NATIONAL INSTRUMENTS™
LabVIEW™

ユーザマニュアル

NATIONAL INSTRUMENTS™

2002 年 01 月版
製品番号 321187D-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 3390 150,
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 E「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

ComponentWorks™、DAQPad™、DataSocket™、HiQ™、HiQ-Script™、LabVIEW™、National Instruments™、NI™、NI-488™、NI-488.2™、ni.com™、NI-DAQ™、NI-FBUS™、NI-VISA™、SCXI™は、National Instruments Corporationの商標です。本書に掲載されている製品および会社名は該当各社の商標または商号です。

特許および商標に関する表示

National Instrumentsの製品を保護する特許については、**ヘルプ→特許**を選択すると表示される製品情報（該当する場合）、このCDにある `patents.txt` ファイル（該当する場合）および/または `www.ni.com/patents` を参照してください。

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。

目次

第 4 章
フロントパネルを作成する

第 5 章
ブロックダイアグラムを作成する

第 6 章
VI の実行とデバッグ

第 7 章
VI およびサブ VI を作成する

第 II 部
VI の作成と編集

第 8 章
ループと Case ストラクチャ

第 11 章
グラフとチャート

第 12 章
グラフィック & サウンド VI

第 13 章
ファイル I/O

第 14 章
VI の文書化と印刷

第 15 章
VI をカスタマイズする

第 16 章
プログラム的に VI を制御する

第 17 章
LabVIEW のネットワーク動作

第 18 章
ActiveX

本書について

本書では、テストや計測、データ集録、計測器制御、データロギング、データ解析、レポート生成などのアプリケーションを LabVIEW で作成するための、LabVIEW のグラフィカルなプログラミング環境とその技術について説明します。

本書では、LabVIEW ユーザインタフェース、プログラミング作業スペース、さらに LabVIEW のパレット、ツール、ダイアログボックスなどの LabVIEW プログラミング機能について説明します。ただし、各パレット、ツール、メニュー、ダイアログボックス、制御器、組み込み VI、組み込み関数などの固有の情報については説明しません。これらの項目の詳細および LabVIEW の機能を使用して特定のアプリケーションを作成するための詳しい手順については、「LabVIEW ヘルプ」を参照してください。「LabVIEW ヘルプ」の詳細とそのアクセス方法については、第 1 章「LabVIEW の概要」の「LabVIEW の関連資料」のセクションを参照してください。

『LabVIEW ユーザマニュアル』は、ポータブルドキュメント形式 (Portable Document Format：PDF) でも入手可能です。**フルインストール**オプションを選択すると、すべての LabVIEW マニュアルの PDF バージョンがインストールされます。PDF バージョンは LabVIEW で**ヘルプ→印刷版マニュアルを表示**を選択すればアクセスできます。

メモ PDF を表示するには、Adobe Acrobat Reader のバージョン 4.0 以降がインストールされている必要があります。Acrobat Reader をダウンロードするには、アドビシステムズ社の Web サイト `www.adobe.co.jp` にアクセスしてください。

「LabVIEW ヘルプ」からも PDF にアクセスできますが、それには Acrobat Reader をインストールしておく必要があります。PDF バージョンの LabVIEW マニュアルをインストールする方法の詳細については、『LabVIEW リリースノート』または『LabVIEW アップグレードノート』を参照してください。LabVIEW ライブラリ (PDF) にアクセスする方法の詳細については、第 1 章「LabVIEW の概要」の「LabVIEW の関連資料」のセクションを参照してください。

本書の構成

『LabVIEW ユーザマニュアル』は 2 部構成になっています。第 I 部「LabVIEW の概念」では、LabVIEW でアプリケーションを作成するためのプログラミング概念について説明します。第 I 部では、LabVIEW のプ

ログラミング環境について説明し、アプリケーションの作成の計画を立てるのに役立つ情報を提供します。

第 II 部 「VI の作成と編集」では、特定の方法でのアプリケーション操作を可能にするために使用する LabVIEW の機能、VI、および関数について説明します。第 II 部では、LabVIEW の各機能の使用法や、VI および関数の各クラスの概要について説明します。

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、ユーザへのアドバイスを表しています。

 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。

第 I 部

LabVIEW の概念

第 I 部では、LabVIEW でアプリケーションを作成するためのプログラミング概念について説明します。各章では LabVIEW プログラミング環境について説明し、アプリケーションの計画に役立つ情報を提供します。

第 I 部「LabVIEW の概念」は、以下の章で構成されています。

- 第 1 章「LabVIEW の概要」では、LabVIEW とその広範なマニュアル、VI の設計および作成に役立つツールについて説明します。
- 第 2 章「バーチャルインスツルメンツ (仮想計測器) VI の概要」では、VI：仮想計測器 (バーチャルインスツルメンツ) の構成要素について説明します。
- 第 3 章「LabVIEW 環境」では、VI のフロントパネルおよびブロックダイアグラムの作成に使用する LabVIEW のパレット、ツール、およびメニューについて説明します。この章では、LabVIEW のパレットをカスタマイズする方法やいくつかの動作環境オプションを設定する方法についても説明します。
- 第 4 章「フロントパネルを作成する」では、VI のフロントパネルの作成方法について説明します。
- 第 5 章「ブロックダイアグラムを作成する」では、VI のブロックダイアグラムの作成方法について説明します。
- 第 6 章「VI の実行とデバッグ」では、VI の動作を構成する方法やブロックダイアグラムの構成やブロックダイアグラムでやり取りされるデータに関する問題を識別する方法について説明します。
- 第 7 章「VI およびサブ VI を作成する」では、VI およびサブ VI の作成方法、VI の配布方法、およびスタンドアロンアプリケーションと共有ライブラリの作成方法について説明します。

1

LabVIEW の概要

LabVIEW は、テキスト行ではなくアイコンを使用してアプリケーションを作成するグラフィカルなプログラミング言語です。命令でプログラムを実行するテキストベースのプログラミング言語とは異なり、LabVIEW では、データの流れでプログラムを実行するデータフロープログラミングを使用します。

LabVIEW では、一連のツールおよびオブジェクトを使用してユーザインタフェースを作成します。ユーザインタフェースをフロントパネルと呼びます。フロントパネルのオブジェクトを制御するには、グラフィカルに表現された関数を使用してコードを追加します。このコードは、ブロックダイアグラムに含まれています。ブロックダイアグラムは、いくつかの点でフローチャートに似ています。

特殊なアプリケーションを開発する場合は、数種類のアドオンソフトウェアツールセットを購入できます。すべてのツールセットはシームレスに LabVIEW に統合されます。これらのツールセットの詳細に関しては、ナショナルインスツルメンツのホームページ `www.ni.com/jp` を参照してください。

LabVIEW の関連資料

LabVIEW には、初心者と上級者を対象にしたさまざまな資料が用意されています。すべての LabVIEW マニュアルおよびアプリケーションノートは PDF でも入手可能です。PDF を表示するには、Adobe Acrobat Reader のバージョン 4.0 以降がインストールされている必要があります。Acrobat Reader をダウンロードするには、アドビシステムズ社の Web サイト `www.adobe.co.jp` にアクセスしてください。最新の関連資料については、ナショナルインスツルメンツの製品マニュアルのページ `ni.com/jp/manuals` を参照してください。

- **『LabVIEW ライブラリ (PDF)』**この PDF ファイル を使用すると、すべての LabVIEW マニュアルおよびアプリケーションノート (PDF バージョン) を検索できます。**ヘルプ→印刷版マニュアルを表示**と選択すると、『LabVIEW ライブラリ』にアクセスできます。
- **『LabVIEW 入門』**このマニュアルには、LabVIEW グラフィカルプログラミング環境と、データ集録および計測器制御アプリケーションの作成に使用する LabVIEW の基本機能が説明されています。

- **『LabVIEW チュートリアル』**このチュートリアルを使用すると、LabVIEW の基本概念を習得できます。チュートリアルでは、複数の作業を通してグラフィカルプログラミングに慣れることができます。『LabVIEW チュートリアル』にアクセスするには、LabVIEW の初期画面に表示される **LabVIEW チュートリアル**ボタンをクリックします。
- **『LabVIEW クイックリファレンスカード』**このカードを使用すると、LabVIEW を短時間で開始することができます。このカードでは、LabVIEW パレット、一般的な編集、配線、デバッグテクニックについて説明します。
- 『LabVIEW ユーザマニュアル』このマニュアルでは、テスト、計測、データ集録、計測器制御、データロギング、データ解析、およびレポート生成アプリケーションを作成するための LabVIEW プログラミングの概念、テクニック、機能、VI、および関数について説明します。
- **「LabVIEW ヘルプ」**このヘルプファイルは、LabVIEW のパレット、メニュー、ツール、VI、および関数に関する情報のリファレンスとして使用します。また、「LabVIEW ヘルプ」では、LabVIEW の機能の使用方法を順を追って説明します。**ヘルプ→ヘルプを表示**を選択して、「LabVIEW ヘルプ」にアクセスできます。

  「LabVIEW ヘルプ」には、以下のリソースへのリンクが含まれています。

  - 『LabVIEW チュートリアル』
  - 『LabVIEW ライブラリ (PDF)』。すべての LabVIEW マニュアルおよびアプリケーションノート (PDF バージョン) が含まれています。
  - ナショナルインスツルメンツのホームページには、Developer Zone、技術サポートデータベース、Product Manuals Library などのテクニカルサポートのリソースがあります。

メモ　**（Macintosh および UNIX）**ナショナルインスツルメンツでは、Netscape 6.0 以降を使用するか、または「LabVIEW ヘルプ」を参照することをお勧めします。

- **『LabVIEW Measurements Manual』**このマニュアルでは、LabVIEW でのデータ集録および計測器制御アプリケーションの作成の詳細について説明します。LabVIEW を初めて使用する場合は、このマニュアルの前に『LabVIEW 入門』と『LabVIEW ユーザマニュアル』をお読みください。
- **『LabVIEW Development Guidelines』**このマニュアルでは、容易に理解、使用、修正できる VI の作成方法について説明します。このマニュアルでは、プロジェクトトラッキング、設計、および文書化のテクニックについて説明します。

メモ　『LabVIEW Development Guidelines』（印刷版）は、LabVIEW プロフェッショナル開発システムのみを対象としたマニュアルです。PDF は LabVIEW のすべてのパッケージにあります。

- **『ポイントバイポイント VI 入門』**このマニュアルでは、LabVIEW のポイントバイポイント解析について説明します。
- **『Using External Code in LabVIEW』**このマニュアルでは、コードインタフェースノード（Code Interface Node）および外部サブルーチンを使用して、テキストベースのプログラミング言語で作成されたコードをインポートする方法について説明します。共有外部サブルーチン、関数ライブラリ、メモリおよびファイル操作ルーチン、および診断ルーチンについて説明します。このマニュアルでは、DLL を呼び出す方法についても説明します。

メモ　『Using External Code in LabVIEW』は PDF でのみ入手可能です。

- **『LabVIEW アプリケーションノート』**LabVIEW アプリケーションノートでは、LabVIEW の高度な概念と用途について説明します。最新のアプリケーションノートに関しては、National Instruments Developer Zone（zone.ni.com）を参照してください。
- **『LabVIEW VXI VI Reference Manual』**このマニュアルでは、LabVIEW の VXI VI について説明します。このマニュアルは、VXI ハードウェアに付属の『NI-VXI Programmer Reference Manual 』とセットになっています。

メモ　**『LabVIEW VXI VI Reference Manual』**は PDF でのみ入手可能です。

LabVIEW のサンプル VI およびツール

LabVIEW のサンプル VI およびツールは、VI の設計と作成に役立ちます。

LabVIEW のサンプル VI

LabVIEW には数百ものサンプル VI があり、それらを使用したり、独自の VI に組み込むことができます。ユーザのアプリケーションに合うようにサンプルを変更したり、1 つまたは複数のサンプルを独自の VI にコピーして貼り付けることもできます。**ヘルプ→サンプルの検索**を選択して、サンプル VI を参照または検索します。追加のサンプル VI については、NI Developer Zone（ni.com）を参照してください。

LabVIEW のツール

LabVIEW には、次のものを含む測定デバイスの迅速な構成に役立つ数多くのツールがあります。これらのツールは、**ツール**メニューからアクセスできます。

- **(Windows)** Measurement & Automation Explorer を使用して、ナショナルインスツルメンツのハードウェアおよびソフトウェアを構成します。
- **(Macintosh)** NI-DAQ 構成ユーティリティを使用して、ナショナルインスツルメンツ DAQ ハードウェアを構成します。
- **(Macintosh)** DAQ チャネルウィザードを使用して、DAQ ハードウェアチャネルに接続されているデバイスのタイプを定義します。チャネルを定義すると、DAQ チャネルウィザードにその設定が保存されます。
- **(Windows および Macintosh)** DAQ Channel Viewer には、構成されている DAQ チャネルがリストされます。
- **(Windows および Macintosh)** DAQ ソリューションウィザードを使用すると一般的な DAQ アプリケーションのソリューションを検索できます。サンプル VI を選択したり、カスタム VI を作成できます。

2

バーチャルインスツルメンツ (仮想計測器) VI の概要

LabVIEW プログラムは、その外観と動作状態がオシロスコープやマルチメータなどの実際の計測器に似ているため、バーチャルインスツルメンツ (仮想計測器)、つまり VI と呼ばれています。各 VI では、ユーザインタフェースや他のソースからの入力値を操作したり、その情報を表示したり、他のファイルやコンピュータにその入力値を渡す関数を使用します。

VI には以下の 3 つの要素があります。

- **フロントパネル**：ユーザインタフェースとして機能します。
- **ブロックダイアグラム**：VI の機能を定義するグラフィカルなソースコードを含んでいます。
- **アイコンおよびコネクタペーン**：VI を別の VI の中で使用できるように識別します。他の VI 内の VI をサブ VI と呼びます。サブ VI は、テキストベースのプログラミング言語のサブルーチンに相当します。

詳細については

VI およびサブ VI の作成の詳細については、「LabVIEW ヘルプ」を参照してください。

フロントパネル

フロントパネルは VI のユーザインタフェースです。図 2-1 はフロントパネルの例を示しています。

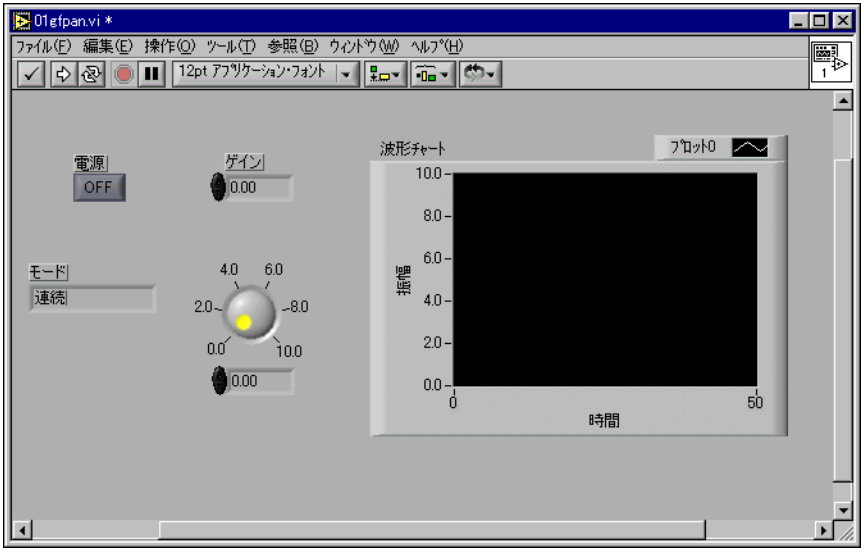


図 2-1　フロントパネルの例

VI の対話式の入力および出力端子である制御器と表示器を用いて、フロントパネルを作成します。制御器には、ノブ、押しボタン、ダイアル、その他の入力デバイスがあります。表示器には、グラフ、LED、その他のディスプレイがあります。制御器は計測器入力デバイスをシミュレーションし、VI のブロックダイアグラムにデータを供給します。表示器は計測器出力デバイスをシミュレーションし、ブロックダイアグラムが集録または生成するデータを表示します。フロントパネルの詳細については、第 4 章「フロントパネルを作成する」を参照してください。

ブロックダイアグラム

フロントパネルを作成したら、グラフィカルに表現された関数を使用してコードを追加し、フロントパネルオブジェクトを制御します。ブロックダイアグラムには、このグラフィカルソースコードが含まれています。フロントパネルオブジェクトは、ブロックダイアグラムでは端子として表示されます。ブロックダイアグラムの詳細については、第 5 章「ブロックダイアグラムを作成する」を参照してください。

図 2-2 の VI は、簡単なブロックダイアグラムオブジェクト（端子、関数、ワイヤ）を示しています。

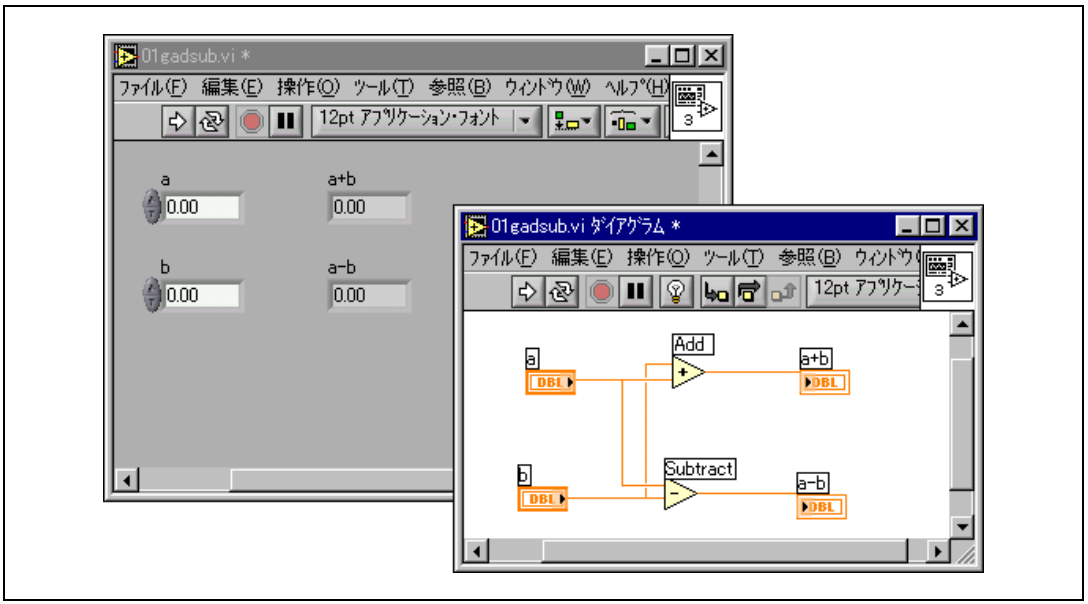


図 2-2　ブロックダイアグラムと対応するフロントパネルの例

端子

端子は制御器または表示器のデータタイプを表します。たとえば、左図に示す DBL 端子は、データタイプが倍精度浮動小数点数の制御器または表示器を表します。LabVIEW のデータタイプとそのグラフィカル表現の詳細については、第 5 章「ブロックダイアグラムを作成する」の「制御器および表示器のデータタイプ」のセクションを参照してください。

端子は、フロントパネルとブロックダイアグラムの間で情報を交換する入出力ポートです。フロントパネルの制御器に入力したデータ (図 2-2 の **a** と **b**) は、制御器端子を介してブロックダイアグラムに入力されます。次に、このデータは Add および Subtract 関数に渡されます。Add および Subtract 関数で内部計算が終了すると新しいデータ値が生成されます。そのデータは表示器端子へ渡り、ブロックダイアグラムを出て、フロントパネルにもう一度入り、フロントパネルの表示器に表示されます。

ノード

ノードは、入力端子や出力端子を持ち、VI の実行時に演算を実行するブロックダイアグラム上のオブジェクトです。テキストベースのプログラミング言語におけるステートメント、演算子、関数、およびサブルーチンに

似ています。図 2-2 では Add および Subtract 関数がノードです。ノードの詳細については、第 5 章「ブロックダイアグラムを作成する」の「ブロックダイアグラムのノード」のセクションを参照してください。

ワイヤ

ブロックダイアグラムオブジェクト間のデータ転送はワイヤを介して行います。図 2-2 では、制御器および表示器の DBL 端子と Add および Subtract 関数がワイヤで接続されています。各ワイヤのデータソースは 2 つですが、そのデータを読み取る多くの VI および関数に配線できます。ワイヤの色、種類、太さはデータタイプによって異なります。不良ワイヤは黒い破線で表示されます。ワイヤの詳細については、第 5 章「ブロックダイアグラムを作成する」の「ブロックダイアグラムオブジェクトをワイヤで 接続する」のセクションを参照してください。

ストラクチャ

ストラクチャは、テキストベースのプログラミング言語におけるループおよびケースステートメントのグラフィカル表現です。コードのブロックを繰り返したり、条件付きでコードを実行したり、特定の順序でコードを実行するには、ブロックダイアグラムでストラクチャを使用します。ストラクチャの詳細と例については、第 8 章「ループと Case ストラクチャ」を参照してください。

アイコンとコネクタペーン

VI フロントパネルとブロックダイアグラムを作成した後でアイコンとコネクタペーンを作成すると、その VI をサブ VI として使用できます。各 VI では、フロントパネルおよびブロックダイアグラムウィンドウの右上隅に左図に示すアイコンが表示されます。アイコンは VI のグラフィカル表現です。アイコンには、テキスト、画像、またはその両方を含めることができます。ある VI をサブ VI として使用する場合、アイコンはその VI のブロックダイアグラム上にあるサブ VI を識別します。アイコンは、ダブルクリックすると、カスタマイズまたは編集できます。アイコンの詳細については、第 7 章「VI およびサブ VI を作成する」の「アイコンを作成する」のセクションを参照してください。

ある VI をサブ VI として使用するには、左図のようなコネクタペーンを作成する必要があります。コネクタペーンはその VI の制御器および表示器に対応する一連の端子であり、テキストベースのプログラミング言語の関数呼び出しのパラメータリストに似ています。コネクタペーンは、VI に配線できる入力端子と出力端子を定義するため、サブ VI として使用することができます。コネクタペーンは、その入力端子で受け取ったデータを

フロントパネルの制御器を介してブロックダイアグラムのコードに渡し、フロントパネルの表示器からの結果を出力端子で受け取ります。

コネクタペーンを初めて表示すると、コネクタパターンが現れます。希望に応じて異なるパターンを選択することができます。一般に、コネクタペーンにはフロントパネルの各制御器または各表示器に 1 つの端子があります。コネクタペーンには最大 28 個の端子を割り当てることができます。新しい入力端子または出力端子が必要となる VI に変更することが予想される場合は、割り当てられていない余分な端子を残してください。コネクタペーンのセットアップの詳細については、第 7 章「VI およびサブ VI を作成する」の「コネクタペーンを設定する」のセクションを参照してください。

メモ　1 つの VI に 17 個以上の端子を割り当てないように注意してください。端子が多すぎると、VI が煩雑でわかりにくくなる可能性があります。

VI およびサブ VI の使用とカスタマイズ

VI を作成してアイコンおよびコネクタペーンを作成すると、その VI をサブ VI として使用できます。サブ VI の詳細については、第 7 章「VI およびサブ VI を作成する」の「サブ VI」のセクションを参照してください。

VI を個々のファイルとして保存したり、複数の VI をグループ化して VI ライブラリに保存することができます。ライブラリに VI を保存する方法の詳細については、第 7 章「VI およびサブ VI を作成する」の「VI を保存する」のセクションを参照してください。

VI の外観と動作をカスタマイズできます。作成するすべての VI に対してカスタムメニューを作成したり、VI のメニューバーの表示／非表示を設定することもできます。VI のカスタマイズの詳細については、第 15 章「VI をカスタマイズする」を参照してください。

3

LabVIEW 環境

LabVIEW のパレット、ツール、メニューを使用して、VI のフロントパネルおよびブロックダイアグラムを作成します。**制御器**パレットと**関数**パレットのカスタマイズおよび、複数の作業環境オプションの設定が可能です。

詳細については

パレット、メニュー、ツールバーの使用と作業環境のカスタマイズについての詳細は、「LabVIEW ヘルプ」を参照してください。

制御器パレット

制御器パレットは、フロントパネル上でのみ使用できます。**制御器**パレットは、フロントパネルの作成に使用する制御器と表示器を含みます。制御器と表示器はそのタイプごとにまとめられたサブパレットにあります。文字列制御器および表示器の詳細については、第 4 章「フロントパネルを作成する」の「フロントパネル制御器および表示器」のセクションを参照してください。

ウィンドウ→制御器パレットを表示を選択するか、フロントパネルの作業スペースを右クリックして、**制御器**パレットを表示します。**制御器**パレットは画面上の任意の場所に配置できます。

制御器パレットの表示方法は変更できます。**制御器**パレットのカスタマイズの詳細については、この章の「制御器および関数パレットをカスタマイズする」のセクションを参照してください。

関数パレット

関数パレットは、ブロックダイアグラム上でのみ使用できます。**関数**パレットには、ブロックダイアグラムを作成する VI と関数があります。VI と関数はそのタイプごとにまとめられたサブパレットにあります。VI と関数のタイプに関する詳細は、第 5 章「ブロックダイアグラムを作成する」の「関数の概要」のセクションを参照してください。

ウィンドウ→関数パレットを表示を選択するか、またはブロックダイアグラムの作業スペースを右クリックして**関数**パレットを表示します。**関数**パレットは画面上の任意の場所に配置できます。

関数パレットの表示方法は変更できます。**関数**パレットのカスタマイズの詳細については、この章の「制御器および関数パレットをカスタマイズする」のセクションを参照してください。

制御器パレットと関数パレットを操作する

制御器および**関数**パレット上の操作ボタンを使用すると、制御器、VI、および関数を操作したり検索できます。サブパレットのアイコンをクリックすると、パレット全体が、選択したサブパレットに置き換わります。サブパレット上の VI アイコンを右クリックし、ショートカットメニューから**VI を開く**を選択して VI を開くこともできます。

制御器および**関数**パレットには以下の操作ボタンがあります。

- **1 つ上のパレットへ移動**：パレット階層の 1 つ上のレベルに移動します。

- **検索**：パレットを検索モードに変更します。検索モードでは、パレット上の制御器、VI、または関数をテキストベースで検索できます。

- **オプション**：パレットの外観を設定できる**関数ブラウザオプション**ダイアログボックスを開きます。

ツールパレット

ツールパレットは、フロントパネルとブロックダイアグラムで使用できます。ツールは、マウスカーソルの特別な操作モードです。カーソルは、パレットで選択されたツールのアイコンに対応します。ツールを使用して、フロントパネルオブジェクトとブロックダイアグラムオブジェクトを操作したり変更します。

ウィンドウ→ツールパレットを表示を選択して、**ツール**パレットを表示します。**ツール**パレットは画面上の任意の場所に配置できます。

ヒント　<Shift> キーを押したまま右クリックすると、カーソルの位置に**ツール**パレットが一時的に表示されます。

自動ツール選択機能が有効になっている場合にフロントパネルまたはブロックパネルのオブジェクトの上にカーソルを移動すると、LabVIEW は**ツール**パレットから自動的に対応するツールを選択します。**ツール**パレッ

トの**自動ツール選択**ボタンをクリックするか、または <Shift-Tab> キーを押すと、自動ツール選択を切り替えることができます。

メニューとツールバー

メニューおよびツールバーを使用して、フロントパネルオブジェクトとブロックダイアグラムオブジェクトを操作したり変更します。ツールバーボタンを使用して VI を実行します。

メニュー

VI ウィンドウの一番上にあるメニューには、他のアプリケーションに共通の**開く**、**保存**、**コピー**、**貼り付け**などの他に、LabVIEW に固有の項目があります。ショートカットキーも併せて表示されるメニューもあります。

(Macintosh) メニューは画面の一番上に表示されます。

メモ　VI が実行モード時には使用できないメニュー項目もあります。

ショートカットメニュー

最も頻繁に使用されるメニューはオブジェクトショートカットメニューです。LabVIEW のすべてのオブジェクトと、フロントパネルおよびブロックダイアグラム上の空白部分は、ショートカットメニューに関連付けられています。ショートカットメニューを使用して、フロントパネルおよびブロックダイアグラムオブジェクトの外観と動作を変更できます。ショートカットメニューにアクセスするには、オブジェクト、フロントパネル、またはブロックダイアグラムを右クリックします。

(Macintosh) <Command> キーを押したまま、オブジェクト、フロントパネル、またはブロックダイアグラムをクリックします。

実行モード時のショートカットメニュー

VI の実行時または実行モードでは、すべてのフロントパネルオブジェクトに一連のショートカットメニューの縮小版があります。オブジェクトの内容の切り取り、コピー、貼り付けを行ったり、オブジェクトの設定をデフォルトに戻したり、オブジェクトの説明を読み取るには、このショートカットメニューの縮小版を使用します。

追加オプションがついた複雑な制御器もあります。たとえば、配列のショートカットメニューには、値の範囲をコピーしたり、最後の配列要素を表示する項目があります。

ツールバー

ツールバーボタンを使用して、VI を実行したり編集します。VI を実行すると、VI のデバッグに使用できるボタンがツールバーに表示されます。

作業環境をカスタマイズする

制御器および**関数**パレットの表示方法を変更できます。また、**オプション**ダイアログボックスを使用して他の作業環境オプションを設定できます。

制御器および関数パレットをカスタマイズする

制御器および**関数**パレットを以下の方法でカスタマイズできます。

- パレットに VI および制御器を追加する。
- ユーザごとに異なるビューを設定する。たとえば、LabVIEW を使用しやすくするために、あるユーザには一部の VI と関数しか表示せず、他のユーザにはパレットをすべて表示する。
- 頻繁に使用する VI と関数にアクセスしやすくするために、組み込みパレットを並び替える。
- 一連の ActiveX 制御器をカスタム制御器に変換し、パレットに追加する。
- ツールセットをパレットに追加する。

VI と制御器をユーザライブラリおよび計測器ライブラリに追加する

制御器および**関数**パレットに VI や制御器を追加する最も簡単な方法は、VI や制御器を `user.lib` ディレクトリに保存することです。LabVIEW を再起動すると、**関数→ユーザライブラリ**および**制御器→ユーザ制御器**パレットは、各ディレクトリ用のサブパレット、VI ライブラリ（`.llb`）、またはメニュー（`.mnu`）ファイルを `user.lib` に含み、各ファイルのアイコンを `user.lib` に含みます。特定のディレクトにファイルを追加したり、特定のディレクトリからファイルを削除した後に LabVIEW を再起動すると、LabVIEW によってパレットが自動的に更新されます。

関数→計測器 I/O →計測器ドライバパレットは、`instr.lib` ディレクトリに対応します。このディレクトリに計測器ドライバを保存すると、**関数**パレットに簡単にアクセスすることができます。

この方法で**制御器**パレットと**関数**パレットに VI または制御器を追加する場合は、パレット上での VI または制御器の位置を正確に指定できません。

パレットビューの作成と編集

制御器パレットと**関数**パレットに追加する VI および制御器の位置を正確に制御するには、パレットビューを作成する必要があります。LabVIEW には 4 つの組み込みパレットビュー、すなわち基本、データ集録、デフォルト、試験 & 計測があります。

LabVIEW では、**制御器**および**関数**パレットの情報を `labview\menus` ディレクトリに格納しています。`menus` ディレクトリには、ユーザが作成またはインストールした各ビューに相当するディレクトリがあります。ネットワーク上で LabVIEW を実行する場合は、ユーザごとに各 `menus` ディレクトリを定義できます。これにより他のユーザに簡単にビューを転送できます。

パレットの新しいビューを作成すると、LabVIEW は、オリジナルのデフォルトビューをコピーします。ユーザはそのコピーに変更を加えます。LabVIEW は、ユーザが変更を加える前に、`labview\menus` ディレクトリにあるオリジナルの組み込みパレットをコピーします。組み込みパレットが保護されているので、元のビューを破壊せずにパレットを試すことができます。

LabVIEW のビューの格納方法

`.mnu` ファイルと `.llb` ファイルには、それぞれ 1 つの**制御器**パレットと 1 つの**関数**パレットが含まれています。さらに、各ファイルには**制御器**および**関数**パレットのアイコンが含まれています。作成した各サブパレットは別の `.mnu` ファイルに格納する必要があります。

ビューを選択すると、LabVIEW はそのビューに対応するディレクトリの `menus` ディレクトリをチェックします。LabVIEW は、ビューを作成するたびに自動的に作成するディレクトリ内の `root.mnu` ファイルから、最上位の**制御器**および**関数**パレットとサブパレットを作成します。

LabVIEW はパレット上に各 VI または制御器のアイコンを作成します。各サブディレクトリ、`.mnu` ファイル、および `.llb` ファイルにはパレット上にサブパレットを作成します。

ActiveX サブパレットを作成する

フロントパネル上の ActiveX 制御器を使用する場合は、**ツール→上級→ActiveX コントロールをインポート**を選択して一連の ActiveX 制御器をカスタム制御器に変換し、それらを**制御器**パレットに追加します。LabVIEW はデフォルトで制御器を `user.lib` ディレクトリに保存します。`user.lib` 内のすべてのファイルとディレクトリはパレットに自動的に表示されるからです。

パレット内にツールセットを表示する

vi.lib\addons にインストールしたツールセットは、LabVIEW の再起動後、**制御器**および**関数**パレットの最上位に自動的に表示されます。他の場所にツールセットをインストールしている場合は、ツールセットを addons ディレクトリに移動すると簡単にアクセスできます。

