTerminationMonitor.vi` の 3 つの VI は、ステップモジュールが呼び出されている TestStand の実行のステータスを監視します。ダイアログボックスの表示など、ステップの中でタスクを実行する際は、現在の TestStand の実行状況を監視する必要があります。実行が終了または中止している場合、ステップモジュール内で実行しているタスクを中止する必要があります。

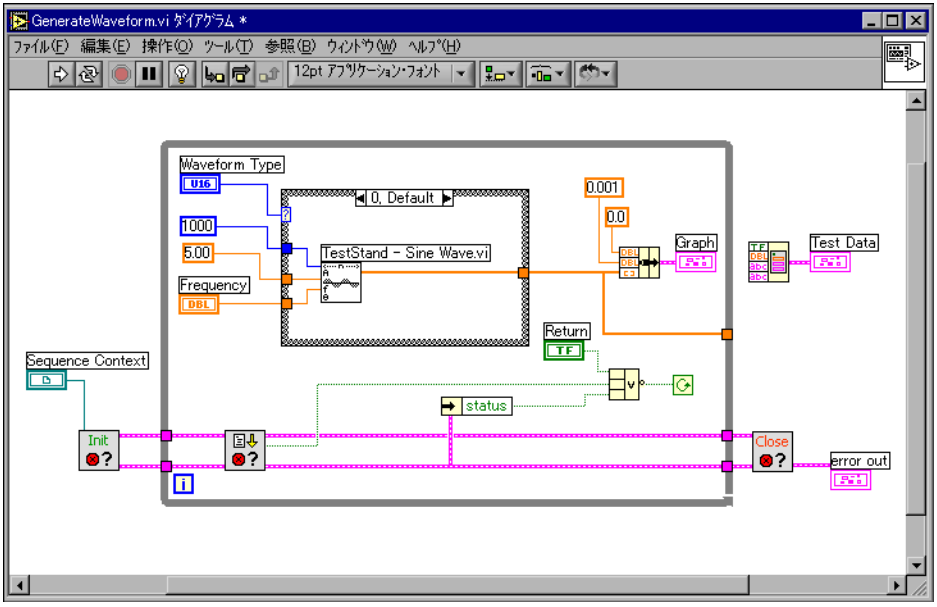

図 10-6　GenerateWaveform.vi ブロックダイアグラム

図 10-7 に示すように、LabVIEW のユーザライブラリから TestStand の関数パレットにアクセスすることができます。

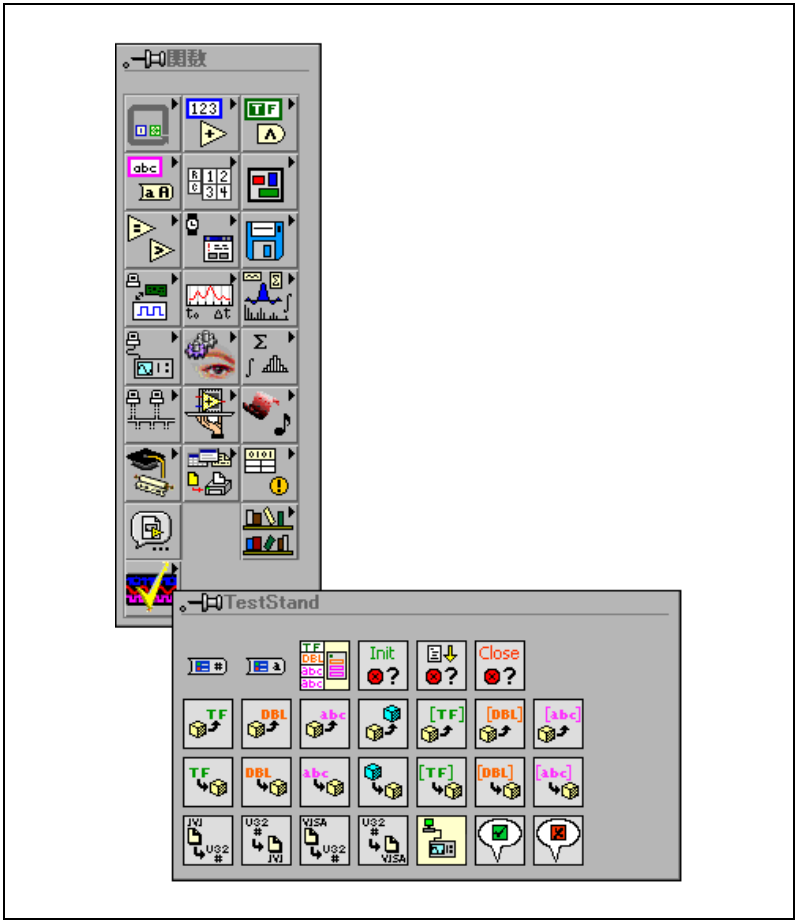

図 10-7　TestStand 関数パレット

b. 2 つの TestStand VI、`Set Property Value (Numeric Array).vi` と `Create Test Data Cluster.vi` を使用して、図 10-8 に示すようにブロックダイアグラムを更新します。

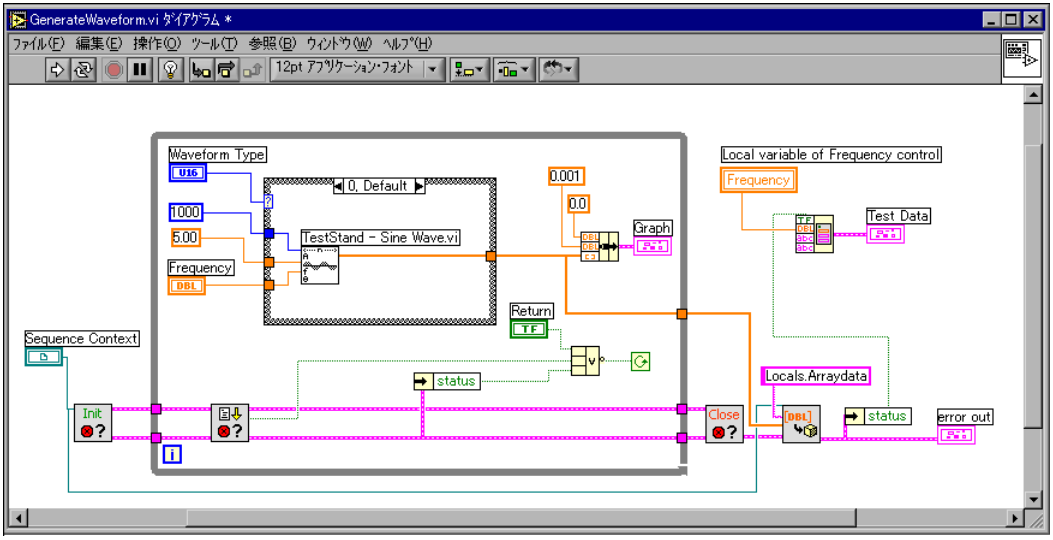

図 10-8　GenerateTestStandWaveform.vi ブロックダイアグラム

メモ　TestStand 関数パレットには、最も一般的に使用される TestStand ActiveX API メソッドおよびプロパティのラッパ VI のみが含まれています。ActiveX API 全般の詳細については、「TestStand Programmer Help」を参照してください。

c. LabVIEW で**ファイル→保存**を選択して、テスト VI への変更を保存します。テスト VI は LabVIEW で開いたままにしておきます。

19. シーケンスエディタの「LabVIEW VI コールを編集」ダイアログボックスに戻ります。
20. **参照**ボタンをクリックして `TestStand\Tutorial` ディレクトリの `GenerateTestStandWaveform.vi` を選択し、ステップの VI モジュールを再度指定します。
21. **OK** をクリックして「LabVIEW VI コールを編集」ダイアログボックスを閉じます。
22. 次の手順で、TestStand 変数に保存したデータをプロットする 2 つめのステップを作成します。
    a. ステップリストの Get Array Data From LabVIEW ステップの下の部分を右クリックして、**ステップを挿入→アクション**を選択し新規アクションステップを挿入します。
    b. 新規ステップに `Display Array Data in LabVIEW` という名前を付けます。

c. 新規ステップを右クリックしてコンテキストメニューから**モジュールを指定**を選択し、「LabVIEW VI コールを編集」ダイアログボックスを表示します。

d. VI パス名制御器に `DisplayWaveform.vi` と入力します。

e. 「シーケンスコンテキスト ActiveX ポインタ」と「呼び出されたら VI のフロントパネルを表示」の各オプションにチェックを付けます。

f. **VI を作成**ソタンをクリックして、LabVIEW でテスト VI を作成し、開きます。シーケンスエディタは、新規 VI を作成するディレクトリを選ぶようプロンプトします。

g. `TestStand\Tutorial` ディレクトリを開きます。新規 VI にはすでに Sequence Contest、Test Data、および error out の各制御器が含まれていることに注意してください。

h. 図 10-9 に示すように、VI のフロントパネルに波形グラフ制御器と Dialog ボタンを追加します。

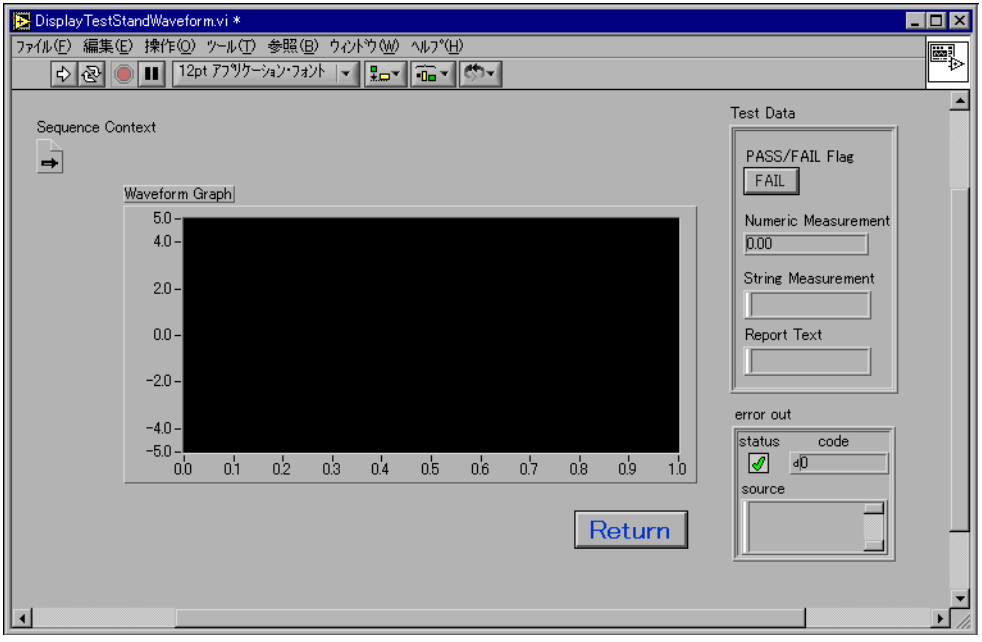

図 10-9　DisplayTestStandWaveform.vi フロントパネル

i. 5 つの TestStand VI、`Get Property Value (Numeric Array).vi`、`Create Test Data Cluster.vi`、`InitializeTerminationMonitor.vi`、`GetMonitorStatus.vi`、および `CloseTermination Monitor.vi` を使用して、図 10-10 に示すようにブロックダイアグラムを更新します。

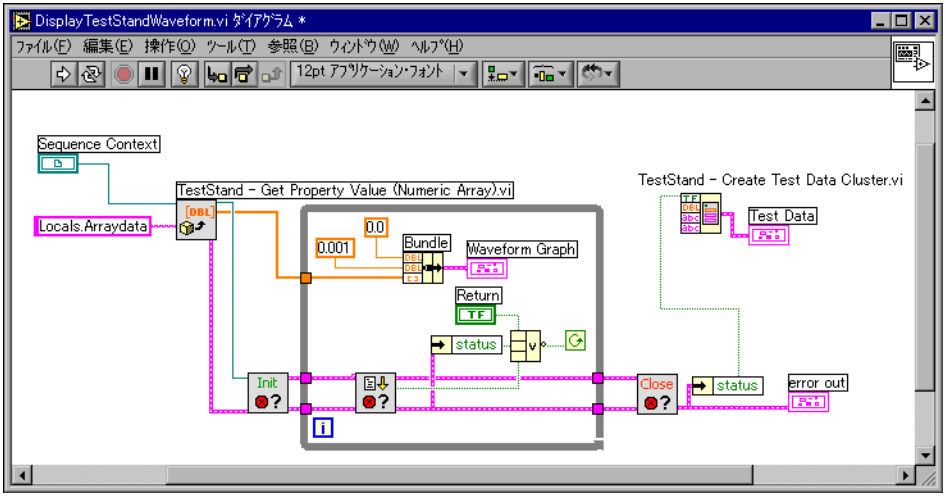

図 10-10　DisplayTestStandWaveform.vi ブロックダイアグラム

j. LabVIEW で**ファイル→保存**を選択して、テスト VI への変更を保存します。テスト VI は LabVIEW で開いたままにしておきます。

23. シーケンスエディタの「LabVIEW VI コールを編集」ダイアログボックスに戻ります。
24. **OK** をクリックしてダイアログボックスを閉じます。
25. メインのシーケンスエディタウィンドウで、**ファイル→保存**を選択してシーケンスファイルの変更を保存します。

シーケンスを実行する

シーケンスを実行して、コードモジュールが配列データを正しく生成し表示していることを確認します。

1. **実行→ MainSequence を実行**を選択してシーケンスを実行します。
2. 2 つの VI テストのフロントパネルが表示されたら、**Return** ボタンをクリックして続行します。2 つめの VI のフロントパネルには、最初のフロントパネルで指定した波形が表示されていることに注目してください。

3. 実行が完了したら実行ウィンドウを閉じます。
4. シーケンスを再度実行して、次の手順で TestStand 配列変数に保存されたデータを表示します。
   a. シーケンスファイルウィンドウでステップを右クリックしてコンテキストメニューから**ブレークポイントのトグル**を選択し、Display Array Data in LabVIEW ステップにブレークポイントを設定します。
   b. **実行→ MainSequence を実行**を選択してシーケンスをもう一度実行します。
   c. 最初の VI テストのフロントパネルが表示されたら、**Return** ボタンをクリックして続行します。
   d. 実行がブレークポイントで停止したら、図 10-11 に示すように、コンテキストタブをクリックして `Locals.Arraydata` 変数を選択します。配列の内容の 0 以外の値に注意してください。

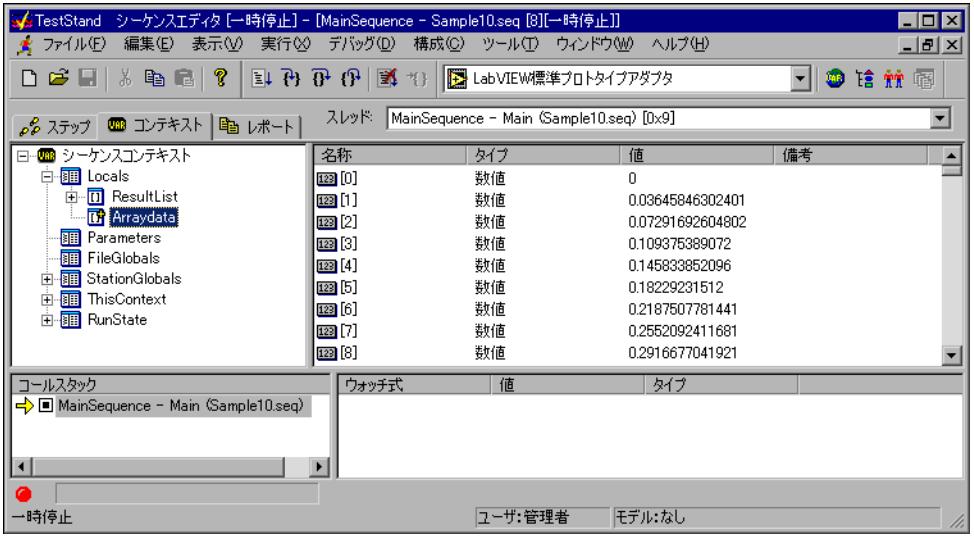

図 10-11　Locals.Arraydata

5. **デバッグ→再開**を選択して続行します。
6. 2 つめの VI テストのフロントパネルが表示されたら、**Return** ボタンをクリックして実行を完了します。
7. 実行が完了したら実行ウィンドウを閉じます。
8. シーケンスエディタのすべてのウィンドウを閉じます。

コードモジュールを LabWindows/CVI で開発している場合は、本章の次のセクションへ進んで「サンプルをセットアップする」セクションを完了してください。それ以外の場合は、本書の第 11 章、「上級開発機能」に進んでください。

LabWindows/CVI コードモジュールで ActiveX を使用する

この練習では、データ配列を生成してそのデータをプロットする 2 つの LabWindows/CVI ステップを持つ新規のシーケンスを作成します。このチュートリアルセッションでは、LabWindows/CVI 開発環境の使用経験があることを前提としています。

メモ　TestStand に対応する正しいバージョンの LabWindows/CVI を使用していることを確認してください。詳細については、`TestStand\Doc` ディレクトリの `readme.txt` ファイルを参照してください。

サンプルをセットアップする

このチュートリアルセッションを完了するためには、シーケンスエディタ内のウィンドウをすべて閉じます。

シーケンスとテストを作成する

この練習では、シーケンスエディタで新規のシーケンスを作成します。

1. **ファイル→新規シーケンスファイル**を選択して新規シーケンスを開きます。
2. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample11.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。ここでシーケンスファイルを保存すると、コードモジュールの絶対パスではなく相対パスを入力することができます。
3. シーケンスファイルウィンドウのローカル変数タブをクリックします。