注意　アップグレード時に LabVIEW が vi.lib ディレクトリのファイルを上書きするので、vi.lib ディレクトリにユーザ独自の VI や制御器を保存しないでください。VI や制御器を**制御器**および**関数**パレットに追加するには user.lib に保存します。

作業環境オプションを設定する

LabVIEW をカスタマイズするには**ツール→オプション**を選択します。**オプション**ダイアログボックスを使用して、パス、パフォーマンス & ディスク、フロントパネル、ブロックダイアグラム、取り消し回数、デバッグツール、色、フォント、印刷、**履歴**ウィンドウ、時間と日付の形式、および他の LabVIEW の機能のオプションを設定します。

オプションダイアログボックス内の一番上のプルダウンメニューを使用して、異なるカテゴリのオプションの中から選択します。

LabVIEW のオプションの格納方法

オプションダイアログボックスを使用するので、オプションを手動で編集したり、その正確な形式を知っておく必要はありません。LabVIEW は各プラットフォームごとに異なる方法でオプションを格納します。

Windows

LabVIEW は LabVIEW ディレクトリの labview.ini ファイルにオプションを格納します。ファイル形式は他の .ini ファイルと同じです。LabVIEW セクションマーカで始まり、offscreenUpdates=True のようにオプション名と値が続きます。

別のオプションファイルを使用する場合は、LabVIEW の起動に使用するショートカット内にそのファイルを指定します。たとえば、labview.ini の代わりにユーザのコンピュータ上の lvrc という名前のオプションファイルを使用するには、デスクトップ上の LabVIEW のアイコンを右クリックして**プロパティ**を選択します。**ショートカット**タブをクリックして**リンク先**テキストボックス内に「labview -pref lvrc」と入力します。

Macintosh

LabVIEW は**システム→初期設定**フォルダ内の LabVIEW Preferences テキストファイルにオプションを格納します。

別のオプションファイルを使用する場合は、LabVIEW Preferences ファイルを **LabVIEW** フォルダにコピーして、**オプション**ダイアログボックス内でオプションを変更します。LabVIEW を起動すると、LabVIEW はまず **LabVIEW** フォルダ内でオプションファイルを探します。LabVIEW フォルダにファイルが見つからないと、**システム**フォルダを探します。そのフォルダでもファイルが見つからないと、**システム**フォルダ内に新規ファイルを作成します。LabVIEW は**オプション**ダイアログボックス内でのすべての変更内容を、LabVIEW が見つけた最初の LabVIEW Preferences ファイルに書き込みます。

UNIX

LabVIEW はユーザのホームディレクトリ内の .labviewrc ファイルにオプションを格納します。**Options** ダイアログボックス内でオプションを変更した場合、LabVIEW は変更内容を .labviewrc ファイルに書き込みます。プログラムディレクトリ内に labviewrc ファイルを作成して、VI 検索パスなど、すべてのユーザに共通のオプションを格納できます。フォントや色の設定など、ユーザごとに異なるオプションを格納するには .labviewrc ファイルを使用します。これはホームディレクトリ内の .labviewrc ファイルのエントリは、プログラムディレクトリ内の競合するエントリよりも優先されるからです。

たとえば、/opt/labview 内に LabVIEW ファイルをインストールした場合、LabVIEW は最初に /opt/labview/labviewrc からオプションを読み取ります。**Options** ダイアログボックス内でアプリケーションフォントなどのオプションを変更した場合、LabVIEW は変更内容を .labviewrc ファイルに書き込みます。次に LabVIEW を起動したとき、LabVIEW は、/opt/labview/labviewrc に定義されているデフォルトのアプリケーションフォントの代わりに、.labviewrc ファイル内のアプリケーションフォントオプションを使用します。

オプションエントリは、オプション名、コロン (:)、値の順に構成されます。オプション名はピリオド (.) で始まり、後ろにオプションが続く実行可能形式です。LabVIEW は、オプション名を検索するときに大文字と小文字を区別します。オプション値を二重引用符または単一引用符で囲むことができます。たとえば、デフォルトの倍精度を使用するには、以下のエントリをホームディレクトリ内の .labviewrc ファイルに追加します。

```
labview.defPrecision : double
```

別のオプションファイルを使用する場合は、LabVIEW を起動するときにコマンドライン上にそのファイルを指定します。たとえば、.labviewrc

の代わりに test ディレクトリ内の lvrc という名前のファイルを使用するには、「labview -pref /test/lvrc」と入力します。LabVIEW は **Options** ダイアログボックス内でのすべての変更内容を lvrc オプションファイルに書き込みます。コマンドライン上でオプションファイルを指定すると、LabVIEW はプログラムディレクトリ内の labviewrc ファイルを読み取りますが、コマンドライン上に指定されたオプションファイルは、プログラムディレクトリ内の競合するエントリよりも優先されます。

4

フロントパネルを作成する

フロントパネルは VI のユーザインタフェースです。通常、最初にフロントパネルを設計し、フロントパネル上に作成した入出力においてタスクを実行するために、ブロックダイアグラムを設計します。ブロックダイアグラムの詳細については、第 5 章「ブロックダイアグラムを作成する」を参照してください。

VI の対話式の入力および出力端子である、制御器と表示器を用いてフロントパネルを作成します。制御器とは、ノブ、押しボタン、ダイアル、その他の入力デバイスです。表示器とは、グラフ、LED、その他のディスプレイです。制御器は計測器入力デバイスをシミュレーションし、VI のブロックダイアグラムにデータを供給します。表示器は計測器出力デバイスをシミュレーションし、ブロックダイアグラムが集録または生成するデータを表示します。

ウィンドウ→制御器パレットを表示を選択して**制御器**パレットを表示し、次に**制御器**パレットから制御器と表示器を選択し、フロントパネル上に配置します。

詳細については

フロントパネルの設計および構成方法の詳細については、「LabVIEW ヘルプ」を参照してください。

フロントパネル上でオブジェクトを構成する

制御器と表示器のショートカットメニューの使用、フロントパネルオブジェクトの順序の設定、およびインポートされたグラフィックの使用により、フロントパネルをカスタマイズできます。フロントパネルオブジェクトを手動でサイズ変更したり、ウィンドウサイズが変更されたときに自動的にサイズ変更するように設定できます。カスタム制御器、表示器およびタイプ定義の作成と使用についての詳細は、『LabVIEW Custom Controls, Indicators, and Type Definitions』アプリケーションノートを参照してください。

オプション項目を表示または非表示にする

フロントパネルの制御器と表示器には、ユーザが表示、非表示を決めることができるオプション項目があります。オブジェクトを右クリックしてショートカットメニューから**項目を表示**を選択し、使用できる項目のリストを表示します。ほとんどのオブジェクトにはラベルとキャプションがあります。ラベルとキャプションの詳細についてはこの章の「ラベルを付ける」のセクションを参照してください。

制御器を表示器に、表示器を制御器に変更する

LabVIEW は、**制御器**パレット内のオブジェクトを、最初は一般的な用途に基づいて制御器または表示器として構成します。たとえば、**制御器→ブール**パレットからトグルスイッチを選択すると、フロントパネル上に制御器として表示されます。通常、トグルスイッチは入力デバイスだからです。LED を選択すると、フロントパネル上に表示器として表示されます。通常、LED は出力デバイスだからです。

一部のパレットには同じタイプまたはクラスのオブジェクト用の制御器と表示器が含まれています。たとえば、**制御器→数値**パレットにはデジタル制御器とデジタル表示器が含まれています。

オブジェクトを右クリックしてショートカットメニューから**制御器に変更**または**表示器に変更**を選択することにより、制御器を表示器に、または表示器を制御器に変更できます。

フロントパネルオブジェクトを入れ替える

フロントパネルオブジェクトを別の制御器または表示器に入れ替えることができます。オブジェクトを右クリックしてショートカットメニューから**入れ換え**を選択すると、**制御器**パレットが既に開いている場合でも、一時的に**制御器**パレットが表示されます。一時的な**制御器**パレットから制御器または表示器を選択して、フロントパネル上の現在のオブジェクトを入れ替えます。

ショートカットメニューから**入れ換え**を選択すると、名前、説明、デフォルトデータ、データフローの方向 (制御器または表示器)、色、サイズなどの、元のオブジェクトに関する可能な限りの情報が保たれます。ただし、新しいオブジェクトは固有のデータタイプを保持します。オブジェクトの端子またはローカル変数からのワイヤはブロックダイアグラム上に残りますが、不良になっている可能性があります。たとえば、数値の端子を文字列の端子に入れ替えると、元のワイヤはブロックダイアグラム上に残りますが、破線になっています。

新しいオブジェクトが入れ替えオブジェクトに似ているほど、元の特性を多く保持できます。たとえば、スライドを別のスタイルのスライドと入れ

替えると、新しいスライドの高さ、スケール、値、名前、説明は同じになります。これに対し、スライドを文字列制御器に入れ替えると、LabVIEW は名前、説明、およびデータフローの方向だけを保持します。スライドは文字列制御器との共通点が少ないからです。

既存のフロントパネル制御器および表示器を入れ替える場合は、クリップボードからオブジェクトを貼り付けることもできます。この方法では古いオブジェクトの特性は何も保持されませんが、ワイヤはオブジェクトに配線されたままです。

制御器のキーボードショートカットを設定する

マウスを使用せずにフロントパネルを操作するために制御器にキーボードショートカットを割り当てることが可能です。制御器を右クリックしてショートカットメニューから**上級→キー操作**を選択し、**キー操作**ダイアログボックスを開きます。

VI の実行中にキーボードショートカットを入力すると、関連付けられている制御器にフォーカスが移動します。制御器がテキスト制御器またはデジタル制御器の場合、LabVIEW はテキストをハイライトし編集可能にします。制御器がブールの場合、値の変更にはスペースバーまたは <Enter> を押してください。

上級→キー操作ショートカットメニュー項目では表示器は淡色表示されます。表示器にはデータを入力できないからです。

キー操作でボタン動作を制御する

フロントパネルの動作を制御するさまざまなボタンにファンクションキーを関連付けることができます。VI のボタンがダイアログボックスと同様に動作するように定義できます。たとえば、<Enter> キーを押すことがデフォルトボタンをクリックすることと同じになるように定義できます。

(Macintosh および Sun) <Return> キーを押すことはデフォルトボタンをクリックすることと同じになります。

<Enter> または <Return> キーをダイアログボックスのボタンに関連付けると、LabVIEW は自動的にそのボタンの周りを太い枠で囲みます。

<Enter> または <Return> キーを制御器に関連付けると、そのフロントパネル上の文字列制御器は復帰文字を受け取ることができません。したがって、そのフロントパネル上のすべての文字列は 1 行に入力されます。より長い文字列を操作するには、スクロールバーを使用できます。

ブール制御器を繰り返し、<Enter> または <Return> キーを押すと、他の制御器が <Enter> あるいは <Return> キーをキーボードショートカット

として使用している場合でも、そのブール制御器は変更されます。割り当てられた <Enter> または <Return> キーボードショートカットは、ブール制御器が選択されていない場合にのみ適用されます。

フロントパネルオブジェクトの操作順序を設定する

フロントパネルオブジェクトの操作順序を設定するには、**編集→タブ順序を設定**を選択します。その後、VI の実行中に <Tab> キーを使用してオブジェクトを操作できます。

フロントパネル上の制御器および表示器はパネル順序と呼ばれる順序を持っていますが、これはフロントパネル上の制御器と表示器の位置とは無関係です。フロントパネル上に最初に作成した制御器または表示器が要素 0、2 番目が 1、以下同様になります。制御器または表示器を削除すると、パネル順序は自動的に調整されます。

パネル順序は、VI 実行時の操作順序を決めます。パネル順序は、フロントパネルデータをロギングするときに、ユーザが作成するデータログファイルのレコードに制御器および表示器が記録される順序も決めます。データのロギングの詳細については、第 13 章「ファイル I/O」の「フロントパネルのデータを記録する」のセクションを参照してください。

VI の実行中にユーザが <Tab> キーを使用して制御器にアクセスするのを防ぐには、**キー操作**ダイアログボックス内の**タブする時この制御器をとばす**チェックボックスをオンにします。

オブジェクトの色を決める

ユーザは多くのオブジェクトの色を変更できますが、すべての色を変更できるわけではありません。たとえば、フロントパネルオブジェクトのブロックダイアグラム端子およびワイヤは、それらが転送するデータのタイプと表現を示す特定の色を使用しています。そのため、ユーザはそれらの色を変更できません。

フロントパネルオブジェクトまたはフロントパネルとブロックダイアグラムの作業スペースの色を追加あるいは変更するには、色付けツールを使用し、オブジェクトまたは作業スペースを右クリックします。**ツール→オプション**を選択し、プルダウンメニューから**色**を選択することにより、ほとんどのオブジェクトのデフォルト色も変更できます。

インポートされたグラフィックを使用する

他のアプリケーションからグラフィックをインポートして、フロントパネルの背景、リング制御器の項目、その他の制御器および表示器の部分として使用できます。制御器内でのグラフィックの使用方法の詳細については、『LabVIEW Custom Controls, Indicators, and Type Definitions』アプリケーションノートを参照してください。

グラフィックをインポートするには、クリップボードにコピーしてフロントパネル上に貼り付けます。**編集→画像からインポート**を選択することによっても可能です。

インポートされたグラフィックを使用した制御器の例については `examples\general\controls\custom.llb` を参照してください。

オブジェクトをグループ化およびロックする

グループ化およびロックするフロントパネルオブジェクトを選択するには、位置決めツールを使用します。ツールバーの**再順序**ボタンをクリックし、プルダウンメニューから**グループ**または**ロック**を選択します。位置決めツールでオブジェクトを移動しサイズを変更する場合、グループ化されたオブジェクトはその相対位置とサイズを維持します。ロックされたオブジェクトはフロントパネル上でその位置を維持し、オブジェクトをアンロックするまで削除できません。オブジェクトのグループ化とロックを同時にすることができます。位置決めツール以外のツールはグループ化またはロックされたオブジェクトに通常どおり作用します。

オブジェクトをサイズ変更する

ほとんどのフロントパネルオブジェクトのサイズは変更できます。位置決めツールをサイズ変更可能なオブジェクトに移動すると、四角形のオブジェクトの場合はそのコーナーにサイズ変更ハンドルが表示され、円形のオブジェクトの場合はその上にサイズ変更円が表示されます。オブジェクトをサイズ変更しても、フォントサイズはそのままです。オブジェクトのグループをサイズ変更するとグループ内のすべてのオブジェクトがサイズ変更されます。

デジタル数値制御器および表示器のような一部のオブジェクトはサイズ変更すると水平または垂直方向にのみサイズが変わります。ノブのような他のオブジェクトはサイズ変更しても同じ比率を保持します。位置決めカーソルは同じように表示されますが、オブジェクトを囲んでいる破線の枠は一方向にしか動きません。

オブジェクトをサイズ変更するとき、手動でサイズ変更の方向を制限できます。オブジェクトを垂直または平行方向にのみサイズ変更、または現在の垂直と水平の比率を維持するには、<Shift> キーを押したままオブジェ

クトをクリックし、ドラッグします。中心点を中心にしてオブジェクトをサイズ変更するには、<Ctrl-Shift> を押したままサイズ変更カーソルをクリックします。

(Macintosh)<Option-Shift> キーを押します。**(Sun)**<Meta-Shift> キーを押します。**(Linux)**<Alt-Shift> キーを押します。

フロントパネルオブジェクトをスケールする

フロントパネルウィンドウをサイズ変更するとき、フロントパネルオブジェクトをスケールしたり、ウィンドウサイズに応じて自動的にサイズが変わるように設定できます。フロントパネル上の 1 つのオブジェクトをスケールするようにも設定できます。また、フロントパネル上のすべてのオブジェクトをスケールするように設定することもできます。ただし、スケールするオブジェクトをすべて設定するか、またはあらかじめオブジェクトをグループ化しないと、フロントパネル上の複数のオブジェクトをスケールするようには設定できません。オブジェクトをスケールするには、オブジェクトを選択し、**編集→オブジェクトをパネルに合わせてスケール**を選択します。

1 つのフロントパネルオブジェクトをスケールするように設定した場合、フロントパネルウィンドウのサイズ変更に応じてオブジェクトは自動的にサイズ変更されます。フロントパネル上の他のオブジェクトはフロントパネル上の以前の配置と一致するように再配置されますが、フロントパネルの新しいウィンドウサイズに合わせてスケールされません。

フロントパネル上の 1 つのオブジェクトを自動的にスケールするように指示するとすぐに、図 4-1 のような灰色の線がフロントパネル上の数個の領域を囲みます。それらの領域は、スケールするオブジェクトを基準にした他のフロントパネルオブジェクトの位置を明確にします。フロントパネルウィンドウをサイズ変更すると、スケールするように設定したオブジェクトは自動的にサイズ変更され、元の位置を基準に再配置されます。
VI を実行すると灰色の線は消えます。

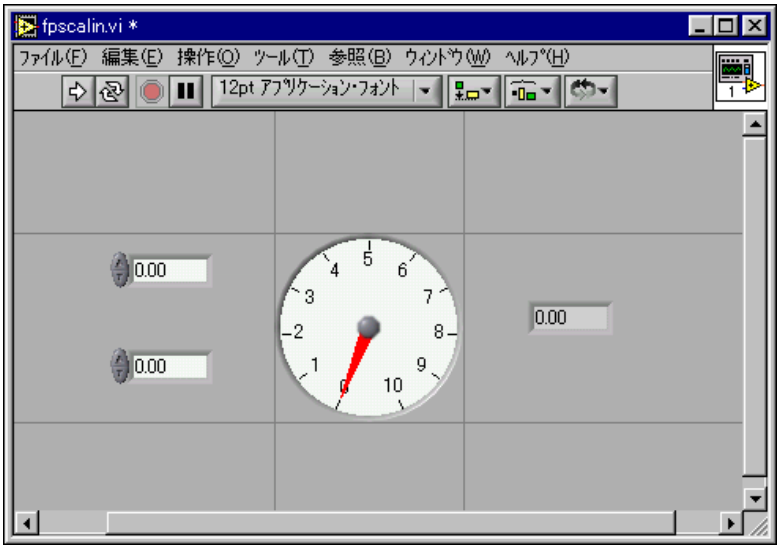


図 4-1　スケールするように設定されたオブジェクトを持つフロントパネル

LabVIEW がオブジェクトを自動的にスケールするときは、ユーザが手動でオブジェクトをサイズ変更したときと同じ規則に従います。たとえば、一部のオブジェクトは水平または垂直方向にのみサイズ変更でき、フォントサイズはオブジェクトがサイズ変更されても同じサイズのままです。

LabVIEW がオブジェクトを自動的にスケールした後、ウィンドウをサイズ変更して元の位置に戻してもオブジェクトが正確に元のサイズに戻らないことがあります。元のフロントパネルウィンドウとオブジェクトのサイズを復元するには、VI を保存する前に**編集→取り消し**を選択します。

配列をスケールするように設定したり、配列内のオブジェクトをスケールするように設定できます。配列をスケールするように設定する場合は、配列内に見ることができる行および列の数を調整します。配列内のオブジェクトをスケールするように設定すると、サイズは異なっても常に同じ数の行および列が配列内に表示されます。

クラスタをスケールするように設定したり、クラスタ内のオブジェクトをスケールするように設定することもできます。クラスタ内のオブジェクトをスケールするように設定すると、クラスタもそれに応じて調整されます。

ウィンドウをサイズ変更せずにフロントパネルにスペースを追加する

ウィンドウをサイズ変更せずにフロントパネルにスペースを追加できます。込み入った状態で配置またはグループ化されたオブジェクト間のスペースを大きくするには、<Ctrl> キーを押し、位置決めツールを使用し

てフロントパネルの作業スペースをクリックします。キーを押したまま、挿入するサイズだけ領域をドラッグして広げます。

(Macintosh)<Option> キーを押します。**(Sun)**<Meta> キーを押します。**(Linux)**<Alt> キーを押します。

破線の枠が付いた四角形により、スペースが挿入される場所が定義されます。キー操作を解除してスペースを追加します。

フロントパネル制御器および表示器

フロントパネルを作成するには、**制御器**パレット上にあるフロントパネル制御器および表示器を使用します。制御器には、ノブ、押しボタン、ダイアル、その他の入力デバイスがあります。表示器には、グラフ、LED、その他のディスプレイがあります。制御器は計測器入力デバイスをシミュレーションし、VI のブロックダイアグラムにデータを供給します。表示器は計測器出力デバイスをシミュレーションし、ブロックダイアグラムで集録または生成されるデータを表示します。

3 次元および旧バージョンの制御器と表示器

多くのフロントパネルオブジェクトの外観はハイカラーの 3 次元です。オブジェクトの外観を良くするために少なくとも 16 ビットの色を表示するようにモニタを設定します。

3 次元フロントパネルオブジェクトはローカラーの 2 次元のオブジェクトにも対応しています。256 色および 16 色のモニタ設定を持つ VI を作成するには、**制御器→旧バージョンの制御器**パレット上にある 2 次元制御器および表示器を使用します。

スライド、ノブ、ダイアル、およびデジタルディスプレイ

スライド、ノブ、ダイアル、およびデジタルディスプレイをシミュレーションするには、**制御器→数値**および**制御器→旧バージョンの制御器→数値**パレット上にある数値制御器および表示器を使用します。このパレットには、色の値を設定するカラーボックスおよびカラーランプも含まれています。数値制御器および表示器を使用して、数値データを入力および表示します。

スライド制御器および表示器

スライド制御器および表示器には、垂直と水平のスライド、タンク、および温度計があります。スライド制御器または表示器の値を変更するには、操作ツールを使用してスライダを新しい位置までドラッグするか、スライドオブジェクトの新しいポイントをクリックするか、あるいはオプション

のデジタルディスプレイを使用します。スライダを新しい位置にドラッグし、その変更の間 VI が実行されている場合、VI がどのくらいの頻度で制御器を読み取るかによって制御器は VI に中間値を渡します。

スライド制御器または表示器は 2 つ以上の値を表示できます。スライダを追加するには、オブジェクトを右クリックし、ショートカットメニューから**スライダを追加**を選択します。複数のスライダを持つ制御器のデータタイプは各数値を含むクラスタです。クラスタの詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「クラスタ」のセクションを参照してください。

ロータリ制御器および表示器

ロータリ制御器および表示器はノブ、ダイアル、ゲージ、およびメータを含んでいます。ロータリオブジェクトはスライド制御器および表示器と同じように動作します。ロータリ制御器または表示器の値を変更するには、指針を移動するか、ロータリオブジェクトのポイントをクリックするか、またはオプションのデジタルディスプレイを使用します。

ロータリ制御器または表示器は 2 つ以上の値を表示できます。新しい指針を追加するには、オブジェクトを右クリックしショートカットメニューから**指針を追加**を選択します。複数の指針を持つ制御器のデータタイプは各数値を含むクラスタです。クラスタの詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「クラスタ」のセクションを参照してください。

デジタル制御器および表示器

デジタル制御器および表示器は、数値データを入力したり表示するのに最も簡単な手段です。これらのフロントパネルオブジェクトを水平に拡張して、より多くの桁を表示することができます。以下の方法でデジタル制御器または表示器の値を変更できます。

- 操作ツールまたはラベリングツールを使用してデジタル表示ウィンドウの内側をクリックし、キーボードから数値を入力します。
- 操作ツールを使用して、デジタル制御器の増分矢印ボタンまたは減分矢印ボタンをクリックします。
- 操作ツールまたはラベリングツールを使用して、変更する桁の右側にカーソルを置き、キーボードの上矢印キーまたは下矢印キーを押します。

カラーボックス

カラーボックスは特定の値に相当する色を表示します。たとえば、カラーボックスを使用して、範囲外の値などさまざまな状態を示すことができます。色の値は RRGGBB 形式の 16 進数で表します。最初の 2 桁は赤の値

を制御します。次の 2 桁は緑の値を制御します。最後の 2 桁は青の値を制御します。

操作ツールまたは色付けツールでカラーボックスをクリックしてカラーパレットを表示し、カラーボックスの色を設定します。

カラーランプ

カラーランプは色で数値を表示します。それぞれに数値とそれに対応する表示色を持つ、少なくとも 2 つの任意のマーカから成るカラースケールを構成します。入力値が変化すると、表示色はその値に対応する色に変わります。ゲージが危険な値に達したときの警告範囲などのように、カラーランプはデータ範囲を視覚的に示すのに便利です。たとえば、カラーランプを使用して強度チャートおよびグラフにカラースケールを設定できます。強度チャートおよびグラフの詳細については、第 11 章「グラフとチャート」の「強度グラフおよびチャート」のセクションを参照してください。

外観、サイズ、色、および色数をカスタマイズするには、カラーランプを右クリックしてショートカットメニューオプションを使用します。

また、フロントパネル上のすべてのノブ、ダイアル、ゲージにカラーランプを追加できます。メータはデフォルトで可視のカラーランプを持ちます。

ボタン、スイッチ、およびライト

ボタン、スイッチ、およびライトをシミュレーションするユーザインタフェースを作成するには、**制御器→ブール**および**制御器→旧バージョンの制御器→ブール**パレット上にあるブール制御器および表示器を使用します。ブール制御器および表示器を使用して、ブール値 (TRUE/FALSE) を入力し表示します。たとえば、測定温度を監視する場合に、温度が一定のレベルを超えたら警告するように、フロントパネルにブール警告ライトを配置できます。

ブールオブジェクトの外観とクリックしたときの動作をカスタマイズするには、ショートカットメニューを使用します。

テキスト入力ボックス、ラベル、およびパス表示

テキスト入力ボックスおよびラベルをシミュレーションしてファイルやディレクトリの位置を入力したりその位置を返すには、**制御器→文字列 & パス**または**制御器→旧バージョンの制御器→文字列 & パス**パレット上にある文字列 & パス制御器および表示器を使用します。

文字列制御器および表示器

フロントパネルの文字列制御器にテキストを入力したり編集するには、操作ツールやラベリングツールを使用します。デフォルトでは、新たに入力したテキストや変更したテキストは、編集セッションを終了するまでブロックダイアグラムには渡されません。編集セッションを終了するには、パネル上の制御器以外の場所をクリックするか、別のウィンドウを表示するか、ツールバーの **Enter** ボタンをクリックするか、または数値キーパッドの <Enter> キーを押します。キーボードの <Enter> キーを押すと復帰文字が入力されます。

(Macintosh および Sun) キーボードの <Return> キーを押すと復帰文字が入力されます。

文字列制御器および表示器の詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「フロントパネルの文字列」のセクションを参照してください。

パス制御器および表示器

パス制御器および表示器を使用して、ファイルやディレクトリの位置を入力したりその位置を返します。パス制御器および表示器の動作は文字列制御器および表示器に似ていますが、LabVIEW は使用しているプラットフォーム標準の構文を使用してパスをフォーマットします。

無効なパス

パスを返す関数が失敗すると、その関数は無効なパス値である無効パスを表示器に返します。パス制御器のデフォルトとして無効パス値を使用すると、ユーザがパスを指定しなかったときにそれを検知して、パスを選択するオプションのあるファイルダイアログボックスを表示することができます。ファイルダイアログボックスを表示するには、File Dialog 関数を使用します。

空のパス

パス制御器の空のパスは、Windows および Macintosh では空の文字列として表示され、UNIX ではスラッシュ (/) として表示されます。パスの指定を促すプロンプトを表示するには、空のパスを使用します。ファイル I/O 関数に空のパスを配線すると、空のパスはコンピュータにマップされたドライブの一覧を参照します。

(Macintosh) 空のパスはマウントされたボリュームを参照します。
(UNIX) 空のパスはルートディレクトリを参照します。

配列とクラスタのそれぞれの制御器と表示器

制御器→配列 & クラスタまたは**制御器→旧バージョンの制御器→配列 & クラスタ**パレット上にある配列 & クラスタ制御器および表示器を使用して、他の制御器および表示器の配列とクラスタを作成します。配列とクラスタの詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「データを配列やクラスタとグループ化する」のセクションを参照してください。

配列 & クラスタパレットには、標準のエラークラスタ制御器および表示器、タブ制御器およびタブ表示器も含まれています。エラークラスタの詳細については、第 6 章「VI の実行とデバッグ」の「不定データを防ぐ」のセクションを参照してください。バリアント制御器の詳細については、第 5 章「ブロックダイアグラムを作成する」の「バリアントデータを処理する」のセクションを参照してください。

タブ制御器

タブ制御器を使用すると、フロントパネル制御器および表示器を狭い領域内で重ねることができます。タブ制御器はページとタブから構成されています。タブ制御器の各ページにフロントパネルオブジェクトを配置し、タブをセレクタとして使用して複数のページを表示します。タブ制御器に配置できるフロントパネルオブジェクトの数に制限はありません。

タブ制御器は、複数のフロントパネルオブジェクトを同時に使用する場合や操作の特定の段階で使用する場合に便利です。たとえば、テストを開始する前にまずいくつかの設定を構成した後、テストの進行中にその内容を修正し、最後に関連データのみを表示して保存できる VI を使用している場合などです。

ブロックダイアグラムでは、タブ制御器はデフォルトで列挙型制御器です。タブ制御器上の制御器および表示器の端子は、他のブロックダイアグラム端子として表示されます。列挙型制御器の詳細については、この章の「列挙型制御器」のセクションを参照してください 。

リストボックス

制御器→リスト & 表または**制御器→旧バージョンの制御器→リスト & 表**パレット上にあるリストボックス制御器を使用して、選択項目のリストを表示します。リストボックスの設定により 1 項目または複数項目の選択の承認ができます。項目サイズおよび作成日時等の各項目の詳細を表示するには、複数列リストボックスを使用してください。

以下のような方法でリスト項目を変更したりリスト項目に情報を追加するには、リストボックス制御器のプロパティノードを使用します。

- 項目文字列を設定する。
- ディレクトリとファイルには異なる記号が付いている**ファイル**ダイアログボックスのように、リスト項目の横に記号を付加する。
- 各リスト項目を無効にする。
- リスト項目の間に区切り線を入れる。
- 制御器の値を読み取って現在選択されている項目を検出する。
- ダブルクリックなどで指定された項目を検出する。

プロパティノードの詳細については、第 16 章「プログラム的に VI を 制御する」の「プロパティノード」のセクションを参照してください。

リストボックスは入力完成機能をサポートします。つまり、最初の数文字を入力するとリストボックス内で一致する項目が検出されます。一致した次の項目に移動するには <Tab> キーを押します。前の項目に戻るには <Shift-Tab> キーを押します。

リストボックスには自動的にスクロールバーが表示されますが、このスクロールバーはリストボックスの情報が表示できる量を超えた場合にのみ有効になります 。

リングおよび列挙型制御器と表示器

制御器→リング &Enum および**制御器→旧バージョンの制御器→リング &Enum** パレット上にあるリングおよび列挙型の制御器および表示器を使用して、繰り返し表示できる文字列の一覧を作成します。

リング制御器

リング制御器は、数値を文字列またはピクチャに関連付ける数値オブジェクトです。リング制御器は、選択肢を繰り返し表示できるプルダウンメニューとして表示されます。

リング制御器は、トリガモードなどの互いに排他的な項目を選択するときに便利です。たとえば、連続、単独、または外部トリガから選択する場合にはリング制御器を使用します。

リング制御器の項目は入力した順に並びます。各項目には 0 ～ *n*–1 の範囲の数値が割り当てられます。ここで *n* は項目の数です。*n*–1 以上の値を入力した場合はリング制御器に最後の項目が表示され、0 以下の値を入力した場合は最初の項目が表示されます。

リング制御器は入力完成機能をサポートしており、プルダウンメニューにはスクロールバーが表示されます。

列挙型制御器

選択する動作の一覧を表示するには、列挙型制御器を使用します。列挙型制御器、つまり enum は文字列のリング制御器に似ています。ただし、リング制御器の場合は値が数字ですが、列挙型制御器の場合は値が文字列です。たとえば、Case ストラクチャでケースを選択するときには列挙型制御器を使用します。Case ストラクチャの詳細については、第 8 章「ループと Case ストラクチャ」の「Case ストラクチャ」のセクションを参照してください。

列挙型制御器のデータタイプは符号なしバイト整数、符号なし 2 バイト整数、または符号なし 4 バイト整数です。制御器のデータタイプを変更するには、列挙型制御器を右クリックしてショートカットメニューから**表記法**を選択します。

上級の列挙型制御器および表示器

Increment および Decrement を除くすべての算術関数は、列挙型制御器を符号なし数値と同じように処理します。Increment は最後の列挙値を最初の値まで増分し、Decrement は最初の列挙値を最後の値まで減分します。符号付き整数を列挙型に強制的に変換する場合、負の値は最初の列挙値に変換され、範囲外の正の値は最後の列挙値に変換されます。範囲外の符号なし整数は常に最後の列挙値に変換されます。

浮動小数点数を列挙型制御器に接続する場合、数値は列挙型表示器内の最も近い列挙項目に変換されます。範囲外数値は前述の説明と同様に処理されます。列挙型制御器を数値に接続する場合、数値は列挙型指標です。列挙型制御器を列挙型表示器に配線するには、列挙項目が一致している必要があります。表示器は、制御器の項目より多い数の項目を含むことができます。

I/O 名制御器および表示器

構成する DAQ チャネル名、VISA リソース名、および IVI 論理名を I/O VI に渡し、計測器や DAQ デバイスと通信するには、**制御器→ I/O** または**制御器→旧バージョンの制御器→ I/O** パレット上にある I/O 名制御器および表示器を使用します。