4. 図 10-12 に示すように、右側のペーンで右クリックして数値配列変数を挿入します。

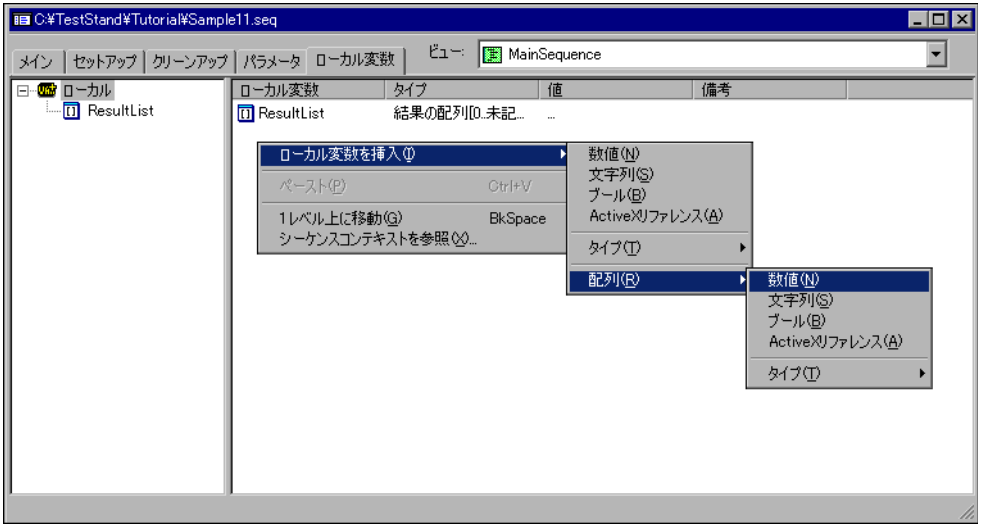

図 10-12　数値のローカル変数配列を挿入

するとシーケンスエディタは、配列範囲ダイアログボックスを表示します。

5. 図 10-13 に示すように、上限制御器に `1023` という値を入力します。

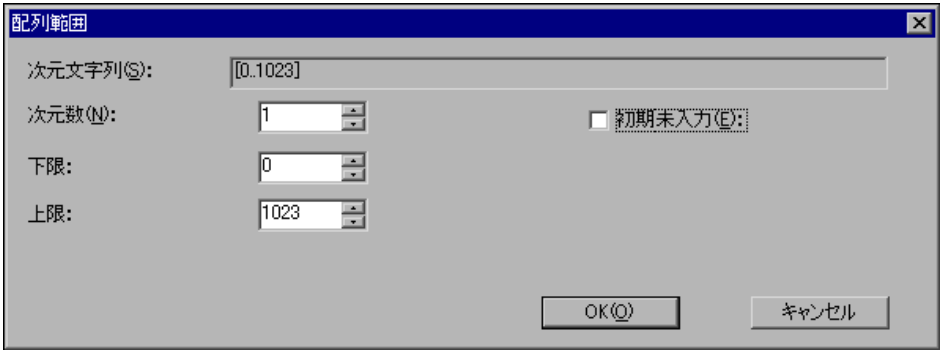

図 10-13　配列範囲ダイアログボックス

6. **OK** をクリックしてダイアログボックスを閉じます。
7. 変数を右クリックして、コンテキストメニューから**名前変更**を選択します。

8. 変数に `Arraydata` という名前を付けます。
9. シーケンスファイルウィンドウのメインタブをクリックします。アダプタ選択リング制御器で C/CVI 標準プロトタイプアダプタを選択します。
10. ステップリストを右クリックして、**ステップを挿入→アクション**を選択し新規アクションステップを挿入します。
11. ステップに `Get Array Data From CVI` という名前を付けます。
12. 新規のステップを右クリックして、コンテキストメニューから**モジュールを指定**コマンドを選択します。
13. モジュールタブのモジュールタイプリング制御器で、Dynamic Link Library（`*.dll`）を選択します。
14. モジュールパス名制御器に `UsingActiveXInCVI.DLL` と入力します。
15. 関数名制御器には、`GenerateTestStandWaveform` と入力します。
16. 「シーケンスコンテキストを渡す」オプションにチェックを付けます。
17. ソースコードタブで、**コードを作成**ボタンをクリックします。アダプタがプロジェクトのパス名を入力するようプロンプトした場合は、`TestStand\Tutorial` ディレクトリの `UsingActiveXInCVI.prj` を入力します。
18. ソースファイルのパス名を入力するようプロンプトするダイアログボックスが表示されたら、`UsingActiveXInCVI.c` と入力します。

    パス名を入力したら、TestStand は以下の動作を行います。

    1. LabWindows/CVI の外部インスタンスを起動。
    2. LabWindows/CVI で新規プロジェクトファイルを作成。
    3. ソースファイルを作成。
    4. ソースファイルと TestStand サポート計測器ドライバをプロジェクトに追加。
    5. ソースファイルにテンプレートの `GenerateTestStandWaveform` 関数を生成。

図 10-14 は、ソースファイルで生成された関数を示します。

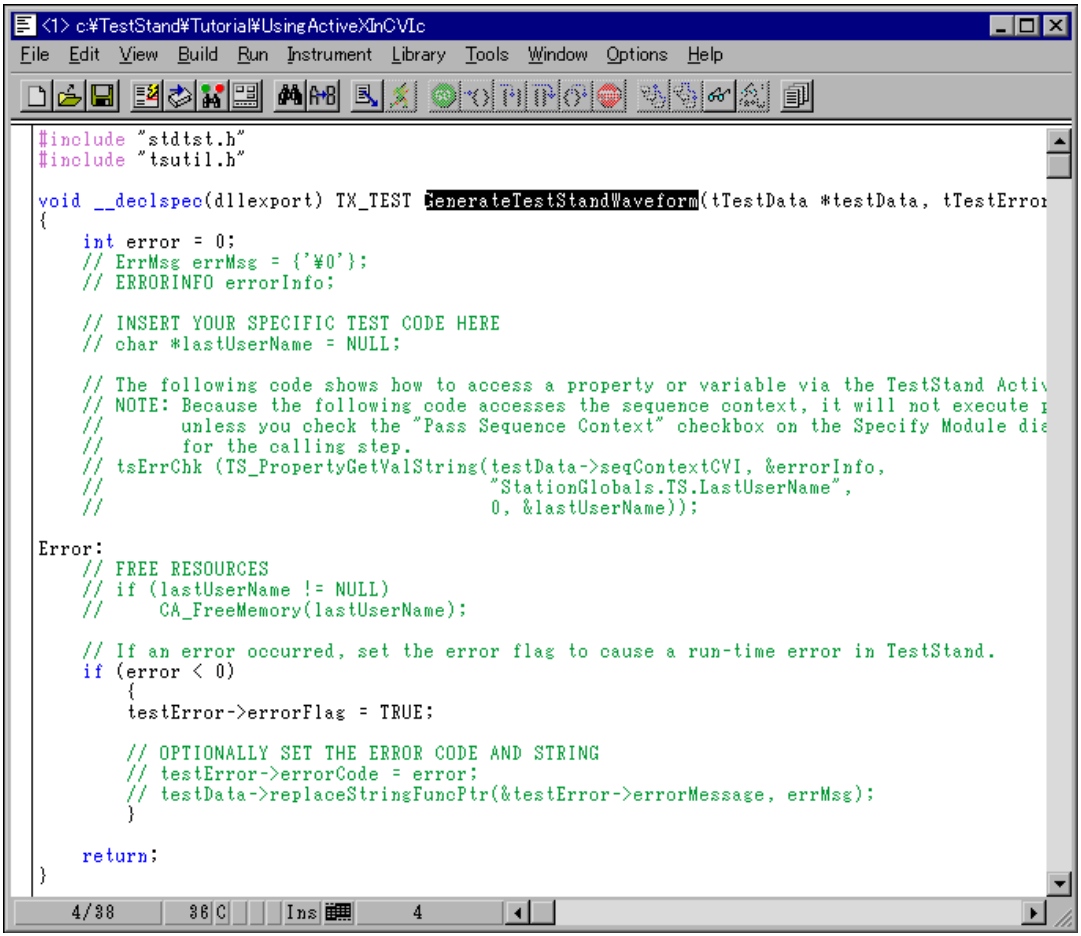

図 10-14　生成された GenerateTestStandWaveform のソース

メモ　このチュートリアルセッションを他の人が以前に完了している場合、プロジェクトとソースファイルがすでに存在することがあります。ファイルの既存のコピーはすべて入れ換えます。LabWindows/CVI が既存の関数を入れ換えるようプロンプトした場合は、**入れ換え**ボタンをクリックしてこのチュートリアルセッションを続行します。

19. `GenerateTestStandWaveform` 関数を更新して、配列を初期化する際の正弦サイクル数を入力するようオペレータにプロンプトするようにします。次の手順で、関数のソースコードを更新します。変更された行は太字で示しています。

```
void __declspec(dllexport) TX_TEST
  GenerateTestStandWaveform(tTestData *testData,
  tTestError *testError)
{
    int error = 0;
    ErrMsg errMsg = {"Error occurred generating
      waveform"};
    ERRORINFO errorInfo;
    int i;
    double cycles = 2.0;
    char buffer[32];
    double Arraydata[1024];
    VARIANT variantData;
    // Prompt for number of cycles
    PromptPopup ("Frequency", "Please input number of
    cycles for the array?", buffer, 32);
    sscanf(buffer, "%lf", &cycles);
    // Initialize C array
    for (i=0; i<1024; i++)
        Arraydata[i] = sin((2*3.14) * i * cycles / 1024);
    // Copy C array to VARIANT
    CA_VariantSet1DArray (&variantData, CAVT_DOUBLE,
      1024, Arraydata);
    // The following code shows how to access a
    // property or variable via the TestStand
    // ActiveX API
    tsErrChk (TS_PropertySetValVariant
      (testData->seqContextCVI, &errorInfo,
      "Locals.Arraydata", 0, variantData));
Error:
    // FREE RESOURCES
    // If an error occurred, set the error flag to
    //   cause a run-time error in TestStand.
    if (error < 0)
    {
        testError->errorFlag = TRUE;
        // OPTIONALLY SET THE ERROR CODE AND STRING
```

```
        testError->errorCode = error;
        testData->replaceStringFuncPtr(&testError->
          errorMessage, errMsg);
    }
    return;
}
```

20. **Build → Compile File** を選択してソースコードをコンパイルし、正しく変更されていることを確認します。
21. コンパイルが正しく行われたらソースコードを保存します。
22. TestStand から配列データを取得してそのデータをプロットするもう 1 つの関数を追加します。ソースファイルに次の関数を追加入力します。

```
void __declspec(dllexport) TX_TEST
  DisplayTestStandWaveform(CAObjHandle   seqContextCVI)
{
    int error = 0;
    ErrMsg errMsg = {'\0'};
    ERRORINFO errorInfo;

    int elements = 0;
    double *Arraydata = NULL;
    VARIANT variantData;

    // The following code shows how to access a
    // property or variable via the TestStand
    // ActiveX API
    tsErrChk (TS_PropertyGetValVariant
      (seqContextCVI, &errorInfo, "Locals.Arraydata",
      0, &variantData));

    // Copy C array to VARIANT
    CA_VariantGet1DArray (&variantData, CAVT_DOUBLE,
      &Arraydata, &elements);

    WaveformGraphPopup ("Waveform From TestStand",
      Arraydata, elements, VAL_DOUBLE, 1.0, 0.0,
      0.0, 1.0);

Error:
    if (Arraydata)
         CA_FreeMemory(Arraydata);

    return;

}
```

23. **Build → Compile File** を選択してソースコードをコンパイルし、正しく変更されていることを確認します。

24. コンパイルが正しく行われたらソースコードを保存します。
25. DLL を構築するには、Project ウィンドウで **Build → Create Debuggable Dynamic Link Library** を選択します。
26. DLL の構築が完成したら、シーケンスエディタに戻ります。
27. **OK** をクリックして C/CVI モジュールコールを編集ダイアログボックスを閉じ、メインステップグループのビューに戻ります。

メモ　DLL 作成の際に LabWindows/CVI がファイル許可エラーを返した場合は、シーケンスエディタに戻って**ファイル**メニューから「すべてのモジュールの解放」を選択します。すると TestStand は、DLL、VI、およびアダプタがロードした他のすべてのモジュールを含む、すべてのステップのコードモジュールを解放します。LabWindows/CVI に戻って DLL を再構築します。

28. アダプタ選択リング制御器で、DLL フレキシブルプロトタイプアダプタを選択します。
29. ステップリストの Get Array Data From CVI ステップの下の部分を右クリックして、**ステップを挿入→アクション**を選択し新規アクションステップを挿入します。
30. ステップに `Display Data In CVI` という名前を付けます。
31. 新規のステップを右クリックして、コンテキストメニューから**モジュールを指定**コマンドを選択します。
32. モジュールタブで、DLL パス名制御器の隣の**参照**をクリックします。
33. 作成した `UsingActiveXInCVI.dll` を選択します。
34. 関数名リング制御器で `DisplayTestStandWaveform` 関数を選択します。その関数では、DLL にパラメータ情報がないという内容のメッセージが表示されます。
35. **新規**ボタンをクリックして、関数のパラメータを作成します。
36. `arg1` パラメータに `seqContextCVI` という名前を付けます。
37. カテゴリリング制御器でオブジェクトを選択します。
38. オブジェクトタイプリング制御器で CVI ActiveX Automation Handle を選択します。
39. 値式制御器に `ThisContext` という式を入力します。

メモ　このセッションは、コードモジュール内でのシーケンスコンテキストの使用方法の説明を目的としています。DLL フレキシブルプロトタイプアダプタを使用すると、配列を関数へのパラメータとして DLL に直接渡すことができます。

40. **OK** をクリックして「DLL コールを編集」ダイアログボックスを閉じます。
41. メインのシーケンスエディタウィンドウで、**ファイル→保存**を選択してシーケンスファイルの変更を保存します。

シーケンスを実行する

この練習では、シーケンスを実行して、コードモジュールが配列データを正しく生成し表示していることを確認します。

1. **実行→ MainSequence を実行**を選択してシーケンスを実行します。
2. 配列データに対して生成する正弦サイクル数に、`5` の値を入力します。
3. **OK** ボタンをクリックして続行します。
4. 2 つめのステップでグラフが表示されたら、**OK** をクリックして続行します。
5. 実行が完了したら実行ウィンドウを閉じます。
6. シーケンスを再度実行して、次の手順で TestStand の配列変数に保存されたデータを表示します。
   a. シーケンスファイルウィンドウでステップを右クリックしてコンテキストメニューから**ブレークポイントのトグル**を選択し、`Display Data In CVI` ステップにブレークポイントを設定します。
   b. **実行→ MainSequence を実行**を選択してシーケンスをもう一度実行します。
   c. 配列で生成する正弦サイクル数に、`5` の値を入力します。
   d. **OK** ボタンをクリックして続行します。
   e. 実行がブレークポイントで停止したら、コンテキストタブをクリックします。

f. 図 10-15 に示すように、`Locals.Arraydata` 変数を選択します。配列の内容の 0 以外の値に注意してください。

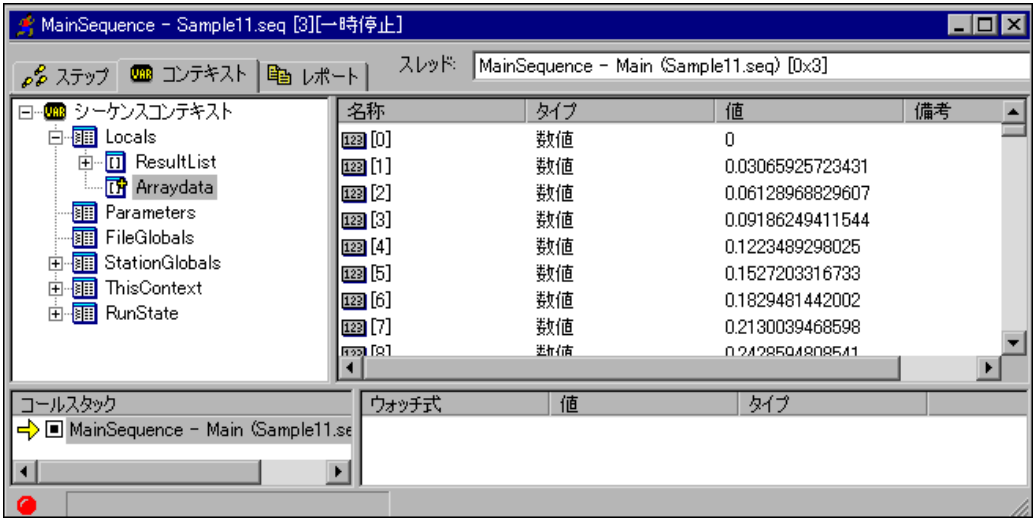

図 10-15　Locals.Arraydata の値

7. **デバッグ→再開**を選択します。
8. グラフウィンドウが表示されたら、**OK** をクリックして続行します。
9. 実行が完了したら実行ウィンドウを閉じます。
10. シーケンスエディタのすべてのウィンドウを閉じます。

このチュートリアルセッションはこれで終わりです。次のセッションでは、シーケンスの開発およびデバッグの際に使用するさらに上級の TestStand 機能について学習します。

11

上級開発機能

本章では、シーケンスの開発およびデバッグの際に使用できるさらに上級の機能について説明します。また、ステップを対話式に実行する方法や、シーケンスを名前によってダイナミックに呼び出す方法についても説明します。

サンプルをセットアップする

このチュートリアルセッションを完了するためには、シーケンスエディタ内のウィンドウをすべて閉じます。

対話式実行

この練習では、シーケンスファイルウィンドウから選択したステップを実行して、ブレークポイントで実行を一時停止した状態でステップを対話式に実行します。

選択したステップを別の実行として実行する

この練習では、選択したステップをシーケンスファイルウィンドウで実行します。

1. `TestStand\Tutorial` ディレクトリから `Sample1.seq` を開きます。
2. シーケンスファイルを開いたら、ステップのアイコンの左側をクリックするか、ステップを右クリックしてコンテキストメニューから**ブレークポイントのトグル**を選択し、Power On ステップにブレークポイントを設定します。
3. <Ctrl> キーを押したまま Power On、ROM、および ROM Diagnostics の各ステップをクリックして、それらのステップを選択します。

4. すると、シーケンスファイルウィンドウは 図 11-1 に示すようになります。

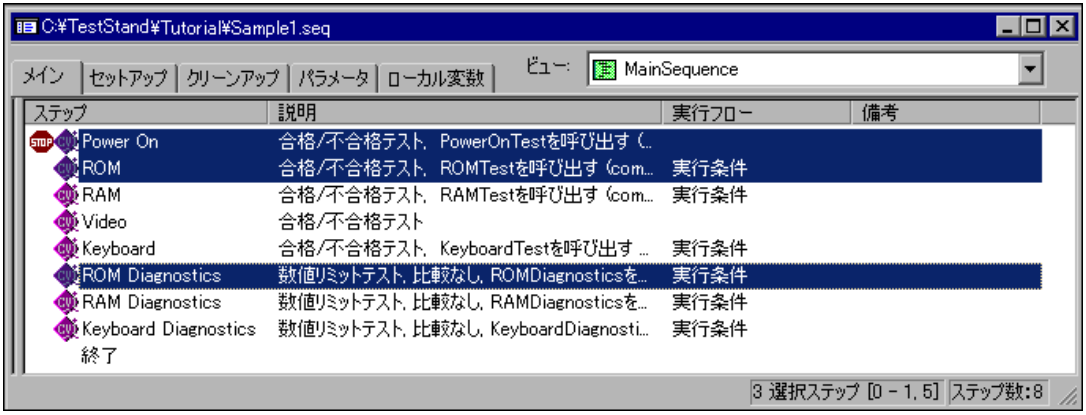

図 11-1　シーケンスファイルウィンドウで複数のステップを選択

5. **実行→選択ステップの実行**を選択します。すると、TestStand は新規の実行を開始します。

   シーケンスファイルウィンドウから選択したステップを実行すると、TestStand はデフォルトでセットアップおよびクリーンアップステップグループとメインステップグループ内のステップを実行します。（ステーションオプションの実行タブで、セットアップとクリーンアップを実行するか選択できます。）

6. Test Simulator ダイアログボックスが表示されたら、**Done** をクリックしてダイアログボックスを閉じます。

7. 図 11-2 に示すように、実行はここで Power On ステップのブレークポイントに入ります。

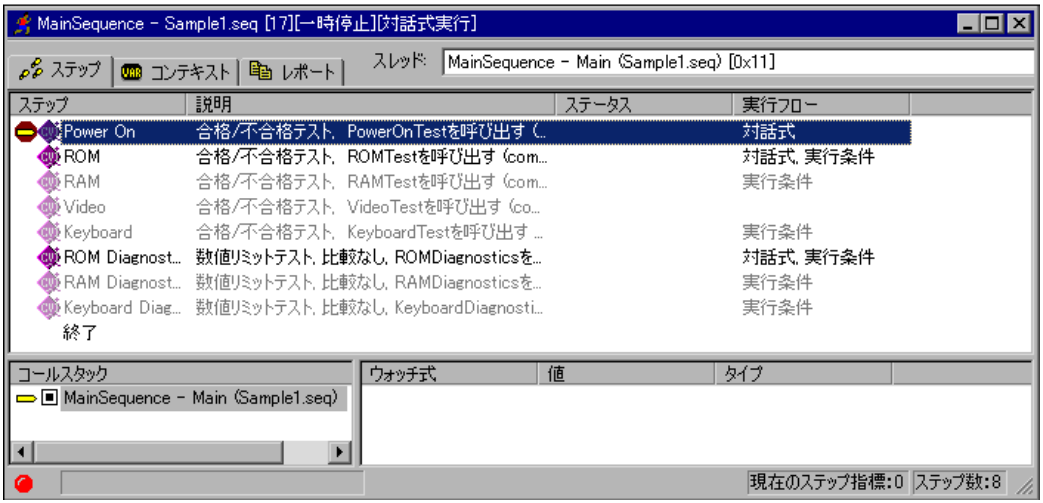

図 11-2　対話式実行でのブレークポイント

対話式実行のポインタは、通常の実行の黄色い矢印ではなく、細い矢印になっていることに注意してください。

8. **デバッグ→ステップオーバー**を選択してシングルステップ処理を 2 回行います。すでに選択したステップのみが実行され、選択していないステップのアイコンはグレー表示になっていることに注意してください。
9. **デバッグ→再開**を選択して実行を完了します。TestStand は ROM Diagnostics ステップの実行条件を無視し、ROM ステップが合格しても ROM Diagnostics ステップを実行していることに注目してください。
10. 実行が完了したら実行ウィンドウを閉じます。
11. 手順 3 から 10 を繰り返します。ただし今回は手順 5 で**実行→エントリポイントを使用して選択ステップを実行→一回実行**を選択します。TestStand は、プロセスモデルエントリポイントの一回実行を使用してステップを実行し、UUT レポートを生成します。

実行中に選択したステップを実行する

この練習では、ブレークポイントで実行を一時停止した状態で、選択したステップを対話式に実行します。

1. **実行→一回実行**を選択して、新規の実行を開始します。
2. Test Simulator ダイアログボックスが表示されたら、ROM test を選択して不合格にします。
3. **Done** をクリックしてダイアログボックスを閉じます。実行は Power On ステップのブレークポイントに入ります。
4. **デバッグ→ステップオーバー**を選択して、RAM Diagnostics ステップに到達するまで実行を継続します。ROM ステップが不合格になります。
5. ステップのアイコンの左側をクリックするか、ステップを右クリックしてコンテキストメニューから**ブレークポイントのトグル**を選択し、実行ウィンドウで ROM ステップにブレークポイントを設定します。
6. <Ctrl> キーを押したまま ROM ステップと ROM Diagnostics ステップをクリックし、それらのステップを選択します。
7. 図 11-3 に示すように、ROM Diagnostics ステップを右クリックしてコンテキストメニューから**選択ステップでループ**を選択します。

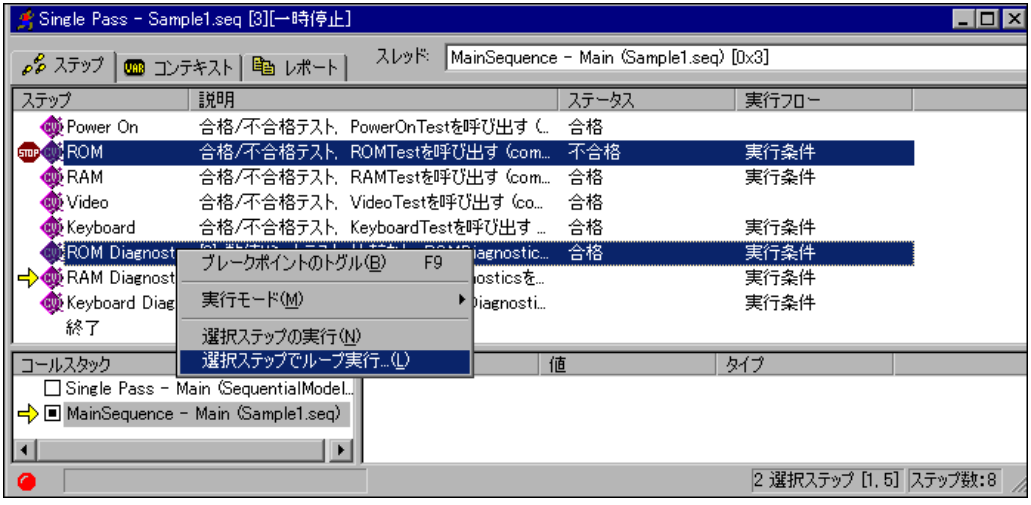

図 11-3　実行中に選択したステップでループ

8. 「選択ステップのループ実行」ダイアログボックスで、ループカウント制御器に `100` と入力します。
9. **OK** をクリックしてダイアログボックスを閉じます。シーケンスエディタは選択したステップの対話式実行を開始し、ROM ステップのブレークポイントで一時停止状態になります。黄色い矢印はまだ

RAM Diagnostics ステップにあり、新規の細い矢印が ROM ステップを指していることに注意してください。

10. **デバッグ→ステップオーバー**を選択して対話式実行をシングルステップ処理します。すると対話式実行は ROM ステップと ROM Diagnostics ステップの間で切り替わります。
11. ROM ステップのステータスは、引き続き不合格となります。対話式実行を 100 回ループで実行する前に、**デバッグ→対話実行の終了**を選択します。すると、実行は RAM Diagnostics ステップで一時停止状態に戻ります。
12. 次の手順で、現在一時停止しているステップ以外のステップから実行を強制的に継続します。
    a. ROM ステップをクリックして、このステップのみをハイライトします。
    b. ROM ステップを右クリックして、コンテキストメニューから**次のステップの設定**を選択します。黄色い矢印が RAM Diagnostics ステップから ROM ステップに移動していることを確認してください。
    c. **デバッグ→ステップオーバー**を選択して一回シングルステップ処理し、RAM Diagnostics ステップではなく ROM ステップが実行されることに注意してください。
13. **デバッグ→再開**を選択して実行を完了します。

    レポートファイルが表示されたら、対話式に実行された各ステップに対する項目が含まれていることを確認します。
14. シーケンスエディタ内のすべてのウィンドウを閉じます。シーケンスファイルへの変更は保存しません。

シーケンスをダイナミックに呼び出してパラメータを渡す

次の練習では、2 つのシーケンスのうちの 1 つをダイナミックに選択して実行するステップをシーケンスに追加します。

ステップをシーケンスに追加する

この練習では、既存のシーケンスを開いて、テストする CPU のタイプと数を入力するようオペレータにプロンプトするステップと、ユーザが指定した CPU タイプによって 2 つの異なるシーケンスのうちの 1 つを呼び出すステップを追加します。

1. `TestStand\Tutorial` ディレクトリから `Sample1.seq` を開きます。
2. シーケンスウィンドウの**ローカル変数**タブをクリックします。

3. 右側のペーンを右クリックして、コンテキストメニューから**ローカル変数を挿入→文字列**を選択します。
4. ローカル変数に `CPUType` という名前を付けます。
5. シーケンスファイルウィンドウのメインタブをクリックして、メインステップグループのステップを表示します。
6. Power On ステップを右クリックして、コンテキストメニューから**ステップを挿入→メッセージポップアップ**を選択します。
7. 新規のステップに `Select CPU Type` という名前を付けます。
8. 新規ステップを右クリックして、コンテキストメニューから**メッセージの設定を編集**を選択します。
9. 「メッセージボックスステップの構成」ダイアログボックスの「テキストとボタン」タブで、次の制御器値を変更します。

タイトル式	`"Select CPU"`
メッセージ式	`"Please select the CPU Type for the UUT."`
ボタン 1	`"INTEL CPU"`
ボタン 2	`"AMD CPU"`
キャンセルボタン	なし

10. **OK** をクリックして「メッセージボックスステップを構成」ダイアログボックスを閉じます。
11. オプションタブを選択して「モーダル」オプションにチェックを付けます。このオプションを有効にすると、メッセージポップアップダイアログボックスがシーケンスエディタウィンドウの後ろに隠れるのを防ぎ、ユーザがメッセージポップアップダイアログボックスを閉じるまでシーケンスエディタを操作しないようにすることができます。
12. Select CPU Type ステップを右クリックして、コンテキストメニューから**プロパティ**を選択します。
13. 式タブをクリックします。
14. ポストステップ式制御器に次の式を入力します。ポストステップ式制御器の隣の**参照**ボタンをクリックし、式参照ダイアログボックスを使って式を作成することもできます。また式参照ダイアログボックスでは、?: などの条件演算子の説明を参照することもできます。

    ```
    Locals.CPUType = ((Step.Result.ButtonHit == 2) ? "AMD" : "INTEL")
    ```

    この式により、ユーザがどちらのボタンをクリックしたかよって、文字列値 `"AMD"` または `"INTEL"` のいずれかがローカル変数に割り当てられます。
15. **OK** をクリックしてプロパティダイアログボックスを閉じます。

16. Select CPU Type ステップを右クリックして、コンテキストメニューから**ステップを挿入→メッセージポップアップ**コマンドを選択します。
17. 新規ステップに `Specify Number of CPUs` という名前を付けます。
18. 新規ステップを右クリックして、コンテキストメニューから**メッセージの設定を編集**コマンドを選択します。
19. 「メッセージボックスステップを構成」ダイアログボックスで次の制御器値を変更します。

タイトル式	`"Number of CPUs"`
メッセージ式	`"Please select the number of CPUs installed for the UUT."`
ボタン 1	`"1"`
ボタン 2	`"2"`
ボタン 3	`"3"`
ボタン 4	`"4"`
キャンセルボタン	なし

20. オプションタブを選択して「モーダル」オプションにチェックを付けます。
21. **OK** をクリックして「メッセージボックスステップを構成」ダイアログボックスを閉じます。
22. Specify Number of CPUs ステップを右クリックして、コンテキストメニューから**ステップを挿入→シーケンスコール**を選択します。
23. `CPU Test` ステップの名前を変更します。
24. CPU Test ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。
25. 「パス名とシーケンスの式を指定」オプションにチェックを付けます。
26. ファイルパス名とシーケンスの各式制御器に次の値を入力します。

ファイルパス名	`Locals.CPUType + "Processor.seq"`
シーケンス	`"MainSequence"`

27. **プロトタイプをロード**ボタンをクリックして、シーケンスコールのプロトタイプを選択します。
28. 「シーケンスプロトタイプをロード」ダイアログボックスの**参照**ボタンをクリックします。
29. `AMDProcessor.seq` シーケンスファイルを選択します。
30. **OK** を 2 回クリックして、「シーケンスファイルを選択」および「シーケンスプロトタイプをロード」ダイアログボックスを閉じます。

    TestStand では、パラメータセクションにシーケンスのパラメータリストが含まれています。

31. `CPUsInstalled` パラメータをクリックします。
32. 「式を入力」オプションを選択します。
33. 文字列制御器に次の式を入力するか、あるいは**参照**をクリックして式参照ダイアログボックスでプロパティを選択します。

```
RunState.Sequence.Main["Specify Number of CPUs"]
  .Result.ButtonHit
```

図 11-4 は、完成したシーケンスコールを編集ダイアログボックスを示します。

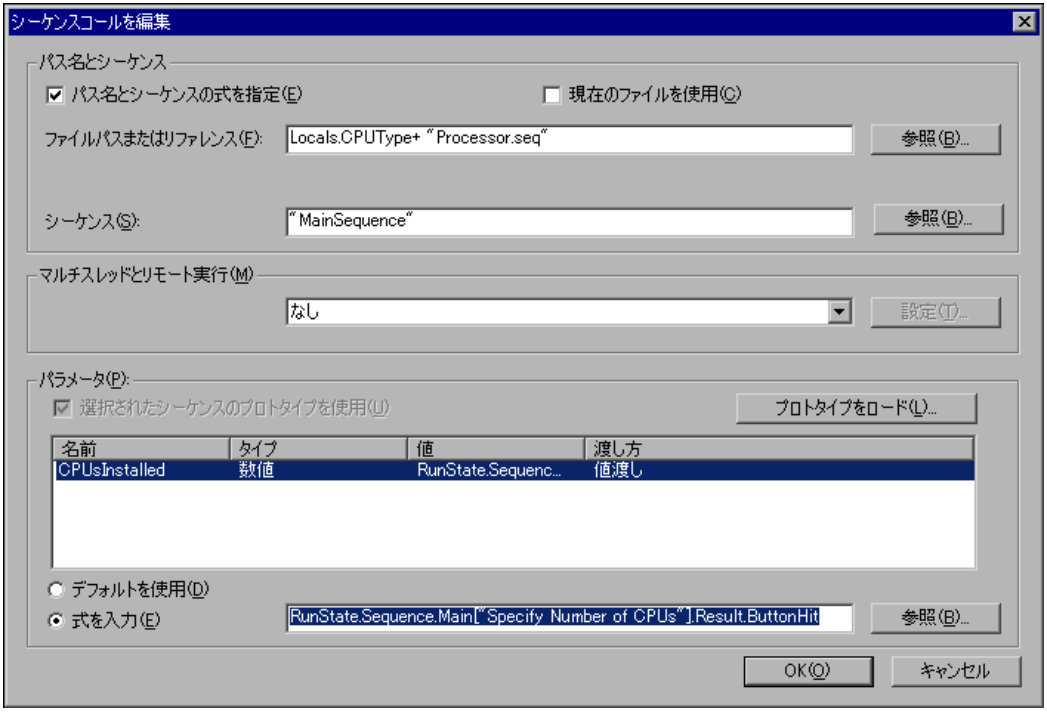

図 11-4　式によってダイナミックに呼び出す

34. **OK** をクリックしてダイアログボックスを閉じます。
35. **ファイル→別名で保存**を選択して、シーケンスを `Sample12.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。

図 11-5 は、完成したシーケンスを示します。

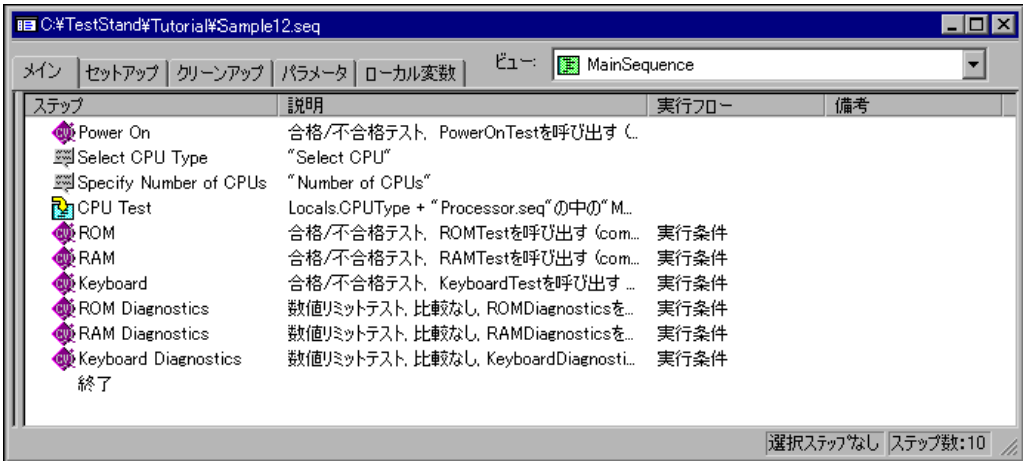

図 11-5　シーケンスをダイナミックに呼び出す

シーケンスを実行する

以下の手順にしたがって、シーケンスをダイナミックに実行します。

1. ステップのアイコンの左側をクリックするか、ステップを右クリックしてコンテキストメニューから**ブレークポイントのトグル**を選択し、CPU Test ステップにブレークポイントを設定します。
2. **実行→一回実行**を選択します。
3. Test Simulator プロンプトで **Done** をクリックします。
4. Select CPU プロンプトで **INTEL CPU** ボタンをクリックします。
5. Number of CPUs プロンプトで **2** ボタンをクリックします。
6. 実行が CPU Test ステップのブレークポイントで一時停止したら、**デバッグ→ステップイン**を選択してサブシーケンスにシングルステップします。図 11-6 に示すように、コールスタックペーンで、`INTELProcessor.seq` がシーケンスコールスタックの一番下に表示されていることに注意してください。