I/O 名定数は、**関数→計測器 I/O** パレットおよび**関数→データ集録**パレットにあります。

(Windows) DAQ チャネル名、VISA リソース名、および IVI 論理名を構成するには、**ツール**メニューからアクセスできる Measurement & Automation Explorer を使用します。

(Macintosh) ナショナルインスツルメンツの DAQ ハードウェアを構成するには、**ツール**メニューからアクセスできる NI-DAQ 構成ユーティリ

ティを使用します。DAQ チャネル名を構成するには、**Tools** メニューからアクセスできる DAQ Channel Wizard を使用します。

(Macintosh および UNIX) VISA リソース名および IVI 論理名を構成するには、計測器の構成ユーティリティを使用します。構成ユーティリティの詳細については、ご使用の計測器のマニュアルを参照してください。

IMAQ セッション制御器は、ハードウェアへの接続を示す固有の識別子です。

波形制御器

波形の個々のデータ要素を操作するには、**制御器→ I/O** または**制御器→ 旧バージョンの制御器→ I/O** パレットにある波形制御器を使用してください。ピクチャデータタイプの詳細については、第 11 章「グラフとチャート」の「波形データタイプ」のセクションを参照してください。

オブジェクトまたはアプリケーションへのリファレンス

ファイル、ディレクトリ、デバイス、およびネットワーク接続を操作するには、**制御器→ Refnum** または**制御器→旧バージョンの制御器→ Refnum** パレット上にある refnum 制御器および表示器を使用します。サブ VI にフロントパネルオブジェクト情報を渡すには制御器 refnum を使用します。制御器 refnum の詳細については、第 16 章「プログラム的に VI を 制御する」の「フロントパネルオブジェクトを制御する」のセクションを参照してください。

リファレンス番号、つまり refnum は、ファイル、デバイス、ネットワーク接続などのオブジェクトに固有の識別子です。ファイル、デバイス、またはネットワーク接続を開くと、それらに関連付けられた refnum が作成されます。開いているファイル、デバイス、またはネットワーク接続で行うすべての操作で、各オブジェクトを識別する refnum が使用されます。VI との間で refnum の受け渡しを行うには、refnum 制御器または表示器を使用します。たとえば、ファイルを閉じて開く操作を行わずに refnum が参照しているファイルの内容を変更するには、refnum 制御器または表示器を使用します。

refnum は開いているオブジェクトを指すテンポラリポインタであるため、そのオブジェクトを開いている間だけ有効です。オブジェクトを閉じると、refnum とオブジェクトの関連付けは解除され、refnum は破棄されます。オブジェクトをもう一度開くと、最初の refnum とは異なる新しい refnum が作成されます。

オブジェクトから読み取った現在の位置、オブジェクトに書き込んだ現在の位置、ユーザのアクセス回数など、各 refnum に関連付けられた情報は保存されるので、2 つのオブジェクトに対して独立した複数の操作を並

行して行うことができます。VI が 1 つのオブジェクトを複数回開くと、開くたびに異なる refnum が返されます。

ダイアログ制御器

作成するダイアログボックス内にある**制御器→ダイアログ制御器**パレット上にあるダイアログ制御器を使用します。ダイアログ制御器および表示器は、ダイアログボックスで使用するために特別に設計されており、リング制御器、ボタン、タブダイアログボックス、チェックボックス、ラジオボタンなどがあります。これらの制御器の外観はフロントパネルに表示される制御器の外観とは異なります。これらの制御器はデスクトップに設定された色で表示されます。

ダイアログ制御器の外観は VI を実行するプラットフォームによって異なるので、作成する VI の制御器の外観はすべての LabVIEW のプラットフォーム上で互換性があります。別のプラットフォームで VI を実行する場合、ダイアログ制御器の色と外観はそのプラットフォームの標準ダイアログボックス制御器に合わせて変わります。

メニューバーとスクロールバーを非表示にし、各プラットフォームの標準ダイアログボックスと同じ外観と動作の VI を作成するには、**ファイル→VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**ウィンドウ外観**を選択します。VI の外観と動作の構成の詳細については、第 15 章「VI をカスタマイズする」の「VI の外観と動作を設定する」のセクションを参照してください。

ラベルを付ける

ラベルは、フロントパネルおよびブロックダイアグラムのオブジェクトを識別するために使用します。

LabVIEW には、所有ラベルとフリーラベルの 2 種類のラベルがあります。所有ラベルは特定のオブジェクトに属し、そのオブジェクトとともに移動します。また、そのオブジェクトにのみ注釈を付けます。所有ラベルは個別に移動できますが、ラベルを所有するオブジェクトを移動すると、ラベルもそのオブジェクトとともに移動します。所有ラベルを隠すことはできますが、ラベル単独でのコピーまたは削除はできません。ショートカットメニューから**項目を表示→単位ラベル**を選択すると、数値制御器に単位ラベルを表示できます。数値単位の詳細については、第 5 章「ブロックダイアグラムを作成する」の「数値単位および厳密類別化チェック」のセクションを参照してください。

フリーラベルはどのオブジェクトにも属していないため、個別に作成、移動、回転、または削除が可能です。フロントパネルおよびブロックダイアグラムに注釈を付けるには、フリーラベルを使用します。

フリーラベルを作成したり、フリーラベルや所有ラベルを編集するには、ラベリングツールを使用します。

キャプション

フロントパネルオブジェクトにはキャプションを付けることもできます。キャプションを表示するには、オブジェクトを右クリックし、ショートカットメニューから**項目を表示→キャプション**を選択します。ラベルとは異なり、キャプションはオブジェクトの名前に影響を与えず、オブジェクトラベルの補足説明として使用できます。キャプションはブロックダイアグラム上には表示されません。

テキストの特性

LabVIEW は、ご使用のコンピュータにインストールされているフォントを使用します。テキストの属性を変更するには、ツールバーの**テキスト設定**プルダウンメニューを使用します。オブジェクトまたはテキストを選択した後に**テキスト設定**プルダウンメニューから選択すると、変更は選択したすべてのものに適用されます。何も選択しないと、変更はデフォルトのフォントに適用されます。デフォルトのフォントが変わっても、既存のラベルのフォントは変わりません。新しいフォントは、変更後に作成したラベルにのみ適用されます。

選択したテキストに特定のフォントスタイルを適用するには、フロントパネル上の**テキスト設定**プルダウンメニューから**フォントダイアログ**を選択します。どのテキストも選択しなかった場合は、**パネルのデフォルト**オプションのチェックマークがオンになります。どのオブジェクトも選択せずにブロックダイアグラムから**テキスト設定→フォントダイアログ**を選択すると、**ダイアグラムのデフォルト**オプションにチェックマークが表示されます。フロントパネルとブロックダイアグラムには、別々のフォントを設定できます。たとえば、ブロックダイアグラムには小さなフォントを表示し、フロントパネルには大きなフォントを表示できます。

テキスト設定プルダウンメニューには、以下の組込みフォントが含まれています。

- **アプリケーションフォント**：**制御器**パレットと**関数**パレット、および新しい制御器のテキストに使用されるデフォルトのフォント。
- **システムフォント**：メニューに使用されるフォント。
- **ダイアログフォント**：ダイアログボックスのテキストに使用されるフォント。

これらの組み込みフォントのいずれかを含む VI を他のプラットフォームに転送する場合、最も近いフォントが使用されます。

テキスト設定プルダウンメニューには、**サイズ**、**スタイル**、**位置調整**、および**色**サブメニュー項目があります。

これらのサブメニューからフォントについて加えた変更は、選択したオブジェクトに適用されます。たとえば、ノブとグラフを選択して別のフォントを選択すると、ラベル、スケール、およびデジタル表示がすべてそのフォントに変更されます。

LabVIEW は、変更時にできるだけ多くのフォント属性を維持します。たとえば、いくつかのオブジェクトを Courier フォントに変更した場合、そのオブジェクトは、可能であればそのサイズとスタイルを保持します。**テキスト設定**ダイアログボックスを使用すると、選択したオブジェクトが選択したテキスト特性に変更されます。組み込みフォントのいずれかまたは現在使用中のフォントを選択すると、選択したオブジェクトは、そのフォントに関連付けられているフォントとサイズに変更されます。

スライドのように複数のセクションにテキストが含まれるオブジェクトを操作する場合、フォントを変更すると現在選択しているオブジェクトまたはテキストが影響を受けます。たとえば、スライド全体を選択して、**テキスト設定**プルダウンメニューから**スタイル→太字**を選択すると、スケール、デジタル表示、およびラベルはすべて太字フォントに変更されます。ラベルだけを選択して**太字**を選択すると、ラベルのみが太字フォントに変更されます。スケールマーカからテキストを選択して**太字**を選択すると、すべてのマーカが太字フォントに変更されます。

ユーザインタフェースを設計する

VI がユーザインタフェースまたはダイアログボックスとして機能する場合、フロントパネルの外観とレイアウトは重要です。フロントパネルは、実行するアクションを容易に識別できるように設計します。フロントパネルは計測器や他のデバイスと同様に設計できます。

ユーザインタフェースの設計についての詳細は『LabVIEW Development Guidelines』を参照してください。

イベントを使用してユーザインタフェースの機能を拡張する方法については、第 8 章「ループと Case ストラクチャ」の「Case ストラクチャとシーケンスストラクチャ」のセクションを参照してください。

フロントパネルの制御器および表示器を使用する

制御器と表示器はフロントパネルの主要なコンポーネントです。フロントパネルを設計する場合は、ユーザがどのように VI と対話するかを考慮に入れて、制御器と表示器を論理的にグループ化します。複数の制御器が関連している場合は、それらの制御器の周囲に装飾フレームを追加したり、

それらをクラスタに入れます。フロントパネルのオブジェクト（ボックス、線または矢印）をグループ化または分割するには、**制御器→装飾**または**制御器→旧バージョンの制御器→装飾**パレット上にある装飾を使用します。これらのオブジェクトは装飾のみでデータを表示しません。

フロントパネルオブジェクトの間隔が狭くならないように配置してください。フロントパネルを読みやすくするため、ある程度の間隔を空けます。また、間隔を空けることによって、関係のない制御器やボタンを誤ってクリックするのを防ぎます。

一般的な用語を使用して、ボタンに特定の名前を割り当てます。OK ではなく開始、停止、保存などの名前を使用します。特定の名前を付けた方が、VI が使用しやすくなります。

デフォルトの LabVIEW フォントおよび色を使用します。LabVIEW は、組み込みフォントを、異なるプラットフォームの類似したフォントグループに置き換えます。選択した別のフォントがコンピュータで使用できない場合は、最も近いフォントに置き換えます。LabVIEW はフォントと同様に色も管理します。コンピュータで使用できない色は、最も近い色に置き換えます。また、システムカラーを使用して、フロントパネルの外観を VI を実行しているコンピュータのシステムカラーに合わせることもできます。

オブジェクト上に他のオブジェクトを配置しないでください。制御器や表示器の全体または一部を覆うようにラベルや他のオブジェクトを配置すると、画面の更新が遅くなり、制御器や表示器が点滅する場合があります。

ダイアログボックスを設計する

画面上の同じ位置に表示される連続したダイアログボックスが VI に含まれている場合は、最初のダイアログボックスのボタンが、次のダイアログボックスのボタンと同じ位置にならないようにダイアログボックスを構成します。最初のダイアログボックスでボタンをダブルクリックしたときに、知らずに次のダイアログボックスのボタンをクリックしてしまっていることがあります。ダイアログ制御器の詳細については、この章の「ダイアログ制御器」のセクションを参照してください。

画面サイズを選択する

VI の設計時には、画面解像度が異なるコンピュータでフロントパネルを表示できるかどうかを考慮に入れます。**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**ウィンドウサイズ**を選択します。**さまざまなモニタ解像度用のウィンドウの比率を維持**チェックボックスをオンにして、画面解像度に対するフロントパネルウィンドウ比率を維持します。

5

ブロックダイアグラムを作成する

フロントパネルを作成したら、グラフィカルに表現された関数を使用してコードを追加し、フロントパネルオブジェクトを制御します。ブロックダイアグラムには、このグラフィカルソースコードが含まれています。

詳細については

ブロックダイアグラムの設計と構成の詳細については、「LabVIEW ヘルプ」を参照してください。

フロントパネルオブジェクトとブロックダイアグラム端子の関係

フロントパネルオブジェクトは、ブロックダイアグラム上では端子として表示されます。ブロックダイアグラム端子をダブルクリックすると、フロントパネル上の対応する制御器または表示器がハイライトされます。

端子は、フロントパネルとブロックダイアグラムの間で情報を交換する入出力ポートです。フロントパネルの制御器に入力したデータは、制御器端子を介してブロックダイアグラムに入力されます。VI の実行が終了すると、出力データは表示器端子に移動します。出力データはブロックダイアグラムを出て、再度フロントパネルに入り、フロントパネル表示器に表示されます。

ブロックダイアグラムオブジェクト

ブロックダイアグラム上のオブジェクトには、端子、ノード、および関数が含まれています。ブロックダイアグラムは、ワイヤでオブジェクトを接続して作成します。

ブロックダイアグラム端子

ブロックダイアグラム端子は、フロントパネル制御器または表示器のデータタイプを表します。たとえば、左図に示す DBL 端子は、データタイプが倍精度浮動小数点数の制御器または表示器を表します。

端子とは、他のワイヤ以外に、ワイヤを接続できるすべてのポイントです。LabVIEW には、制御器および表示器端子、ノード端子、定数、およびフォーミュラノード上の入出力端子などのストラクチャ専用端子があります。ワイヤを使用して端子を接続し、データを他の端子に渡します。端子を表示するには、ブロックダイアグラムオブジェクトを右クリックし、ショートカットメニューから**項目を表示→端子**を選択します。端子を隠すには、オブジェクトを右クリックし、もう一度**項目を表示→端子**を選択します。このショートカットメニュー項目は、拡張可能 (ドラッグして端子の数を変更できる) な関数では使用できません。

制御器および表示器のデータタイプ

表 5-1 は、さまざまなタイプの制御器端子および表示器端子の記号を示しています。各端子の色と記号は制御器や表示器のデータタイプを示します。制御器端子の枠は表示器端子より太くなっています。また、端子が制御器である場合には右側に空の矢印が表示され、端子が表示器である場合には左側に矢印が表示されます。

表 5-1　制御器および表示器端子

制御器	表示器	データタイプ	色
SGL	SGL	単精度浮動小数点数	オレンジ
DBL	DBL	倍精度浮動小数点数	オレンジ
EXT	EXT	拡張精度浮動小数点数	オレンジ
CSG	CSG	単精度浮動小数点複素数	オレンジ
CDB	CDB	倍精度浮動小数点複素数	オレンジ
CXT	CXT	拡張精度浮動小数点複素数	オレンジ
I8	I8	符号付き 8 ビット整数	青
I16	I16	符号付き 16 ビット整数	青
I32	I32	符号付き 32 ビット整数	青
U8	U8	符号なし 8 ビット整数	青
U16	U16	符号なし 16 ビット整数	青
U32	U32	符号なし 32 ビット整数	青
◆	◆	列挙型	青
TF	TF	ブール	緑色
abc	abc	文字列	ピンク

表 5-1　制御器および表示器端子（続き）

制御器	表示器	データタイプ	色
		配列：要素のデータタイプを角括弧で囲み、そのデータタイプの色で表現されます。	データタイプによって異なる
 	 	クラスタ：複数のデータタイプを囲みます。クラスタデータタイプは、クラスタの要素のデータタイプが同じ場合は茶色、異なる場合はピンクです。	茶色またはピンク
		パス	水色
		波形：波形のデータ、開始時間および Δt（時間分解能）を含む要素のクラスタピクチャデータタイプの詳細については、第 11 章「グラフとチャート」の「波形データタイプ」のセクションを参照してください。	茶色
		リファレンス番号 (refnum)	水色
		バリアント：制御器または表示器の名前、データタイプ情報、およびデータそのものが含まれています。バリアントデータタイプの詳細については、この章の「バリアントデータを処理する」のセクションを参照してください。	紫
		多形性：VI または関数が複数のデータタイプを受け入れることを示します。多形性 VI および関数の詳細については、この章の「多形性 VI および関数」のセクションを参照してください。	紫
		I/O 名：構成する DAQ チャネル名、VISA リソース名、および IVI 論理名を I/O VI に渡し、計測器や DAQ デバイスと通信します。I/O 名データタイプの詳細については、第 4 章「フロントパネルを作成する」の「I/O 名制御器および表示器」のセクションを参照してください。	紫
		ピクチャ：線、円、テキストなどの画像形状を含むことができるピクチャを表示します。ピクチャデータタイプの詳細については、第 12 章「グラフィック & サウンド VI」の「ピクチャ表示器を使用する」のセクションを参照してください。	青

多くのデータタイプは、データを処理できる一連の関数に対応しています。各データタイプでどの関数を使用するかについては、この章の「関数の概要」のセクションを参照してください。

定数

定数は、ブロックダイアグラムに固定データ値を与えるブロックダイアグラム上の端子です。ユニバーサル定数は、pi (π) や無限 (∞) などの固定値を持つ定数です。ユーザ定義定数は、VI の実行前にユーザが定義して編集する定数です。

定数にラベルを付けるには、定数を右クリックし、ショートカットメニューから**項目を表示→ラベル**を選択します。ユニバーサル定数にはあらかじめ決まったラベルが付けられていますが、操作ツールまたはラベリングツールを使用して編集できます。

ほとんどの定数はパレットの一番下か一番上にあります。

ユニバーサル定数

ユニバーサル定数は、数値計算や、文字列またはパスのフォーマットに使用します。LabVIEW には以下のタイプのユニバーサル定数があります。

- **ユニバーサル数値定数**：自然数や光速など、よく使用される高精度の数学的および物理的な値。ユニバーサル数値定数は、**関数→数値→その他の数値定数**パレットにあります。
- **ユニバーサル文字列定数**：改行文字や復帰文字など、よく使用される非表示文字列。ユニバーサル文字列定数は、**関数→文字列**パレットにあります。
- **ユニバーサルファイル定数**：無効パス、Not a Refnum、デフォルトディレクトリなど、よく使用されるファイルパス値のセットです。ユニバーサルファイル定数は、**関数→ファイル I/O →ファイル定数**パレットにあります。

ユーザ定義定数

関数パレットには、ブール値、数値、リング、列挙型、カラーボックス、リストボックス、文字列、配列、クラスタ、パス定数などのタイプによって構成された定数があります。

VI または関数の入力端子を右クリックして、ショートカットメニューから**定数を作成**を選択して、ユーザ定義定数を作成します。VI の実行時には、ユーザ定義定数の値を変更できません。

また、定数は、フロントパネル制御器をブロックダイアグラムにドラッグして作成することもできます。作成された定数には、フロントパネル制御器をブロックダイアグラムにドラッグしたときの制御器の値が含まれています。フロントパネル制御器はフロントパネル上に残ります。制御器の値を変更しても、定数の値には影響しません。逆に、定数の値を変更しても制御器の値には影響がありません。

操作ツールかラベリングツールを使用して定数をクリックし、値を編集します。自動ツール選択機能が有効になっている場合には、定数をダブルクリックしてラベリングツールに切り替え、値を編集してください。

ブロックダイアグラムのノード

ノードは、入力端子や出力端子を持ち、VI の実行時に演算を実行するブロックダイアグラム上のオブジェクトです。テキストベースのプログラミング言語におけるステートメント、演算子、関数、およびサブルーチンに似ています。LabVIEW には以下のタイプのノードがあります。

- **関数**：組み込み実行要素。演算子、関数、またはステートメントに相当します。LabVIEW で使用可能な関数の詳細については、この章の「関数の概要」のセクションを参照してください。
- **サブ VI**：他の VI のブロックダイアグラムで使用される VI。サブルーチンに相当します。ブロックダイアグラム上のサブ VI の使用方法については、第 7 章「VI およびサブ VI を作成する」の「サブ VI」のセクションを参照してください。
- **ストラクチャ**：シーケンスストラクチャ、Case ストラクチャ、For ループ、While ループなどのプロセス制御要素。ストラクチャの使用方法については、第 8 章「ループと Case ストラクチャ」を参照してください。
- **フォーミュラノード**：ブロックダイアグラムに方程式を直接入力するためのサイズ変更可能なストラクチャ。フォーミュラノードの使用方法については、第 20 章「フォーミュラと方程式」の「フォーミュラノード」のセクションを参照してください。
- **プロパティノード**：クラスのプロパティを設定したり検索します。プロパティノードの使用方法については、第 16 章「プログラム的に VI を 制御する」の「プロパティノード」のセクションを参照してください。
- **インボークノード**：クラスのメソッドを実行します。インボークノードの使用方法については、第 16 章「プログラム的に VI を 制御する」の「数値関数」のセクションを参照してください。
- **コードインタフェースノード (CIN)**：テキストベースのプログラミング言語からコードを呼び出します。テキストベースのプログラミング言語からのコードの呼び出しの詳細については、第 19 章「テキストベースのプログラミング言語からのコード呼び出し」の「コードインタフェースノード」のセクションを参照してください。

VI のフロントパネルおよびブロックダイアグラムを作成した後、その VI をサブ VI として使用できるように、左図に示すようなコネクタペーンを作成します。コネクタペーンは、VI の制御器および表示器に対応する端子のセットで、テキストベースのプログラミング言語における関数呼び出しのパラメータリストに似ています。コネクタペーンは、VI に配線でき

る入力端子と出力端子を定義するため、サブ VI として使用することができます。コネクタペーンのセットアップの詳細については、第 7 章「VI およびサブ VI を作成する」の「コネクタペーンを設定する」のセクションを参照してください。

関数の概要

関数は LabVIEW の重要な操作要素です。**関数**パレットにある関数アイコンの背景色は淡い黄色で、前景色は黒です。関数にはフロントパネルやブロックダイアグラムはありませんが、コネクタペーンがあります。

また、**関数**パレットには LabVIEW に組み込みの VI も含まれています。これらの VI は、データ集録、計測器制御、通信、およびその他の VI を作成するときにサブ VI として使用します。組み込み VI の使用方法については、第 7 章「VI およびサブ VI を作成する」の「組み込み VI および関数を使用する」のセクションを参照してください。

数値関数

数値について算術演算、三角関数、対数関数、および複素演算を作成または実行したり、数値のデータタイプを変換するには、**関数→数値**パレットにある数値関数を使用します。

ブール関数

以下のようなタスクを実行するには、**関数→ブール**パレットにあるブール関数を使用して、1 つのブール値またはブール値の配列について論理演算を実行します。

- TRUE 値を FALSE 値に変更したり、FALSE 値を TRUE 値に変更する。
- 複数のブール値を受け取った場合に、どのブール値を返すかを決める。
- ブール値を 1 または 0 の数値に変換する。
- 複数のブール値で複合演算を実行する。

文字列関数

以下のタスクを実行するには、**関数→文字列**パレットにある文字列関数を使用します。

- 複数の文字列を連結します。
- 文字列から文字列のサブセットを抽出します。
- 文字列の文字またはサブセットを検索および置き換えます。

- データを文字列に変換します。
- ワープロや表計算アプリケーションで使用するために文字列をフォーマットします。

文字列関数の使用方法については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「文字列」のセクションを参照してください。

配列関数

以下のような配列を作成して操作するタスクでは、**関数→配列**パレットにある配列関数を使用します。

- 配列から個々のデータ要素を抽出する。
- 配列に個々のデータ要素を追加する。
- 配列を分割する。

配列関数の使用方法の詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「配列」のセクションを参照してください。

クラスタ関数

以下のようなクラスタを作成して操作するタスクでは、**関数→クラスタ**パレットにあるクラスタ関数を使用します。

- クラスタから個々のデータ要素を抽出する。
- クラスタに個々のデータ要素を追加する。
- クラスタを個々のデータ要素に分割する。

クラスタ関数の使用方法については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「クラスタ」のセクションを参照してください。

比較関数

ブール値、文字列、数値、配列、およびクラスタを比較するには、**関数→比較**パレットにある比較関数を使用します。

比較関数の使用方法については、付録 C「比較関数」を参照してください。

時間 & ダイアログ関数

以下のタスクを実行するには、**関数→時間 & ダイアログ**パレットにある時間 & ダイアログ関数を使用します。

- 演算の実行速度を操作する。
- コンピュータの時計から時刻と日付の情報を取り出す。
- ユーザに指示を与えるダイアログボックスを作成する。

このパレットにはエラー処理 VI も含まれています。エラー処理 VI の使用方法については、第 6 章「VI の実行とデバッグ」の「エラーチェックとエラー処理」のセクションを参照してください。

ファイル I/O 関数

以下のタスクを実行するには、**関数→ファイル I/O** パレットにあるファイル I/O 関数を使用します。

- ファイルを開いたり閉じる。
- ファイルから読み取ったり、ファイルに書き込む。
- パス制御器で指定するディレクトリとファイルを作成する。
- ディレクトリ情報を取り出す。
- 文字列、数値、配列、およびクラスタをファイルに書き込む。

また、**ファイル I/O** パレットには、一般的なファイル I/O タスクを実行する VI も含まれています。ファイル I/O VI および関数の使用方法の詳細については、第 13 章「ファイル I/O」を参照してください。

波形関数

以下のタスクを実行するには、**関数→波形**にある波形関数を使用します。

- 波形値、チャネル情報およびタイミング情報を含む波形を作成する。
- 波形から個々のデータ要素を抽出する。
- 波形の個々のデータ要素を編集する。

VI での波形の作成および使用方法については、『LabVIEW Measurements Manual』の Part II「DAQ Basics」を参照してください。

アプリケーション制御関数

VI および LabVIEW アプリケーションのプログラムによる制御をローカルコンピュータ上や、ネットワーク経由で行うには、**関数→アプリケーション制御**パレットにあるアプリケーション制御関数を使用します。アプリケーション制御関数の使用方法については、第 16 章「プログラム的に VI を 制御する」を参照してください。

上級関数

ダイナミックリンクライブラリ (DLL) などのライブラリからのコードの呼び出し、他のアプリケーションで使用するための LabVIEW データの操作、Windows レジストリキーの作成と操作、およびテキストベースのプログラミング言語から一部のコードの呼び出しを行うには、**関数→上級**パレットにある上級関数を使用します。上級関数の使用方法については、『Using External Code in LabVIEW』マニュアルを参照してください。

ブロックダイアグラム関数に端子を追加する

関数には端子の数を変更できるものもあります。たとえば、10 個の要素を持つ配列を作成するには、10 個の端子を追加する必要があります。

拡張可能な関数に端子を追加するには、位置決めツールを使用してその関数のコーナーをドラッグします。また、位置決めツールを使用して拡張可能な関数から端子を削除することもできますが、既に配線されている端子は削除できません。端子を削除するには、まず既存のワイヤを削除する必要があります。

また、アイコン上のいずれかの端子を右クリックし、ショートカットメニューから**入力端子を追加**、**出力端子を追加**、**入力端子を削除**、または**出力消去**を選択すると、端子を追加または削除できます。関数によっては、入力、出力、または refnum 制御器の端子を追加できます。**入力端子を追加**および**出力端子を追加**ショートカットメニュー項目を選択すると、右クリックした端子のすぐ後に新しい端子が追加されます。**入力端子を削除**および**出力消去**ショートカットメニュー項目を選択すると、右クリックした端子が削除されます。ショートカットメニュー項目を使用して、配線された端子を削除すると、端子が削除され、ワイヤが切断されます。

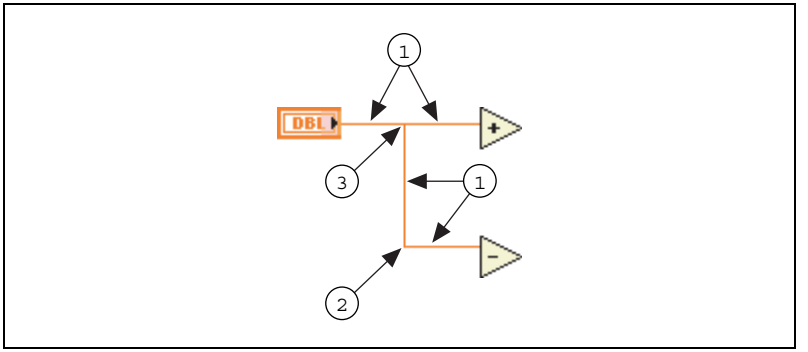
ブロックダイアグラムオブジェクトをワイヤで接続する

ブロックダイアグラムオブジェクト間のデータ転送はワイヤを介して行います。各ワイヤのデータソースは 1 つですが、そのデータを読み取る複数の VI および関数に接続できます。ワイヤの色、スタイル、および太さは、データタイプによって異なります。不良ワイヤは黒い破線で表示されます。ワイヤのデータタイプの詳細については、『LabVIEW クイックリファレンスカード』を参照してください。

ワイヤスタブとは、配線ツールを VI または関数ノード上で移動したときに未配線の端子の隣に現れる短いワイヤです。ワイヤスタブは各端子のデータタイプを示します。端子の名前を記述するヒントラベルも表示され

ます。端子を配線すると、ノード上を配線ツールが移動しても、その端子のワイヤスタブは表示されません。

ワイヤセグメントは水平方向または垂直方向の 1 本のワイヤです。ワイヤの屈折点は 2 つのセグメントの結合点です。3 本または 4 本のワイヤセグメントの結合点を接点といいます。ある接点から別の接点までのセグメント、ある端子から次の接点までのセグメント、端子間に接点がない場合は端子から端子までのセグメントはすべて、1 つのワイヤブランチに含まれます。図 5-1 は、ワイヤセグメント、屈折点、および接点を示しています。



1　セグメント　　2　屈折点　　3　接点

図 5-1　ワイヤセグメント、屈折点、および接点

最初に配線ツールを移動した方向によって、端子を水平方向または垂直方向に配線できます。端子のどこをクリックしても、ワイヤは端子の中心に接続されます。端子をクリックした後に、スペースバーを押して水平方向と垂直方向を切り替えます。

端子を配線する際にカーソルを水平方向または垂直方向に移動すると、一度だけワイヤを直角に曲げることができます。ワイヤを複数の方向に曲げるには、マウスボタンをクリックしてワイヤを固定した後、さらに新しい方向にドラッグします。ワイヤを固定して新しい方向にドラッグするという操作は繰り返し行うことができます。

固定したワイヤの最終点を取り消すには、<Shift> キーを押したままブロックダイアグラム上でクリックします。

(Macintosh) の場合は、<Option> キーを押したままクリックします。**(UNIX および Linux)** の場合は、中央マウスボタンをクリックします。

ワイヤを交差させると、最初に描いたワイヤに小さなすき間が表示されて、最初のワイヤが 2 本目のワイヤの下にあることを示します。

注意　ワイヤを交差させると、ブロックダイアグラムが煩雑になり、デバッグが難しくなる場合があります。

オブジェクトを自動配線する

自動配線機能を有効にしておくと、オブジェクトはブロックダイアグラム上に配置したとおりに自動的に配線されます。また、既にブロックダイアグラム上にあるオブジェクトを自動的に配線することもできます。LabVIEW では、最適な端子のみが接続され、不適切な端子は接続されません。

選択したオブジェクトをブロックダイアグラム上の別のオブジェクトの近くに移動すると、有効な接続を示すワイヤが一時的に描かれます。マウスボタンを離してブロックダイアグラム上にオブジェクトを配置すると、ワイヤが自動的に接続されます。

位置決めツールを使用してオブジェクトを移動する場合は、スペースバーを押すことによって自動配線機能を切り替えることができます。

関数パレットからオブジェクトを選択したり、既にブロックダイアグラム上にあるオブジェクトを <Ctrl> キーを押したままドラッグしてコピーすると、デフォルトで自動配線機能が有効になります。位置決めツールを使用して既にブロックダイアグラム上にあるオブジェクトを移動すると、デフォルトで自動配線は無効になります。

(Macintosh)<Option> キーを押します。**(Sun)**<Meta> キーを押します。**(Linux)**<Alt> キーを押します。

オブジェクトを手動で配線する

ブロックダイアグラムノードの端子を他のブロックダイアグラムノードの端子に手動で接続するには、配線ツールを使用します。ツールのカーソルポイントは、糸巻きから引き出されたワイヤの先端です。ノード上に配線ツールを移動すると端子が点滅し、VI および関数上ではその端子の名前がヒントラベルに表示されます。

ヘルプ→ヘルプを表示を選択して**ヘルプ**ウィンドウを表示します。このウィンドウでは、VI または関数の各端子のリストが表示されてどこにワイヤを接続する必要があるかを示します。**ヘルプ**ウィンドウには、Build Array 関数などの拡張可能な関数の端子は表示されません。

ワイヤを選択する

ワイヤを選択するには、位置決めツールを使用してワイヤを 1 回、2 回、または 3 回クリックします。ワイヤを 1 回クリックするとワイヤの 1 つ

のセグメントが選択されます。ダブルクリックするとブランチが選択されます。3 回クリックすると、ワイヤ全体が選択されます。

壊れたワイヤを削除する

不良ワイヤは黒い破線で表示されます。ワイヤは、データタイプに互換性がない 2 つのオブジェクトを配線しようとした場合など、さまざまな理由で壊れます。壊れたワイヤ上に配線ツールを移動すると、ワイヤが壊れた理由を示すヒントラベルが表示されます。壊れたワイヤを削除するには、位置決めツールでワイヤを 3 回クリックし、<Delete> キーを押します。

メモ　黒い破線のワイヤと緑の点線のワイヤを混同しないでください。緑の点線のワイヤはブール値データタイプを表します。

編集→不良ワイヤを削除を選択すると、壊れたワイヤをすべて削除できます。

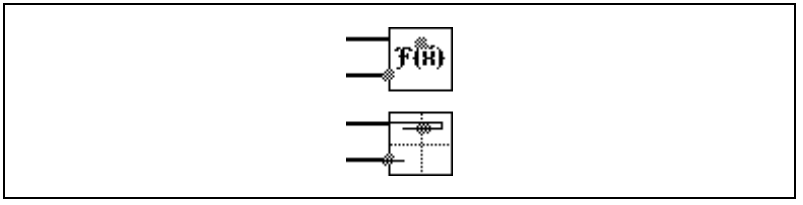
注意　壊れたワイヤをすべて削除するときは注意してください。ブロックダイアグラムの配線が完了していないためにワイヤが壊れているように見える場合があります。

強制ドット

異なる 2 つのデータタイプを配線すると、警告を示す強制ドットがブロックダイアグラムノードに表示されます。ドットは、ノードに渡された値が異なる表記法に変換されたことを示しています。たとえば、倍精度浮動小数点表記法の値 3.02 を持つ制御器を、整数表記法を持つ表示器に配線すると、表示器に強制ドットが表示され、表示器は 3 になります。

ブロックダイアグラムでは変換が発生した端子の枠に強制ドットが配置されて、自動数値変換が行われたことを示します。

VI および関数は多くの端子を持つことができるため、ある端子と別の端子を配線した場合、次の例に示すように強制ドットがアイコンの内部に表示されることがあります。



強制ドットによって VI でより多くのメモリが使用されるため、実行時間が長くなります。VI では、データタイプの統一を心掛けてください。

多形性 VI および関数

多形性 VI および関数は、異なるデータの入力に適応します。ほとんどの LabVIEW ストラクチャが多形性であるように、VI および関数の中にも多形性のものがあります。

多形性 VI

多形性 VI では、1 つの入力端子または出力端子で異なるデータタイプを受け入れます。多形性 VI は、同じコネクタペーンパターンを持つサブ VI の集合体です。各サブ VI は多形性 VI のインスタンスです。

たとえば、Read Key VI は多形性 VI です。その**デフォルト値**端子は、ブール値、倍精度浮動小数点数値、符号付き 32 ビット整数値、パス、文字列、または符号なし 32 ビット整数値のデータを受け入れます。

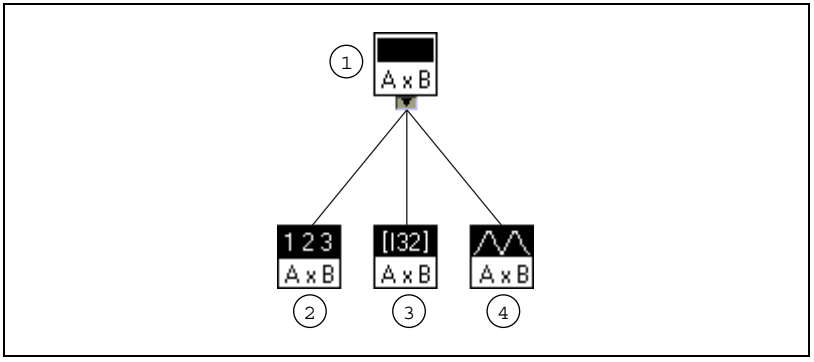
多形性 VI の入力に配線したデータタイプは使用するインスタンスを決定します。多形性 VI にデータタイプのサブ VI がない場合、壊れたワイヤが表示されます。多形性 VI を右クリックし、ショートカットメニューで**タイプを選択**を選択し、サブ VI を選択すれば、インスタンスはデフォルトインスタンスとして選択できます。

多形性 VI を作成する

異なるデータタイプで同じ操作を実行する場合は、独自の多形性 VI を作成します。多形性 VI を作成できるのは、LabVIEW プロフェッショナル開発システムのみです。

たとえば、単精度浮動小数点数、数値の配列、または波形で同じ数値演算を実行する場合は、数値演算 、配列演算、および波形演算の 3 つの独立した VI を作成できます。演算を実行する必要がある場合は、入力として使用するデータタイプに応じて、これらの VI のいずれかをブロックダイアグラムに配置します。

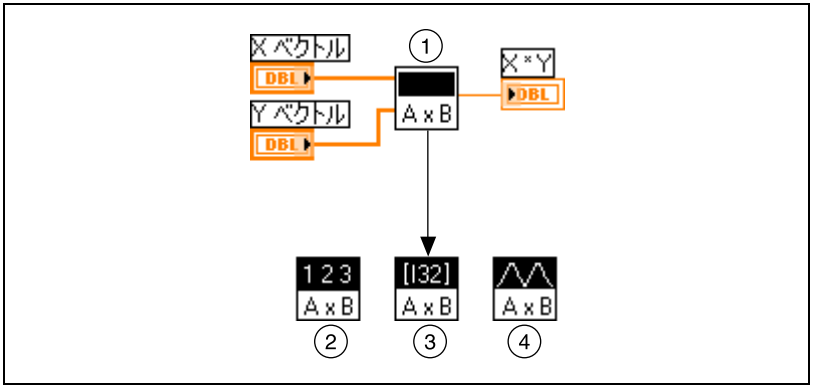
あるバージョンの VI をブロックダイアグラム上に手動で配置する代わりに、1 つの多形性 VI を作成して使用できます。図 5-2 のように、多形性の Compute VI には VI の 3 つのインスタンスが含まれています。



1　演算　　2　数値演算　　3　配列演算　　4　波形演算

図 5-2　多形性 VI の例

Compute VI は、図 5-3 のように、ブロックダイアグラム上で Compute サブ VI に配線したデータタイプに基づいて、VI の正しいインスタンスをスタティックにリンクします。



1　演算　　2　数値演算　　3　配列演算　　4　波形演算

図 5-3　サブ VI を呼び出す多形性 VI

多形性 VI は他のほとんどの VI と異なり、ブロックダイアグラムやフロントパネルがありません。

多形性 VI を作成する場合は以下の問題を考慮に入れてください。

- 多形性 VI に含めるすべての VI は、同じコネクタペーンパターンを持つ必要があります。多形性 VI のコネクタペーンとその多形性 VI の作成に使用する VI のコネクタペーンは一致している必要があるからです。

- VI の各インスタンスのコネクタペーン上にある入力端子と出力端子は、多形性 VI のコネクタペーン上にある入力端子と出力端子に対応する必要があります。
- 多形性 VI の作成に使用する VI は、同じサブ VI および関数から構成されている必要はありません。
- VI の各フロントパネルのオブジェクト数が同じである必要はありません。ただし、各フロントパネルは、多形性 VI のコネクタペーンを構成するのと同じ数以上の制御器および表示器を持つ必要があります。
- 多形性 VI のアイコンを作成できます。
- 多形性 VI を他の多形性 VI の中で使用することはできません。

多形性のサブ VI を含む VI の完全なドキュメントを作成する場合は、多形性 VI と、その多形性 VI によって呼び出される VI が、**サブ VI のリスト**に表示されます。

多形性関数

程度の差はありますが関数は多形性です。関数の入力がすべて多形性でなかったり、一部または全部の入力が多形性である場合があります。関数の入力には、数値またはブール値を受け入れるものがあります。数値や文字列を受け入れるものもあります。スカラ数値だけでなく、数値配列、数値クラスタ、数値クラスタの配列などを受け入れるものもあります。また、1 次元配列しか受け入れないものもありますが、配列要素はどのタイプでもかまいません。複素数値を含むすべてのタイプのデータを受け入れる関数もあります。多形性関数の詳細については、付録 B「多形性関数」を参照してください。

バリアントデータを処理する

バリアントデータは特定のデータタイプには適応せず、チャネル名やチャネル単位などの属性を持っています。LabVIEW はバリアントデータタイプによってバリアントデータを表現します。バリアントデータタイプは、制御器または表示器の名前、そのデータタイプの情報、およびデータそのものを含んでいるという点で他のデータタイプとは異なります。

バリアントデータの作成および操作を行うには、**関数→上級→データ操作→バリアント**パレットにあるバリアント関数を使用します。LabVIEW のすべてのデータタイプはバリアントデータタイプに変換可能なため、他の VI および関数でバリアントデータを使用できます。たとえば、文字列をバリアントデータに変換すると、そのバリアントデータタイプはテキストを格納し、そのテキストが文字列であることを示します。

データタイプとは無関係にデータを操作する必要がある場合は、バリアントデータを使用します。データを文字列に平坦化することによって、バリアントデータタイプを使用しなくてもタイプとは無関係にデータを表すことができます。たとえば、**Get All Control Values** メソッドは、VI からの制御器および表示器に関する情報をクラスタの配列として返します。配列内の各クラスタには、各制御器または表示器のデータタイプと値が含まれています。図 5-4 に示すように、クラスタには、制御器または表示器の名前が文字列で、データタイプが 17 ビット整数の配列で、またデータが平坦化された文字列の形で含まれています。

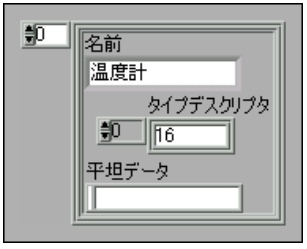


図 5-4　平坦化された数値データのクラスタ

しかし、LabVIEW はデータの平坦化を強制できないため、平坦化されたデータの使用には限界があります。また、平坦化された整数を拡張精度浮動小数点数として非平坦化しようとするとうまくいきません。バリアントデータタイプは、データを平坦化するのではなく、データタイプとは無関係にデータを操作するのに使用します。データの平坦化および非平坦化の詳細については、『LabVIEW Data Storage』アプリケーションノートの「Flattened Data」のセクションを参照してください。

属性を追加すると、バリアントデータをより細かく識別できます。たとえば、バリアントデータタイプの属性によって、データを取り込んだデータ集録デバイスチャネルを識別できます。

バリアントデータは、LabVIEW のメモリでの読み書き、スタック (LIFO：後入れ先出し法)、キュー (FIFO：先入れ先出し法)、スマートバッファ、またはツリーが関連する操作を実行するときにも便利です。これらの操作は、データタイプとは無関係にデータを処理します。

数値単位および厳密類別化チェック

浮動小数点表記法を持つ数値制御器には、メートルやキロメートル／秒などの物理的な測定単位を関連付けることができます。

制御器の単位は、単位ラベルと呼ばれる独立した所有ラベルに表示されます。単位ラベルを表示するには、制御器を右クリックし、ショートカットメニューから**項目を表示→単位ラベル**を選択します。

単位ラベルには、標準の略号を単位として入力できます。たとえば、メートルは `m`、フィートは `ft`、秒は `s` と入力できます。

メモ　フォーミュラノードでは単位を使用できません。

単位および厳密類別化チェック

オブジェクトに単位を関連付けた場合は、その単位と互換性のある単位を持つオブジェクトにのみ配線できます。LabVIEW では、厳密類別化チェックを使用して単位間に互換性があるかどうかを確認します。単位に互換性がない 2 つのオブジェクトを配線すると、エラーが返されます。たとえば、単位タイプがマイルのオブジェクトを、単位タイプがリットルのオブジェクトに配線するとエラーが返されます。これは、マイルが距離の単位、リットルが量の単位であるためです。

図 5-5 は、単位に互換性があるオブジェクトの配線を示しています。この図では表示器の単位がキロメートルであるため、LabVIEW は**距離**表示器のスケールを自動的に変更して、メートルではなくキロメートルを表示します。

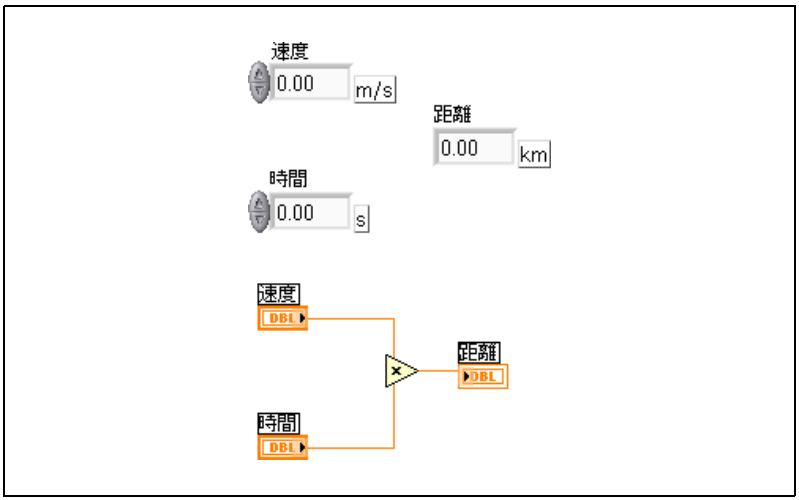


図 5-5　単位に互換性があるオブジェクトの配線

図 5-6 では、**距離**表示器の単位タイプが秒であるため、エラーが発生します。エラーを修正するには、図 5-5 のように、秒をキロメートルなどの距離の単位に変更します。

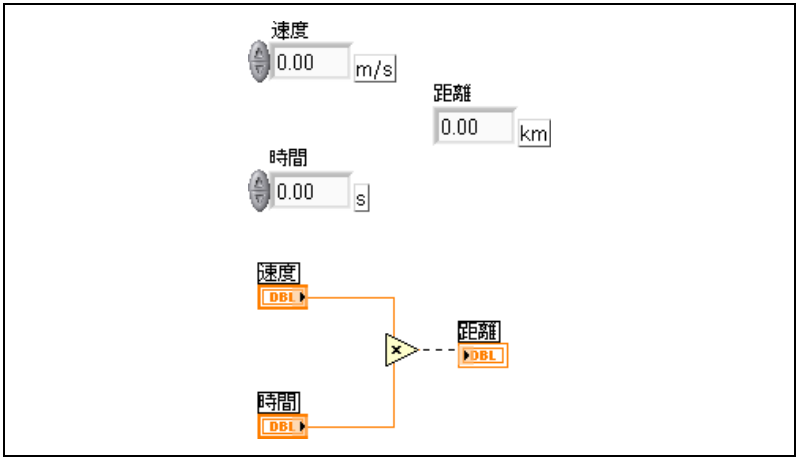


図 5-6　単位に互換性がないオブジェクトの配線により壊れたワイヤ

エラーを表示するには、壊れたワイヤ上に配線ツールを移動してヒントを表示させるか、壊れたワイヤを右クリックしてショートカットメニューから**エラーリスト**を選択します。**エラーリスト**ウィンドウには次のエラーが表示されます。

互換性がない単位を持つ数値データタイプを接続しました。

VI および関数には単位に対してあいまいなものもあります。これらの VI および関数は、単位を持つ他の端子では使用できません。たとえば、Increment 関数は単位に対してあいまいです。距離単位を使用する場合、Increment 関数は、1 メートル、1 キロメートル、あるいは 1 フィート加算するのか区別できません。このあいまいさのため、Increment 関数や単位に関連付けられたデータを持つ値を増減する他の関数は使用できません。

この例のようなあいまいさを避けるには、図 5-7 のように、適切な単位を持つ数値定数と Add 関数を使用して、独自の増分単位関数を作成します。

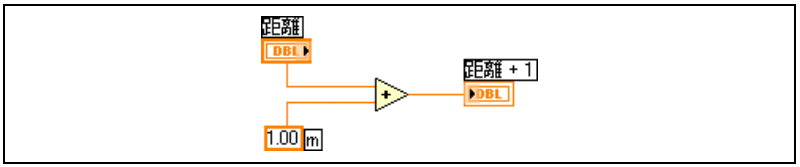


図 5-7　単位を持つ増分関数の作成

ブロックダイアグラムのデータフロー

LabVIEW はデータフローモデルに従って VI を実行します。ブロックダイアグラムのノードは、そのすべての入力が使用可能なときに実行されます。ノードは実行を完了すると、データをその出力端子に渡し、その出力データをデータフローパスの次のノードに渡します。

Visual Basic、C++、JAVA、その他のほとんどのテキストベースのプログラミング言語は、プログラム実行の制御フローモデルに従います。制御フローでは、プログラム要素の順序によってプログラムの実行順序が決まります。

LabVIEW では、コマンドの順序ではなく、データフローによってブロックダイアグラム内の要素の実行順序が決まるため、並列処理されるブロックダイアグラムを作成できます。LabVIEW はマルチタスクおよびマルチスレッドシステムであるため、複数の実行スレッドと複数の VI を同時に実行します。LabVIEW でタスクを同時に実行する方法の詳細については、『Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability』アプリケーションノートを参照してください。

データ依存と人工データ依存

実行の制御フローモデルは命令によって駆動されます。データフローの実行はデータによって駆動されます。つまり、データに依存しています。他のノードからデータを受け取るノードは常に、他のノードの実行が終了した後で実行されます。

ワイヤで接続されていないブロックダイアグラムのノードは、任意の順序で実行できます。『LabVIEW Development Guidelines』マニュアルでは、左から右、上から下にレイアウトすることを推奨していますが、ノードは必ずしも左から右、上から下に実行しません。

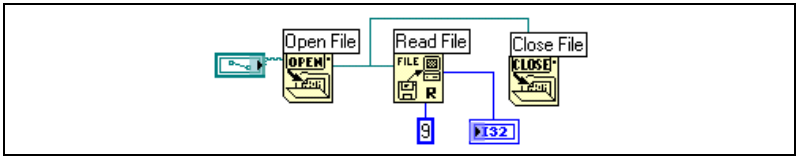
シーケンスストラクチャを使用して、自然なデータ依存が存在しない場合の実行順序を制御します。Case ストラクチャの詳細については、第 8 章「ループと Case ストラクチャ」の「シーケンスストラクチャ」のセクションを参照してください。フロースルーパラメータを使用して実行順序を制御することもできます。フロースルーパラメータの詳細については、第 13 章「ファイル I/O」の「フロースルーパラメータ」のセクションを参照してください。

受信ノードが実際に受信したデータを使用していない 人工データ依存も作成できます。その代わり、受信側ノードはデータの到着によってその実行をトリガします。人工データ依存の使用例については、`examples\general\structs.llb` 内の Timing Template (data dep) VI を参照してください。

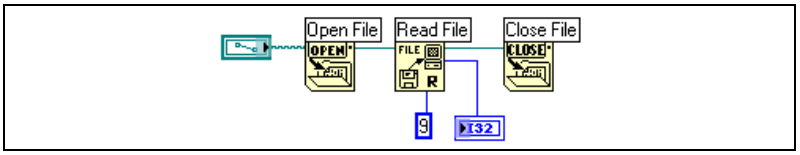
データ依存が存在しない場合

データ依存が存在しない場合は、左から右または上から下へ実行されるとは限りません。必要なときは必ず、データフローを配線することによってイベントのシーケンスを明示的に定義してください。

次の例では、Read File 関数が Close File 関数に配線されていないため、Read File 関数と Close File 関数との間にはデータ依存がありません。この例では、最初に実行される関数が決定できないため、期待どおりに機能しない場合があります。Close File 関数が最初に実行された場合、Read File 関数は機能しません。



次のブロックダイアグラムでは、Read File 関数の出力を Close File 関数に配線することによってデータ依存を確立します。Close File 関数は、Read File 関数の出力を受け取るまで実行されません。



データフローとメモリ管理

データフロー実行モデルでは、制御フロー実行モデルよりも簡単にメモリを管理できます。LabVIEW では、変数を割り当てたり変数に値を割り当てることはありません。その代わりに、データの移動を表すワイヤを使用してブロックダイアグラムを作成します。

データを生成する VI と関数がそのデータにメモリを自動的に割り当てます。VI や関数がそのデータを使用しなくなると、LabVIEW は関連付けられているメモリの割り当てを解除します。新しいデータを配列や文字列に追加すると、LabVIEW は新しいデータを管理するために十分なメモリを割り当てます。

LabVIEW ではメモリ管理が自動的に処理されるため、メモリの割り当てや割り当て解除の制御はほとんど必要ありません。ただし、VI が大量のデータセットを処理する場合は、メモリの割り当てがいつ行われるかを認識する必要があります。関連する原理を理解していれば、最小限のメモリ使用量で VI を作成できます。メモリ使用を最小限に抑えると、VI の実行速度が向上します。メモリ割り当ての詳細については、『LabVIEW Performance and Memory Management』アプリケーションノートを参照してください。

ブロックダイアグラムを設計する

ブロックダイアグラムを設計するには、次のガイドラインに従ってください。

- 左から右、上から下にレイアウトしてください。ブロックダイアグラム内の要素の位置によって実行順序が決まるわけではありませんが、ブロックダイアグラムをわかりやすく配列するために、右から左には配線しないでください。実行順序は、ワイヤとストラクチャのみによって決まります。
- 複数の画面を占めるブロックダイアグラムは作成しないでください。ブロックダイアグラムが大きく複雑になると、わかりにくく、デバッグが困難になる可能性があります。

- ブロックダイアグラム内のコンポーネントを他の VI で再利用できるかどうか、またはブロックダイアグラムの一部を論理的な要素としてまとめることができるかどうかを判断してください。それらが可能な場合は、ブロックダイアグラムを、特定のタスクを実行するサブ VI に分割してください。サブ VI を使用すると、変更の管理やブロックダイアグラムのデバッグを速やかに実行できます。サブ VI の詳細については、第 7 章「VI およびサブ VI を作成する」の「サブ VI」のセクションを参照してください。
- ブロックダイアグラム内のエラーを管理するには、エラー処理 VI、関数、およびパラメータを使用してください。エラー処理の詳細については、第 6 章「VI の実行とデバッグ」の「エラーチェックとエラー処理」のセクションを参照してください。
- ブロックダイアグラムの外観を改善するには、効率よく配線してください。ワイヤが効率よく配線されていなくても、エラーが発生することはないかもしれませんが、ブロックダイアグラムが読みにくく、デバッグが困難になり、VI が実際は実行していないことを実行しているように見える可能性があります。
- ストラクチャの枠の下や重なり合ったオブジェクトの間には配線しないでください。LabVIEW では、そのようなワイヤの一部のセグメントが表示されない場合があります。
- ワイヤの上にオブジェクトを配置しないでください。ワイヤはクリックされたオブジェクトだけに接続されます。ワイヤの上に端子やアイコンを配置すると、接続されていなくても接続されているかのように見えます。
- 作業スペースの色を追加したり変更するには、色付けツールを使用して ブロックダイアグラムの作業スペースを右クリックします。
- 密集したオブジェクト間の間隔を空けるには、<Ctrl> キーを押したまま、位置決めツールでブロックダイアグラムの作業スペースをクリックします。キーを押したまま、挿入するサイズだけ領域をドラッグアウトします。

  (Macintosh)<Option> キーを押します。**(Sun)**<Meta> キーを押します。**(Linux)**<Alt> キーを押します。

  破線の枠が付いた四角形により、スペースが挿入される場所が定義されます。キー操作を解除してスペースを追加します。

ユーザインタフェースの設計についての詳細は『LabVIEW Development Guidelines』を参照してください。

6

VI の実行とデバッグ

VI を実行するには、端子に対して適切なデータタイプを持つすべてのサブ VI、関数、およびストラクチャを配線する必要があります。VI は、予想外の方法でデータを生成したり、動作する場合があります。LabVIEW を使用すると、VI の実行方法を構成したり、ブロックダイアグラムの構成やブロックダイアグラムで受け渡されるデータに関する問題を確認できます。

詳細については

VI のデバッグの詳細については、「LabVIEW ヘルプ」を参照してください。

VI を実行する

VI を実行すると、VI を設計した際に目的とした操作が実行されます。左図のように、ツールバーの**実行**ボタンが白く塗りつぶされた矢印として表示されている場合は VI を実行できます。また、白く塗りつぶされた矢印は、VI のコネクタペーンを作成した場合にその VI をサブ VI として使用できることを示します。

ブロックダイアグラムツールバーの**実行**ボタン、**連続実行**ボタン、またはシングルステップボタンをクリックすると、VI が実行されます。**実行**ボタンをクリックすると、VI は一度実行されます。VI は、そのデータフローを完了すると停止します。**実行継続** ボタン をクリックすれば、手動で停止するまで VI を継続して実行できます。シングルステップボタンをクリックすると、VI は 1 ステップずつ実行されます。シングルステップボタンを使用して VI をデバッグする方法の詳細については、この章の「シングルステップ」のセクションを参照してください。

メモ **実行停止**ボタンで VI を停止することは避けてください。VI のデータフローを完了するか、プログラムで VI を停止する方法を設計してください。そうすることで VI は既知の状態になります。たとえば、クリックすると VI が停止するボタンをフロントパネルに配置します。

VI の実行方法を構成する

VI の実行方法を構成するには、**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**実行**を選択します。たとえば、VI を開くとすぐに実行するように構成したり、サブ VI として呼び出されると一時停止するように構成することができます。また、さまざまな優先順位で実行するように VI を構成できます。たとえば、他の操作が完了するのを待たずに VI を実行する必要がある場合、時間重視 (最高) の優先順位で実行するように VI を構成します。マルチスレッド VI の作成については、『Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability』アプリケーションノートを参照してください。VI の実行方法の構成については、第 15 章「VI をカスタマイズする」を参照してください。

壊れた VI を修正する

VI が実行されない場合、その VI は壊れています。つまり、それは実行不能な VI です。VI を作成または編集すると、左図のように、**実行**ボタンが壊れた状態で表示されることがよくあります。ブロックダイアグラムの配線の終了時にボタンがまだ壊れたままの場合、その VI は壊れているため実行されません。

壊れた VI の原因を調べる

VI が壊れている理由を調べるには、壊れた**実行**ボタンをクリックするか、**ウィンドウ→エラーリストを表示**を選択します。**エラーリスト**ウィンドウにすべてのエラーがリストされます。**VI リスト**セクションには、エラーが発生したメモリ内のすべての VI の名前がリストされます。**エラーと警告**セクションに、**VI リスト**セクションで選択した VI に関するエラーと警告がリストされます。**詳細**セクションにエラーが記述され、場合によってはエラーを修正したり、エラーの詳細を調べるための推奨方法が示されます。LabVIEW エラーとそれらの説明をリストするオンラインヘルプファイルを開くには、**ヘルプ**を選択します。

エラーを表示ボタンをクリックするか、またはエラーの説明をダブルクリックすると、関連するブロックダイアグラムまたはフロントパネルが表示され、エラーを含むオブジェクトがハイライトされます。

VI に警告が含まれていて、**エラーリスト**ウィンドウ内の**警告を表示**チェックボックスがオンになっている場合、ツールバーに左図のような**警告**ボタンが表示されます。

常に**エラーリスト**ウィンドウに警告を表示するように LabVIEW を構成するには、**ツール→オプション**を選択し、一番上のプルダウンメニューから**デバッグ**を選択して、**デフォルトとして警告をエラーボックス内に表示す**

るチェックボックスをオンにします。**エラーリスト**ウィンドウを開いたままこの変更を行うと、変更をすぐに表示できます。

警告が表示されても、VI の実行は妨げられません。これらの警告は、VI 内の潜在的な問題を避ける目的で設けられています。

壊れた VI の一般的な原因

VI の編集時に VI が壊れる一般的な理由を以下にリストします。

- ブロックダイアグラムに、データタイプの不一致や未接続配線による壊れたワイヤが含まれている。ブロックダイアグラムオブジェクトの配線方法の詳細については、第 5 章「ブロックダイアグラムを作成する」の「ブロックダイアグラムオブジェクトをワイヤで 接続する」のセクションを参照してください。
- 必要なブロックダイアグラムの端子が配線されていない。ブロックダイアグラムノードに必要なパラメータを確認するには、**ヘルプ→ヘルプを表示**を選択するか、「LabVIEW ヘルプ」を参照してください。
- サブ VI が壊れているか、あるいは VI のブロックダイアグラム上にそのアイコンを配置した後でそのコネクタペーンを編集した。サブ VI の詳細については、第 7 章「VI およびサブ VI を作成する」の「サブ VI」のセクションを参照してください。

デバッグ方法

VI は壊れていないのに、予想外のデータが得られた場合、以下の方法で VI またはブロックダイアグラムのデータフローに関する問題を確認し、修正できます。

- ほとんどの組み込み VI と関数の一番下にある**エラー入力**および**エラー出力**パラメータを配線します。これらのパラメータはブロックダイアグラム上の各ノード内で発生したエラーを検出し、エラーが発生したかどうかとその発生場所を示します。作成した VI 内でこれらのパラメータを使用することもできます。これらのパラメータの使用方法の詳細については、この章の「エラー処理」のセクションを参照してください。
- ブロックダイアグラム内でのデータの移動を監視するには、実行のハイライトを使用します。
- ブロックダイアグラム上で VI の各動作を表示するには、VI をシングルステップします。
- VI の実行時に中間値をチェックするには、プローブツールを使用します。
- シングルステップを実行したり、プローブを挿入するために実行を一時停止するには、ブレークポイントを使用します。

- 制御器や表示器の値を編集したり、実行回数を制御したり、サブ VI の実行の開始時点に戻るには、サブ VI の実行を中断します。
- あるセクションがない場合に VI のパフォーマンスが向上するかどうかを調べるには、ブロックダイアグラムのそのセクションをコメントとして除外します。

実行のハイライト

ブロックダイアグラムの実行を動画表示するには、左図に示す**実行のハイライト**ボタンをクリックします。実行のハイライトは、ワイヤに沿って移動するバブルを使用して、ブロックダイアグラム上のノードからノードへのデータ移動を示します。VI 全体でノードからノードにデータがどのように移動するかを調べるには、シングルステップとともに実行のハイライトを使用します。

メモ　実行のハイライトを使用すると、VI の実行速度が大幅に低下します。

エラーアウトクラスタがエラーをレポートすると、エラー値が赤で **エラーアウト**の横に表示されます。エラーが起こらない場合は、**OK** が緑で**エラーアウト**の横に表示されます。エラークラスタの詳細については、この章の「エラークラスタ」のセクションを参照してください。

シングルステップ

VI の実行時にブロックダイアグラム上の VI の各動作を表示するには、VI をシングルステップで実行します。左図に示すシングルステップボタンが機能するのは、シングルステップモードでの VI またはサブ VI 内の実行に限られます。シングルステップモードに切り替えるには、ブロックダイアグラムのツールバーにある**飛び越える**または**中に入る**ボタンをクリックします。そのボタンをクリックした場合の次の手順を説明するヒントラベルを表示するには、**飛び越える**、**中に入る**、または**外に出る**ボタンの上にカーソルを移動します。サブ VI は、シングルステップで実行するか通常どおりに実行できます。

実行のハイライトをオンにした状態で VI をシングルステップで実行する場合は、左図のような実行グリフが現在実行中のサブ VI のアイコン上に表示されます。

プローブツール

VI の実行時にワイヤ上の中間値をチェックするには、左図に示すプローブツールを使用します。一連の操作が含まれている複雑なブロックダイアグラムを作成し、そのいずれかの操作が間違ったデータを返す可能性がある場合にプローブツールを使用します。実行のハイライト、シングルステップ、およびブレークポイントとともにプローブツールを使用して、

データに誤りがないか、およびその場所を確認します。データが利用可能な場合、シングルステップの実行時またはブレークポイントで一時停止したときにプローブはすぐに更新されます。シングルステップまたはブレークポイントのために実行がノードで一時停止している場合は、実行直後のワイヤにプローブを挿入して、そのワイヤを通った値を確認することもできます。

また、カスタムプローブを作成して、プローブを挿入したデータを表示するための表示器を指定できます。たとえば、数値データを表示する場合は、プローブ内のチャートにそのデータを表示するように選択できます。ワイヤを右クリックして**カスタムプローブ**を選択し、使用する表示器を指定します。

ブレークポイント

ブロックダイアグラム上の VI、ノード、またはワイヤ上にブレークポイントを配置し、その位置で実行を一時停止するには、左図に示すブレークポイントツールを使用します。ワイヤ上にブレークポイントを設定すると、データがワイヤを通った後で実行が一時停止します。ブロックダイアグラム上のすべてのノードの実行後に一時停止するようにするには、ブロックダイアグラムの作業スペースにブレークポイントを配置します。

VI がブレークポイントで一時停止すると、LabVIEW はブロックダイアグラムを表示し、マーキーを使用して、ブレークポイントを含むノードかワイヤをハイライトします。ブレークポイントの上でカーソルを移動すると、ブレークポイントツールのカーソルの黒色部分が白色で表示されます。

実行中にブレークポイントに到達したときは、以下の処置をとることができます。

- シングルステップボタンを使用して実行をシングルステップする。
- ワイヤにプローブを置いて中間値をチェックする。
- 次のブレークポイントまたは VI が実行を終了するまで実行を続ける。

LabVIEW は VI とともにブレークポイントを保存しますが、VI を実行する場合にのみブレークポイントはアクティブになります。

実行を中断する

制御器や表示器の値を編集したり、呼び出し側 VI に返すまでの実行回数を制御したり、サブ VI の実行開始時点に戻るには、サブ VI の実行を中断します。実行を中断した状態でサブ VI へのすべての呼び出しを開始したり、サブ VI への特定の呼び出しを中断することができます。

サブ VI へのすべての呼び出しを中断するには、サブ VI を開き、**操作→呼び出されたら中断する**を選択します。他の VI がそのサブ VI を呼び出す

と、そのサブ VI は自動的に中断されます。シングルステップの実行中にこのメニュー項目を選択した場合は、サブ VI はすぐには中断されません。サブ VI は呼び出されたときに中断されます。