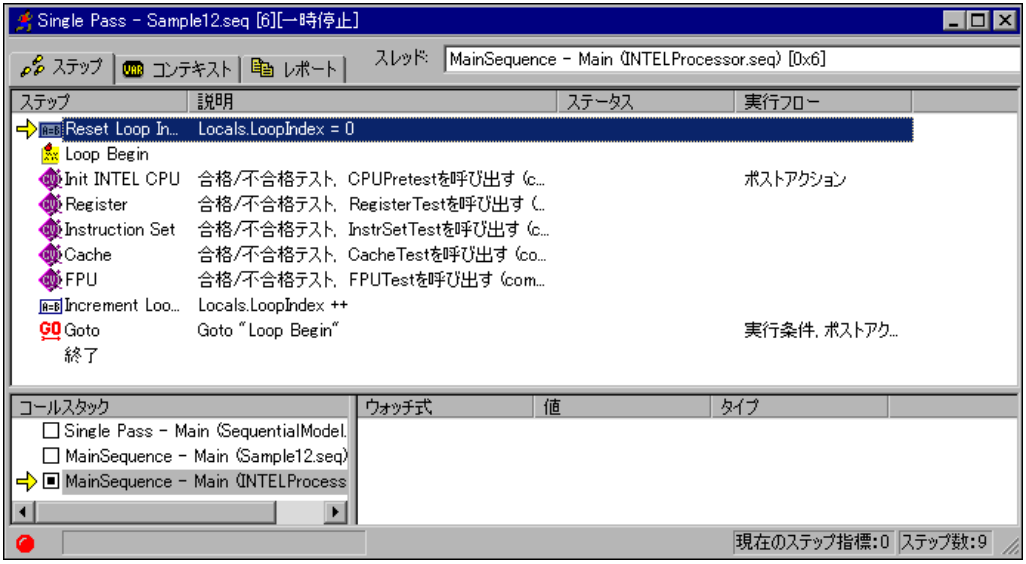

図 11-6　コールスタックペーンに表示された INTELProcessor.seq

7. コンテキストタブをクリックして、図 11-7 に示すように、シーケンスの 2 つのパラメータの値を確認してください。

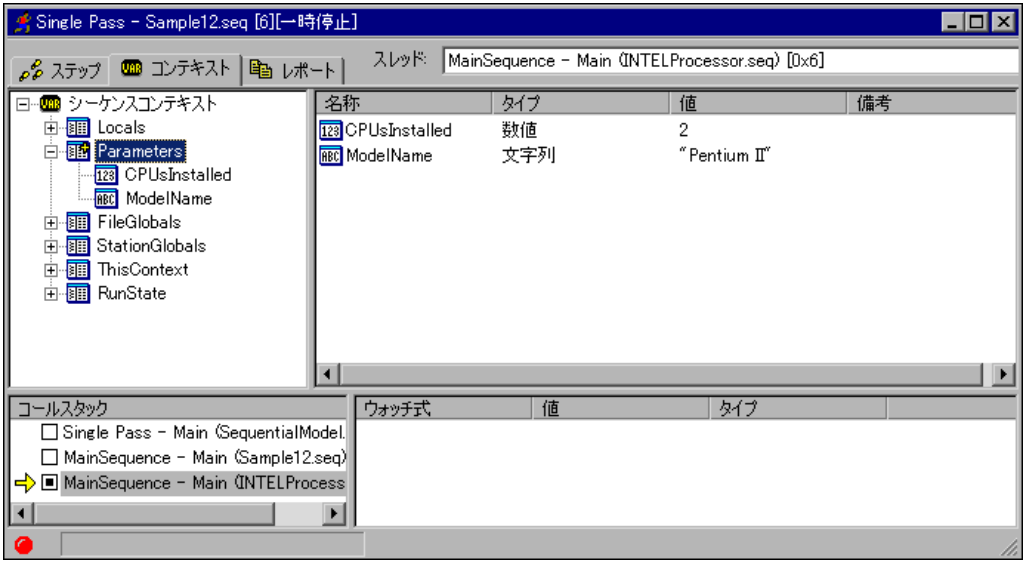

図 11-7　コンテキストタブのシーケンスパラメータ

CPUsInstalled パラメータの値は Specify Number of CPUs プロンプトでクリックしたボタンの値と同じになっています。INTELProcessor.seq の MainSequence にはまた ModelName パラメータが必要です。作成したシーケンスコールステップではこのパラメータを指定していないため、エンジンはそのパラメータ値をデフォルトに初期化します。

8. **デバッグ→再開**を選択して実行を完了します。
9. レポートファイルが表示されたら、レポートを確認します。ただし実行ウィンドウは閉じません。
10. **実行→再実行**を選択して、再度実行します。
11. Test Simulator プロンプトで **Done** をクリックします。
12. Select CPU プロンプトで **AMD CPU** ボタンをクリックします。
13. Number of CPUs プロンプトで **3** ボタンをクリックします。
14. 実行が CPU Test ステップのブレークポイントで一時停止したら、**デバッグ→ステップイン**を選択してサブシーケンスにステップインします。

    コールスタックペーンで、AMDProcessor.seq がコールスタックの一番下に表示されていることに注意してください。
15. **デバッグ→再開**を選択して実行を完了し、レポートを確認します。
16. 実行ウィンドウとシーケンスファイルウィンドウを閉じます。

このチュートリアルセッションはこれで終わりです。次のセッションでは、TestStand で生成されるレポートのカスタマイズ方法について学習します。

12

レポートをカスタマイズする

本章では、TestStand でのレポート生成のカスタマイズ方法を学習します。第 8 章、「コールバックを使用する」で説明したコールバック構造により、独自のテストレポートコールバックルーチンを作成して、どのような形式のレポートでも生成することができます。レポート生成の変更は一般的に行われるため、TestStand では独自のコールバックを作成せずにテストレポートの形式を構成できるオプションをいくつか用意しています。それらの各オプションについて本章で説明します。

サンプルをセットアップする

このチュートリアルセッションを完了するためには、シーケンスエディタ内のウィンドウをすべて閉じます。

テストレポートオプションを構成する

1. `TestStand\Tutorial` ディレクトリから `Sample1.seq` を開きます。
2. **構成→レポートオプション**を選択します。図 12-1 に示すように、「レポートオプションを編集」ダイアログボックスが表示されます。

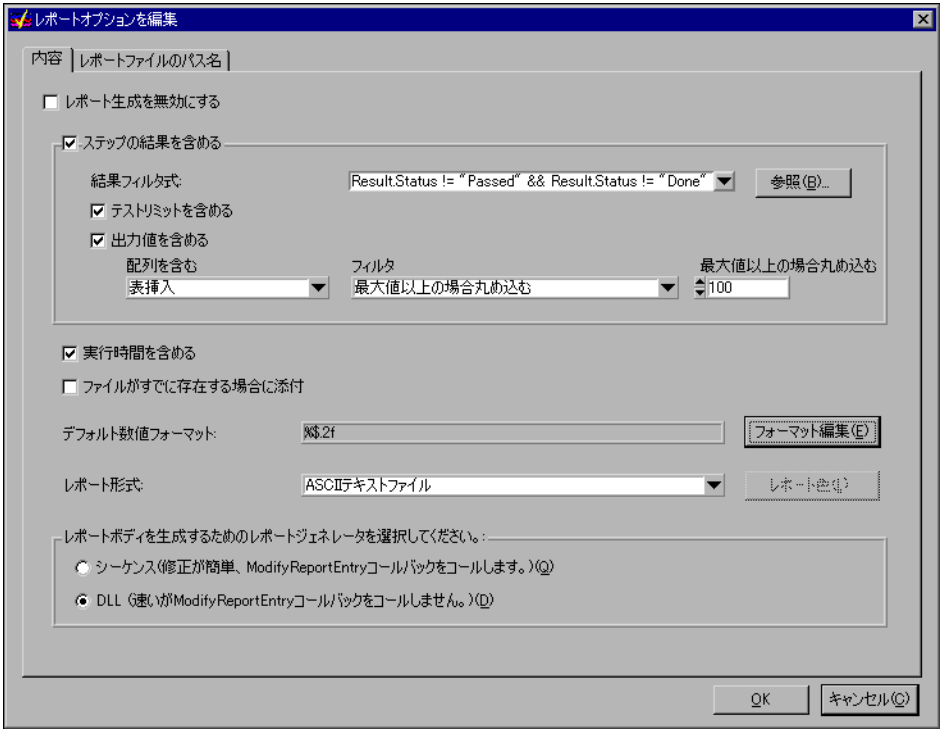

図 12-1　UUT レポート設定

3. テストレポートオプションを図 12-1 のように構成します。
   a. 「ステップの結果を含める」オプションをチェックして、ステップ結果の設定を以下のように構成します。
      - 結果フィルタ式の右側の矢印をクリックして、「合格 / 終了 / スキップ以外」を選択します。

        結果フィルタ式により、結果がテストレポートに記録される前に満たされるべき条件が設定されます。この例では、合格しなかったステップや、ステータスなしで終了したステップの結果のみを記録するよう構成します。TestStand は、レポート生成の際にランタイムで式を評価します。各ステップの結果は、評価が `True` の場合のみテストレポートに記録されます。
      - 「テストリミットを含める」オプションにチェックを付けます。

- 「出力値を含める」オプションにチェックを付けます。

  この例では、測定値が配列の場合、配列を表として含めるように TestStand を構成します。ウェブページレポートを生成する場合は、配列をグラフとして含めることもできます。

b. 「実行時間を含める」オプションにチェックを付けます。

c. **フォーマット編集**ボタンをクリックして数値形式ダイアログボックスを開きます。デフォルトで、TestStand は 13 桁精度の浮動小数点数を記録するよう数値形式を構成します。少数桁数を 2 に変更して、**OK** ボタンをクリックします。

d. レポート形式制御器を ASCII テキストファイルに設定します。この設定により、テストレポートは標準の ASCII 形式で作成されます。

4. 「レポートファイルのパス名」タブをクリックします。

   このタブを使用すると、テストレポートファイルの名前とパスを構成することができます。たとえば、各 UUT に対して新しいファイルを作成したり、レポートファイル名に時間や日付を入れることもできます。

5. 「レポートファイルのパス名」タブのオプションをそのままにして、**OK** をクリックしダイアログボックスを閉じます。

6. **実行→ UUT をテスト**を選択してシーケンスを実行します。

7. Video および CPU テスト以外のコンポーネントを不合格にするよう選択して、シーケンスを何度か実行してください。

8. UUT 情報ダイアログボックスの**停止**ボタンをクリックして、シーケンスの実行を停止します。

9. テストレポートの内容を確認し、不合格になった UUT に対して「不合格ステップへのリンク」が含まれていることに注目してください。「不合格ステップへのリンク」には、UUT の不合格の原因となったステップが表示されています。また、実行が不合格のステップに到達した際のシーケンスコールステップも表示されます。図 12-2 は、`Sample1.seq` の RAM ステップの不合格によって UUT が不合格になった場合の不合格ステップへのリンクを示します。さらに、不合格になったステップがレポートに表示されている唯一のステップ結果であることにも注意してください。レポート形式は、図 12-2 のようになります。

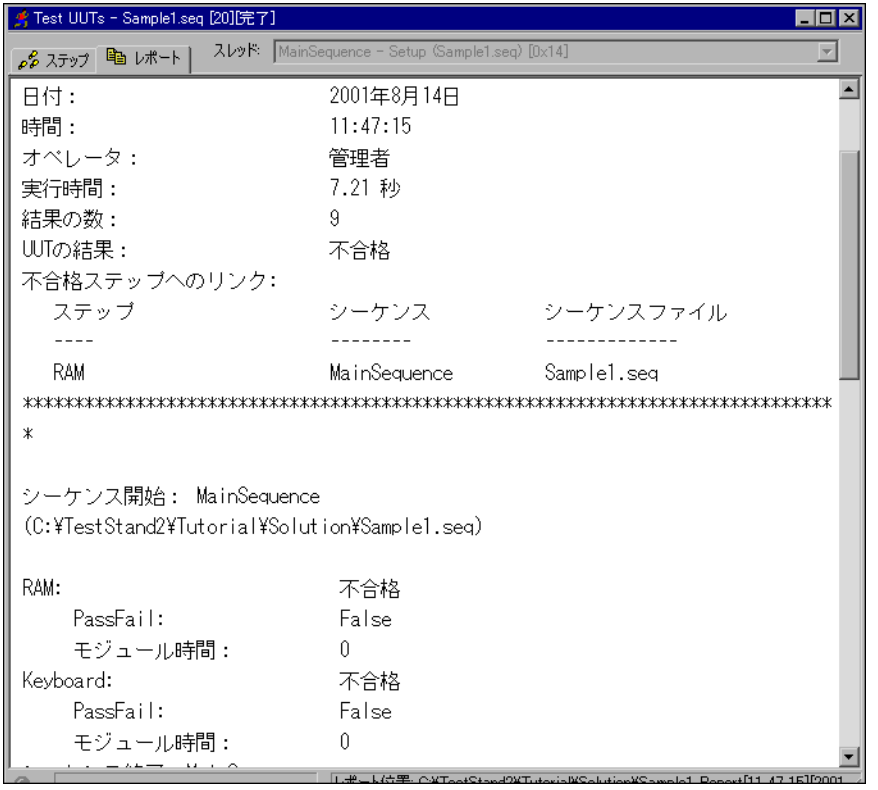

図 12-2　テキスト形式のテストレポート

10. 実行ウィンドウを閉じます。
11. **構成→レポートオプション**を選択します。
12. レポート形式を Web ページに変更します。
13. **OK** をクリックしてダイアログボックスを閉じます。
14. **実行→ UUT をテスト**を選択してシーケンスを実行します。
15. Video および CPU テスト以外のテストを不合格にするよう選択して、シーケンスを何度か実行してください。
16. **UUT 情報**ダイアログボックスの**停止**ボタンをクリックして、シーケンスを停止します。
17. テストレポートの内容を確認して、HTML レポートでは、「不合格ステップへのリンク」の各ステップ名がそのステップの結果を表示する

レポートセクションへのハイパーリンクになっていることに注目してください。レポート形式は、図 12-3 のようになります。

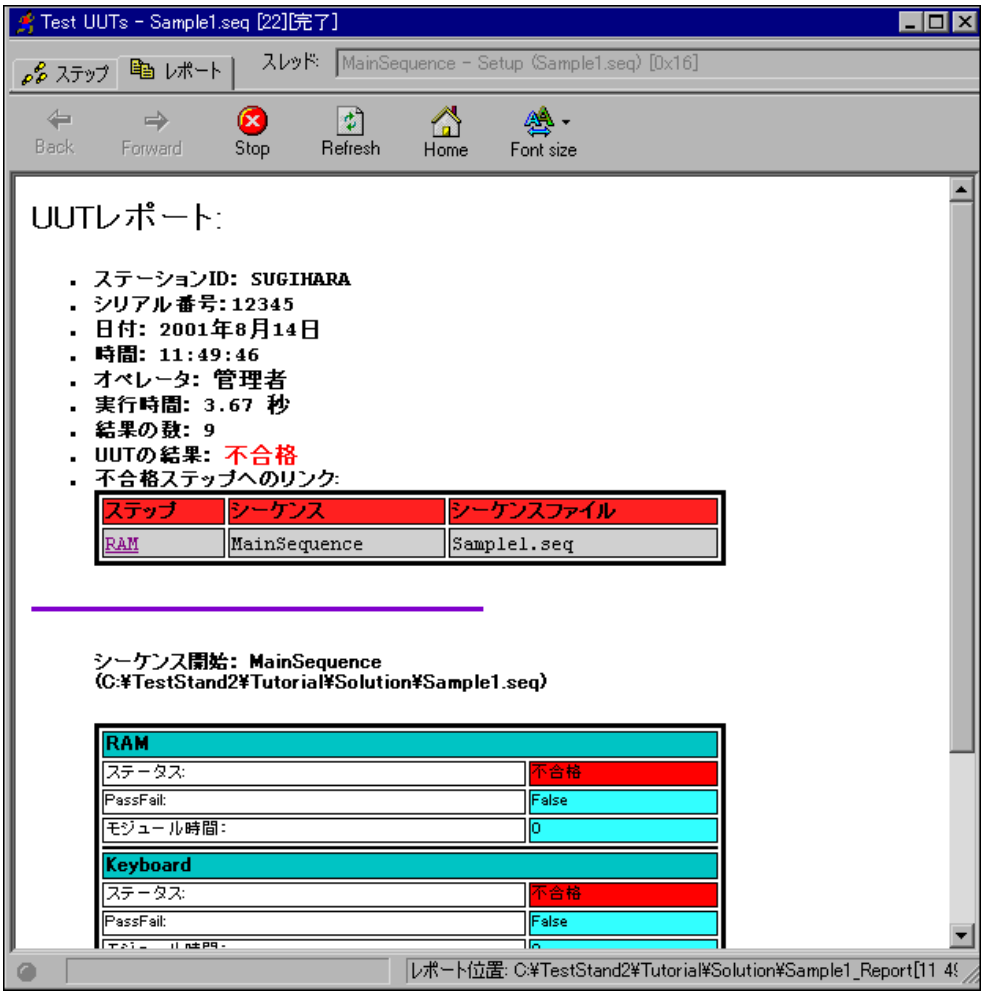

図 12-3　HTML 形式のテストレポート

18. 実行ウィンドウを閉じます。
19. **構成→レポートオプション**を選択します。
20. レポート形式制御器を ASCII テキストファイルに戻し、結果フィルタ式の矢印のプルダウンリストから「すべての結果」を選択して `True` に戻します。
21. **OK** をクリックして「レポートオプションを編集」ダイアログボックスを閉じます。

外部レポートビューワを使用する

Microsoft Word や Microsoft Excel など、テキストの表示や編集により適した外部アプリケーションでテストレポートを表示したい場合もあるでしょう。TestStand では、そのような外部アプリケーションを外部レポートビューワといいます。

デフォルトで、Windows 2000/NT/Me/9*x* オペレーティングシステムはアプリケーションをファイル拡張子と関連付けることができます。たとえば、Microsoft はデフォルトで `.doc` ファイル拡張子を Microsoft WordPad アプリケーションと関連付けます。Microsoft Word をシステムにインストールすると、Word インストーラは `.doc` ファイルタイプの WordPad ファイルタイプとの関連付けを置き換えます。

1. レポートのファイル拡張子を変更するには、次の手順に従ってください。
   a. **構成→レポートオプション**を選択します。
   b. 「レポートファイルのパス名」タブを選択します。
   c. 「レポート形式に標準拡張子を使用」オプションのチェックを外します。
   d. 拡張子文字列制御器に `doc` と入力します。これにより TestStand は、テストレポートを `.doc` ファイル拡張子で作成します。
   e. **OK** をクリックして「レポートオプションを編集」ダイアログボックスを閉じます。
2. `.doc` ファイル拡張子に関連付けられた外部ビューワを自動的に起動するよう TestStand を構成します。
   a. **構成→外部ビューワ**を選択します。
   b. 「デフォルト外部ビューワを自動的に起動」チェックボックスにチェックを付けます。
   c. **OK** をクリックして「外部ビューワを構成」ダイアログボックスを閉じます。
3. **実行→ UUT をテスト**を選択してシーケンスを実行します。
4. シーケンスの反復を数回実行してください。
5. **UUT 情報**ダイアログボックスの**停止**ボタンを選択して、実行を停止します。

   これにより TestStand はテキストレポートを生成し、ワードパッドまたは Microsoft Word を起動してテストレポートを表示するようになります。
6. テストレポートの内容を確認して外部レポートビューワアプリケーションを閉じます。

7. レポート設定を次のように戻します。
   a. **構成→レポートオプション**を選択します。
   b. 内容タブで、レポート形式を Web ページに変更します。
   c. 「レポートファイルのパス名」タブで、「レポート形式に標準拡張子を使用」にチェックを付けます。
   d. **OK** をクリックしてダイアログボックスを閉じます。
8. 外部ビューワ設定を次のように戻します。
   a. **構成→外部ビューワ**を選択します。
   b. 「デフォルト外部ビューワを自動的に起動」チェックボックスのチェックを外します。
   c. **OK** をクリックして「外部ビューワを構成」ダイアログボックスを閉じます。

ファイルの関連付けをオペレーティングシステムとは関係なく定義したい場合は、**構成→外部ビューワ**を使用して構成します。外部ビューワに関する詳細は、『TestStand User Manual』の Chapter 4、「Sequence Editor Menu Bar」を参照してください。

新規のステッププロパティをレポートに追加する

ステップタイプは、そのタイプの各ステップのプロパティと動作のリストを定義します。TestStand には、いくつかの定義済みステップタイプが含まれています。それらのステップタイプの機能が目的に合わない場合は、目的に合うものを作成することができます。

この練習では、カスタム数値配列プロパティを持つ新規ステップタイプを作成してこのプロパティをレポートに含めます。

サンプルをセットアップする

このサンプルを完了するためには、シーケンスエディタ内のウィンドウをすべて閉じます。

ステップタイプを作成する

1. **ファイル→新規シーケンスファイル**を選択して新規シーケンスを開きます。
2. **ファイル→別名で保存**を選択してシーケンスを保存します。シーケンスを `Sample13.seq` という名前で `<TestStand>\Tutorial` ディレクトリに保存します。ここでシーケンスファイルを保存すると、コードモジュールの絶対パスではなく相対パスを指定することができます。

3. シーケンスファイルウィンドウの「ビュー」リング制御器で「シーケンスファイルタイプ」を選択します。図 12-4 に示すように、デフォルトでステップタイプタブが選択されています。
4. 右クリックしてコンテキストメニューから**ステップタイプを挿入**を選択します。ステップタイプに `NumericArray` という名前を付けます。
5. 数値配列であるカスタムステッププロパティを挿入します。左側のツリー表示ペーンで、NumericArray を展開して Result ノードをハイライトします。実行中、TestStand はすべてのプロパティの値を自動的にステップの Result コンテナに収集します。そしてそれらを Locals.ResultList 配列の要素として格納します。それらの値は、下記に説明するようにレポートに含めることができます。右側のリスト表示ペーンで Error プロパティを右クリックして、図 12-4 コンテキストメニューから**フィールドを挿入→配列→数値**を選択します。

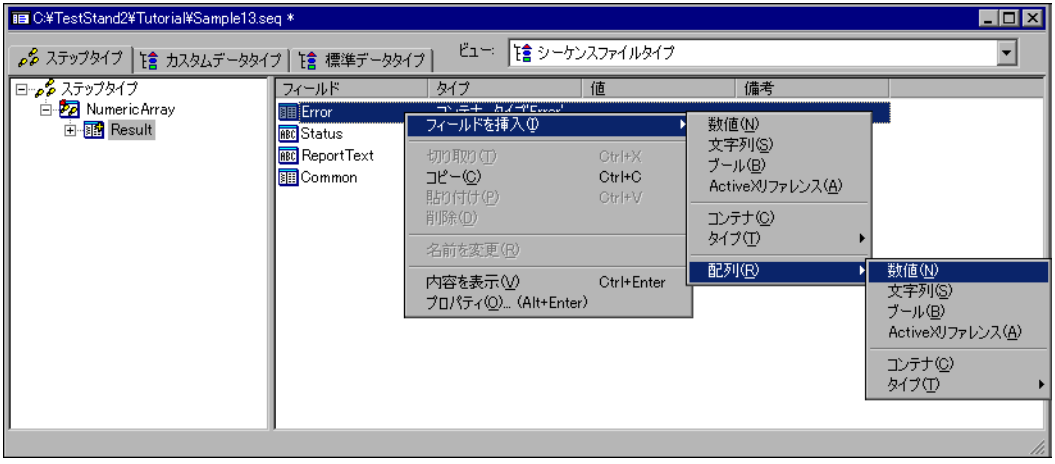

図 12-4　数値配列を挿入コンテキストメニュー

6. 挿入したプロパティの配列範囲ダイアログボックスで、「初期は未入力」オプションにチェックを付け、**OK** ボタンをクリックします。プロパティに `NumArray` という名前を付けます。このプロパティに配列を書き込むと、TestStand は配列を 1 次元配列サイズに自動的に変更します。

7. NumArray を右クリックして、図 12-5 に示すようにコンテキストメニューから**プロパティ**を選択します。

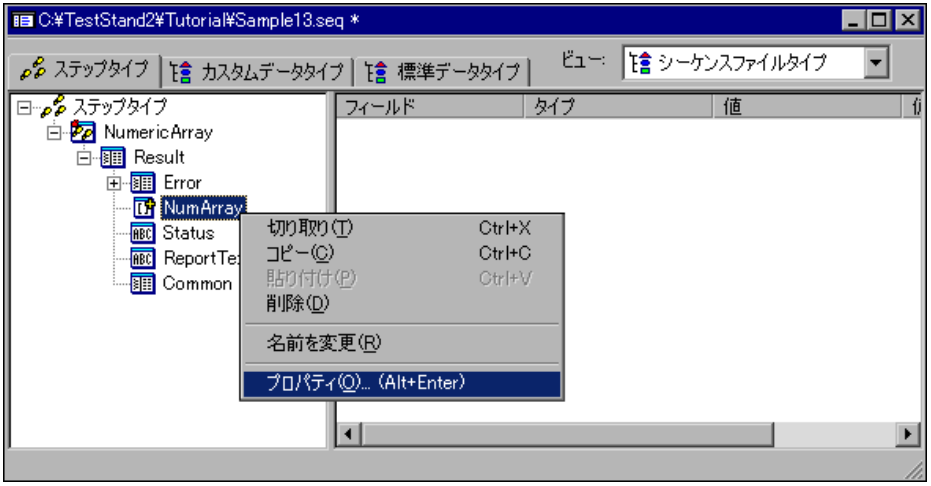

図 12-5　数値配列プロパティコンテキストメニュー

8. NumArray プロパティダイアログボックスで**上級**ボタンをクリックます。
9. 「フラグを編集」ダイアログボックスで、使用できるフラグのリスト内の PropFlags_IncludeInReport にチェックを付けます。このフラグにより、プロパティの値をレポートに追加するよう指定されます。
10. **OK** を 2 回クリックして「フラグを編集」ダイアログボックスと「NumArray プロパティ」ダイアログボックスを閉じます。
11. NumericArray ステップタイプを右クリックして、コンテキストメニューから**プロパティ**を選択します。NumericArray プロパティダイアログボックスでは、実行オプション、ループオプション、式、ポストアクションなど、ステップのすべてのデフォルトプロパティ値を含むステップタイプの動作を構成することができます。ユーザ定義のステップモジュールの前後に、コードモジュールを呼び出すサブステップを作成することができます。サブステップは、ステップタイプの各インスタンスで発生する標準の動作を定義します。ステップタイプの詳細については、『TestStand User Manual』の Chapter 9、「Types」を参照してください。

12. 一般タブで、該当する制御器に次の値を入力します。

制御器名	制御器値
デフォルトステップ名式	“Numeric Array Step”
ステップ説明式	“My Description” + “%ModuleDescription”

他の制御器はデフォルト設定のままにしておきます。

13. **OK** ボタンをクリックして NumericArray プロパティダイアログボックスを閉じます。他のステップタイプ設定は変更しません。
14. **ファイル→保存**を選択してシーケンスファイルを保存します。

これで、ステップモジュールから数値配列を受け取るプロパティを持つステップタイプが作成されました。ステップタイプの Result コンテナでこのプロパティを見つけて PropFlags_IncludeInReport フラグを有効にすると、ステップタイプの各インスタンスの数値配列がレポートに表示されます。

この他にも、この練習で使用しないステップタイプ機能があります。一般的なステップタイプの変更には、数値配列を評価してステップの合否を判定する 1 つまたは複数のポストサブステップまたはステータス式などが含まれることがあります。ステップタイプにはまた、テストシーケンスを実行する前に合否基準を構成することを可能にする編集サブステップが 1 つまたは複数ある場合もあります。ステップタイプのインスタンスで、各編集サブステップに対してコンテキストメニュー項目が作成されます。また、配列を返すコードモジュールをユーザが作成できるようにコードテンプレートを作成することもできます。

LabVIEW 標準プロトタイプアダプタを使用してステップモジュールを作成する

この練習では、ステップタイプのインスタンスと数値配列を返す VI ステップモジュールを作成します。LabVIEW を使用していなくて、LabWindows/CVI を使用している場合は、このセクションをスキップして本章の「C/CVI 標準プロトタイプアダプタを使用してステップモジュールを作成する」または「DLL フレキシブルプロトタイプアダプタを使用してステップモジュールを作成する」のセクションに進んでけっこうです。

1. `<TestStand>\Tutorial\Sample13.seq` を開きます。
2. 「ビュー」リング制御器で MainSequence を選択します。
3. ツールバーのアダプタ選択リングをクリックして、LabVIEW 標準プロトタイプアダプタを選択します。

4. MainSequence のメインステップグループで、右クリックしてコンテキストメニューから**ステップを挿入→ NumericArray** を選択します。デフォルトのステップ名とステップの説明が、ステップタイプ作成時に入力した値であることを確認してください。
5. Numeric Array ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。
6. 「LabVIEW VI コールを編集」ダイアログボックスで、シーケンスコンテキスト ActiveX ポインタにチェックを付けます。
7. **VI を作成**ボタンをクリックすると、TestStand はステップのコードモジュールのパス名を選択するようユーザにプロンプトします。
8. `<TestStand>\Tutorial` ディレクトリを選択します。ファイル名制御器に `ReturnNumArray.vi` という名前を入力します。このチュートリアルセッションを他の人が以前に行っていると、この VI はすでにあるかもしれません。
9. **OK** をクリックして、VI パス名の選択に使用したダイアログボックスを閉じます。
10. TestStand は、`ReturnNumArray.vi` を作成して LabVIEW で開きます。VI のダイアグラムを図 12-6 のように完成させます。ルックアップ文字列 `Step.Result.NumArray` を入力する際の構文エラーに注意してください。

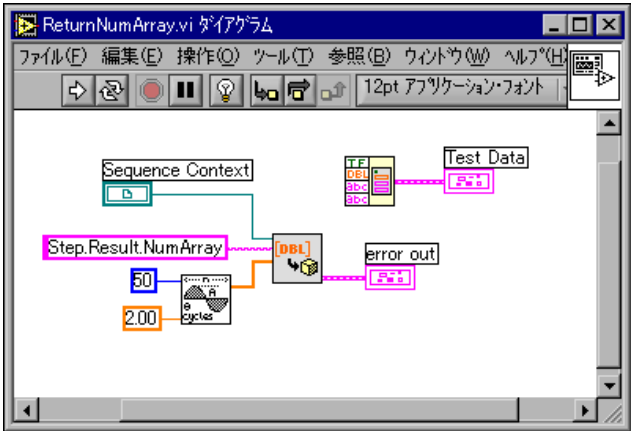

図 12-6　ReturnNumArray.vi ダイアグラム

11. VI の構築が終了したら、LabVIEW で**ファイル→保存**を選択して VI を保存します。
12. VI のダイアグラムとフロントパネルを閉じます。
13. シーケンスエディタに戻り、**OK** をクリックして「LabVIEW VI コールを編集」ダイアログボックスを閉じます。

14. **ファイル→別名で保存**を選択してシーケンスファイルを保存します。
15. **実行→一回実行**を選択してシーケンスを実行します。レポートオプションで、「レポート形式」を Web ページに、「配列を含む」制御器をグラフ挿入に設定している場合、配列データは図 12-7 のようになります。レポートオプションの制御器の値を変えてみて、レポートにどのような変化が現れるか確認してください。

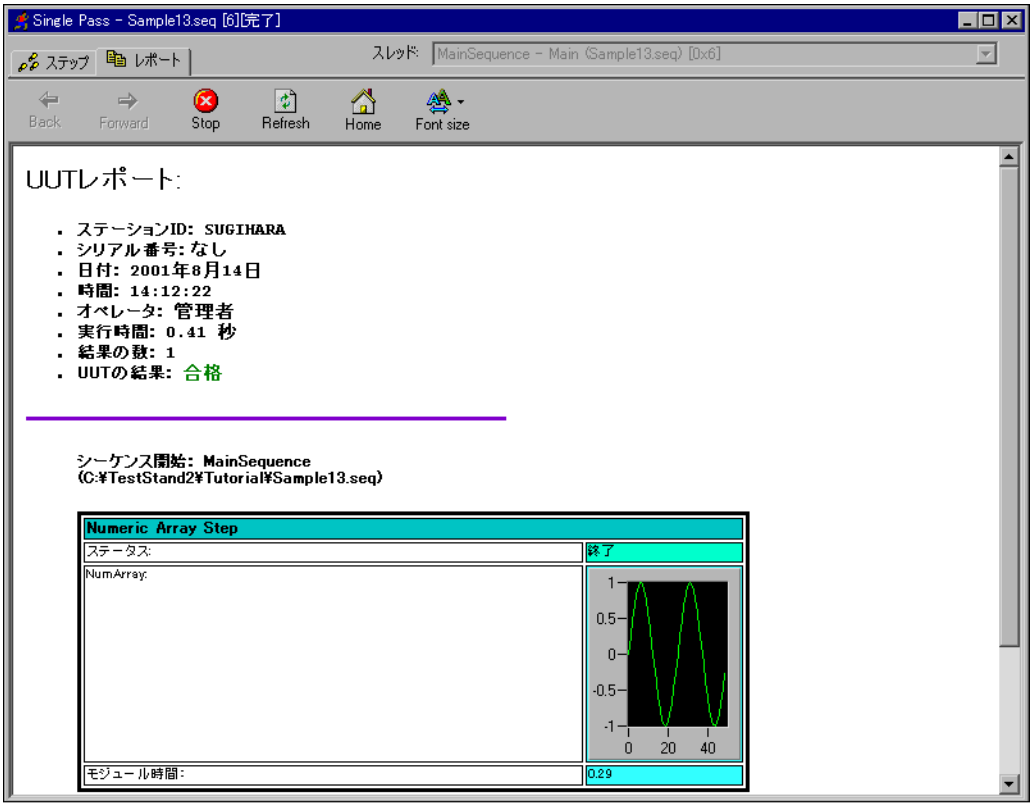

図 12-7　グラフを使用した数値配列レポート

C/CVI 標準プロトタイプアダプタを使用してステップモジュールを作成する

この練習では、ステップタイプのインスタンスと数値配列を返す DLL コードモジュールを作成します。LabVIEW を使用している場合は、このセクションをスキップして第 13 章、「LabVIEW および LabWindows/CVI テスト エグゼクティブシーケンス を変換する」に進んでけっこうです。