特定のサブ VI の呼び出しを中断するには、ブロックダイアグラム上のサブ VI ノードを右クリックし、ショートカットメニューから**サブ VI ノード設定**を選択します。サブ VI のそのインスタンスだけで実行を中断するには、**呼び出されたら中断する**チェックボックスをオンにします。

参照→ VI 階層を表示を選択することによって表示される**階層ウィンドウ**は、VI が一時停止または中断されているかどうかを示します。矢印グリフは、通常どおりに実行している VI またはシングルステップで実行している VI を示します。一時停止グリフは、一時停止または中断されている VI を示します。緑の一時停止グリフ、または白黒の場合中抜きのグリフは、呼び出されたら一時停止する VI を示します。赤の一時停止グリフ、または白黒の場合塗りつぶされたグリフは、現在一時停止している VI を示します。感嘆符グリフはサブ VI が中断されていることを示します。VI を同時に中断または一時停止できます。

サブ VI の現在のインスタンスを調べる

サブ VI を一時停止すると、ツールバーの**呼び出しリスト**プルダウンメニューが、最上位 VI からサブ VI に至る呼び出し側 VI のチェーンをリストします。このリストは、**参照→この VI の発呼者**を選択すると表示されるリスト（VI が現在実行中かどうかにかかわらず、すべての呼び出し側 VI をリストで表示）とは異なります。ブロックダイアグラムに複数のインスタンスが含まれている場合にサブ VI の現在のインスタンスを調べるには、**呼び出しリスト**メニューを使用します。**呼び出しリスト**メニューから VI を選択すると、そのブロックダイアグラムが開き、LabVIEW は現在のサブ VI のインスタンスがハイライトします。

ブロックダイアグラムのセクションをコメントとして除外する

ブロックダイアグラムのセクションを無効にした状態で、VI を実行できます。これは、テキストベースのプログラミング言語でコードのセクションをコメントとして除外することに似ています。あるセクションがない場合に VI のパフォーマンスが向上するかどうかを調べるには、ブロックダイアグラムのそのセクションを無効にします。

無効にするセクションを Case ストラクチャ内に入れ、ブール定数を使用して両方のケースを実行します。Case ストラクチャの使用方法の詳細については、第 8 章「ループと Case ストラクチャ」の「Case ストラクチャ」のセクションを参照してください。VI のコピーを作成し、コピーからブロックダイアグラムのそのセクションを削除することもできます。使用しない方の VI は破棄します。

デバッグツールを無効にする

デバッグツールを無効にすると、メモリの必要条件が低減されるためパフォーマンスがわずかに向上します。コネクタペーンを右クリックし、**VI プロパティ**を選択します。**カテゴリ**プルダウンメニューの**実行**を選択し、**デバッギングを許可**チェックボックスをオフにします。

不定データまたは予想外のデータ

不定データ、すなわち NaN(数値以外) または Inf(無限大) は、後続のすべての操作を無効にします。浮動小数点演算は、誤った計算や意味のない結果を示す以下の 2 つの記号値を返します。

- NaN(数値以外) は、負の数値の平方根を求めるような無効な演算が生成する浮動小数点値を表します。
- Inf(無限大) は、0 で数値を除算するといった演算が生成する浮動小数点値を表します。

LabVIEW は、整数値がオーバーフローまたはアンダーフローの状態になっているかどうかをチェックしません。浮動小数点値のオーバーフローまたはアンダーフローは、IEEE 754 の「Standard for Binary Floating-Point Arithmetic」を参照してください。

浮動小数点演算は、NaN および Inf を忠実に伝達します。NaN または Inf を明示的または暗示的に整数またはブール値に変換すると、値は意味のない値になります。たとえば、1 を 0 で除算すると Inf になります。Inf を 16 ビット整数に変換すると、値は 32,767 となり、通常の値のように見えます。数値の変換方法の詳細については、付録 B「多形性関数」の「数値変換」のセクションを参照してください。

データを整数データタイプに変換する前に、中間の浮動小数点値の有効性をチェックするには、プローブツールを使用します。比較関数 Not A Number/Path/Refnum? を、有効性が疑わしい値に配線することによって、NaN がないかをチェックします。

ループ内の予想外のデータとデフォルトデータ

自動指標付けされた For ループの繰り返し回数が 0 のとき、For ループは予想外の値を生成します。シフトレジスタが初期化されていないとき、While ループはデフォルトデータを生成します。

For ループ

For ループのカウント端子に 0 を配線した場合や、自動指標付けが有効になった状態で空の配列を入力として For ループに配線した場合、For ループは予想外の値を生成します。ループは実行せず、自動指標付けが無効になった状態の出力トンネルにのみ予想外の値が含まれます。ループが実行するかどうかに関わらずループから値を転送するには、シフトレジスタを使用してください

For ループ、自動指標付け、およびシフトレジスタの詳細については、第 8 章「ループと Case ストラクチャ」の「For ループおよび While ループのストラクチャ」のセクションを参照してください。

While ループ

While ループ上でシフトレジスタを初期化しない場合、出力はパラメータのデフォルト値 (0、FALSE、空の文字列など) か、あるいは最後に VI を実行したときにシフトレジスタに最後にロードされた値になります。

配列内のデフォルトデータ

配列の範囲を超えて指標付けを行うと、配列要素パラメータのデフォルト値が生成されます。配列のサイズを調べるには、Array Size 関数を使用します。配列の詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「配列」のセクションを参照してください。指標付けの詳細については、第 8 章「ループと Case ストラクチャ」の「ループに自動指標付けを行う」のセクションを参照してください。While ループを使用して最後の要素を超えて配列を指標付けしたり、Index Array 関数の**指標**入力に大き過ぎる値を入力したり、Index Array 関数に空の配列を指定すると、誤って配列の範囲を超えて指標付けする可能性があります。

不定データを防ぐ

VI が不定データを生成するかどうかを調べるのに、NaN、Inf、空の配列などの特別な値ばかりを使用しないでください。その代わり、不定データを生成しそうな状況に直面したときに VI にエラーを報告させることによって、VI が確定データを生成することを確認します。

たとえば、入力配列を使用して For ループを自動指標付けする VI を作成する場合は、入力配列が空のときに VI がどのように動作するかを決めます。出力エラーコードを生成するか、あるいはループが生成するデータを、確定データで置き換えます。

エラーチェックとエラー処理

LabVIEW で自動的に有効になるデバッグ機能により、ブロックダイアグラムのノードにエラーがないかどうかをチェックします。デバッグ設定を確認するには、**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニュー内の**実行**を選択します。**デバッギングを許可**チェックボックスはデフォルトでオンになっています。各エラーには数値によるコードとそれに対応するエラーメッセージがあります。エラーを管理するには、LabVIEW エラー処理 VI、関数、およびパラメータを使用します。たとえば、LabVIEW がエラーを検出した場合は、ダイアログボックスにエラーメッセージを表示できます。デバッグツールとともにエラー処理機能を使用して、エラーを検出し、管理します。ナショナルインスツルメンツでは、エラー処理を行うことを強くお勧めします。

エラーをチェックする

作成した VI に自信があっても、ユーザが直面するすべての問題は予測できません。エラーをチェックするメカニズムがないと、VI が正しく機能しないということしかわかりません。エラーチェック機能があると、エラーが発生した理由と場所がわかります。

どのような入出力 (I/O) 動作を行う場合も、エラーが発生する可能性を考慮する必要があります。ほとんどすべての I/O 関数はエラー情報を返します。特に I/O 動作 (ファイル、シリアル、計測器、データ集録、および通信) に関して VI 内にエラーチェック機能を組み込み、エラーを正しく処理するメカニズムを装備してください。

VI 内のエラーをチェックすると、以下の問題を確認できます。

- 通信を間違えて初期化したか、あるいは外部機器に不適切なデータを書き込んだ場合。
- 外部機器の電源が入っていないか、故障しているか、あるいは正しく機能していない場合。
- オペレーティングシステムソフトウェアをアップグレードした際に、ファイルへのパスや、VI やライブラリの機能が変更された場合。VI またはシステムプログラムの問題に気付くはずです。

エラー処理

LabVIEW はエラーを自動的に処理しません。使用するエラー処理方法をユーザが選択します。たとえば、ブロックダイアグラム内の I/O VI がタイムアウトになっても、アプリケーション全体を停止させたくない場合があります。また、一定時間、再実行したい場合もあります。LabVIEW では、このようなエラー処理の判断をブロックダイアグラム内で行えます。

VI と関数は、数値によるエラーコードかエラークラスタのいずれかの方法でエラーを返します。関数は通常数値によるエラーコードを使用し、VI は通常エラー入出力とともにエラークラスタを使用します。エラークラスタの詳細については、この章の「エラークラスタ」のセクションを参照してください。

LabVIEW 内のエラー処理はデータフローモデルに従います。データが VI 内を移動する場合と同様に、エラー情報も VI 全体を移動できます。エラー情報を VI の最初から最後に配線してください。エラーが発生せずに VI が実行されたかどうかを調べるには、VI の最後にエラー処理 VI を組み込みます。使用または作成する各 VI 内で**エラー入力**と**エラー出力**クラスタを使用して、VI 内でエラー情報を渡します。

VI の実行時、LabVIEW は各実行ノードでエラーがないかをテストします。エラーが検出されなかった場合、ノードは正常に実行されます。エラーが検出された場合、ノードは実行されずにエラーを次のノードに渡します。次のノードは同じ動作を繰り返します。実行フローの最後にエラーが報告されます。

エラークラスタ

制御器→配列とクラスタパレット上にあるエラークラスタには以下の情報コンポーネントが含まれています。

- **ステータス**はブール値であり、エラーが発生した場合に TRUE を報告します。ほとんどの VI、関数とブールを受け取る構造は、このパラメータを認識します。たとえば、エラークラスタを停止（Stop）、Quit LabVIEW、Select 関数のブール入力に配線することができます。エラーが発生した場合、エラークラスタは関数に TRUE の値を渡します。
- **コード**は符号付き 32 ビット整数で、番号でエラーを識別します。**ステータス** FALSE を伴う 0 以外のエラーコードは、致命的なエラーでなく警告であることを知らせます。
- **ソース**はエラーが発生した場所を識別する文字列です。

エラー処理に While ループを使用する

エラークラスタを While ループの条件端子に配線すると、While ループの繰り返しを停止できます。エラークラスタを条件端子に配線すると、エラークラスタの**ステータス**パラメータの TRUE または FALSE 値だけが端子に渡されます。エラーが発生すると、While ループは停止します。

エラークラスタが条件端子に配線されているとき、ショートカットメニュー項目 **True の場合停止**と **True の場合、継続**は**エラーで停止**と**エラーの場合、継続**に変わります。

エラー処理に Case ストラクチャを使用する

Case ストラクチャのセレクタ端子にエラークラスタを配線すると、ケースセレクタラベルには 2 つのケース、すなわちエラーとエラーなしが表示されます。Case ストラクチャの枠の色は、エラーの場合は赤に変わり、エラーなしの場合は緑に変わります。エラーが発生すると、Case ストラクチャはエラーサブダイアグラムを実行します。Case ストラクチャの使用については、第 8 章「ループと Case ストラクチャ」の「Case ストラクチャ」のセクションを参照してください。

7

VI およびサブ VI を作成する

フロントパネルとブロックダイアグラムの作成方法を習得すると、独自の VI やサブ VI を作成したり、VI を配布したり、スタンドアロンアプリケーションおよび共有ライブラリを作成できます。

開発時の一般的な落とし穴やプロジェクトの開発に使用できるツールの情報など、プロジェクトの計画の詳細については、『LabVIEW Development Guidelines』マニュアルを参照してください。

詳細については

サブ VI の作成および使用方法、VI の保存方法、スタンドアロンアプリケーションおよび共有ライブラリの作成方法の詳細については、「LabVIEW ヘルプ」を参照してください。

プロジェクトの計画と設計

独自の VI を開発する前に、ユーザが実行する必要があるタスクのリストを作成します。ユーザインタフェースのコンポーネントと、解析結果などを表示する、データ解析に必要な制御器と表示器の数およびタイプを決めます。アプリケーションを使用するユーザやプロジェクトチームの他のメンバとともに、ユーザが関数や機能にアクセスする必要が生じる状況とその方法を考察し検討します。アプリケーションを使用するユーザやプロジェクトチームのメンバに提示するためのサンプルフロントパネルを作成し、ユーザがそのフロントパネルを使用してタスクを実行できるかどうかを検討します。このような対話式のプロセスを経て、必要に応じユーザインタフェースを改良していきます。

アプリケーションを適切な位置で、管理可能な断片に分割します。まず、アプリケーションのメインコンポーネントが含まれている上位ブロックダイアグラムから分割します。たとえば、ブロックダイアグラムには、構成のためのブロック、データ集録のためのブロック、集録データを解析するためのブロック、解析結果を表示するためのブロック、データをディスクに保存するためのブロック、エラーを処理するためのブロックなどが含まれます。

上位ブロックダイアグラムを設計したら、入出力を定義します。次に、上位ブロックダイアグラムのメインコンポーネントを構成するサブ VI を設計します。サブ VI を使用すると、上位ブロックダイアグラムの読み取り、デバッグ、理解、および管理が容易になります。また、一般的な操作または頻繁に行われる操作のために再利用可能なサブ VI を作成できます。サブ VI は、作成しながらテストを行います。それより上位のテストルーチンを作成できますが、小さいモジュールでエラーを検出する方が複数の VI の階層をテストするよりも簡単です。上位ブロックダイアグラムの初期設計の不完全な部分が見つかる可能性もあります。下位のタスクを実行するためにサブ VI を使用すると、アプリケーションの変更や再構成も簡単に行えます。サブ VI の詳細については、この章の「サブ VI」のセクションを参照してください。

ブロックダイアグラムとサブ VI の例については、**ヘルプ→サンプルの検索**を選択してください。

複数の開発者とともにプロジェクトを設計する

複数の開発者が同じプロジェクトに取り組む場合は、開発プロセスとアプリケーションが確実に機能するように、プログラミングの責任範囲、インタフェース、およびコーディング基準をあらかじめ定義しておきます。

1 台のコンピュータにプロジェクト VI のマスタコピーを保存し、ソースコードの管理方針を定めます。ファイルの共有を容易にするソースコード管理ツールが装備されている、LabVIEW プロフェッショナル開発システムの使用を検討してください。このツールには、VI を比較し、VI のバージョン間で行われた変更を表示するユーティリティも用意されています。ソースコード管理ツールの使用方法の詳細については、『LabVIEW Development Guidelines』マニュアルの Chapter 2「Incorporating Quality into the Development Process」の「Source Code Control」のセクションを参照してください。

組み込み VI および関数を使用する

LabVIEW には、データ集録 VI および関数、他の VI にアクセスする VI、他のアプリケーションと通信する VI など、特定のアプリケーションの作成に役立つ VI および関数が用意されています。これらの VI をアプリケーション内でサブ VI として使用すると、開発時間を短縮できます。サブ VI の詳細については、この章の「サブ VI」のセクションを参照してください。

計測器制御およびデータ集録 VI および関数を作成する

組み込み VI および関数は、オスシロスコープなどの外部計測器を制御したり、熱電対からの読み取りなどのデータ集録に使用できます。

外部計測器を制御するには、**関数→計測器 I/O** パレット上にある計測器 I/O VI および関数を使用します。LabVIEW で計測器を制御するには、ハードウェアを正しくインストールし、電源をオンにして、コンピュータ上で動作させる必要があります。計測器の制御に使用する VI および関数は、ご使用のハードウェアがサポートする計測器の通信プロトコルによって異なります。計測器を制御する VI の作成方法の詳細については、『LabVIEW Measurements Manual』を参照してください。

DAQ デバイスからデータを集録するには、**関数→データ集録**パレットにあるデータ集録 VI および関数を使用します。これらの VI を使用するには、NI-DAQ ドライバソフトウェアと DAQ ハードウェアをインストールする必要があります。NI-DAQ ドライバおよび DAQ ハードウェアのインストール方法とデータを集録する VI の作成方法の詳細については、『LabVIEW Measurements Manual』を参照してください。データを集録したら、組み込み解析 VI、レポート生成 VI、および数学 VI および関数を使用して、そのデータに対する解析、レポート生成、および数学演算を実行できます。LabVIEW で使用する解析や数学的概念の詳細については、『Analysis Concepts』マニュアルを参照してください。

他の VI にアクセスする VI を作成する

サブ VI として呼び出された VI やユーザによって実行された VI の動作を制御するには、**関数→アプリケーション制御**パレットにあるアプリケーション制御 VI および関数を使用します。これらの VI および関数を使用すると、複数の VI を同時に構成できます。また、他の LabVIEW ユーザとネットワーク接続されている場合は、これらの VI と関数を使用して VI にリモートでアクセスして制御できます。VI をリモートで制御する方法の詳細については、第 16 章「プログラム的に VI を 制御する」を参照してください。

他のアプリケーションと通信する VI を作成する

Microsoft Excel などの他のアプリケーションとの間でデータを読み書きするには、**関数→ファイル I/O** パレットにあるファイル I/O VI および関数を使用します。これらの VI および関数を使用すると、レポートを生成したり、他のアプリケーションからのデータを VI に取り込むことができます。他のアプリケーション間でデータをやり取りする方法の詳細については、第 13 章「ファイル I/O」を参照してください。

FTP などの通信プロトコルを使用して LabVIEW データをウェブ上に転送したり、通信プロトコルを使用してクライアントサーバアプリケーション

を作成するには、**関数→通信**パレットにある通信 VI および関数を使用します。ネットワークまたはウェブ上で他のアプリケーションと通信する方法の詳細については、第 17 章「LabVIEW のネットワーク動作」を参照してください。

(Windows) ActiveX オブジェクトを VI に追加したり、ActiveX 対応アプリケーションを制御するには、**関数→通信→ ActiveX** パレットにある ActiveX 関数を使用します。ActiveX テクノロジの使用方法の詳細については、第 18 章「ActiveX」を参照してください。

サブ VI

VI を作成し、そのアイコンとコネクタペーンを作成したら、その VI を他の VI 内で使用できます。他の VI のブロックダイアグラムから呼び出された VI をサブ VI といいます。サブ VI は、テキストベースのプログラミング言語におけるサブルーチンに相当します。サブ VI のノードは、テキストベースのプログラミング言語におけるサブルーチンコールに相当します。プログラム内のサブルーチンコールステートメントがサブルーチンそのものではないのと同様に、ノードはサブ VI そのものではありません。同じサブ VI ノードが複数含まれているブロックダイアグラムでは、同じサブ VI が数回呼び出されます。

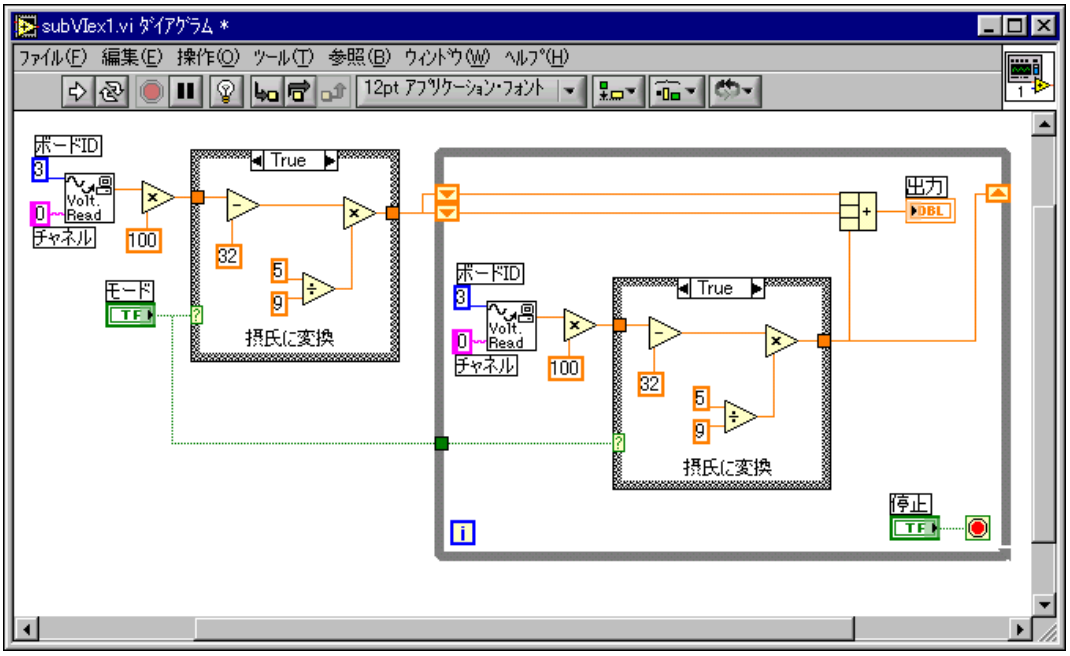
サブ VI の制御器と表示器は、呼び出し側 VI のブロックダイアグラムとの間でデータをやり取りします。その VI へのサブ VI 呼び出しを作成するには、**関数→ VI を選択**パレットにある VI を選択して、ブロックダイアグラム上に配置します。

メモ　VI をサブ VI として使用するには、その前にコネクタペーンを設定する必要があります。コネクタペーンの設定方法の詳細については、この章の「コネクタペーンを設定する」のセクションを参照してください。

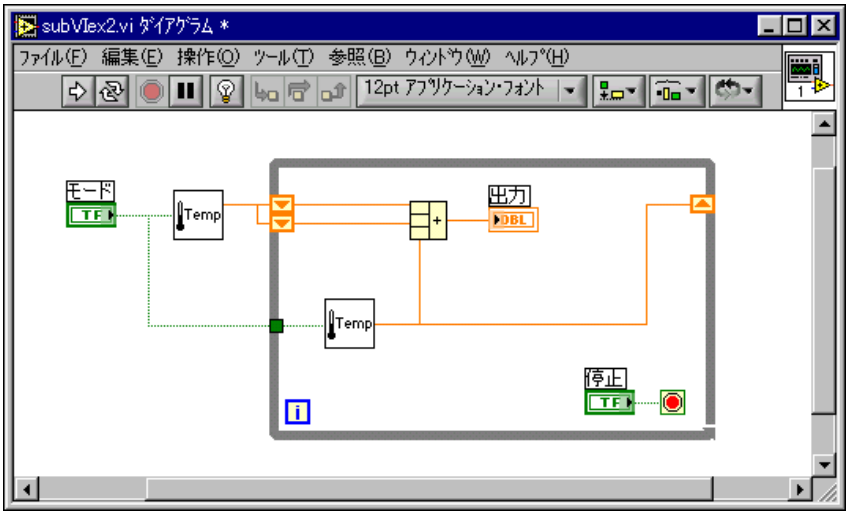
操作ツールまたは位置決めツールでブロックダイアグラム上のサブ VI をダブルクリックすると編集することができます。サブ VI を保存すると、現在のインスタンスだけでなく、サブ VI に対するすべての呼び出しに変更の影響があります。

共通の操作を監視する

VI の作成後、特定の操作を頻繁に行うことに気付く場合があります。サブ VI またはループを使用してその操作を繰り返し行うことを検討してください。たとえば、次のブロックダイアグラムには 2 つの同じ操作が含まれています。



次のブロックダイアグラムのように、その操作を実行するサブ VI を作成し、そのサブ VI を 2 回呼び出すことができます。



他の VI 内でサブ VI を再利用することもできます。ループを使用して共通の操作を結合する方法の詳細については、第 8 章「ループと Case ストラクチャ」を参照してください。

コネクタペーンを設定する

VI をサブ VI として使用するには、左図に示すコネクタペーンを作成する必要があります。コネクタペーンは、VI の制御器および表示器に対応する端子のセットで、テキストベースのプログラミング言語における関数呼び出しのパラメータリストに似ています。コネクタペーンは、VI に配線できる入力端子と出力端子を定義するため、サブ VI として使用することができます。コネクタペーンの詳細については、第 2 章「バーチャルインスツルメンツ (仮想計測器) VI の概要」の「アイコンとコネクタペーン」のセクションを参照してください。

接続を定義するには、フロントパネル上の制御器または表示器をコネクタペーンの各端子に割り当てます。コネクタペーンを定義するには、フロントパネルウィンドウの右上コーナーにあるアイコンを右クリックし、ショートカットメニューから**コネクタを表示**を選択します。これでコネクタペーンがアイコンに置き換えられます。コネクタペーン上の各四角形は端子を表します。この四角形を使用して入出力を割り当てます。LabVIEW がコネクタペーン上に表示する端子の数は、フロントパネル上の制御器と表示器の数によって異なります。

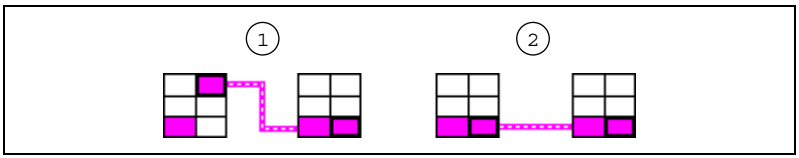
コネクタペーンには最大 28 個の端子があります。プログラムで使用する制御器および表示器がフロントパネル上に 29 以上ある場合は、制御器および表示器のいくつかを 1 つのクラスタにグループ化して、このクラスタをコネクタペーン上の端子に割り当てます。クラスタを使用してデータをグループ化する方法の詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「クラスタ」のセクションを参照してください。

メモ　1 つの VI に 17 個以上の端子を割り当てないように注意してください。端子が多すぎると、VI が煩雑でわかりにくくなる可能性があります。

VI に対して別の端子パターンを選択するには、コネクタペーンを右クリックし、ショートカットメニューから**パターン**を選択します。追加端子を持つコネクタペーンパターンを選択します。追加端子は、それらが必要になるまで未接続にしておくことができます。このように柔軟性を持たせることによって、VI の階層に対する影響を最小限に抑えた状態で変更を行うことができます。

頻繁に併用するサブ VI のグループを作成する場合は、各入力を配置する位置を思い出せるように、同じ位置に共通の入力端子を持つ統一されたコネクタペーンをサブ VI に指定します。他のサブ VI が入力として使用する出力を生成するサブ VI を作成する場合は、配線パターンをわかりやすくするために入出力接続を揃えます。フロントパネルの左下コーナーに**エラー入力**クラスタを配置し、右下コーナーに**エラー出力**クラスタを配置します。

図 7-1 は、不適切に揃えられたエラークラスタの例と適切に揃えられたエラークラスタの例を示しています。



1　不適切な例　　2　適切な例

図 7-1　不適切に揃えられたエラークラスタと適切に揃えられたエラークラスタ

必須、推奨、および任意の入出力を設定する

ユーザがサブ VI 接続の配線を忘れないように、必須、推奨、および任意の入出力を指定できます。

コネクタペーン内の端子を右クリックし、ショートカットメニューから**らこの接続は**を選択します。チェックマークはその端子の現在の設定を示します。必須、推奨、または任意を選択します。

端子入力の場合、必須とは、必須入力が配線されないと、サブ VI が配置されたブロックダイアグラムが壊れることを意味します。必須は、端子出力には使用できません。端子入力および出力の場合、推奨は、入力が配線されていなくても、サブ VI が配置されたブロックダイアグラムは実行可能であることを意味します。ただし、入力を配線しないと、**警告**ダイアログボックスに入力が配線されていないという警告が生成されます。任意は、サブ VI を配置されたブロックダイアグラムが実行可能で、端子入力または出力が配線されていなくても警告が生成されないことを意味します。

`vi.lib` 内の VI の入出力は、**必須**、**推奨**、または**任意**として既にチェックマークが付けられています。作成した VI の入出力はデフォルトで**推奨**に設定されます。VI が正しく動作するためにその入力または出力が必要な場合にのみ、端子の設定を必須に設定します。

ヘルプウィンドウで**詳細**表示が選択されている場合、必須接続は太字、推奨接続は通常のテキスト、任意接続は淡色表示で示されます。**シンプル**表示が選択されている場合は表示されません。

アイコンを作成する

各 VI では、フロントパネルおよびブロックダイアグラムウィンドウの右上コーナーに左図のようなアイコンが表示されます。アイコンは VI のグラフィカル表現です。アイコンには、テキスト、画像、またはその両方を含めることができます。ある VI をサブ VI として使用する場合、アイコンはその VI のブロックダイアグラム上にあるサブ VI を識別します。

デフォルトのアイコンには、LabVIEW の起動後に開いた新しい VI の数を示す数字が含まれています。カスタムアイコンを作成してデフォルトアイコンと置き換えるには、フロントパネルまたはブロックダイアグラムの右上コーナーにあるアイコンを右クリックして、ショートカットメニューから**アイコンを編集**を選択するか、あるいはフロントパネルの右上コーナーにあるアイコンをダブルクリックします。

ファイルシステムの任意の場所からグラフィックをドラッグして、フロントパネルまたはブロックダイアグラムの右上コーナーにドロップすることができます。LabVIEW は、グラフィックを 32 × 32 ピクセルのアイコンに変換します。

使用するモニタのタイプに従って、モノクロ、16 色、および 256 色モードで別々のアイコンを設計できます。カラープリンタを使用していない場合は、モノクロのアイコンが印刷に使用されます。

VI の一部からサブ VI を作成する

VI の一部をサブ VI に変換するには、位置決めツールを使用して、再利用するブロックダイアグラムの部分を選択し、**編集→選択範囲をサブ VI に変換**を選択します。ブロックダイアグラムの選択された部分が新しいサブ VI のアイコンに置き換えられます。LabVIEW は新しいサブ VI の制御器と表示器を作成し、サブ VI を既存のワイヤに配線します。

選択範囲からサブ VI を作成すると便利ですが、VI の論理階層を作成する場合には慎重に計画する必要があります。選択範囲内のどのオブジェクトを含めるかを検討し、その結果作成される VI の機能が変わらないようにしてください。

サブ VI を設計する

ユーザがサブ VI のフロントパネルを表示する必要がない場合は、色、フォントなど、その外観の作成に時間をかける必要はありません。ただし、VI をデバッグするときにフロントパネルを表示しなければならない場合があるので、フロントパネルの構成は重要です。

コネクタペーンに表示されるとおりに制御器と表示器を配置します。フロントパネルの左側に制御器を配置し、右側に表示器を配置します。フロントパネルの左下に**エラー入力**クラスタを配置し、右下に**エラー出力**クラスタを配置します。コネクタペーンの設定方法の詳細については、この章の「コネクタペーンを設定する」のセクションを参照してください。

制御器がデフォルト値を持つ場合は、そのデフォルト値を括弧で囲んで制御器名の一部とします。また、該当する場合は制御器の名前に測定単位を指定します。たとえば、制御器が上限温度を設定する場合にデフォルト値が 75 °F のときは、制御器に**上限温度 (75 degF)** という名前を付けます。

複数のプラットフォーム上でそのサブ VI を使用する場合は、制御器の名前に特殊文字を使用しないでください。たとえば、°**F** の代わりに **degF** を使用してください。

ユーザが TRUE 状態での制御器の動作を判断できるように、ブール制御器に名前を付けます。**キャンセル**、**リセット**、**初期化**のように、実行される動作を記述する名前を使用します。

VI の階層を表示する

階層ウィンドウには、タイプの定義やグローバル変数など、メモリ内にあるすべての VI の呼び出し側階層のグラフィック表現が表示されます。**階層ウィンドウ**を表示するには、**参照→ VI 階層を表示**を選択します。このウィンドウを使用して、現在の VI を構成するサブ VI や他のノードを表示します。

階層ウィンドウのオブジェクト上に操作ツールを移動すると、各 VI の名前が表示されます。VI を他の VI 内でサブ VI として使用するには、位置決めツールを使用して**階層ウィンドウ**からブロックダイアグラムにその VI をドラッグします。また、1 つまたは複数のノードを選択し、クリップボードにコピーして、他のブロックダイアグラム上に貼り付けることもできます。その VI のフロントパネルを表示するには、**階層ウィンドウ**内の VI のノードをダブルクリックします。

サブ VI が含まれている VI には、その枠に矢印ボタンが表示されます。サブ VI を表示または非表示にするには、この矢印ボタンをクリックします。赤い矢印ボタンは、すべてのサブ VI が非表示になっていることを示します。黒い矢印ボタンは、すべてのサブ VI が表示されていることを示します。

VI を保存する

VI は、個別ファイルとして保存したり、複数の VI をグループ化して VI ライブラリ内に保存できます。VI ライブラリファイルの拡張子は `.llb` です。特に複数の開発者が同じプロジェクトに取り組んでいる場合は、ディレクトリに分類された個別ファイルとして VI を保存することをお勧めします。

VI を個別ファイルとして保存する場合の利点

VI を個別ファイルとして保存する理由を以下に示します。

- ファイルシステムを使用して個別ファイルを管理できる。
- サブディレクトリを使用できる。
- 同じファイルにプロジェクト全体を保存するよりも、個別ファイルに VI および制御器を保存する方が信頼性が高い。

- プロフェッショナル開発システムの組み込みソースコード管理ツールやサードパーティのソースコード管理ツールを使用できる。

VI をライブラリとして保存する場合の利点

VI をライブラリとして保存する理由を以下に示します。

- ファイル名に最大 255 文字を使用できる。

  (Macintosh) MacOS 9.*x* 以前のバージョンではファイル名が 31 文字に制限されていますが、ライブラリ名には字数制限はありません。
- 他のプラットフォームに VI を転送する場合は、個々の VI を何度か転送するよりも VI ライブラリを転送する方が簡単である。
- VI ライブラリはファイルを圧縮するため、プロジェクトのサイズを若干小さくできる。
- ライブラリで最上位 VI としてマークを付け、ライブラリを開いたときに LabVIEW が自動的にそのライブラリのすべての最上位 VI を開くようにすることができる。