1. `<TestStand>\Tutorial\Sample13.seq` を開きます。
2. 「ビュー」リング制御器で MainSequence を選択します。
3. ツールバーのアダプタ選択リングをクリックして、C/CVI 標準プロトタイプアダプタを選択します。
4. MainSequence のメインステップグループで、右クリックしてコンテキストメニューから**ステップを挿入→ NumericArray** を選択します。デフォルトのステップ名とステップの説明が、ステップタイプ作成時に入力した値であることを確認してください。
5. Numeric Array ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。「C/CVI モジュールコールを編集」ダイアログボックスのモジュールタブで、モジュールタイプを Dynamic Link Library に設定します。モジュールパス名制御器に `NumericArray.dll` と入力します。
6. 関数名制御器に `GetNumArray` と入力します。
7. 「C/CVI モジュールコールを編集」ダイアログボックスで「シーケンスコンテキストを渡す」オプションにチェックを付けます。
8. 「C/CVI モジュールコールを編集」ダイアログボックスのソースコードタブで、「関数を含むソースファイルのパス名」制御器に `NumericArray.c` と入力します。
9. 「開く CVI プロジェクトファイルのパス名」制御器に `NumericArray.prj` と入力します。
10. **コードを作成**ボタンをクリックします。すると、TestStand は LabWindows/CVI プロジェクトファイルのパス名を選択するようプロンプトします。
11. `<TestStand>\Tutorial` ディレクトリを選択して **OK** ボタンをクリックし、「CVI プロジェクトファイルのパス名を選択」ダイアログボックスを閉じます。このチュートリアルセッションを他の人が以前に行っている場合、このファイルはすでにあるかもしれません。
12. TestStand は、LabWindows/CVI ソースファイルのパス名を選択するようプロンプトします。`<TestStand>\Tutorial` ディレクトリを選択して **OK** ボタンをクリックし、「ソースファイルのパス名を選択」ダイアログボックスを閉じます。

パス名を入力したら、TestStand は以下の動作を行います。

a. LabWindows/CVI の外部インスタンスを起動。
b. LabWindows/CVI で新規プロジェクトファイルを作成。
c. ソースファイルを作成。
d. ソースファイルと TestStand サポート計測器ドライバをプロジェクトに追加。
e. ソースファイルにテンプレートの `GetNumArray` 関数を生成。

13. 倍精度配列を Step.Result.NumArray プロパティに返すように `GetNumArray` を更新します。次の手順で、関数のソースコードを更新します。変更された行は太字で示しています。

```
void __declspec(dllexport) TX_TEST
    GetNumArray(tTestData *testData, tTestError
    *testError)

{
    int                 error = 0;
    ErrMsg              errMsg = {'\0'};
    ERRORINFO           errorInfo;
    VARIANT             tmpVariant;
    double              sineArray[50];


    tmpVariant = CA_VariantEmpty();
    errChk(SinePattern (50, 1.0, 0.0, 2, sineArray));
    errChk(CA_VariantClear(&tmpVariant));

    // Create a Safe Array from the 1-D array, and
    // store the Safe Array in the VARIANT.
    errChk(CA_VariantSet1DArray(&tmpVariant,
    CAVT_DOUBLE, 50, sineArray));

    // Set the value of the property the lookupString
    // parameter specifies with a variant.
    // Use this method to set the value
    // of an entire array at once.
    tsErrChk(TS_PropertySetValVariant(testData->
        seqContextCVI, &errorInfo,
        "Step.Result.NumArray", 0, tmpVariant));

Error:
    // FREE RESOURCES
    CA_VariantClear(&tmpVariant);
```

```
    // If an error occurred, set the error flag to
    // cause a run-time error in TestStand.
    if (error < 0)
        {
        testError->errorFlag = TRUE;
        testError->errorCode = error;
        testData->replaceStringFuncPtr(&testError->
        errorMessage, errMsg);
        }
    return;
}
```

14. **Build → Compile File** を選択してソースコードをコンパイルし、正しく変更されていることを確認します。
15. コンパイルが正しく行われたらソースコードを保存します。
16. DLL を構築するには、Project ウィンドウで **Build → Create Debuggable Dynamic Link Library** を選択します。

メモ　DLL 作成の際に LabWindows/CVI がファイル許可エラーを返した場合は、シーケンスエディタに戻って**ファイル→すべてのモジュールの解放**を選択します。すると TestStand は、DLL、VI、およびアダプタがロードした他のすべてのモジュールを含む、すべてのステップのコードモジュールを解放します。LabWindows/CVI に戻って DLL を再構築します。

17. DLL の構築が完成したら、シーケンスエディタに戻ります。
18. **OK** をクリックして C/CVI モジュールコールを編集ダイアログボックスを閉じ、メインステップグループビューに戻ります。
19. **ファイル→別名で保存**を選択してシーケンスファイルを保存します。
20. **実行→一回実行**を選択してシーケンスを実行します。レポートオプションで、「レポート形式」を Web ページに、「配列を含む」制御器をグラフ挿入に設定している場合、配列データは図 12-8 のようになります。レポートオプションの制御器の値を変えてみて、レポートにどのような変化が現れるか確認してください。

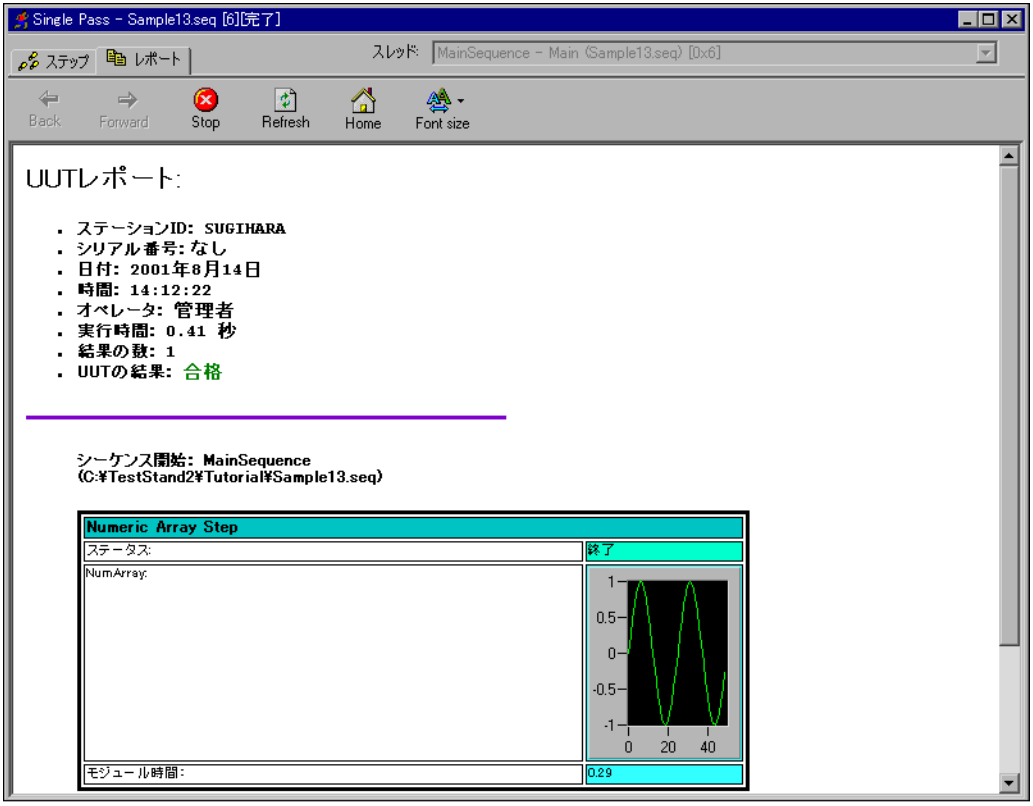

図 12-8　グラフを使用した数値配列レポート

DLL フレキシブルプロトタイプアダプタを使用してステップモジュールを作成する

この練習では、ステップタイプのインスタンスと数値配列を返す DLL ステップモジュールを作成します。LabVIEW を使用している場合は、このセクションをスキップして第 13 章、「LabVIEW および LabWindows/CVI テスト エグゼクティブシーケンス を変換する」に進んでけっこうです。

1. `<TestStand>\Tutorial\Sample13.seq` を開きます。
2. 「ビュー」リング制御器で MainSequence を選択します。
3. ツールバーのアダプタ選択リングをクリックして、DLL フレキシブルプロトタイプアダプタを選択します。
4. MainSequence のメインステップグループで、右クリックしてコンテキストメニューから**ステップを挿入→ NumericArray** を選択しま

す。デフォルトのステップ名とステップの説明が、ステップタイプ作成時に入力した値であることを確認してください。

5. Numeric Array ステップを右クリックして、コンテキストメニューから**モジュールを指定**を選択します。「DLL コールを編集」ダイアログボックスのモジュールタブで、「DLL とパス名」制御器に `NumericArray.dll` と入力します。
6. 関数名制御器に `GetNumericArray` と入力します。
7. 呼び出し規約制御器の値は標準コールのままにしておきます。
8. ソースコードタブで、「関数を含むソースファイルのパス名」に `NumericArray.c` と入力します。
9. **コードを作成**ボタンをクリックします。
10. ソースファイルがない場合、TestStand はソースファイルのパス名を選択するようプロンプトします。`<TestStand>/Tutorial` ディレクトリを選択して、**OK** ボタンをクリックします。
11. TestStand は、`*.c` 拡張子を持つファイルを開くように登録されているシステム内のアプリケーションを使って `.c` ファイルを開くか、あるいはメモ帳を起動して作成した `.c` ファイルを表示するかどうか尋ねます。`.c` ファイルが表示されたら、シーケンスエディタに戻って **OK** ボタンをクリックし、「DLL コールを編集」ダイアログボックスを閉じます。
12. **スタート→プログラム→ National Instruments → Measurement Studio → CVI IDE** を選択して、LabWindows/CVI を起動します。
13. 前の練習を完了している場合は、LabWindows/CVI プロジェクトの `NumericArray.prj` を開きます。
14. 前の練習を完了していない場合は、DLL モジュールを構築するための LabWindows/CVI プロジェクトを作成する必要があります。
    a. **File → New → Project** を選択して、新規のプロジェクトを開きます。LabWindows/CVI でプロジェクトがすでにロードされている場合は、現在のプロジェクトをアンロードするかどうか尋ねるプロンプトが表示されます。**Yes** ボタンをクリックします。LabWindows/CVI は、現在のプロジェクトのオプションを新規のプロジェクトに移行するかどうか尋ねるプロンプトを表示します。すべてのオプションのチェックを外して **OK** ボタンをクリックします。

b. **Edit → Add Files to Project → All Files (*.*)** を選択して、Add Files to Project ダイアログボックスを開きます。下記の各ファイルを選び、Add ボタンを使って Selected Files 制御器に追加します。

- <TestStand>\Tutorial\ NumericArray.c
- <TestStand>\API\CVI\tsapicvi.fp
- <TestStand>\API\CVI\tsutil.fp

すべてのファイルを追加したら、**OK** ボタンをクリックしてファイル名がリストに表示されているプロジェクトウィンドウに戻ります。

c. **File → Save** を選択して、プロジェクトを NumericArray.prj という名前で <TestStand>\Tutorial ディレクトリに保存します。

15. プロジェクトウィンドウでファイル名をダブルクリックして、NumericArray.c ファイルを開きます。
16. 倍精度配列を Step.Result.NumArray プロパティに返すように GetNumericArray を更新します。次の手順で、関数のソースコードを更新します。変更された行は太字で示しています。

```
void __declspec(dllexport) __stdcall GetNumericArray
    (CAObjHandle seqContextCVI, short *errorOccurred,
    long *errorCode, char errorMsg[1024])
{
int error = 0;
ErrMsg      errMsg = {'\0'};
ERRORINFO   errorInfo;
VARIANT     tmpVariant;
double      sineArray[50];

tmpVariant = CA_VariantEmpty();
errChk(SinePattern (50, 1.0, 0.0, 2, sineArray));
errChk(CA_VariantClear(&tmpVariant));

// Create a Safe Array from the 1-D array,
// and stores the Safe Array in the VARIANT.
errChk(CA_VariantSet1DArray(&tmpVariant,
CAVT_DOUBLE, 50, sineArray));

// Set the value of the property the lookupString
// parameter specifies with a variant.
// Use this method to set the value
// of an entire array at once.
```

```
    tsErrChk(TS_PropertySetValVariant(seqContextCVI,
    &errorInfo, "Step.Result.NumArray", 0, tmpVariant));

Error:
    // FREE RESOURCES
    CA_VariantClear(&tmpVariant);

    // If an error occurred, set the error flag to cause
    // a run-time error in TestStand.
    if (error < 0)
        {
        testError->errorFlag = TRUE;
        *errorCode = error;
        strcpy(errorMsg, errMsg);
        }
    return;
}
```

17. **Build → Compile File** を選択してソースコードをコンパイルし、正しく変更されていることを確認します。
18. コンパイルが正しく行われたらソースコードを保存します。
19. プロジェクトで DLL を構築するために、プロジェクトウィンドウで **Build → Target Type → Dynamic Link Library** を選択します。
20. **Build → Create Debuggable Dynamic Link Library** を選択して DLL を構築します。

メモ　DLL 作成の際に LabWindows/CVI がファイル許可エラーを返した場合は、シーケンスエディタに戻って**ファイル→すべてのモジュールの解放**を選択します。すると TestStand は、DLL、VI、およびアダプタがロードした他のすべてのモジュールを含む、すべてのステップのコードモジュールを解放します。LabWindows/CVI に戻って DLL を再構築します。

21. シーケンスエディタに戻り、**ファイル→保存**を選択してシーケンスファイルを保存します。
22. これで、DLL 関数を呼び出すシーケンスを実行できる状態になりました。**実行→一回実行**を選択してシーケンスを実行します。
23. レポートオプションで、「レポート形式」を Web ページに、「配列を含む」制御器をグラフ挿入に設定している場合、配列データは図 12-9 のようになります。レポートオプションの制御器の値を変えてみて、レポートにどのような変化が現れるか確認してください。

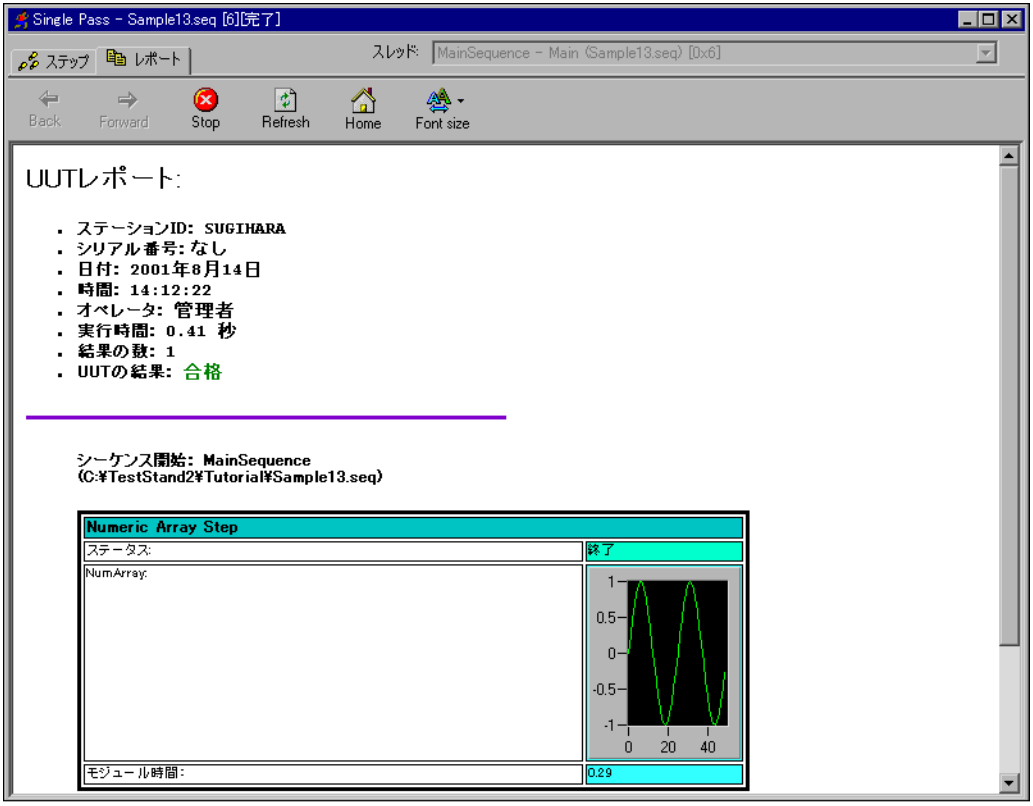

図 12-9　グラフを使用した数値配列レポート

この練習では、シーケンスコンテキストポインタを使用して、DLL から TestStand に数値配列を渡しました。DLL フレキシブルプロトタイプアダプタを使用すると、関数パラメータを使って数値配列を直接渡すこともできます。その方法についての詳細は、`<TestStand>\Examples\AccessingArrays\PassingArrayParametersToDLL` のサンプルを参照してください。

コールバックを使用してレポートを追加する

HTML レポートのデフォルトのヘッダは、図 12-3 のようになっています。この練習では、プロセスモデルのレポートコールバックを使用して HTML レポートのヘッダにロゴを追加します。

1. `<TestStand>\Tutorial\Sample1.seq` を開きます。
2. シーケンスファイルウィンドウをアクティブウィンドウにして、**編集→シーケンスファイルコールバック**を選択し、`Sample1.seq` コールバックダイアログボックスを開きます。
3. `ModifyReportHeader` コールバックを選択します。
4. **追加**ボタンをクリックして、シーケンスファイルにコールバックを追加します。

   図 12-10 は、完成したダイアログボックスを示します。

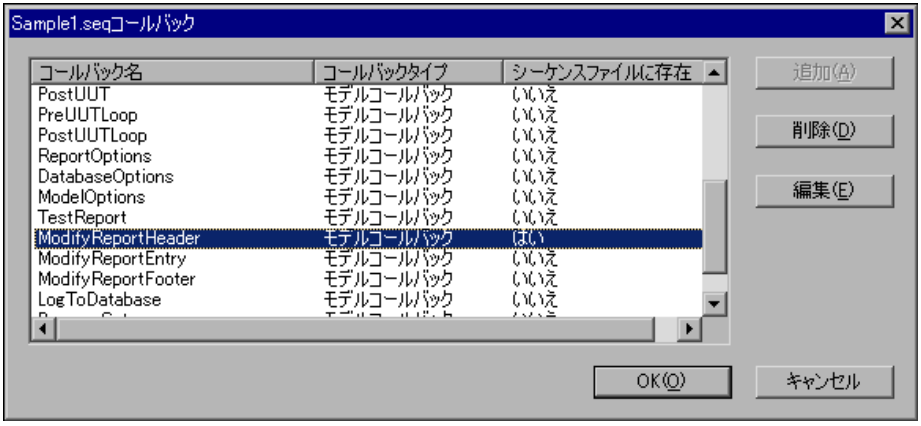

図 12-10　コールバックダイアログボックス

5. **編集**ボタンをクリックしてダイアログボックスを閉じ、新規の `ModifyReportHeader` コールバックシーケンスを編集します。図 12-11 に示すように、「ビュー」プルダウンリングのリストに `ModifyReportHeader` が表示されていることに注目してください。

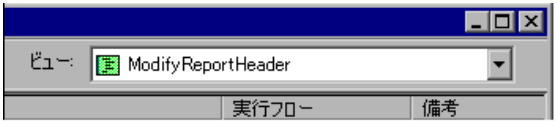

図 12-11　シーケンスのビューに表示された ModifyReportHeader

6. ローカル変数タブをクリックし、図 12-12 に示すように、右側のペーンを右クリックして文字列ローカル変数を挿入します。

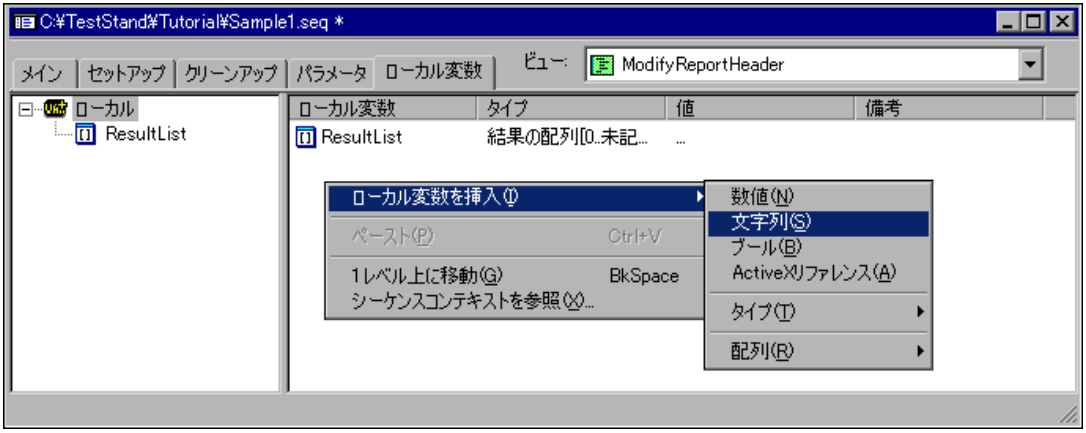

図 12-12　ModifyReportHeader で文字列ローカル変数を挿入

7. 変数に `AddToHeader` という名前を付けます。
8. AddToHeader をダブルクリックします。
9. 文字列プロパティダイアログボックスの値フィールドに次の値を入力します。

```
<IMG ALT='Logo Goes Here' SRC='Logo.jpg'><br><br>
<A HREF='http://www.ni.com'>Visit Our Web
  Site</A><br>
```

10. **OK** をクリックしてダイアログボックスを閉じます。
11. メインタブをクリックして、メインステップグループを表示します。これは空白になっています。
12. ステップのリストを右クリックして、コンテキストメニューから**ステップを挿入→ステートメント**を選択し、ステートメントステップを挿入します。
13. このステップに `Add Custom Logo` という名前を付けます。
14. Add Custom Logo ステップを右クリックして、コンテキストメニューから**式を編集**を選択します。
15. 次の式を入力します。

```
Parameters.ReportHeader = Locals.AddToHeader +
  Parameters.ReportHeader
```

16. **OK** をクリックして「ステートメントステップを編集」ダイアログボックスを閉じます。

17. **ファイル→別名で保存**を選択して、シーケンスを `Sample14.seq` という名前で `TestStand\Tutorial` ディレクトリに保存します。

    図 12-13 は、完成したシーケンスファイルを示します。

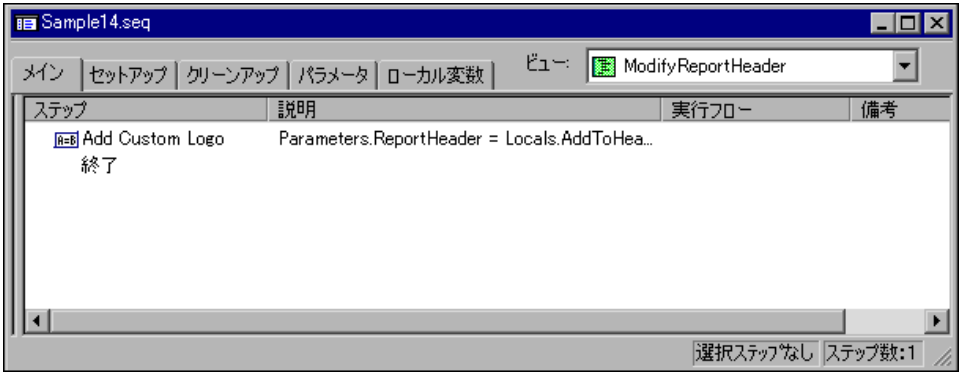

図 12-13　完成した ModifyReportHeader シーケンス

18. **実行→一回実行**を選択します。
19. Test Simulator プロンプトで **Done** をクリックします。

20. 実行が完了したら、レポートを表示して、図 12-14 に示すように UUT レポートの上部に新しいロゴの画像が表示されていることを確認します。

図 12-14　新たに作成した HTML ヘッダ

21. 実行ウィンドウとシーケンスファイルウィンドウを閉じます。

これでこのチュートリアルの最後のセッションは終了です。レポートのカスタマイズの詳細については、『TestStand User Manual』の Chapter 15、「Managing Reports」を参照してください。

13

LabVIEW および LabWindows/CVI テストエグゼクティブシーケンスを変換する

本章では、既存のテストエグゼクティブを TestStand に変換する方法を説明します。TestStand では、LabVIEW テストエグゼクティブおよび LabWindows/CVI テストエグゼクティブのシーケンスファイルを TestStand シーケンスファイルに変換するための変換ユーティリティが用意されています。

LabVIEW テストエグゼクティブシーケンスを変換する

LabVIEW テストエグゼクティブ 5.1 のシーケンスファイルを TestStand シーケンスファイルに変換するには、次の手順に従ってください。

1. TestStand シーケンスエディタまたは任意のオペレータインタフェースのいずれかで、**ツール→シーケンスファイルコンバータ→LabVIEW テストエグゼクティブシーケンスの変換**を選択します。
2. 表示されるファイルダイアログボックスで、変換する LabVIEW テストエグゼクティブシーケンスファイルを選択します。
3. 変換後のテストシーケンスの新しいファイル名を入力します。

   変換がうまく行われたかどうかを示すメッセージが表示されます。

シーケンスの変換についての詳細は、`TestStand\Doc` ディレクトリにあるオンラインヘルプドキュメント、Converting from the LabVIEW Test Executive to TestStand (`LVTECompatibility.hlp`) を参照してください。

メモ LabVIEW テストエグゼクティブのバージョン 4.0 または 5.0 を使用している場合は、シーケンスファイルを TestStand に切り替える前にまずバージョン 5.1 に変換する必要があります。

LabWindows/CVI テストエグゼクティブシーケンスを変換する

LabWindows/CVI テストエグゼクティブ 2.0.1 のシーケンスファイルを TestStand シーケンスファイルに変換するには、次の手順に従ってください。

1. TestStand シーケンスエディタまたは任意のランタイムオペレータインタフェースのいずれかで、**ツール→シーケンスファイルコンバータ→ CVI テストエグゼクティブシーケンスの変換**を選択します。
2. 表示されるファイルダイアログボックスで、変換する LabWindows/CVI テストエグゼクティブシーケンスファイルを選択します。
3. 変換後のテストシーケンスの新しいファイル名を入力します。

   変換がうまく行われたかどうかを示すメッセージが表示されます。

シーケンスの変換についての詳細は、`TestStand\Doc` ディレクトリにあるオンラインヘルプドキュメント、Converting from the LabWindows/CVI Test Executive to TestStand (`CVITECompatibility.hlp`) を参照してください。

メモ　2.0.1 以前のバージョンの LabWindows/CVI テストエグゼクティブを使用している場合は、まずシーケンスエディタでシーケンスファイルをロードし LabWindows/CVI テストエグゼクティブ 2.0.1 用に保存し直す必要があります。

A

技術サポートのリソース

ウェブサポート

インストール、構成、アプリケーションに関わる問題および疑問を解決するには、まず弊社ウェブサイトの「サポート」のページをクリックしてください。問題を解決・診断するオンラインリソースには、よくある質問に対する答え、技術サポートデータベース、製品別のトラブルシューティングウィザード、マニュアル、ドライバ、ソフトウェアのアップデート等の情報があります。ウェブサポートをご利用になるには、ni.com/jp の「サポート」のページにアクセスしてください。

NI Developer Zone

ni.com/zone の NI Developer Zone には、自動計測システムの構築に不可欠なリソースがあります。NI Developer Zone では、開発者独自の技術を共有するための開発者コミュニティだけでなく、最新のサンプルプログラム、システムコンフィギュレータ、チュートリアル、および技術ニュース等に簡単にアクセスできます。

カスタマートレーニング

ナショナルインスツルメンツは、お客様のトレーニングの要望にお応えするための様々な方法を提供しております。お客様自身のペースで学習できるチュートリアル、ビデオ、対話式 CD や世界各地で開催中のインストラクタによる実践コース等をご用意しております。コースのスケジュール、摘要、トレーニングセンター、およびクラスへの登録については、ni.com/jp で「セミナー／イベント」をクリックしてください。

システムインテグレーション

時間的制約がある場合、社内の技術リソースに制限がある場合等は、コンサルティングまたはシステムインテグレーションサービスをご利用いただけます。弊社のアライアンスプログラムメンバーのネットワークを通じて、様々な専門技術や知識を得ることができます。アライアンスプログラムのシステムインテグレーションソリューションの詳細については、ni.com/jp の「ソリューション」を参照してください。

世界各地でのサポート

ナショナルインスツルメンツは、お客様のサポートの要望にお応えするため世界各地に支社を配置しております。ni.com の Worldwide Offices から各支社のウェブサイトにアクセスできます。これらのウェブサイトでは、最新の連絡先、サポートの電話番号、E メールアドレス、および現在のイベントについての情報を提供しています。

弊社ウェブサイトの技術サポートリソースを検索しても必要な情報が得られない場合は、最寄の営業所またはナショナルインスツルメンツ本社にお問い合わせください。世界各国の支社の電話番号については、本書の最初のページをご覧ください。

用語集

接頭辞	意味	値
p-	ピコ	10^{-12}
n-	ナノ	10^{-9}
µ-	マイクロ	10^{-6}
m-	ミリ	10^{-3}
k-	キロ	10^{3}
M-	メガ	10^{6}
G-	ギガ	10^{9}
t-	テラ	10^{12}

A

ActiveX サーバ — ActiveX 規格によって他のアプリケーションで利用可能になる実行可能コード。ActiveX では、クライアントがサーバからオブジェクトを要求し、そのオブジェクトに対して動作を実行するようサーバに指示するクライアント／サーバの関係が基本になっています。

ActiveX リファレンスのプロパティ — ActiveX オブジェクトへのリファレンスを保持する情報のコンテナ。TestStand では、プロパティの値を `IDispatch` または `IUnknown` ポインタとして保持します。

ASCII — 情報交換用米国標準コード。

C

CPU — 中央処理装置。

D

DLL — ダイナミックリンクライブラリ。

G

G	LabVIEW アプリケーションの開発に使用するグラフィカルプログラミング言語。
GUI	ランタイムオペレータ インタフェースを参照。

H

hex	16 進法。

L

LabVIEW	Laboratory Virtual Instrument Engineering Workbench の略。プログラミング言語 G をベースにし、一般にテストおよび計測の目的で使用するプログラム開発アプリケーション。

M

MB	メモリのメガバイト。
MFC	Microsoft Foundation Class Library。
MHz	メガヘルツ。

R

RAM	ランダムアクセスメモリ。
ROM	読み取り専用メモリ。
RTF	リッチテキスト形式。

S

s	秒。
SVGA	Super VGA。

V

VI	仮想計測器。
VI ライブラリ	特定の目的を持つ関連 VI が集められたタイプ .LLB の特殊ファイル。

あ

アウトオブプロセス	実行可能ファイル内の ActiveX サーバなど、実行可能コードがクライアントと同じプロセススペースで実行しないこと。
アクティブウィンドウ	ある時点においてユーザ入力の影響を受けるウィンドウ。アクティブウィンドウのタイトルはハイライト表示されています。
アダプタ	ステップが使用する TestStand エンジンのサービスの 1 つで、他のシーケンスまたは他のコードモジュールでコードを呼び出します。アダプタは、コードモジュールのタイプ、その呼び出し方法、およびパラメータを渡す方法についての情報を持っています。
アプリケーション開発環境（ADE)	LabVIEW、LabWindows/CVI、Microsoft Visual C など、テストモジュールやランタイム実行オペレータインタフェースを作成できるプログラミング環境。
アプリケーションプログラミングインタフェース（API）	TestStand エンジンなど、特定のサービスを制御するのに使用する一連のクラス、メソッド、およびプロパティ。

い

インスタンスステップタイププロパティ	各ステップのインスタンスに存在する組み込みのステッププロパティ。ステップタイプで作成する各ステップには、プロパティの独自のコピーがあります。TestStand は、ステップタイプのインスタンスプロパティに対して指定した値を、作成する新規の各ステップのプロパティの初期値として使用します。通常、ステップの作成後、そのインスタンスのプロパティ値は変更が可能です。
インプロセス	ダイナミックリンクライブラリ（DLL）の中の ActiveX サーバのように、実行可能コードがクライアントと同じプロセススペース内で実行すること。

う

ウォッチウィンドウ　現在アクティブになっているユーザ選択可能な変数や式の値を示すウィンドウ。

ウィンドウ　プログラムの開発や実行に関連する特定のタスクをサポートする作業領域。

え

エラー発生フラグ　ステップでランタイムエラーが発生したことを示す `Step.Result.Error.Occurred` というブールフラグ。

エンジン　テストエグゼクティブエンジンを参照。

エンジンコールバック　実行中の特定の時点で TestStand が呼び出すシーケンス。エンジンコールバックを使用して、個々のステップの実行の前後、対話式実行の前後、シーケンスファイルのロード後、およびシーケンスファイルのアンロード前に特定のシーケンスを呼び出すよう TestStand に指示します。

エントリポイント　`UUT` をテスト（Test UUTs）、一回実行（Single Pass）、およびレポートオプションなど、TestStand でメニュー項目として表示されるプロセスモデルファイル内のシーケンス。

お

オブジェクト　ActiveX サーバによりクライアントに提供されるサービス。

オペレータ　一般的にテストステーションの操作に関連するすべての特権は持つが、シーケンス実行のデバッグ、シーケンスファイルの編集、またはユーザ特権、ステーションオプション、レポートオプション、データベースオプションの構成は行えないユーザプロファイル。

か

開発者　通常の場合シーケンスおよびシーケンスファイルの操作、デバッグ、および開発に関連するすべての特権を含むが、ユーザ特権、レポートオプション、またはデータベースオプションの構成は行えないユーザプロファイル。

カスタムプロパティ　ステップタイプでユーザが定義するプロパティ。ステップタイプで作成する各ステップには、カスタムプロパティの独自のコピーがあります。TestStand は、ステップタイプのカスタムプロパティに対して入力した値を、作成する新規の各ステップのプロパティの初期値として使用します。通常、ステップの作成後、そのステップのプロパティ値は変更が可能です。

カスタム名付きデータタイプ　ユーザが定義し名前を付けるデータタイプ。たとえば、`NumChannels` や `PowerLevel` などのサブプロパティを含む `Transmitter` データタイプを作成する場合があります。

管理者　一般にテストステーションに対するすべての特権を含むユーザプロファイル。

き

技術者　通常の場合シーケンスおよびシーケンスファイルの操作およびデバッグに関連するすべての特権を含むが、シーケンスファイルの編集や、ユーザ特権、レポートオプション、またはデータベースオプションの構成は行えないユーザプロファイル。

く

組み込みのステップタイププロパティ　同じタイプのすべてのステップに共通のプロパティ。組み込みのステップタイププロパティは、クラスステップタイププロパティまたはインスタンスステップタイププロパティのいずれかです。

組み込みのプロパティ　すべてのステップまたはシーケンスに含まれるプロパティ。ステップの実行モードプロパティはその一例です。TestStand ではそのようなプロパティはシーケンスエディタで通常非表示になっていますが、ダイアログボックスを通して変更することはできます。

クライアントシーケンスファイル　UUT をテストするためにプロセスモデルが呼び出すメインシーケンスが含まれたシーケンスファイルです。各クライアントシーケンスファイルには、`MainSequence` というシーケンスが含まれます。プロセスモデルはテストプロセスにおいて一定であることを定義するのに対し、クライアントシーケンスファイルは実行する異なるタイプのテストに対して固有のステップを定義します。

クラス　そのクラスのインスタンスとして作成したオブジェクトに関して使用できるメソッドおよびプロパティのリストを定義します。クラスは、データタイプの定義に似ていますが、変数ではなくオブジェクトに適用されます。

クラスステップタイププロパティ　ステップタイプ自体にのみ存在する組み込みのステッププロパティ。TestStand は、これらのプロパティを使用して、ステップのすべてのインスタンスに対するステップタイプの動作を定義します。ステップのインスタンスには、クラスプロパティの独自のコピーは含まれません。

クラスタ　順序付けられていて指標付けはされていない LabVIEW データ要素で、数値、ブール、文字列、配列、またはクラスタを含む任意のデータタイプを持つことができます。要素はすべて制御器かすべて表示器でなければなりません。

クリップボード　作業領域から切り取り、コピー、または削除されたテキストを保持するためにオペレーティングシステムが使用する一時保管領域。

グローバル変数　TestStand では、シーケンスファイルグローバルとステーショングローバルの 2 つのタイプのグローバル変数が定義されています。シーケンスファイルグローバルは、シーケンスファイルの任意のシーケンスまたはステップからアクセス可能です。ステーショングローバルは、ステーションにロードされた任意のシーケンスファイルからアクセス可能です。ステーショングローバル変数の値は、別の実行や TestStand の異なる呼び出しでも一定しています。