メモ　LabVIEW では、すべてのプラットフォーム上で記憶位置を統一するため、組み込み VI とサンプル VI の多くが VI ライブラリ内に保存されています。

VI ライブラリを使用する場合は、アプリケーションを複数の VI ライブラリに分割することを検討してください。つまり、ある VI ライブラリに上位 VI を配置し、それ以外のライブラリは機能別に VI を含むように設定してください。ライブラリの VI に変更を保存する場合、オペレーティングシステムが大きなファイルに変更を書き込まなければならないため、個別の VI に変更を保存する場合より時間がかかります。大きなライブラリに変更を保存すると、メモリ容量が増え、性能が低下する可能性があります。各ライブラリのサイズは約 1 MB までとしてください。

ライブラリ内で VI を管理する

VI ライブラリ内でのファイルのコピー、名前の変更、および削除を簡単に行うには、VI ライブラリマネージャを使用します。このツールにアクセスするには、**ツール→ VI ライブラリマネージャ**を選択します。このツールを使用すると、新しい VI ライブラリとディレクトリを作成し、そのディレクトリ間で VI ライブラリを変換することもできます。ソースコード管理ツールを使用して VI を管理する必要がある場合は、新しい VI ライブラリおよびディレクトリを作成してそのディレクトリ間で VI ライブラリを変換することが重要になります。

VI ライブラリマネージャを使用する前に、既にメモリ内にある VI に対してファイル操作が行われないように、影響を受ける可能性があるすべての VI を閉じてください。

VI に名前を付ける

VI を保存するときは、わかりやすい名前を使用してください。`Temperature Monitor.vi` や `Serial Write & Read.vi` のようなわかりやすい名前を使用すると、VI を簡単に識別でき、その使用方法が一目でわかります。`VI#1.vi` のようなあいまいな名前を使用すると、特に複数の VI を保存した場合に VI の識別が困難になる可能性があります。

ユーザが他のプラットフォームでその VI を実行する可能性があるかどうかを考慮してください。`\`、`:`、`/`、`?`、`*`、`<>`、`#` など、一部のオペレーティングシステムで特別な目的のために予約されている文字は使用しないでください。

(Macintosh) ユーザが MacOS 9.*x* 以前のバージョンで VI を実行する場合は、VI の名前を 30 文字以下にしてください。

旧バージョンの形式で保存する

VI は、旧バージョンの形式で保存できます。これにより、LabVIEW のアップグレードが便利になり、必要に応じて VI を 2 つのバージョンの LabVIEW 用に維持しておくことができます。新しいバージョンにアップグレードしても、旧バージョンの VI に戻ることができます。

旧バージョンの形式で VI を保存する際、LabVIEW はその VI だけでなく、`vi.lib` ファイルを除くその階層内のすべての VI を変換します。

VI には旧バージョンの LabVIEW では提供されていなかった機能が使用されていることがあります。この場合、LabVIEW はできるだけ多くの VI を保存し、変換できなかった VI についてはレポートを生成します。そのレポートは、生成されるとすぐに**警告**ダイアログボックスに表示されます。これらの警告を確認してダイアログボックスを閉じるには、**OK** ボタンをクリックします。**保存**ボタンをクリックして警告をテキストファイルに保存しておくと、後で調べることができます。

VI を配布する

他のコンピュータや他のユーザに VI を配布する場合は、ユーザが編集可能なブロックダイアグラムのソースコードを含めるか、あるいはブロックダイアグラムを非表示または削除するかを検討してください。ファイルサイズを小さくし、ユーザがソースコードを変更できないようにするには、**ファイル→オプション付き保存**を選択して、ブロックダイアグラムなしで VI を保存します。ブロックダイアグラムなしで VI を保存すると、ユーザは VI から他のプラットフォームに移動したり、LabVIEW の今後のバージョンに VI をアップグレードすることもできなくなります。

注意　ブロックダイアグラムなしで VI を保存する場合、元のバージョンの VI に上書きしないでください。VI を別のディレクトリに保存するか、別名で保存してください。

ブロックダイアグラムを削除しない場合は、ブロックダイアグラムにパスワード保護を割り当てることができます。ブロックダイアグラムは使用可能ですが、ユーザはブロックダイアグラムを表示または編集するためにパスワードを入力する必要があります。

VI を配布するための別のオプションとして、スタンドアロンアプリケーションまたは共有ライブラリを作成する方法があります。ユーザが VI を編集する可能性がない場合は、スタンドアロンアプリケーションまたは共有ライブラリの使用が適しています。ユーザはアプリケーションを実行したり共有ライブラリを使用できますが、ブロックダイアグラムの編集や表示はできません。スタンドアロンアプリケーションには簡易メニューが用意されています。

プラットフォーム間での VI のポート、および VI のローカライズの詳細については、『LabVIEW VI の移植とローカライズ』アプリケーションノートを参照してください。

スタンドアロンアプリケーションと共有ライブラリを作成する

アプリケーションビルダを使用してスタンドアロンアプリケーションおよびインストーラまたは VI の共有ライブラリ (DLL) を作成するには、**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。テキストベースのプログラミング言語を使用して共有ライブラリ内の VI を呼び出す場合は、共有ライブラリを使用します。テキストベースのプログラミング開発環境で VI で作成した機能を共有したい場合に、共有ライブラリを使用すると便利です。

メモ　LabVIEW プロフェッショナル開発システムにはアプリケーションビルダが含まれています。LabVIEW 基本パッケージまたは LabVIEW 開発システムを使用している場合は、アプリケーションビルダを別途購入できます。

作成するアプリケーションまたは共有ライブラリ (DLL) のさまざまな設定を構成するには、**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タブを使用します。これらの設定を行ったら、必要に応じてアプリケーションを簡単に再構築できるように、それらの設定をスクリプトとして保存します。

ユーザはアプリケーションを実行したり、共有ライブラリを使用できますが、ブロックダイアグラムの編集や表示はできません。

アプリケーションビルダのインストール方法の詳細については、『LabVIEW アプリケーションビルダリリースノート』を参照してください。

第 II 部

VI の作成と編集

第 II 部では、アプリケーションを特定の方法で動作可能にするために使用する LabVIEW の機能、VI、および関数について説明します。各章では、LabVIEW の各機能の有効性や、VI および関数の各クラスの概要について説明します。

第 II 部「VI の作成と編集」は以下の章で構成されています。

- 第 8 章「ループと Case ストラクチャ」では、ブロックダイアグラム上のストラクチャを使用してコードのブロックを繰り返したり、条件や特定の順序に従ってコードを実行する方法について説明します。
- 第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」では、データをグループ化するための文字列、配列、およびクラスタの使用法について説明します。
- 第 10 章「ローカルおよびグローバル 変数」では、ローカルおよびグローバル変数を使用してワイヤで接続できないアプリケーションの位置の間で情報をやり取りする方法について説明します。
- 第 11 章「グラフとチャート」では、グラフおよびチャートを使用してグラフ形式でデータのプロットを表示する方法について説明します。
- 第 12 章「グラフィック & サウンド VI」では、VI の画像や音を表示したり変更する方法について説明します。
- 第 13 章「ファイル I/O」では、ファイル I/O 操作を実行する方法について説明します。
- 第 14 章「VI の文書化と印刷」では、VI の文書化や印刷の方法について説明します。
- 第 15 章「VI をカスタマイズする」では、アプリケーションのニーズに応じて動作するように VI とサブ VI を構成する方法について説明します。

- 第 16 章「プログラム的に VI を 制御する」では、VI や LabVIEW の他のインスタンスと通信し、プログラムによって VI および LabVIEW を制御する方法について説明します。
- 第 17 章「LabVIEW のネットワーク動作」では、VI を使用して他のアプリケーションやリモートコンピュータで実行中のものを含め他のプロセスとの通信およびネットワークを介した通信を行う方法について説明します。
- 第 18 章「ActiveX」では、他の Windows アプリケーションがアクセスできるようにオブジェクト、コマンド、および関数のセットをパブリッシュする方法について説明します。
- 第 19 章「テキストベースのプログラミング言語からのコード呼び出し」では、テキストベースのプログラミング言語からコードを呼び出したり、DLL を使用する方法について説明します。
- 第 20 章「フォーミュラと方程式」では、VI での式の使用法について説明します。

8

ループと Case ストラクチャ

ストラクチャは、テキストベースのプログラミング言語におけるループおよびケースステートメントのグラフィック表現です。コードのブロックを繰り返したり、条件付きでコードを実行したり、特定の順序でコードを実行するには、ブロックダイアグラムでストラクチャを使用します。

他のノードと同様に、ストラクチャには端子があります。これらの端子はストラクチャをブロックダイアグラムの他のノードに接続し、入力データを受け取ると自動的に実行して、実行が終了するとそのデータを出力ワイヤに渡します。

各ストラクチャにはサイズ変更可能な特有の枠があり、ストラクチャの規則に従って実行されるブロックダイアグラムの一部を囲みます。ストラクチャの枠で囲まれたブロックダイアグラムの部分をサブダイアグラムと呼びます。ストラクチャにデータを受け渡す端子をトンネルと呼びます。トンネルはストラクチャの枠上にある接続ポイントです。

詳細については

ストラクチャの使用方法の詳細については、「LabVIEW ヘルプ」を参照してください。

ブロックダイアグラムがプロセスをどのように実行するかを制御するには、**関数→ストラクチャ**パレット上にある以下のストラクチャを使用します。

- **For ループ**：設定された回数だけサブダイアグラムを実行します。
- **While ループ**：条件が一致するまでサブダイアグラムを実行します。
- **Case ストラクチャ**：複数のサブダイアグラムが含まれており、ストラクチャに渡された入力値に従って、そのうちの 1 つのみが実行されます。
- **シーケンスストラクチャ**：1 つまたは複数のサブダイアグラムが含まれており、順番に実行されます。
- **フォーミュラノード**：入力された数値に基づいて数学演算を実行します。フォーミュラノードの使用方法の詳細については、第 20 章「フォーミュラと方程式」の「フォーミュラノード」のセクションを参照してください。

- **イベントストラクチャ**：ユーザが VI と対話する方法に応じて実行する 1 つまたは複数のサブダイアグラムが含まれています。

ショートカットメニューを表示するには、ストラクチャの枠を右クリックします。

For ループおよび While ループのストラクチャ

繰り返し操作を制御するには、For ループと While ループを使用します。

For ループ

左図に示す For ループは、設定された回数だけサブダイアグラムを実行します。

左図に示すカウント端子 (入力端子) の値は、サブダイアグラムの実行を繰り返す回数を示します。回数を明示的に設定するには、ループの外側からカウント端子の左側または上側に値を配線するか、自動指標付け機能を使用して自動的に回数を設定します。回数を自動的に設定する方法の詳細については、この章の「自動指標付けで For ループの回数を設定する」のセクションを参照してください。

左図に示す繰り返し端子 (出力端子) には、実行された繰り返しの回数が表示されます。繰り返しのカウントは常にゼロ (0) から始まります。最初の繰り返しでは、繰り返し端子は 0 を返します。

カウント端子と繰り返し端子はともに符号付き倍長整数です。浮動小数点数をカウント端子に配線すると、LabVIEW は、浮動小数点数を範囲内の数値に強制的に丸めます。カウント端子に 0 または負の数を配線すると、ループは実行されません。

現在の繰り返しから次の繰り返しにデータを渡すには、For ループにシフトレジスタを追加します。シフトレジスタをループに追加する方法の詳細については、この章の「ループ内のシフトレジスタ」のセクションを参照してください。

While ループ

テキストベースのプログラミング言語における Do ループまたは Repeat-Until ループと同様に、左図に示す While ループは条件が一致するまでサブダイアグラムを実行します。

While ループは、入力端子である条件端子が特定のブール値を受け取るまでサブダイアグラムを実行します。条件端子のデフォルトは、**True の場合、継続** (左図) です。条件端子が **True の場合、継続**の場合、While ループは条件端子が FALSE 値を受け取るまでそのサブダイアグラムを実

行します。条件端子の動作と外観を変更するには、端子または While ループの枠を右クリックし、左図に示す **True の場合停止**を選択します。また、操作ツールを使用して、条件端子をクリックし、条件を変更することもできます。条件端子が **True の場合停止**の場合、While ループは条件端子が TRUE 値を受け取るまでそのサブダイアグラムを実行します。VI は各繰り返しの最後に条件端子をチェックするので、While ループは少なくとも 1 回は実行されます。条件端子を配線しないと、VI は壊れます。

また、While ループの条件端子を使用すると、基本的なエラー処理を実行できます。条件端子にエラークラスタを配線すると、エラークラスタの**ステータス**パラメータの TRUE または FALSE 値だけが端子に渡されます。また、ショートカットメニュー項目が、**True の場合停止**と **True の場合、継続**から**エラーで停止**と**エラーの場合、継続**に変わります。エラークラスタとエラー処理の詳細については、第 6 章「VI の実行とデバッグ」の「エラーチェックとエラー処理」のセクションを参照してください。

左図に示す繰り返し端子 (出力端子) には、実行された繰り返しの回数が表示されます。繰り返しのカウントは常にゼロ (0) から始まります。最初の繰り返しでは、繰り返し端子は 0 を返します。

現在の繰り返しから次の繰り返しにデータを渡すには、While ループにシフトレジスタを追加します。シフトレジスタをループに追加する方法の詳細については、この章の「ループ内のシフトレジスタ」のセクションを参照してください。

無限 While ループを回避する

図 8-1 のように、ブール制御器の端子が While ループの外側に配置され、ループの開始時の条件端子が **True の場合停止**の場合に制御器が FALSE に設定されていると、無限ループが発生します。ループの外側の制御器が TRUE に設定され条件端子が **True の場合、継続**の場合も、無限ループが発生します。

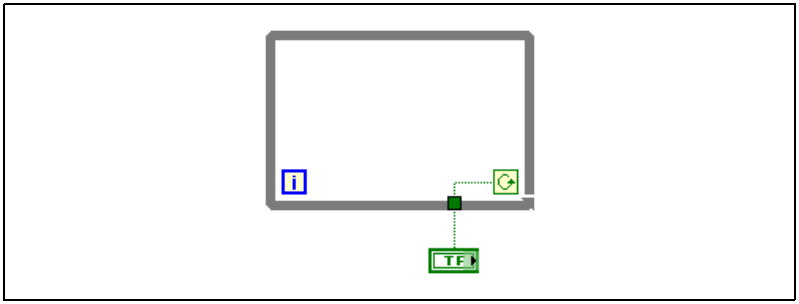


図 8-1　無限 While ループ

制御器の値はループの開始前に一度読み取られるだけなので、その値を変更しても無限ループは停止しません。無限ループを停止するには、ツールバーの**実行停止**ボタンをクリックして VI を中断する必要があります。

ループに自動指標付けを行う

For ループまたは While ループ入力トンネルに配列を配線する場合に自動指標付けを有効にすると、その配列内のすべての要素を読み取って処理できます。配列の詳細については、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」を参照してください。

ループの枠上の入力トンネルに配列を配線し、その入力トンネルで自動指標付けを有効にすると、その配列の要素は最初の要素から 1 つずつループに入ります。自動指標付けが無効になっていると、配列全体がループに渡されます。配列の出力トンネルを自動指標付けすると、出力配列はループのすべての繰り返しから新しい要素を受け取ります。したがって、自動指標付けされた出力配列のサイズは常に繰り返し回数と等しくなります。

自動指標付けを有効または無効にするには、ループの枠上のトンネルを右クリックし、ショートカットメニューから**指標付け使用**または**指標付け不使用**を選択します。While ループの自動指標付けはデフォルトで無効になっています。

ループは 1 次元配列のスカラ要素、2 次元配列の 1 次元要素、というように指標を付けます。出力トンネルではその逆の操作が行われます。スカラ要素は 1 次元配列内に、1 次元配列は 2 次元配列内に、というように順番に配置されます。

自動指標付けで For ループの回数を設定する

For ループに入力された配列で自動指標付けを有効にすると、LabVIEW がカウント端子を配列のサイズに設定するため、カウント端子を配線する必要はありません。For ループを使用すると配列を一度に 1 要素ずつ処理できるので、For ループに配線するすべての配列に関して LabVIEW はデフォルトで自動指標付けを有効にします。配列を、一度に 1 要素ずつ処理する必要がない場合は、自動指標付けを無効にします。

複数のトンネルに対して自動指標付けを有効にする場合や、カウント端子を配線する場合、回数は選択した小さい方の値になります。たとえば、それぞれ 10 個と 20 個の要素を持つ自動指標付けされた 2 つの配列がループに入り、値 15 をカウント端子に配線している場合、ループは 10 回実行し、2 番目の配列の最初の 10 個の要素だけを指標付けします。1 つのグラフ上で 2 つのデータソースからデータをプロットし、最初の 100 個の要素をプロットする場合は、カウント端子に 100 を配線します。ただし、要素が 50 個しか含まれていないデータソースがある場合は、ループ

は 50 回実行し、最初の 50 個の要素にのみ指標付けします。配列のサイズを調べるには、Array Size 関数を使用します。

配列の出力トンネルを自動指標付けすると、出力配列はループのすべての繰り返しから新しい要素を受け取ります。したがって、自動指標付けされた出力配列のサイズは常に繰り返し回数と等しくなります。たとえば、ループが 10 回実行される場合、出力される配列には 10 個の要素が入っています。出力トンネルで自動指標付けを無効にすると、ループの最後の繰り返しの要素だけがブロックダイアグラム内の次のノードに渡されます。自動指標付けが有効になっていることを示すために、括弧付きのグリフがループの枠に表示されます。出力トンネルと次のノード間のワイヤの太さも、ループが自動指標付けを使用しているかどうかを示します。自動指標付けを使用すると、ワイヤにはスカラではなく配列が入るためワイヤが太くなります。

While ループで自動指標付けを行う

While ループに入る配列に対して自動指標付けを有効にすると、While ループは For ループの場合と同様に配列に指標を付けます。ただし、While ループは特定の条件が一致するまで繰り返されるため、While ループが実行する繰り返し回数が配列のサイズによって制限されることはありません。While ループが入力配列の最後を過ぎても指標を付けるときは、配列要素タイプのデフォルト値がループに渡されます。Array Size 関数を使用すると、デフォルト値が While ループに渡されないようにすることができます。Array Size 関数は配列内にある要素の数を示します。While ループが配列サイズと同じ回数だけ繰り返した時点で実行を停止するように設定します。

注意　出力配列のサイズはあらかじめ決定できないため、While ループで使用するよりも、For ループの出力に対して自動指標付けを有効にする方が効率的です。あまり多くの繰り返しを行うと、システムのメモリが不足する可能性があります。

ループ内のシフトレジスタ

ループのある繰り返しから次の繰り返しに値を転送するには、For ループおよび While ループでシフトレジスタを使用します。シフトレジスタは、テキストベースのプログラミング言語におけるスタティック変数に類似しています。

左図に示すシフトレジスタは、ループ枠の左右に一対の端子があり、互いに向かい合っています。右の端子には上矢印が付いており、繰り返し処理完了時のデータが格納されます。LabVIEW は、レジスタの右側に接続されたデータを次の繰り返しに転送します。シフトレジスタを作成するには、ループの右または左の枠を右クリックし、ショートカットメニューから**シフトレジスタを追加**を選択します。

シフトレジスタはどのタイプのデータでも転送でき、シフトレジスタに配線された最初のオブジェクトのデータタイプに自動的に適応します。各シフトレジスタの端子に配線するデータは、同じタイプである必要があります。ストラクチャ上に複数のシフトレジスタを作成できます。前の複数回の繰り返しでの値を記憶させておくために、左側に複数の端子を使用できます。

ループの実行後、シフトレジスタに格納されている最後の値は右の端子に残されます。右の端子をループの外側に配線すると、シフトレジスタに格納されている最後の値が転送されます。

レジスタを初期化しない場合、ループは最後に実行されたときにレジスタに書き込まれた値か、ループが実行されなかった場合のデータタイプのデフォルト値を使用します。

初期化されていないシフトレジスタとともにループを使用し、VI が実行されるたびに、シフトレジスタの初期出力が前の実行からの最後の値になるように VI を繰り返し実行します。以降の VI の実行間で状態情報を保持するには、初期化されていないシフトレジスタを使用してください。

タイミングを制御する

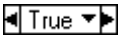
データがチャートにプロットされる速度などのプロセスの実行速度を制御する必要がある場合があります。ループ内で Wait 関数を使用するとループが再実行されるまでの待機時間（ミリ秒単位) だけ待つことができます。

Case ストラクチャとシーケンスストラクチャ

Case ストラクチャ、シーケンスストラクチャ、およびイベントストラクチャには複数のサブダイアグラムが含まれていますが、一度に表示できるのはこの中の 1 つだけです。Case ストラクチャは、ストラクチャに渡された入力値に従って 1 つのサブダイアグラムを実行します。シーケンスストラクチャはそのすべてのサブダイアグラムを順番に実行します。

Case ストラクチャ

左図に示す Case ストラクチャには、複数のサブダイアグラム、すなわちケースが含まれています。サブダイアグラムは一度に 2 つしか表示できず、またストラクチャは一度に 1 つしかケースを実行しません。入力値によって、どのサブダイアグラムが実行されるかが決まります。Case ストラクチャは、テキストベースのプログラミング言語におけるケースステートメントまたは `if...then...else` ステートメントに似ています。



Case ストラクチャの上部にあるケースセレクタラベルには、左図のように、中央にケースに対応するセレクタ値の名前があり、その両側に減分矢印と増分矢印があります。使用可能なケースをスクロールするには、減分矢印／増分矢印をクリックします。ケース名の横にある下矢印をクリックして、プルダウンメニューからケースを選択することもできます。

どのケースが実行されるかを確認するには、入力値、すなわちセレクタを左図に示すセレクタ端子に配線します。整数、ブール値、文字列、または列挙型の値をセレクタ端子に配線する必要があります。セレクタ端子は、Case ストラクチャの左側の枠上の任意の場所に配置できます。セレクタ端子のタイプがブールの場合、ストラクチャには TRUE ケースと FALSE ケースがあります。セレクタ端子が整数、文字列、または列挙型の値の場合には、ストラクチャに任意の数のケースを使用することができます。

範囲外の値を処理するために Case ストラクチャにデフォルトケースを指定します。そうでない場合には、可能な入力値をすべてリストする必要があります。たとえば、セレクタが整数で、1、2、および 3 に対してケースを指定する場合、入力値が 4 またはその他の有効な整数値であれば、実行するためにデフォルトケースを指定する必要があります。

ケースセレクタの値とデータタイプ

ケースセレクタラベルには、1 つの値またはリスト、および値の範囲を入力することができます。リストの場合は、カンマを使用して値を区切ります。数値の範囲の場合、`10..20` で範囲を指定します。これは、10 ～ 20 のすべての数値を意味します。また、より大まかな範囲も使用できます。たとえば、`..100` は、100 以下のすべての数値を示します。また、`..5, 6, 7..10, 12, 13, 14` など、リストや範囲を結合することもできます。同じケースセレクタラベルに重なる範囲が含まれる値を入力すると、Case ストラクチャは、コンパクトな形式でラベルを再表示します。前述の例は **`..10, 12..14`** として再表示されます。文字列の範囲の場合、`a..c` の範囲は、a および b のすべての含みますが、c は含みません。範囲 `a..c,c` は、最後の値 c も含みます。

ケースセレクタラベルに文字列や列挙された値を入力すると、**`"red"`**、**`"green"`**、**`"blue"`** などのように、値は引用符で囲まれて表示されます。ただし、文字列や列挙された値にカンマまたは範囲記号 (`","` または `".."`) が含まれていない場合は、値を入力するときに引用符を入力する必要はありません。文字列の値には、英数字以外の文字に特別な円コードを使用します。たとえば、復帰文字には `\r`、改行文字には `\n`、タブには `\t` を使用します。これらの円コードのリストについては、「LabVIEW ヘルプ」を参照してください。

Case ストラクチャのセレクタ端子に接続されたワイヤのデータタイプを変更した場合、可能であれば、Case ストラクチャはケースセレクタの値

を自動的に新しいデータタイプに変換します。たとえば、数値 19 を文字列に変換すると、文字列値は "19" になります。文字列を数値に変換する場合、LabVIEW は数値を表す文字列の値だけを変換します。他の値は文字列のまま残されます。数値をブール値に変換する場合、0 は FALSE に変換され、1 は TRUE に変換されます。それ以外のすべての数値は文字列になります。

セレクタ端子に配線されたオブジェクトとはタイプが異なるセレクタ値を入力すると、その値は赤で表示されて、ストラクチャを実行する前に値を削除または編集する必要があることを示し、VI は実行されません。また、浮動小数点演算特有の四捨五入での誤差が生じる可能性があるため、浮動小数点値はケースセレクタ値として使用できません。浮動小数点値をケースに配線すると、LabVIEW は最も近い整数に値を四捨五入します。ケースセレクタラベルに浮動小数点値を入力すると、その値は赤で表示されて、ストラクチャを実行する前に値を削除または編集する必要があることを示します。

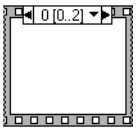
入力トンネルと出力トンネル

Case ストラクチャに複数の入力トンネルと出力トンネルを作成することができます。入力はすべてのケースに利用できますが、ケースに各入力を使用する必要はありません。しかし、出力トンネルはケースごとに定義する必要があります。あるケースに出力トンネルを作成すると、他のすべてのケースの枠上の同じ位置にトンネルが現れます。1 つも出力トンネルが配線されていないと、そのストラクチャのすべての出力トンネルが白い正方形で表示されます。各ケース内の同じ出力トンネルに異なるデータソースを定義することができますが、各ケースのデータタイプに互換性がある必要があります。出力トンネルを右クリックしてショートカットメニューから**未配線の場合はデフォルトを使用**を選択することもできます。

エラー処理に Case ストラクチャを使用する

Case ストラクチャのセレクタ端子にエラークラスタを配線すると、ケースセレクタラベルには 2 つのケース、すなわちエラーとエラーなしが表示されます。Case ストラクチャの枠の色は、エラーの場合は赤に変わり、エラーなしの場合は緑に変わります。Case ストラクチャは、エラー状態に基づいて適切なケースサブダイアグラムを実行します。エラー処理の詳細については、第 6 章「VI の実行とデバッグ」の「エラー処理」のセクションを参照してください。

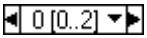
シーケンスストラクチャ



左図に示すシーケンスストラクチャには、順番に実行される 1 つまたは複数のサブダイアグラム、すなわちフレームが含まれています。シーケンスストラクチャの上部のフレームラベルは、Case ストラクチャのケース

セレクタラベルに似ています。フレームラベルには中央にフレーム番号があり、両側に増分矢印と減分矢印があります。使用可能なフレームをスクロールするには、減分矢印と増分矢印をクリックします。フレーム番号の隣にある下矢印をクリックして、プルダウンメニューからフレームを選択することもできます。ケースセレクタラベルの場合とは異なり、フレームラベルに値を入力することはできません。シーケンスストラクチャにフレームを追加したり、ストラクチャからフレームを削除したり、フレームの配置を変えたりすると、LabVIEW はフレームラベルの番号を自動的に調整します。

シーケンスストラクチャはフレーム 0 を実行し、次にフレーム 1、その次にフレーム 2 というように、最後のフレームまで実行を続けます。シーケンスストラクチャは、最後のフレームが終了するまで実行を完了せず、いかなるデータも返しません。



Case ストラクチャの上部にあるケースセレクタ識別子には、左図のように中央にケースセレクタ識別子があり、その両側に減分ボタンと増分ボタンがあります。たとえば、左図に示すシーケンスセレクタ識別子では、`0` が現在のフレーム番号であり、`[0..2]` がフレームの範囲となります。使用可能なフレームをスクロールするには、減分／増分矢印ボタンをクリックします。

シーケンスストラクチャを使用して、自然なデータ依存が存在しない場合の実行順序を制御します。他のノードからデータを受け取るノードは、データに関して他のノードに依存しているため、常に他のノードの実行が終了した後で実行されます。

ブロックダイアグラムの他の部分と同様に、シーケンスストラクチャの各フレーム内では、データ依存によってノードの実行順序が決まります。データ依存の詳細については、第 5 章「ブロックダイアグラムを作成する」の「データ依存と人工データ依存」のセクションを参照してください。

Case ストラクチャとは異なり、シーケンスストラクチャのトンネルにはデータソースを 1 つしか使用できません。出力はどのフレームからでも可能ですが、データは、個々のフレームの実行が完了したときではなく、すべてのフレームの実行が完了したときにのみシーケンスストラクチャから渡されます。Case ストラクチャの場合と同様に、入力トンネルのデータはすべてのフレームに利用可能です。

あるフレームから後続のフレームにデータを渡すには、左図に示すシーケンスローカル端子を使用します。データソースを含むフレームのシーケンスローカル端子に、外側を向いた矢印が表示されます。以降のフレーム内の端子には内向き矢印が含まれ、その端子がそのフレームのデータソースであることを示します。シーケンスローカルを配線した最初のフレームよりも前にあるフレーム内では、シーケンスローカル端子を使用できません。

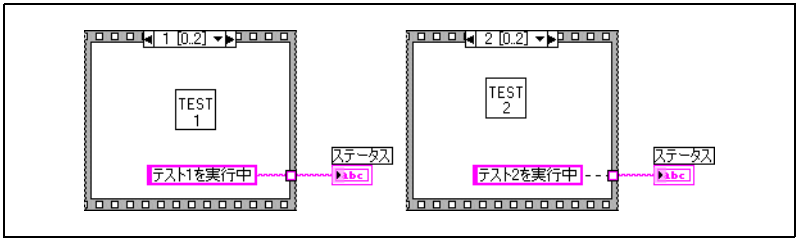
シーケンスストラクチャの使用例については、`examples\general\structs.llb` を参照してください。

シーケンスストラクチャの使いすぎを避ける

LabVIEW が持つ並列処理機能を活用するために、シーケンスストラクチャは使いすぎないでください。シーケンスストラクチャによって実行順序は保証されますが、並列処理が妨げられます。たとえば、PXI、GPIB、シリアルポート、DAQ デバイスなどの I/O デバイスを使用する非同期タスクは、シーケンスストラクチャによって妨げられない限り、他の動作と並行して実行できます。シーケンスストラクチャにより、ブロックダイアグラムの一部が隠され、また左から右への自然なデータフローが妨げられます。

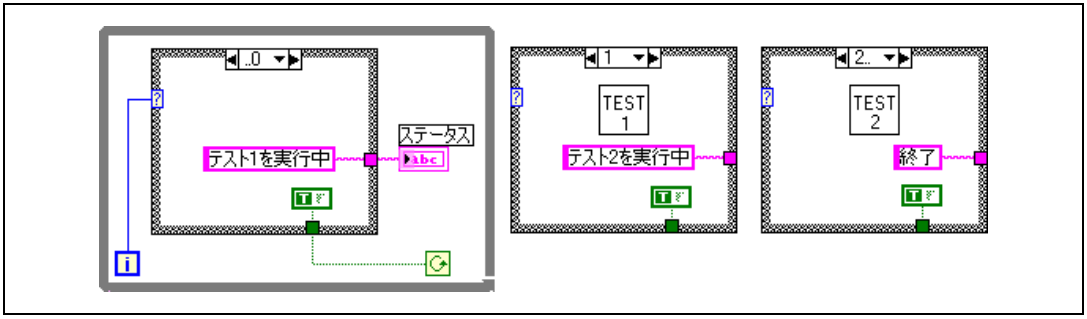
実行順序を制御する必要がある場合は、ノード間のデータ依存を確立することを検討してください。たとえば、エラー I/O を使用して、I/O 動作の実行順序を制御できます。エラー I/O の詳細については、第 6 章「VI の実行とデバッグ」の「エラー処理」のセクションを参照してください。

また、シーケンスストラクチャの異なるフレームから表示器を更新する場合には、シーケンスストラクチャを使用しないでください。たとえば、テストアプリケーションで使用される VI には、現在進行中のテストの名前を表示する**ステータス**表示器がある場合もあります。各テストが異なるフレームから呼び出されたサブ VI である場合、以下のブロックダイアグラムの壊れたワイヤに示されるように、各フレームから表示器を更新することはできません。



シーケンスストラクチャのすべてのフレームは、データがストラクチャから出て行く前に実行されるため、**ステータス**表示器に値を割り当てられるのは 1 つのフレームのみです。

代わりに、以下の例に示すように、Case ストラクチャと While ループを使用してください。



Case ストラクチャの各ケースは、シーケンスストラクチャのフレームと同じです。While ループの各繰り返しは、次のケースを実行します。**ステータス**表示器は、各ケースについて VI のステータスを表示します。データは、ケースが実行するたびにストラクチャから出て行くため、**ステータス**表示器は、対応するサブ VI を呼び出すケースの前のケースで更新されます。

シーケンスストラクチャの場合とは異なり、Case ストラクチャは、ケースの実行中に While ループを終了するためにデータを渡すことができます。たとえば、最初のテストの実行中にエラーが発生した場合、Case ストラクチャは、条件端子に FALSE を渡してループを終了することができます。ただし、シーケンスストラクチャの場合には、エラーが発生した場合でもすべてのフレームを実行する必要があります。

イベント駆動型プログラミング

LabVIEW は、データフローによってブロックダイアグラムの要素の実行順序が決まるデータフロー型プログラミング環境です。LabVIEW のイベントでは、ユーザによるフロントパネル上のオブジェクトとの直接対話およびブロックダイアグラムの要素の順序によってデータフローが決まります。

メモ　イベント駆動型のプログラミングは、LabVIEW 開発システムおよびプロフェッショナル開発システムでのみ使用できます。

イベントとは

イベントは、ユーザが実行するアクションによって発生します。たとえば、マウスをクリックするとマウスイベントが発生し、キーボードのキーを押すとキーボードイベントが発生します。イベント駆動型プログラムでは、プログラムはまずイベントの発生を待機し、イベントが発生するとそのイベントに対応して、その後、再び次のイベントを待機します。プログラムの反応は、そのイベントのコードによって決まります。イベント駆動

型プログラムの実行順序は、発生するイベントと、イベントの発生順序によって決まります。

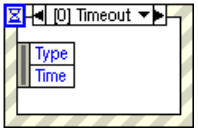
LabVIEW でイベントを使用する

イベントは、ユーザインタフェースを設計する上で非常に便利です。イベントを使用して、フロントパネルでのアクションとブロックダイアグラムの実行を同期させます。イベントを使用して、ユーザがいつ値を変更したか、フロントパネルを閉じたか、アプリケーションを終了したか、などを検知することができます。イベントストラクチャを使用して、イベント専用のコードを実行し、ユーザが実行したアクションを決定するために、ブロックダイアグラムがフロントパネルをポーリングする必要を減らすことができます。これによって処理に要する時間が短縮され、ブロックダイアグラムが簡略になります。

メモ　イベントは、フロントパネル上でのユーザの直接的な動作のみを待機します。VI サーバ、グローバル変数、ローカル変数などを使用して、VI またはフロントパネルのオブジェクトをプログラム的に変更してもイベントは動作しません。

イベントストラクチャを使用する際の注意および使用可能なイベントの詳細については、「LabVIEW ヘルプ」を参照してください。

イベントストラクチャ



左図に示すイベントストラクチャを使用して、アプリケーションでイベントを処理します。Case ストラクチャの場合と同様に、イベントストラクチャにも複数のケースを追加することができます。その後、1 つまたは複数のイベントを処理するように各ケースを構成できます。このようなイベントが発生すると、LabVIEW は対応するケースを実行します。終了条件が発生するまで一連のイベントを処理するには、イベントストラクチャを While ループ内に配置します。

注意　イベントストラクチャを While ループ内に配置して、終了条件で While ループが停止した場合、イベントの生成を継続することはできますが、イベントストラクチャはイベントを実行および処理しません。これによってインタフェースがロックする可能性があります。

メモ　LabVIEW 開発システムまたはプロフェッショナル開発システムを使用して、イベントストラクチャを含む VI を作成した場合、その VI を LabVIEW 基本パッケージで実行することができます。この場合も、イベントストラクチャの機能は失われません。ただし、LabVIEW 基本パッケージでイベントストラクチャを編集することはできません。

左図のように、イベントストラクチャの左上に Timeout 端子を使用して、イベントストラクチャがイベントの発生を待機する時間（ミリ秒）を指定します。デフォルトは -1 で、これは無限に待機することを示します。イベントが発生する前にタイムアウトした場合、タイムアウトイベントを処理するようにケースを構成してあれば、LabVIEW はタイムアウトイベントを生成します。

左図に示す Event Data Node は、Unbundle By Name 関数に外見が似ています。このノードは各イベントケースの右側と左側の枠の内側に表示されます。このノードは横方向にサイズ変更したり、Unbundle By Name 関数の場合と同様に、ノードの各要素を任意の Event Data Field にアクセスするように設定できます。ノードは、イベントストラクチャの各ケースがどのイベントを処理するように構成されているかによって、異なるデータを表示します。ある 1 つのケースが複数のイベントを処理するように構成した場合、処理されたすべてのイベントタイプに共通するデータのみを使用することができます。

左図に示すイベントストラクチャの上部のイベントセレクタラベルには、現在選択されているケースが、そのケースで処理されるイベントを含めて表示されます。

メモ　Case ストラクチャの場合と同様に、イベントストラクチャでもトンネルを使用することができます。ただし、デフォルトでは、各ケースに対してイベントストラクチャの出力トンネルを配線する必要はありません。未配線のすべてのトンネルは、そのトンネルデータタイプのデフォルト値を使用します。トンネルを右クリックして、ショートカットメニューで**未配線の場合はデフォルトを使用**を選択解除すると、すべてのケースでトンネルを配線する必要があるデフォルトの Case ストラクチャの動作に戻ります。

イベントは、アプリケーション、VI、および制御器などのクラスに分類されます。イベントデータには常にそのイベントをトリガしたオブジェクトへのリファレンスが含まれます。ある 1 つのケースが異なる VI サーバクラスの制御器の複数のイベントを処理する場合、イベントソース制御リファレンスは、すべてのクラスに共通の親クラスとなります。たとえば、イベントストラクチャの 1 つのケースがデジタル数値制御器およびカラーランプ制御器のイベントを処理するように構成した場合、デジタル数値とカラーランプ制御器は数値クラスであるため、イベントソース制御リファレンスは、数値となります。

イベントを構成する

イベントストラクチャの枠を右クリックして、ショートカットメニューから**このケースで処理されるイベントを編集**を選択すると、**イベントを編集**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を使用して、1 つまたは複数のケースを編集することができます。

フィルタおよび通知イベント

通知とフィルタの 2 種類のイベントがあります。通知イベントは、LabVIEW に対して、ユーザが制御器の値を変更した場合など、ユーザのアクションがすでに発生していることを知らせます。たとえば、ユーザがフロントパネルのボタンをクリックしたときに、ブロックダイアグラムのイベントストラクチャに通知する VI を作成することができます。ユーザがボタンをクリックすると値が変更されるため、イベントストラクチャに通知されます。イベントストラクチャは、複数の通知イベントを一度に処理することができます。ある 1 つのケースが複数の通知イベントを処理するように構成した場合、処理されたすべてのイベントタイプに共通するデータのみが利用可能になります。

フィルタイベントを使用して、インタフェースの動作を制御することができます。また、フィルタイベントを使用して、ユーザインタフェースが処理する前にイベントデータを検証したり、変更することや、イベントデータ全体を破棄して、VI に変更が適用されないようにすることができます。たとえば、ユーザが VI のフロントパネルを対話式に閉じることを許可しないように Menu Selection イベントを構成することができます。また、Key Down イベントを構成して、文字列制御器に入力されたすべての文字を大文字に変更したり、不要な文字をフィルタするようにイベントを変更することができます。

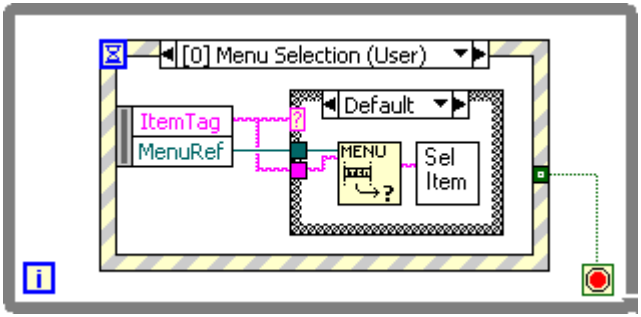
Discard 端子など、イベントストラクチャの右側の端子によって、フィルタイベントを識別することができます。

メモ　通知イベントの場合とは異なり、イベントストラクチャの 1 つのケースが複数のフィルタイベントを処理できるのは、イベントが返すデータが同一の場合のみです。Key Down と Mouse Down など、同一のデータを返さない 2 つのイベントを 1 つのグループに分類しようとすると、VI は壊れます。

イベントの例

以下に、Menu Selection (User) イベントを使用して構成されたイベントストラクチャの例を示します。このイベントは、`sample.rtm` という名前のユーザ定義メニューを使用して作成されたメニュー選択を捕捉します。**itemTag** は、どのメニュー項目が選択されているかを返し、

MenuRef はメニューバーへの refnum を返します。イベントのその他の使用例については、`examples\general\uievents.llb` を参照してください。



メモ　イベントストラクチャがメニューイベントを処理するように構成し、同じメニュー項目に対して Get Menu Selection 関数を構成した場合、イベントストラクチャが優先され、LabVIEW は Get Menu Selection 関数を無視します。すべての VI で、イベントストラクチャまたは Get Menu Selection 関数を使用する必要があります。

9

文字列、配列、およびクラスタを使用してデータをグループ化する

データをグループ化するには、文字列、配列、およびクラスタを使用します。文字列は一連の ASCII 文字をグループ化します。配列は同じタイプのデータ要素をグループ化します。クラスタは混合タイプのデータ要素をグループ化します。

詳細については

文字列、配列、およびクラスタを使用してデータをグループ化する方法の詳細については、「LabVIEW ヘルプ」を参照してください。

文字列

文字列とは、表示可能または表示不可能な一連の ASCII 文字です。文字列は、プラットフォームに異存しない情報およびデータのフォーマットを提供します。一般的な文字列の使用例としては以下のようなものがあります。

- 単純なテキストメッセージの作成する。
- 数値データを文字列として計測器に渡し、その後文字列を数値に変換する。
- 数値データをディスクに保存する。数値を ASCII ファイルに保存するには、数値をディスクファイルに書き込む前に、まず数値を文字列に変換する必要があります。
- ダイアログボックスでユーザに指示を出す。

フロントパネルでは、文字列は表、テキスト入力ボックス、およびラベルとして表示されます。ブロックダイアグラムの文字列関数を使用して文字列を編集および操作します。ワードプロセッサや表計算アプリケーションで使用するために、または他の VI や関数で使用するために文字列をフォーマットします。

フロントパネルの文字列

制御器→文字列 & パスパレットの文字列制御器および表示器を使用して、テキスト入力ボックスやラベルをシミュレートします。文字列制御器および表示器の詳細については、第 4 章「フロントパネルを作成する」の「文字列制御器および表示器」のセクションを参照してください。

文字列表示タイプ

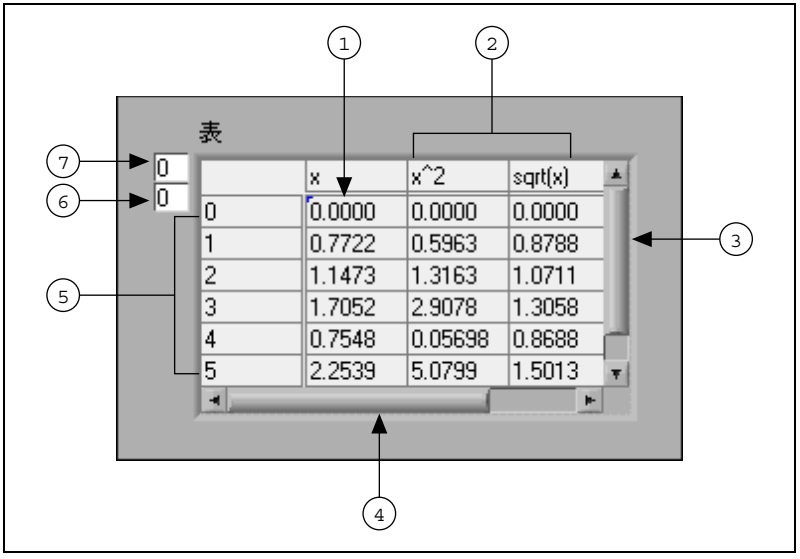
フロントパネルで文字列制御器または表示器を右クリックして、表 9-1 に示された表示タイプから選択します。表には各表示タイプのサンプルメッセージも表示されます。

表 9-1　: 文字列表示タイプ

表示タイプ	説明	メッセージ
通常の表示	制御器のフォントを使用して印刷可能な文字を表示します。印刷できない文字はボックスで表示します。	4 つの表示タイプがあります。 「¥」は円記号です。
¥（円）コード表示	印刷できないすべての文字を円コードで表示します。	`There¥sare¥sfour¥sdisplay¥stypes.¥n¥¥¥sis¥sa¥yen¥smark.`
パスワード表示	スペースを含む各文字をアスタリスク（*）で表示します。	`**************************** ****************`
16 進表示	各文字の ASCII 値を文字ではなく 16 進数で表示します。	`5468 6572 6520 6172 6520 666F 7572 2064 6973 706C 6179 2074 7970 6573 2E0A 5C20 6973 2061 2062 6163 6B73 6C61 7368 2E`

表

制御器→リスト & 表パレットの表制御器を使用して、フロントパネルに表を作成します。表の各セルは文字列であり、各セルは列と行で指定されます。したがって、表は 2 次元の文字列を表示します。図 9-1 は、表および表の各部を示します。配列の詳細については、この章の「配列」のセクションを参照してください。



1　指標値で示されるセル	5　行見出し
2　列見出し	6　水平指標
3　横スクロールバー	7　垂直指標
4　縦スクロールバー	

図 9-1　表の各部

文字列をプログラム的に編集する

文字列を編集するには、以下の手順に従って、**関数→文字列**パレットにある文字列関数を使用します。

- 文字列内の文字またはサブ文字列を検索、取り出し、および置換する。
- 文字列内のすべてのテキストを大文字または小文字に変更する。
- 文字列内の一致パターンを検索して取り出す。
- 文字列から 1 行を取り出す。
- 文字列内のテキストを回転または逆にする。
- 複数の文字列を連結する。
- 文字列から文字を削除する。

文字列関数を使用して文字列を編集する方法のサンプルは、`examples\general\strings.llb` にあります。

文字列をフォーマットする

他の VI、関数、またはアプリケーションでデータを使用する場合、データを文字列に変換した後、文字列を VI、関数、またはアプリケーションが読める形式にフォーマットする必要がある場合が頻繁にあります。たとえば、Microsoft Excel の場合、数字や語をセルに区切るために区切り文字を使用するため、区切り文字を含む文字列を使用する必要があります。

たとえば、Write File 関数を使用して数値の 1 次元配列をスプレッドシートに書き込む場合、配列を文字列にフォーマットし、各数値をタブなどの区切り文字で分割する必要があります。また、Write to Spreadsheet File VI を使用して数値の配列をスプレッドシートに書き込むには、Array to Spreadsheet String 関数を使用して配列をフォーマットし、フォーマットと区切り文字を指定する必要があります。

以下のタスクを実行するには、**関数→文字列**パレットにある文字列関数を使用します。

- 複数の文字列を連結する。
- 文字列から文字列のサブセットを抽出する。
- データを文字列に変換する。
- ワープロや表計算アプリケーションで使用するために文字列をフォーマットする。

文字列をテキストおよびスプレッドシートファイルに保存するには、**関数→ファイル I/O** パレットにある File I/O VI および関数を使用します。

指示子をフォーマットする

多くの場合、文字列をフォーマットするために、1 つまたは複数の指示子を文字列関数の**形式文字列**パラメータに入力する必要があります。フォーマット指示子コードは、文字列をデータに、またはデータを文字列に変換する方法を示します。LabVIEW は、変換コードを使用して、テキストフォーマットのパラメータを決定します。たとえば、フォーマット指示子 `%x` は、16 進整数を文字列に、文字列を 16 進整数に変換します。

Format Into String 関数および Scan From String 関数は、**形式文字列**パラメータに複数のフォーマット指示子（拡張可能な関数に対する各入力または出力に 1 つずつ）を使用することができます。

Array To Spreadsheet String 関数および Spreadsheet String To Array 関数は、変換する入力が 1 つしかないため、**形式文字列**パラメータに 1 つのフォーマット指定子しか使用しません。LabVIEW は、これらの関数にユーザが挿入した余分な指定子を特別な意味のない文字列として扱います。

数値と文字列

文字列データは ASCII 文字であり、数値データは ASCII 文字ではないという点で、数値データと文字列データは異なります。テキストファイルとスプレッドシートファイルには文字列のみが使用できます。テキストファイルまたはスプレッドシートファイルに数値データを書き込むには、まず数値データを文字列に 変換する必要があります。

数値のセットを既存の文字列に追加するには、数値データを文字列に変換し、Concatenate Strings 関数または他の文字列関数を使用して、新しい文字列を既存の文字列に追加します。**関数→文字列→文字列 / 数値変換**パレットにある文字列 / 数値変換関数を使用して、数値を文字列に変換します。

文字列にはグラフまたはチャートに表示する数値のセットを含むこともできます。たとえば、チャートにプロットする数値のセットを含むテキストファイルを読み込むことができます。ただし、これらの数値は ASCII テキストであるため、まず数値を文字列として読み込んでから、数値をチャートにプロットする前に、その文字列を数値のセットに変換する必要があります。

図 9-2 は、数値のセットを含み、文字列を数値に変換し、数値の配列を作成し、数値をグラフにプロットする文字列を示します。

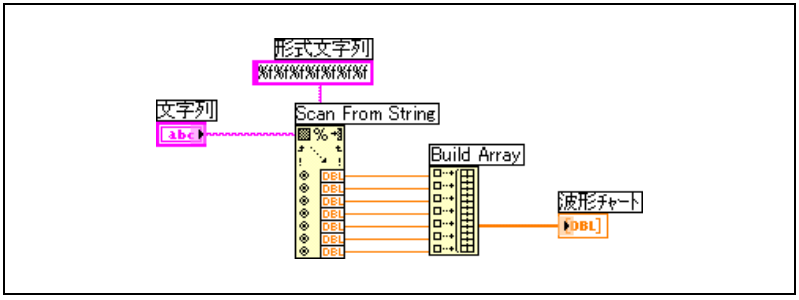


図 9-2　文字列を数値に変換する

データタイプを XML に変換する

拡張マークアップ言語（XML）は、データを説明するためにタグを使用するフォーマット規約です。HTML タグとは異なり、XML タグはブラウザに対して 1 つのデータをフォーマットする方法を指定しません。その代わりに、XML タグは 1 つのデータを識別します。

たとえば、Web で書籍を販売している書店の場合に、各書籍を以下の基準でライブラリに分類する必要があるとします。

- 本の種類（フィクションまたはノンフィクション）
- タイトル
- 著者
- 出版社
- 価格
- ジャンル
- あらすじ
- ページ数

各書籍に対して、XML ファイルを作成することができます。『Touring Germany's Great Cathedrals』というタイトルの本の XML ファイルは、以下のようなものになります。

```
<nonfiction>
<Title>Touring Germany's Great Cathedrals</Title>
<Author>Tony Walters</Author>
<Publisher>Douglas Drive Publishing</Publisher>
<PriceUS>$29.99</PriceUS>
<Genre>Travel</Genre>
<Genre>Architecture</Genre>
<Genre>History</Genre>
<Synopsis>This book fully illustrates twelve of Germany's
most inspiring cathedrals with full-color photographs,
scaled cross-sections, and time lines of their
construction.</Synopsis>
<Pages>224</Pages>
</nonfiction>
```

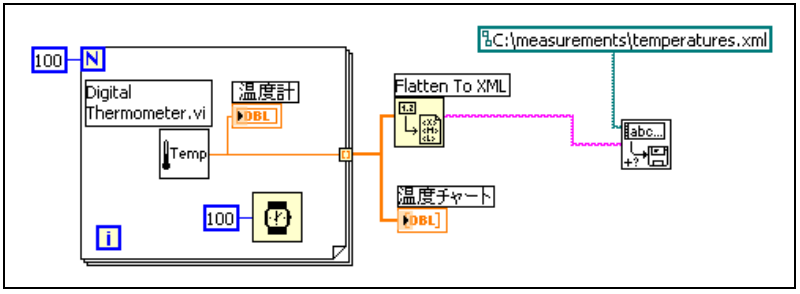
同様に、LabVIEW のデータを名前、値、およびタイプで分類することができます。ユーザ名の文字列制御器を XML で以下のように表現することができます。

```
<String>
<Name>User Name</Name>
<Val>Reggie Harmon</Val>
</String>
```

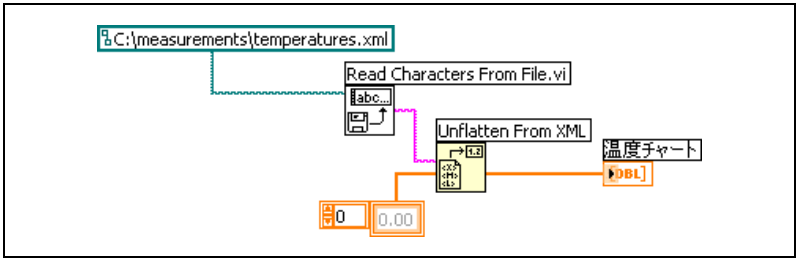
XML ベースのデータタイプ

LabVIEW のデータを XML に変換すると、データをファイルに保存したときに、データを説明するタグからデータの値、名前、およびタイプを簡単に識別できるように、データがフォーマットされます。たとえば、温度の値の配列を XML に変換して、データをテキストファイルに保存すると、各温度を識別する `<Value>` タグを見つけることによって温度を簡単に識別することができます。

関数→上級→データ操作パレットにある Flatten to XML 関数を使用して、LabVIEW データタイプを XML フォーマットに変換します。以下の例は、シミュレーションされた温度を 100 個生成し、温度の配列をチャートにプロットした後、数字の配列を XML フォーマットに変換して XML データを `temperatures.xml` ファイルに書き込みます。



関数→上級→データ操作パレットにある Unflatten from XML 関数を使用して、XML フォーマットのデータタイプを LabVIEW データに変換します。以下の例は、`temperatures.xml` ファイルの 100 個の温度を読み込んで、温度の配列をチャートにプロットします。



メモ　LabVIEW Variant データを XML に平坦化することができますが、変数データを XML から非平坦化しようとすると、空の LabVIEW 変数が生成されます。

LabVIEW XML スキーマ

LabVIEW は、確立された XML スキーマにデータを変換します。現在、カスタマイズスキーマを作成したり、各データに対して LabVIEW がタグを付ける方法を制御するとはできません。また、VI または関数全体を XML に変換することもできません。

データを配列やクラスタとグループ化する

データをグループ化するには、**制御器→配列とクラスタ**、**関数→配列**、および**関数→クラスタ**パレットにある配列およびクラスタの制御器と関数を使用します。配列は同じタイプのデータ要素をグループ化します。クラスタは混合タイプのデータ要素をグループ化します。

配列

配列は要素と次元で構成されています。要素は配列を構成するデータです。次元は、配列の長さ、高さ、または深さです。配列は、1 以上の次元を持ち、メモリ容量が許すかぎり、一次元あたり $2^{31}-1$ の要素を持つことができます。

数値、ブール値、パス、文字列、波形、およびクラスタデータタイプの配列を作成できます。同じタイプのデータの集合を処理する場合や同じ演算を繰り返し行う場合は、配列を使用することを検討して下さい。配列は、波形から収集したデータや、ループで生成されたデータ (各ループで配列の要素が 1 つずつ生成されます) を格納するのに適しています。

配列の配列は作成できません。しかし、複数次元の配列を使用したり、各クラスタが 1 つ以上の配列を含んでいるクラスタの配列を作成できます。配列に使用できる要素のタイプの詳細については、この章の「配列に関する制約」のセクションを参照してください。クラスタの詳細については、「クラスタ」のセクションを参照してください。

指標

配列内の特定の要素をみつけるには、次元ごとに 1 つの指標が必要です。LabVIEW では、指標を使用すると、配列内を移動してブロックダイアグラム上の配列から要素、行、列およびページを取り出します。

配列の例

簡単な配列の例としては、太陽系の 9 つの惑星をリストしたテキストの配列があります。LabVIEW は、これを 9 つの要素を持つ文字列の 1 次元配列として表現します。

配列の要素には順番が付けられます。配列の指標を使用して、すべての特定の要素にすばやくアクセスできます。指標は 0 から始まります。つまり、指標は 0 ～ *n*–1 の範囲になります。ここで、*n* は配列内の要素の数です。9 つの惑星の例では *n* = 9 で、指標は 0 ～ 8 の範囲になります。地球は 3 番目の惑星なので、指標は 2 になります。

配列のもう 1 つの例として、図 9-3 のように、連続した各要素が連続した時間間隔の電圧値を表す数値配列として表現された波形があります。

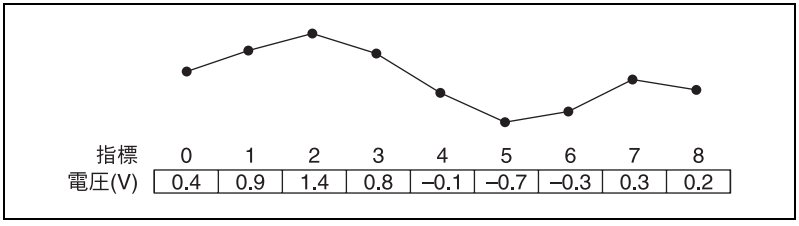


図 9-3　数値配列の波形

より複雑な配列の例として、図 9-4 のように、点の配列として表現されたグラフがありますが、ここで各点は X および Y 座標を表す 2 つの数値を含むクラスタになっています。

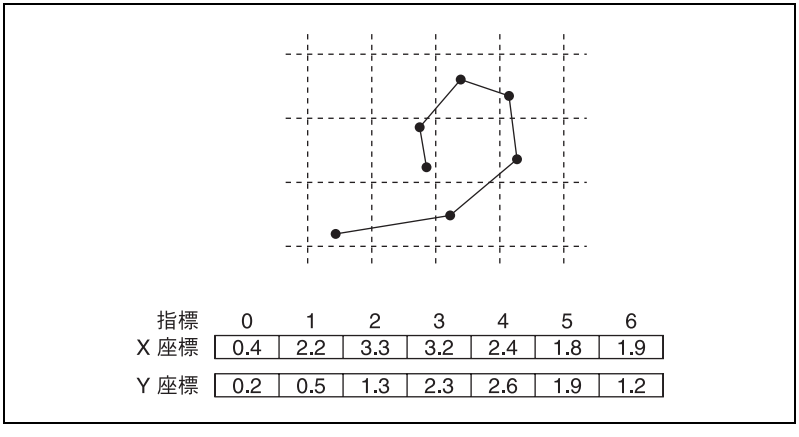


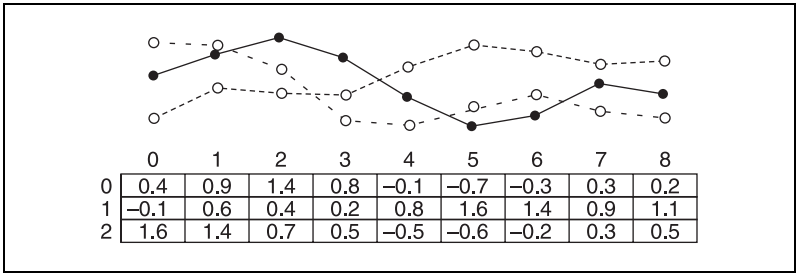
図 9-4　点の配列のグラフ

前の例は 1 次元配列を使用しています。2 次元配列は要素をグリッドに格納します。要素を見つけるには（0 から始まる）列指標と行指標が必要です。図 9-5 は 6 列× 4 行の 2 次元配列を示しており、6 × 4 = 24 の要素を含んでいます。

図 9-5　6 列 × 4 行の 2 次元配列

たとえば、チェス盤には 8 列× 8 行の総計 64 の位置があります。各位置は空かチェスの駒が 1 つあるかのどちらかです。チェス盤は文字列の 2 次元配列として表現できます。各文字列は、盤上の対応する位置に置かれた駒の名前か、またはその場所に駒がない場合は空の文字列です。

配列に行を追加することによって、図 9-3 および図 9-4 の 1 次元配列の例を 2 次元に一般化できます。図 9-6 は、数値の 2 次元配列として表現された波形の集合を示しています。行指標で波形を選択し、列指標で波形上の点を選択します。



	0	1	2	3	4	5	6	7	8
0	0.4	0.9	1.4	0.8	–0.1	–0.7	–0.3	0.3	0.2
1	–0.1	0.6	0.4	0.2	0.8	1.6	1.4	0.9	1.1
2	1.6	1.4	0.7	0.5	–0.5	–0.6	–0.2	0.3	0.5

図 9-6　数値の 2 次元配列内の複数波形

その他の配列の例については、`examples\general\arrays.llb` を参照してください。

配列に関する制約

以下の例外を除いて、ほとんどすべてのデータタイプの配列を作成できます。

- 配列の配列は作成できません。しかし、複数次元の配列を使用したり、Build Cluster Array 関数を使用して、各クラスタが 1 つ以上の配列を含んでいるクラスタの配列を作成することができます。
- グラフは配列データタイプで、配列は他の配列を含むことができないため、XY グラフ以外のグラフの配列を作成することはできません。しかし、グラフがクラスタ内にある場合には、XY グラフ以外のグラフの配列を作成することもできます。
- チャートの配列は作成することができません。

配列制御器、表示器、および定数を作成する

図 9-7 のようにフロントパネルに配列シェルを配置し、データオブジェクトまたは要素（数値、ブール値、文字列、パス、refnum、クラスタ制御器、またはクラスタ表示器など）を配列シェル内にドラッグして、フロントパネル上に配列制御器または表示器を作成します。

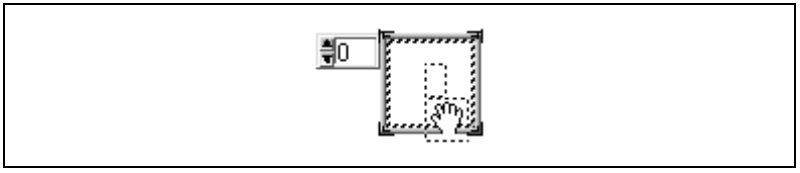


図 9-7　配列シェル

配列シェルは、小さなブール制御器でも大きな 3 次元グラフでも、新しいオブジェクトに合わせて自動的にサイズを変更します。

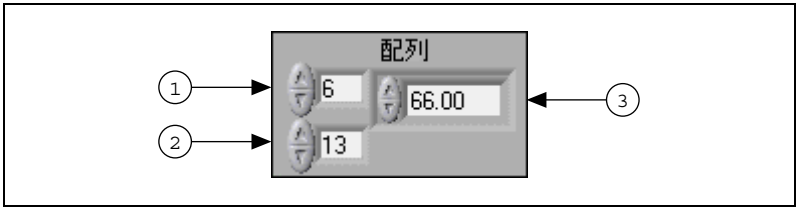
フロントパネル上に特定の要素を表示するには、指標表示に指標番号を入力するか、指標表示上の矢印を使用して、該当する番号にナビゲートします。

ブロックダイアグラム上に配列定数を作成するには、**関数→配列→配列定数**を選択してフロントパネルに配列シェルを置き、次に文字列定数、数値定数、またはクラスタ定数を配列シェル内に置きます。配列定数は、他の配列と比較する基準として使用できます。

配列指標表示

2 次元配列には行と列があります。図 9-8 のように、左側の 2 つのボックスの上の表示は行指標で、下の表示は列指標です。行と列の表示の右側に

ある表示は、指定された位置の値を示しています。図 9-8 は、6 行目および 13 列目の値が 66 であることを示しています。



1　行　　2　列　　3　行、列での値

図 9-8　配列制御器

行と列は 0 から始まり、1 番目の列は 0、2 番目の列は 1、以下同様に続きます。以下の配列の指標表示を行 1 および列 2 に変更すると、値 6 が表示されます。

0	1	2	3
4	5	6	7
8	9	10	11

配列次元の範囲を外れた列や行を表示しようとすると、配列制御器は淡色表示され、定義された値がないことを示し、データタイプのデフォルト値が表示されます。データタイプのデフォルト値は配列のデータタイプによって異なります。

一度に複数の行または列を示すには位置決めツールを使用します。

配列関数

以下のような配列を作成して操作するタスクでは、**関数→配列**パレットにある配列関数を使用します。

- 配列から個々のデータ要素を抽出する。
- 配列内のデータ要素を挿入、削除、または置き換える。
- 配列を分割する。

配列関数を自動的にサイズ変更する

Index Array、Replace Array Subset、Insert Into Array、Delete From Array、および Array Subset 関数は、配線された入力配列の次元に合わせて自動的にサイズ変更されます。たとえば、1 次元配列をこれらの関数の 1 つに配線すると、その関数は 1 つの指標入力を示します。2 次元配列

を同じ関数に配線すると、その関数は、行と列に対してそれぞれ 1 つずつ、合計 2 つの指標入力を示します

位置決めツールを使用して、手動で関数をサイズ変更することにより、これらの関数で複数の要素、つまりサブ配列 (行、列、またはページ) にアクセスすることができます。これらの関数の 1 つを拡張するとき、これらの関数は、その関数に配線された配列の次元によって決められた増分だけ拡張します。1 次元配列をこれらの関数の 1 つに配線すると、この関数は 1 つの指標入力分拡張します。2 次元配列を同じ関数に配線すると、関数は、行と列にそれぞれ 1 つずつ、合計 2 つの指標入力を拡張します。

配線する指標入力によって、アクセスまたは変更するサブ配列の形状が決まります。たとえば、Index Array 関数への入力が 2 次元配列であり、**行**入力にのみ配線した場合、配列の 1 次元行すべてが抽出されます。**列**入力にのみ配線すると、配列の 1 次元列すべてを取り出します。**行**入力と列入力を配線した場合は、配列の 1 つの要素が抽出されます。入力グループはそれぞれ独立しており、配列のすべての次元のどの部分にもアクセスできます。