こ

構成エントリポイント　プロセスモデルの機能を構成するプロセスモデルファイル内のシーケンス。構成エントリポイントは通常、`TestStand\cfg` ディレクトリにある `.ini` ファイルに構成情報を保存します。デフォルトで、構成エントリポイントは**構成**メニューに表示されます。たとえば、デフォルトのプロセスモデルには `Config Report Options` 構成エントリポイントが含まれます。`Config Report Options` エントリポイントは、**構成**メニューの**レポートオプション**に表示されます。

コードテンプレート　スケルトンコードを含むソースファイル。スケルトンコードは、ステップタイプを使用するステップのコードモジュールを開発する際の開始点となります。

コードモジュール　特定のテストや他の動作を実行する 1 つまたは複数の関数を含む、Windows Dynamic Link Library (`.dll`) または LabVIEW VI (`.vi`) などのプログラムモジュール。

コールスタック　ネストされたサブシーケンスが完了するのを待機しているアクティブシーケンスのチェーン。

コールバック　シリアル番号の照会やレポートの記録など、一般的なタスクを処理するのに使用するシーケンス。

コネクタ　入力端子と出力端子を含む LabVIEW の VI または関数ノードの一部で、そこを通ってデータがノードに受け渡しされます。

コンテキストメニュー　オブジェクトをクリックしてアクセスするメニュー。メニューオプションは、そのオブジェクト特定のものです。

コンテナプロパティ　値を持たないプロパティで、一般的に複数のサブプロパティを含んでいます。コンテナプロパティは、C/C++ のストラクチャおよび LabVIEW のクラスタと同意語です。

さ

サブシーケンス　他のシーケンスが呼び出すシーケンス。サブシーケンスコールは、呼び出しシーケンスのステップとして指定します。

サブステップ　ステップモジュールを呼び出す以外に、ステップタイプがステップに対して実行するアクション。サブステップは、アダプタを選択しモジュールコールを指定して定義します。TestStand は、編集サブステップ、プリステップサブステップ、およびポストステップサブステップの 3 つの異なるタイプを定義します。

サブステップモジュール　編集、プリステップ、またはポストステップサブステップが呼び出すコードモジュール。

し

シーケンス　実行に対して特定の順序で指定する一連のステップ。ステップが実行されるかどうかおよびそのタイミングは、前のステップの結果に依存します。

シーケンスエディタ　シーケンスの作成、編集、およびデバッグのためのグラフィカルユーザインタフェースを提供するプログラム。

シーケンスグローバル変数　シーケンスファイル全体に関連したデータを保存するための変数。各シーケンスおよびシーケンスフィル内のステップは、これらのグローバル変数に直接アクセスできます。

シーケンスコンテキスト　アクティブシーケンスのすべてのグローバル変数、すべてのローカル変数、およびステッププロパティへのリファレンスを含む TestStand オブジェクト。シーケンスコンテキストの内容は、現在実行中のシーケンスおよびステップによって異なります。

シーケンスファイル　1 つまたは複数のシーケンスの定義を含むファイル。

シーケンスファイルウィンドウ	シーケンスエディタ内でシーケンスファイルが表示される別個のウィンドウ。
式	複数の変数またはプロパティの値から新しい値を計算するための公式。式では、TestStand が式を評価する際にアクティブになっているシーケンスコンテキストのすべての変数およびプロパティにアクセスすることができます。式の例を以下に示します。`Locals.MidBandFrequency = (Step.HighFrequency + Step.LowFrequency) / 2`
実行	TestStand がシーケンス、そのステップ、および呼び出すすべてのサブシーケンスを実行するのに必要な全情報を含むオブジェクト。一般的に、TestStand シーケンスエディタは各実行に対して新規のウィンドウを作成します。
実行ウィンドウ	実行するステップを表示するシーケンスエディタ内のウィンドウ。実行が中止されると、実行ウィンドウは次に実行するステップを表示し、シングルステップオプションを選べる状態になります。コールスタックのどのアクティブシーケンスコンテキストでも変数やプロパティを表示することができます。
実行エントリポイント	UUT に対してテストを実行するプロセスモデル内のシーケンス。実行エントリポイントは、クライアントシーケンスファイルの `MainSequence` を呼び出します。デフォルトプロセスモデルには、`UUT` をテストおよび一回実行の 2 つの実行エントリポイントがあります。デフォルトで、実行エントリポイントは実行メニューに表示されます。実行エントリポイントは、`MainSequence` を持つシーケンスファイルがアクティブウィンドウに含まれている場合のみメニューに表示されます。
実行条件	TestStand がシーケンスの通常の実行フローでステップを実行するために true になる必要があるステップの条件の組み合わせ。
実行ポインタ	ステップタブで現在実行中のステップを指すことにより、実行の進行状況を示す黄色のポインタアイコン。
実行モード	通常、スキップ、強制的に合格、強制的に不合格など、ステップを実行するモード。
終了	通常の実行フローを中断して、シーケンスまたはコールスタックのすべてのクリーンアップステップグループを実行することにより、実行を停止すること。

す

数値プロパティ　IEEE 754 形式の 64 ビット浮動小数点数値。

ステーショングローバル変数　異なる実行間、およびシーケンスエディタやランタイムオペレータインタフェースの別の呼び出し間で一貫している変数。TestStand エンジンは、ステーショングローバル変数の値をランタイムコンピュータ上のファイルに保持しています。

ステーションコールバックシーケンスファイル　ステーションコールバックシーケンスを含むシーケンスファイル。ステーションコールバックは、通常実行または対話式実行でエンジンが各ステップを実行する前後に実行されます。

ステーションモデル　ステーションのすべてのシーケンスファイルで使用するために選択するプロセスモデル。TestStand のインストールプログラムは、`SequentialModel.seq` をデフォルトのステーションモデルファイルとして設定します。ステーションオプションダイアログボックスを使用して、別のステーションモデルを選択することができます。

ステップ　テストモジュールを呼び出して特定のテストを行うなど、他のアクションのシーケンスの中に含めることのできる何らかのアクション。

ステップグループ　シーケンス内の一連のステップ。シーケンスには、セットアップ、メイン、およびクリーンアップの 3 つのステップグループがあります。TestStand がシーケンスを実行すると、最初にセットアップグループのステップ、次にメイングループのステップ、最後にクリーンアップグループのステップが実行します。

ステップ結果　ステップの結果プロパティからのサブプロパティとステップ名やシーケンス内での位置など実行に関する追加情報のコピーを含むコンテナプロパティ。TestStand は、各ステップが実行すると自動的にステップ結果を作成し、そのステップ結果をレポート生成に使用される結果リストに入れます。

ステップステータス　実行中のステップのステータスを示す文字列値。TestStand の各ステップには、`Result.Status` プロパティがあります。ステップまたはそのコードモジュールがステータスプロパティを設定する値に対して制限はありませんが、TestStand および組み込みステップタイプは定義済みの値を使用し認識します。

ステップタイプ　そのタイプの各ステップに対し一連のカスタムステッププロパティと標準動作を定義するコンポーネント。同じタイプのステップはすべて同じプロパティを持ちますが、プロパティの値は異なることがあります。ステップタイプはサブステップを使って標準動作を定義します。

ステップタイプ特定のダイアログボックス　編集サブステップが呼び出されたときにステップタイプによって表示されるダイアログボックス。このダイアログボックスを使用して、ステップタイプ特定のステッププロパティを変更することができます。このダイアログボックスは、コンテキストメニューの**モジュールを指定**より上に表示されるメニュー項目で呼び出します。たとえば、数値リミットテストステップのコンテキストメニューには「リミットの編集」項目が表示され、合格／不合格テストステップのコンテキストメニューには「合格／不合格ソースの編集」項目が表示されます。

ステッププロパティ　ステップのプロパティ。

ステップモジュール　ステップが呼び出すコードモジュール。

せ

制御器　パネルまたはウィンドウに表示されるデータを入力するための入出力デバイス。

制御フロー　実行順序を決定する命令の順序。

そ

ソースコードテンプレート　ステップのコードモジュールを開発するにあたって開始点となるスケルトンコードを含む一連のソースファイル。開発者がステップの「モジュールを指定」ダイアログボックスのソースコードタブで**コードを作成**ボタンをクリックすると、TestStand はコードテンプレートを使用します。

た

ダイアログボックス　コマンドを実行するために必要な追加の情報を入力するためのプロンプトメカニズム。

対話モード　シーケンスから 1 つか複数のステップを選択してコンテキストメニューまたはメニューバーから**選択ステップを実行**または**選択ステップをループ実行**を選択し、ステップを実行すること。シーケンス内の選択されたステップは、シーケンスにどのような分岐ロジックが含まれるかにかかわらず実行します。選択されたステップは、シーケンスに表示された順序で実行します。

単一値プロパティ　単一の値を持つプロパティ。TestStand には、数値プロパティ、文字列プロパティ、ブールプロパティ、および ActiveX リファレンスプロパティの 4 つの単一値プロパティがあります。

端子　LabVIEW VI 上のデータが渡されるオブジェクトまたは領域。

ち

中断　コールスタックの実行でシーケンスのクリーンアップステップグループを一切実行せずに実行を停止すること。実行を中断すると、レポートは生成されません。

チェックボックス　2 つの選択可能なオプションの間で切り替えるためのダイアログボックスの入力。

つ

通常実行　**シーケンス名を実行**を選択するか、**実行**メニューからプロセスモデルエントリポイントの 1 つを選択して、シーケンスエディタで実行を開始すること。

通常のシーケンスファイル　UUT をテストするシーケンスを含むすべてのシーケンスファイル。

て

テストエグゼクティブエンジン　シーケンスの作成、編集、実行、およびデバッグ用の API を提供する 1 つまたは一連のモジュール。シーケンスエディタまたはランタイムオペレータインタフェースは、テストエグゼクティブエンジンのサービスを使用します。

テストモジュール　テストを実行するコードモジュール。

テストユニット（UUT）　テストするデバイスまたはコンポーネント。

テンプレート　コードテンプレートを参照。

な

名前付きデータタイプ　ユーザにより固有の名前を与えられた変数またはプロパティのタイプ。データタイプには通常複数のサブプロパティが含まれるため、任意の複素データ構造が作成されます。データタイプを使用するすべての変数やプロパティは同じデータ構造ですが、含まれる値は異なることがあります。

ね

ネスト　他のステップまたはシーケンスにより呼び出されること。シーケンスがサブシーケンスを呼び出した場合、そのサブシーケンスは呼び出しシーケンスの呼び出しによりネストされたということになります。

ネストされた対話式実行　ブレークポイントで中断した通常実行の実行ウィンドウからステップを対話式に実行すること。実行が中断されたシーケンスおよびステップグループでのみステップを実行できます。選択したステップは、通常実行の状況で実行されます。

は

ハイライト　入力フォーカスが TestStand 画面に表示される方法；項目に入力フォーカスを置くこと。

配列プロパティ　同じタイプの単一値のプロパティの配列を含むプロパティ。

破棄　メモリのクリーンアップなしで実行のスレッドを終了し、実行、終了、または停止操作を中止すること。これにより TestStand は不確実な状態になることがあります。

ひ

標準指定データタイプ　TestStand で定義し指定しているデータタイプ。標準データタイプにサブプロパティを追加することはできますが、組み込みサブプロパティを削除することはできません。標準指定データタイプは、`Path`、`Error`、および `CommonResults` です。

ふ

プリステップサブステップ　ステップモジュールを呼び出す前にエンジンが呼び出すサブステップ。たとえば、プリステップサブステップは、計測構成パラメータを取り出し、それらをステップモジュールで使用するためにステッププロパティに保存するコードモジュールを呼び出すことがあります。

ブロックダイアグラム　プログラムまたはアルゴリズムの図による記述または表記。LabVIEW では、ノードと呼ばれる実行可能なアイコンとデータをノードからノードへ運ぶワイヤから構成されるブロックダイアグラムが、VI のソースコードです。ブロックダイアグラムは VI のダイアグラムウィンドウに常駐します。

プロセスモデル　テストエグゼクティブがテストを行うシーケンスを実行する前後の一連の操作。共通の操作としては、UUT の識別、オペレータへの合否の通知、テストレポートの生成、および結果の記録などがあります。

プロパティ　オブジェクトの設定または属性を保存し保持する情報のコンテナ。プロパティには、単一の値や同じタイプの値の配列がある場合や、値がまったくない場合もあります。プロパティには、サブプロパティをいくつでも含めることができます。各プロパティには名前が付いています。

プロパティ－配列プロパティ　単一のタイプのサブプロパティ配列である値を含むプロパティ。サブプロパティの配列に加えて、プロパティ－配列プロパティは他のタイプのサブプロパティをいくつでも含むことができます。

フロントエンドコールバック　シーケンスエディタおよびランタイムオペレータインタフェースが呼び出す共通のシーケンス。フロントエンドコールバックを使用すると、複数のアプリケーションで特定の操作の同じ実装を共有することができます。TestStand はフロントエンドコールバックシーケンス、`LoginLogout` を含むシーケンスファイル、`FrontEndCallback.seq` をインストールします。

フロントエンドコールバックシーケンスファイル　フロントエンドコールバックを含むシーケンスファイル。TestStand はフロントエンドコールバックシーケンス、`LoginLogout` を含むシーケンスファイル、`FrontEndCallback.seq` をインストールします。

フロントパネル　LabVIEW VI の対話式ユーザインタフェース。物理的な計測器のフロントパネルを模したもので、スイッチ、スライド、メータ、グラフ、チャート、ゲージ、LED、および他の制御器や表示器で構成されています。

ブレークポイント　プログラムの実行における中断。

へ

編集サブステップ　ステップを編集する際にエンジンが呼び出すサブステップ。コンテキストメニューの**モジュールを指定**より上に表示されるメニュー項目でサブステップを呼び出します。編集サブステップにより表示されるダイアログボックスで、シーケンスの開発者がカスタムステッププロパティの値を編集します。たとえば、数値リミットテストステップのコンテキストメニューには「リミットの編集」項目が表示され、合格／不合格テストステップのコンテキストメニューには「合格／不合格ソースの編集」項目が表示されます。

変数 ある特定の文脈において自由に作成できるプロパティ。変数は、シーケンスファイルに対してグローバルであったり、特定のシーケンスに対してローカルであったりすることができます。また、ステーショングローバル変数を持つこともできます。

変数ウィンドウ 現在アクティブになっているすべての変数またはプロパティの値を表示するウィンドウ。

ほ

ポストアクション ステップの合否状況またはステップの実行後にエンジンが評価するカスタム条件によって TestStand がとるアクション。ポストアクションにより、コールバックを実行したりステップの実行後に他のステップにジャンプしたりすることができます。

ポストステップサブステップ ステップモジュールを呼び出した後にエンジンが呼び出すサブステップ。ポストステップサブステップは、ステップモジュールがステッププロパティに保存した値を、編集サブステップが他のステッププロパティに保存したリミット値に対して比較するコードモジュールを呼び出すことがあります。

ボタン ダイアログボックスにあり、選択することによってそのダイアログボックスに関連するコマンドを実行する項目。

ポップアップメニュー コンテキストメニューを参照。

め

メインシーケンス UUT のテストを開始するシーケンス。プロセスモデルは全テストプロセスの一部としてメインシーケンスを呼び出します。プロセスモデルはテスティングプロセスにおいて一定であることを定義するのに対し、メインシーケンスは実行する異なるタイプのテストに対して固有のステップを定義します。

メソッド オブジェクトに対し操作や関数を実行します。

メニューバー メインメニューの名前を含む水平バー。

も

モジュールアダプタ　TestStand エンジンが使用するコンポーネントの 1 つで、他のシーケンスまたは他のコードモジュールでコードを呼び出します。コードモジュールでコードを呼び出す際、アダプタはその呼び出し方法とパラメータを渡す方法についての情報を持っています。

モデルコールバック　シーケンスファイルがプロセスモデルのシーケンスのデフォルト動作をカスタマイズすることを可能にするメカニズム。

モデルシーケンスファイル　プロセスモデルシーケンスを含む特別なタイプのシーケンスファイル。シーケンスファイル内のシーケンスが、UUT のテストの際に実行の高レベルシーケンスフローを方向付けます。

ゆ

ユーザマネージャ　ユーザのリスト、ユーザ名とパスワード、および特権を管理する TestStand エンジンのコンポーネント。ユーザマネージャには、シーケンスエディタのユーザマネージャウィンドウからアクセスします。

ら

ランタイムエラー　実行が強制的に終了されるエラー状態。シーケンス実行中にエラーが発生すると、TestStand はクリーンアップステップにジャンプし、エラーはすべての呼び出しシーケンスへ伝えられ、さらに最上位シーケンスまで伝えられます。

ランタイムオペレータインタフェース　製造ステーションでシーケンスを実行するためのグラフィカルユーザインタフェースを提供するプログラム。シーケンスエディタとランタイムオペレータインタフェースは、同じプログラムの別の局面であることもあります。

り

リストボックス　選択可能な項目のリストを表示するダイアログボックスの要素。

リソース文字列　アプリケーションを直接変更することなく文字列を変更できるようにするために、外部ファイルに保存されたテキスト文字列。

リファレンスカウント　各 ActiveX オブジェクトは、それを参照するものの数を追跡します。それにより、オブジェクトは使用するリソースをいつ解放すべきか決めることができます。

リファレンスプロパティ　ActiveX リファレンスのプロパティを参照。

る

ルート対話式実行　独立した実行でシーケンスファイルウィンドウから選択したステップを実行すること。ルート対話式実行は、プロセスモデルを呼び出しません。

ろ

ローカル変数　値または追加のサブプロパティを保持するシーケンスのプロパティ。プロパティ値に直接アクセスできるのは、シーケンス内のステップのみです。

わ

ワイヤ　ソース端子とシンク端子の間のデータの進路を定義するツール。

索引

D

H

L

N

O

T

V

あ

う

え

か

き

け

こ

さ

し

す

せ

た

つ

て

と

は

ふ

へ

ほ

め

も

ゆ

ら

る

れ

ろ