図 9-9 のようなブロックダイアグラムでは、Index Array 関数を使用して 2 次元配列から行と要素を取り出します。

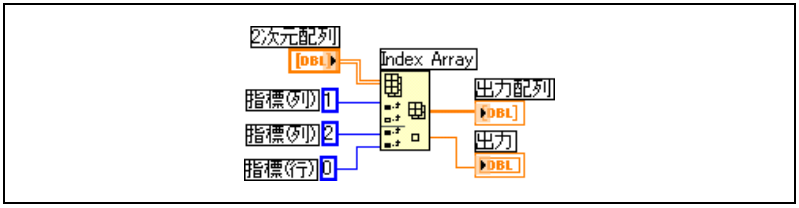


図 9-9　2 次元配列に指標を付ける

配列内の複数の連続した値にアクセスするには、Index Array 関数を拡張します。ただし、増分ごとに値を指標入力に配線しないでください。たとえば、2 次元配列から 1 番目、2 番目、および 3 番目の行を取り出すには、Index Array 関数を 3 回の増加で拡張し、1 次元配列表示器を各サブ配列出力に配線します。

クラスタ

電話ケーブル内の束線のような混合タイプのクラスタグループデータ要素では、ケーブル内の各ワイヤはクラスタの異なる要素を表します。クラスタは、テキストベースのプログラミング言語におけるレコードまたは構造体に似ています。

いくつかのデータ要素をクラスタにまとめることによって、ブロックダイアグラム上のワイヤの混雑を取り除き、サブ VI に必要なコネクタペーン

端子の数を減らします。コネクタペーンには最大 28 個の端子があります。プログラムで使用する制御器および表示器がフロントパネル上に 29 以上ある場合は、制御器および表示器のいくつかを 1 つのクラスタにグループ化して、このクラスタをコネクタペーン上の端子に割り当てます。

クラスタおよび配列要素には両方とも順番が付いていますが、1 つずつ要素に指標を付けるのではなく、すべてのクラスタ要素を一度にアンバンドルする必要があります。また、Unbundle By Name 関数を使用して特定のクラスタ要素にアクセスすることもできます。クラスタは、固定サイズであるという点でも配列と異なります。配列の場合と同様に、クラスタは制御器または表示器のいずれかです。クラスタでは、制御器と表示器を併用することはできません。

ブロックダイアグラム上のほとんどのクラスタは、ピンク色の配線パターンとデータタイプアイコンを持っています。数値のクラスタは (ポイントと呼ぶこともあります)、茶色の配線パターンとデータタイプアイコンを持っています。茶色の数値クラスタを Add や Square Root のような数値関数に配線し、クラスタのすべての要素に対して同じ演算を同時に実行することができます。

クラスタ要素には、その要素のシェル内の位置に無関係の論理的順序があります。クラスタに配置する最初のオブジェクトは要素 0、2 番目のオブジェクトが要素 1 などのようになります。要素を削除すると、この順序は自動的に調整されます。クラスタの順序は、その要素がブロックダイアグラムの Bundle 関数と Unbundle 関数の端子として表示される順序を決定します。クラスタの枠を右クリックして、ショートカットメニューから**クラスタ内制御器の並べ替え**を選択して、クラスタの順序を表示および変更することができます。

クラスタを配線する場合、両方のクラスタの要素数が同じになる必要があります。対応する要素はクラスタ順で決まり、データタイプも適合している必要があります。たとえば、1 つのクラスタの倍精度浮動小数点数値がクラスタ順で他のクラスタの文字列に対応する場合、ブロックダイアグラム上の配線は破線で表示され、VI は実行されません。異なる表現の数値の場合、LabVIEW はこれらの数値を同じ表現に強制的に変換します。数値変換の詳細については、付録 B「多形性関数」の「数値変換」のセクションを参照してください。

以下のようなクラスタを作成して操作するタスクでは、**関数→クラスタ**パレットにあるクラスタ関数を使用します。

- クラスタから個々のデータ要素を抽出する。
- クラスタに個々のデータ要素を追加する。
- クラスタを個々のデータ要素に分割する。

10

ローカルおよびグローバル変数

LabVIEW では、ブロックダイアグラム端子を使用して、フロントパネルオブジェクトとの間でデータを読み書きします。しかし、フロントパネルオブジェクトにはブロックダイアグラム端子が 1 つしかないので、アプリケーションは、複数の位置からこの端子内のデータにアクセスする必要があります。

ローカルおよびグローバル変数は、ワイヤで配線できないアプリケーションの位置の間で情報をやり取りします。1 つの VI の複数の位置からフロントパネルオブジェクトにアクセスするにはローカル変数を使用します。複数の VI 間でアクセスし、データをやり取りするにはグローバル変数を使用します。

詳細については

ローカルおよびグローバル変数の使用方法の詳細については、「LabVIEW ヘルプ」を参照してください。

ローカル変数

1 つの VI の複数の位置からフロントパネルオブジェクトにアクセスし、ワイヤで配線できないブロックダイアグラムストラクチャ間でデータをやり取りするには、ローカル変数を使用します。

ローカル変数を使用すると、フロントパネルの制御器や表示器との間でデータの読み書きができます。ローカル変数への書き込みは、他の端子へデータを渡すのに似ています。しかし、ローカル変数を使用すると、データを制御器に書き込んだり、表示器からデータを読み取ることができます。ローカル変数を使用すると、事実上、入力と出力の両方としてフロントパネルオブジェクトにアクセスできます。

たとえば、ログインするのにパスワードなどを入力する必要がある場合、新しいユーザがログインするたびに**ログイン**および**パスワード**プロンプトを入力しなおしてもよいのですが、ローカル変数を使用すると、ログイン

するときは**ログイン**および**パスワード**文字列制御器から読み取り、ログアウトするときはこれらの制御器に空の文字列を書き込みます。

ローカル変数を作成する

既存のフロントパネルオブジェクトまたはブロックダイアグラム端子を右クリックし、ショートカットメニューから**作成→ローカル変数**を選択して、ローカル変数を作成します。そのオブジェクトのローカル変数がブロックダイアグラムに表示されます。

また、**関数→ストラクチャ→ローカル変数**と選択して、ローカル変数をブロックダイアグラムに配置することもできます。左図に示すローカル変数ノードは、制御器または表示器に関連付けられていません。ローカル変数ノードを右クリックして、**選択項目**ショートカットメニューからフロントパネルオブジェクトを選択します。ショートカットメニューに所有ラベルを持つすべてのフロントパネルオブジェクトのリストが表示されます。

LabVIEW は、所有ラベルを使用してローカル変数をフロントパネルオブジェクトと関連付けるので、フロントパネルの制御器や表示器には、わかりやすい所有ラベルを付けてください。所有ラベルとフリーラベルの詳細については、第 4 章「フロントパネルを作成する」の「ラベルを付ける」のセクションを参照してください。

グローバル変数

同時に実行される複数の VI 間でアクセスしてデータをやり取りするにはグローバル変数を使用します。グローバル変数は組み込み LabVIEW オブジェクトです。グローバル変数を作成すると、フロントパネルだけがあってブロックダイアグラムのない特殊なグローバル VI が自動的に作成されます。制御器と表示器をグローバル VI のフロントパネルに追加し、VI が含むグローバル変数のデータタイプを定義します。事実上、このフロントパネルはいくつかの VI がデータにアクセスできるコンテナです。

たとえば、同時に 2 つの VI が実行中であると想定します。各 VI は While ループを含んでおり、データ点を波形チャートに書き込みます。最初の VI には、両方の VI を終了させるためのブール制御器が含まれています。1 つのブール制御器で両方のループを終了させるには、グローバル変数を使う必要があります。両方のループが同じ VI 内の 1 つのブロックダイアグラム上にあると想定した場合は、ローカル変数を使用してループを終了できます。

グローバル変数を作成する

関数→ストラクチャ→グローバル変数を選択して、左図に示すグローバル変数をブロックダイアグラムに配置します。グローバル変数ノードをダブルクリックして、グローバル VI のフロントパネルを表示します。標準フロントパネルの場合と同様に、このフロントパネル上に制御器と表示器を置きます。

LabVIEW は、所有ラベルを使用してグローバル変数を識別するので、フロントパネルの制御器や表示器には、わかりやすい所有ラベルを付けてください。所有ラベルとフリーラベルの詳細については、第 4 章「フロントパネルを作成する」の「ラベルを付ける」のセクションを参照してください。

それぞれが 1 つのフロントパネルオブジェクトを持つ複数の単一グローバル VI を作成するか、または複数のフロントパネルオブジェクトを持つ 1 つのグローバル VI を作成できます。複数オブジェクトを持つグローバル VI は、関連する変数をグループ化できるため、より効率的です。VI のブロックダイアグラムは、グローバル VI のフロントパネル上の制御器や表示器と関連付けられたいくつかのグローバル変数ノードを含むことができます。これらのグローバル変数ノードは、グローバル VI のブロックダイアグラム上に置いた最初のグローバル変数ノードのコピーか、または現在の VI 上に置いたグローバル VI のグローバル変数ノードのいずれかです。サブ VI を他の VI 上に置く場合と同じやり方で、グローバル VI を他の VI 上に置きます。ブロックダイアグラム上に新しいグローバル変数ノードを置くたびに、そのグローバル変数ノードおよびそのコピーにのみ関連付けられた新しい VI が作成されます。サブ VI の詳細については、第 7 章「VI およびサブ VI を作成する」の「サブ VI」のセクションを参照してください。

グローバル VI のフロントパネル上にオブジェクトを置いた後、保存して、元の VI のブロックダイアグラムに戻ります。次に、アクセスするグローバル VI のオブジェクトを選択する必要があります。グローバル変数ノードを右クリックして、**選択項目**ショートカットメニューからフロントパネルオブジェクトを選択します。ショートカットメニューに所有ラベルを持つすべてのフロントパネルオブジェクトのリストが表示されます。

変数の読み書き

ローカル変数またはグローバル変数を作成すると、変数とのデータの読み書きが可能になります。デフォルトで、新しい変数はデータを受け取ります。この種の変数は表示器として機能し、書き込みローカルまたはグローバルとなります。新しいデータをローカルまたはグローバル変数に書き込むと、関連付けられているフロントパネルの制御器や表示器は新しいデータに更新されます。

また、変数は、データソースあるいは読み取りローカルまたはグローバルとして動作するようにも構成できます。変数を右クリックして、ショートカットメニューから**読み取りに変更**を選択し、変数を制御器として動作するように構成します。このノードを実行すると、VI は関連付けられているフロントパネルの制御器または表示器のデータを読み取ります。

データを与えるのではなくブロックダイアグラムからデータを受け取るように変数を変更するには、変数を右クリックして、ショートカットメニューから**書き込みに変更**を選択します。

ブロックダイアグラムでは、制御器と表示器の場合と同様に、読み取りローカルまたはグローバルと書き込みローカルまたはグローバルを見分けることができます。読み取りローカルまたはグローバルの境界線は、制御器と同じ太い線です。書き込みローカルまたはグローバルの境界線は、表示器と同じ細い線です。

ローカルおよびグローバル変数の使用方法の例については、`examples\general\locals.llb` および `examples\general\globals.llb` を参照してください。

ローカルおよびグローバル変数を慎重に使用する

ローカルおよびグローバル変数は高度な LabVIEW の概念です。これらの変数は本来、LabVIEW データフロー実行モデルの一部ではありません。これらの変数を使用すると、ブロックダイアグラムでの読み取りが難しくなる可能性があるため、変数は慎重に使用してください。ローカル変数やグローバル変数をコネクタペーンの代わりに使用したり、シーケンスストラクチャの各フレームの値にアクセスするために使用するなど、これらの変数の使用法を誤ると、VI が予期しない動作をする可能性があります。ブロックダイアグラム上に長いワイヤを配線するのを避けるためや、データフローの代わりに使用するなど、ローカル変数やグローバル変数を多用しすぎると、性能が低下します。LabVIEW データフロー実行モデルの詳細については、第 5 章「ブロックダイアグラムを作成する」の「ブロックダイアグラムのデータフロー」のセクションを参照してください。

ローカルおよびグローバル変数を初期化する

VI を実行する前に、ローカルおよびグローバル変数に既知のデータ値が含まれていることを確認します。そうしないと、VI が正しく動作しないデータが変数に含まれている可能性があります。

変数を初期化しないまま、VI が最初に変数を読み取ると、その変数にはフロントパネルオブジェクトに関連付けられているデフォルト値が含まれています。

競合状態

VI はデータフロー実行モデルに従うため、LabVIEW のローカルおよびグローバル変数は、テキストベースのプログラミング言語のローカルおよびグローバル変数のようには動作しません。競合状態は、並列に実行される 2 つ以上のコードが、ローカルまたはグローバル変数で代表されるような同じ共有リソースの値を変更する場合に発生します。図 10-1 は競合状態の例を示しています。

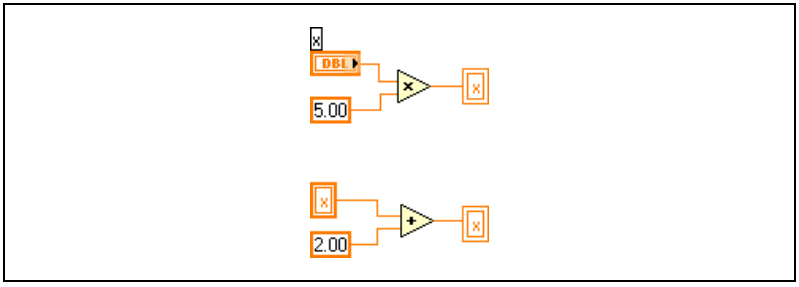


図 10-1　競合状態

この VI の出力は、演算が実行される順番によって異なります。2 つの演算の間にはデータ依存がないため、どちらが先に実行するかを決める方法がありません。競合状態を避けるには、読み取った変数と同じ変数に書き込まないことです。データ依存の詳細については、第 5 章「ブロックダイアグラムを作成する」の「データ依存と人工データ依存」のセクションを参照してください。

並列に実行される VI でグローバル変数を使用する場合、グローバル変数のデータがいつ変更されたかを示すためにブールのグローバル変数を追加することができます。他の VI はこのブールグローバル変数を監視して、データが変更されたかどうかを判断し、新しい値を読み込むことができます。

ローカル変数を使用する際のメモリへの配慮

サブ VI を作成するとき、サブ VI とのデータのやり取りの方法を記述したコネクタペーンを作成します。コネクタペーンは、データバッファのコピーを呼び出し側 VI からは作成しません。

ローカル変数はデータバッファのコピーを作成します。ローカル変数から読み取る場合は、その変数が関連付けられている制御器からのデータ用に新しいバッファを作成します。

ローカル変数を使用してブロックダイアグラム上のある場所から他の場所に大量のデータを転送する場合、通常はより多くのメモリを使用するため、ワイヤを使用してデータを転送する場合よりも実行速度が低下します。

グローバル変数を使用する場合のメモリへの配慮

グローバル変数から読み取ると、LabVIEW は、そのグローバル変数に格納されているデータのコピーを作成します。

大量の配列や文字列を操作する場合は、グローバル変数の操作に必要な時間とメモリがかなりの量になることがあります。1 つの配列要素を変更する場合でも LabVIEW は配列全体を格納するため、配列の場合のグローバル変数の操作は特に効率が悪くなります。アプリケーション内のいくつかの場所からグローバル変数を読み取る場合、複数のメモリバッファを作成しますが、このような方法は効率が悪く、パフォーマンスを低下させます。

11

グラフとチャート

グラフ形式でデータのプロットを表示するにはグラフやチャートを使用します。

グラフとチャートでは、データの表示方法と更新方法が異なります。グラフを持つ VI は通常、データを配列で収集して、そのデータをグラフにプロットします。これは、データを格納した後にそのプロットを生成するスプレッドシートに似ています。これに対し、チャートは、すでに表示されているデータ点に新しいデータ点を追加します。チャートでは、現在の読み取り値または測定値を以前に集録したデータと関連させて見ることができます。

詳細については

グラフとチャートの使用方法の詳細については、「LabVIEW ヘルプ」を参照してください。

グラフとチャートのタイプ

制御器→グラフパレットにあるグラフとチャートには以下のタイプがあります。

- **波形チャートとグラフ**：一定のレートで集録されるデータを表示します。
- **XY グラフ**：トリガが発生したときに集録されるデータなど、不定レートで集録されるデータを表示します。
- **強度チャートと強度グラフ**：第 3 の次元の値を色で表示して 2 次元プロット上に 3 次元データを表示します。
- **デジタル波形グラフ**：パルスまたはデジタルラインのグループとしてデータを表示します。コンピュータは、デジタルデータを他のコンピュータにパルスで転送します。
- **(Windows) 3 次元グラフ**：フロントパネル上の ActiveX オブジェクトの 3 次元プロット上に 3 次元データを表示します。

グラフとチャートの例については、`examples\general\graphs` を参照してください。

グラフおよびチャートのオプション

グラフとチャートは異なる方法でデータをプロットしますが、それらには共通するいくつかのオプションがあり、ショートカットメニューからアクセスできます。グラフのみまたはチャートのみで使用可能なオプションの詳細については、この章の「グラフをカスタマイズする」のセクションと「チャートをカスタマイズする」のセクションを参照してください。

波形および XY グラフとチャートには、強度、デジタル、および 3 次元のグラフとチャートとは異なるオプションがあります。強度、デジタル、および 3 次元のグラフとチャートのオプションの詳細については、この章の「強度グラフおよびチャート」、「3 次元グラフ」、および「デジタルグラフ」のセクションを参照してください。

グラフおよびチャート上の複数の x スケールと y スケール

すべてのグラフとチャートは、複数の x スケールおよび y スケールをサポートします。共通の x スケールまたは y スケールを持たない複数のプロットを表示するには、グラフまたはチャート上で複数のスケールを使用します。波形のグラフまたはチャートの x スケールまたは y スケールを右クリックし、ショートカットメニューから**スケール複製**を選択して、グラフまたはチャートに複数の x スケールまたは y スケールを追加します。

グラフとチャートのエイリアス除去ラインプロット

エイリアス除去ラインを使用すると、チャートおよびグラフ内のラインプロットの外観を改善できます。エイリアス除去ラインの描画を有効にすると、ラインプロットはより滑らかに表示されます。エイリアス除去ラインの描画によって、ライン幅、ラインスタイル、ポイントスタイルなどが変更されることはありません。

メモ　エイリアス除去ラインの描画は、デジタル波形グラフでは使用できません。

エイリアス除去ラインプロットを有効にするには、プロット凡例を右クリックし、ショートカットメニューから**エイリアス除去**を選択します。プロット凡例が表示されていない場合は、チャートまたはグラフを右クリックし、ショートカットメニューから**項目を表示→プロット凡例**を選択します。

メモ　エイリアス除去ラインの描画は演算主導なので、エイリアス除去ラインプロットを使用すると性能が低下します。

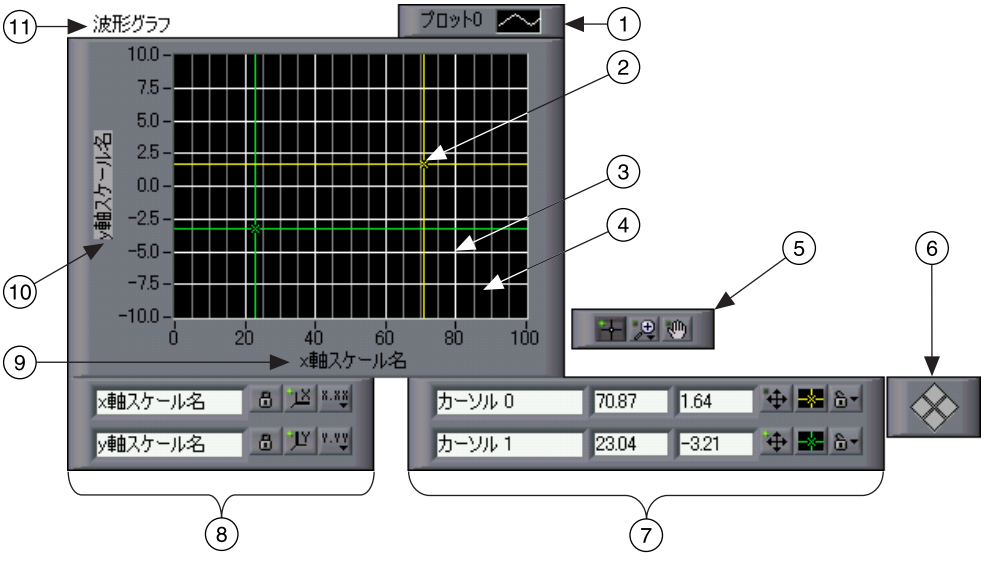
グラフとチャートの外観をカスタマイズする

オプションを表示または非表示にすることによって、グラフとチャートの外観をカスタマイズします。グラフまたはチャートを右クリックし、ショートカットメニューから**項目を表示**を選択して、以下のオプションを表示または非表示にします。

- **プロット凡例**：プロットの色とスタイルを定義します。凡例をサイズ変更し、複数のプロットを表示します。
- **スケール凡例**：スケールのラベルを定義し、スケールのプロパティを構成します。
- **グラフパレット**：VI の実行中にスケーリングとフォーマット処理を変更します。
- **X スケールと Y スケール**：x スケールと y スケールをフォーマットします。スケールの詳細については、この章の「スケールオプション」のセクションを参照してください。
- **カーソルの凡例 (グラフのみ)**：定義された点座標にマーカを表示します。グラフ上に複数のカーソルを表示できます。
- **スクロールバー (チャートのみ)**：チャート内のデータをスクロールします。スクロールバーを使用して、バッファが現在表示していないデータを表示します。

グラフをカスタマイズする

グラフカーソルの動作、スケーリングオプション、およびグラフ軸を変更できます。図 11-1 はグラフの要素を示しています。



1　プロット凡例	4　最小グリッドマーク	7　カーソルの凡例	10　Y スケール
2　カーソル (グラフのみ)	5　グラフパレット	8　スケール凡例	11　ラベル
3　グリッドマーク	6　カーソルムーバ	9　X スケール	

図 11-1　グラフの要素

前述の凡例に示した大部分の項目は、グラフを右クリックし、ショートカットメニューから**項目を表示**を選択して、該当する要素を選択することによって追加できます。

グラフカーソル

グラフ上のカーソルによって、プロット上の点の正確な値を読み取ることができます。カーソル値はカーソルの凡例内に表示されます。カーソルを追加するには、グラフを右クリックし、ショートカットメニューから**項目を表示→カーソルの凡例**を選択して、カーソルの凡例の行の任意の場所をクリックし、カーソルをアクティブにします。複数のカーソルを追加するには、位置決めツールを使用して、カーソルの凡例を拡張します。

すべてのグラフ上にカーソルとカーソルディスプレイを配置したり、プロット上のカーソルにラベルを付けることができます。カーソルをプロット上にロックしたり、複数のカーソルを同時に移動することもできます。グラフにはカーソルをいくつでも使用できます。

カーソル値を読み取り、カーソルを使用してプログラムでグラフを拡大／縮小する方法の例については、`examples\general\graphs\zoom.llb` 内の Zoom Graph VI を参照してください。

スケールオプション

グラフは、水平と垂直のスケールを自動的に調整して、配線したデータを反映させることができます。この動作をオートスケーリングと呼びます。グラフを右クリックし、ショートカットメニューから **X スケール→ X 軸オートスケール** または **Y スケール→ Y 軸オートスケール**を選択して、オートスケーリングをオンまたはオフにします。デフォルトでは、グラフに対するオートスケーリングは有効です。ただし、オートスケーリングによって性能が低下します。

水平または垂直のスケールを直接変更するには、操作ツールまたはラベリングツールを使用します。

波形グラフのスケール凡例

スケールにラベルを付け、スケールのプロパティを構成するには、スケール凡例を使用します。

操作ツールを使用して、左図に示す**スケール形式**ボタンをクリックし、形式、精度、およびマッピングモードを構成します。

各スケールのオートスケーリング、スケールの表示、スケールラベル、およびプロットを切り替えたり、スケールラベル、グリッド、グリッド線、およびグリッドカラーをフォーマットするには、左図に示す**スケールロック**ボタンを使用します。

グラフのスケール形式

グラフのスケールをフォーマットして、絶対時間または相対時間、あるいは振幅を表すことができます。絶対時間形式を使用してスケールの時刻、日付、あるいは両方を表示します。LabVIEW で日付を想定しないようにする場合は、相対時間形式を使用します。絶対時間形式または相対時間形式を選択するには、グラフを右クリックし、変更するスケールを選択して、ショートカットメニューから形式を選択します。表示された形式ダイアログボックスでグラフスケールの異なるプロパティを指定します。デフォルトでは、x 軸は相対時間を表し、y 軸は振幅を表します。

スムーズアップデートを使用する

オフスクリーンバッファを使用してフラッシュの回数を最小にするには、グラフを右クリックし、ショートカットメニューから**上級→スムーズアップデート**を選択します。**スムーズアップデート**を使用すると、ご使用のコンピュータおよびビデオシステムによっては性能が低下します。

チャートをカスタマイズする

既に格納されているデータを上書きして波形全体を表示するグラフとは異なり、チャートは既に格納されているデータの記録を定期的に更新して維持します。

制御器→グラフパレット上にある波形チャートと強度チャートをカスタマイズし、データ表示条件に合わせたり、詳細な情報を表示できます。チャートで使用可能なオプションには、スクロールバー、凡例、パレット、デジタル表示、および時間に関するスケール表現が含まれます。チャート記録の長さ、更新モード、およびプロット表示の動作を変更できます。

チャート記録の長さ

既にチャートに追加されているデータ点をバッファ、つまりチャート記録に格納します。チャート記録バッファのデフォルトのサイズは 1,204 データ点です。チャートを右クリックし、ショートカットメニューから**チャート記録の長さ**を選択すると、記録バッファを構成できます。チャートのスクロールバーを使用して、それまでに収集したデータを表示できます。スクロールバーを表示するには、チャートを右クリックし、ショートカットメニューから**項目を表示→スクロールバー**を選択します。

チャートの更新モード

チャートは 3 種類のモードを使用して、データをスクロールします。チャートを右クリックし、ショートカットメニューから**上級→更新モード**を選択します。**ストリップチャート**、**スコープチャート**、または **スイープチャート**を選択します。デフォルトのモードは**ストリップチャート**です。ストリップチャートは、チャートを左から右へスクロールして、実行中のデータを連続的に表示します。スコープチャートは、チャートを左から右へ部分的にスクロールして、パルスや波形などのデータ項目を一つ表示します。スイープ表示は EKG 表示に似ています。スイープはスコープと同様に動作しますが、垂直の線で分けられた右側に古いデータを表示し、左側に新しいデータを表示する点だけ異なります。

オーバレイ対スタックプロット

1 つの垂直スケールを使用して複数のプロットをチャートに表示することをオーバレイプロットと呼び、複数の垂直スケールを使用してチャート上に表示することをスタックプロットと呼びます。図 11-2 は、オーバレイプロットとスタックプロットの例を示しています。

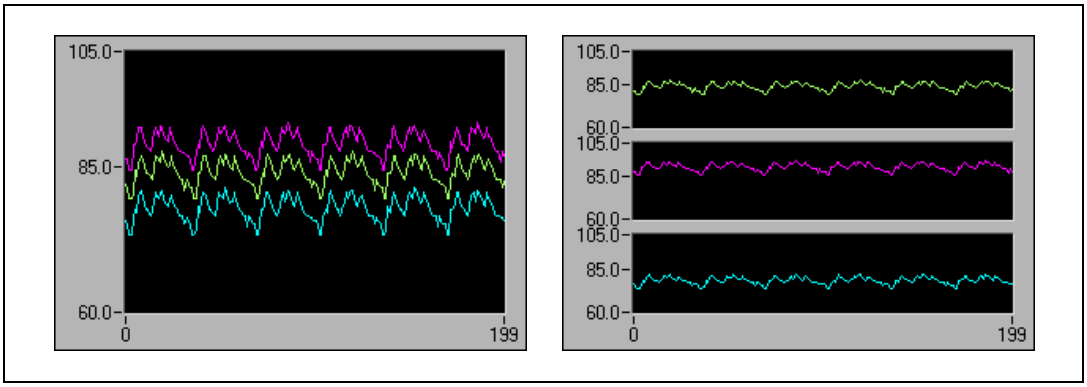


図 11-2　オーバレイおよびスタックプロットを持つチャート

さまざまな種類のチャートの例と受け入れ可能なデータタイプについては、`examples\general\graphs\charts.llb` 内の Charts VI を参照してください。

波形グラフと XY グラフ

波形グラフは、均一にサンプリングされた測定値を表示します。XY グラフは、均一または不均一にサンプリングされた一組の点を表示します。図 11-3 は波形グラフと XY グラフの例を示しています。

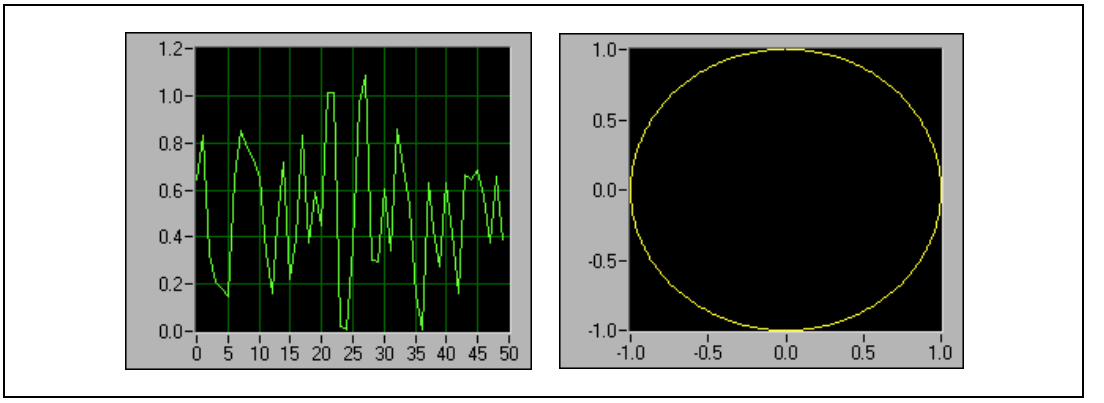


図 11-3　波形グラフと XY グラフ

波形グラフは、時間変化集録波形のように x 軸に沿って均一に分布した点で、*y*= f(*x*) のような一価関数のみをプロットします。XY グラフは、円形やタイムベースが変化する波形などの多価関数をプロットする汎用デカルトグラフ作成オブジェクトです。どちらのグラフも、任意の数の点を含むプロットを表示できます。

これらのタイプのグラフはどちらも、複数のデータタイプを受け入れます。これによって、表示の前に行う必要のあるデータの処理を最小限に抑えられます。

シングルプロット波形グラフデータタイプ

波形グラフは、シングルプロット波形グラフに対して 2 つのデータタイプを受け入れます。

グラフは値の配列を 1 つ受け入れ、そのデータをグラフ上の点として解釈し、さらに *x*= 0 で始まる *x* 指標を 1 つ増分します。また、グラフは *x* の初期値、Δ*x*、および *y* データの配列のクラスタを受け入れます。

シングルプロット波形グラフが受け入れるデータタイプの例については、`examples\general\graphs\gengraph.llb` 内の Waveform Graph VI を参照してください。

マルチプロット波形グラフ

マルチプロット波形グラフは 2 次元配列の値を受け入れます。ここで、配列の各行はシングルプロットです。グラフはデータをグラフ上の点として解釈し、*x* = 0 で始まる *x* 指標を 1 つ増分します。その配列の各列をプロットとして扱うには、2 次元配列のデータタイプをグラフに配線し、グラフを右クリックして、ショートカットメニューから**配列を転置**を選択します。DAQ デバイスは各チャネルを個別の列として格納した 2 次元配列としてデータを返すため、これは DAQ デバイスから複数のチャネルをサンプリングするときに特に便利です。このデータタイプを受け入れるグラフの例については、`examples\general\graphs\gengraph.llb` 内の Waveform Graph VI の (Y) マルチプロット 1 グラフを参照してください。

マルチプロット波形グラフは、*x* 値、Δ*x* 値、および *y* データの 2 次元配列のクラスタも受け入れます。グラフは *y* データをグラフ上の点として解釈し、*x*= 0 で始まる *x* 指標を Δ*x* ずつ増分します。このデータタイプは、すべて均一の規則的なレートでサンプリングした複数の信号を表示するのに便利です。このデータタイプを受け入れるグラフの例については、`examples\general\graphs\gengraph.llb` 内の Waveform Graph VI の (Xo, dX, Y) マルチプロット 3 グラフを参照してください。

マルチプロット波形グラフは、クラスタを含む配列のプロット配列を受け入れます。各クラスタには、*y* データを含む点配列が含まれています。内部配列はプロット内の点を記述し、外部配列は各プロットに対して 1 つのクラスタがあります。図 11-4 は、**y** クラスタのこの配列を示しています。

図 11-4　クラスタ y の配列

各プロット内の要素数が異なる場合は、2 次元配列ではなくマルチプロット波形グラフを使用します。たとえば、チャネル別に異なる時間を使用して、複数のチャネルからデータをサンプリングする場合は、2 次元配列ではなくこのデータストラクチャを使用します。これは、2 次元配列では各行の要素数が同じでなくてはならないためです。クラスタの配列の内部配列の要素の数は同じでなくてもかまいません。このデータタイプを受け入れるグラフの例については、`examples\general\graphs\gengraph.llb` 内の Waveform Graph VI の (Y) マルチプロット 2 グラフを参照してください。

マルチプロット波形グラフは、*x* の初期値のクラスタ、Δ*x* 値、およびクラスタを持つ配列を受け入れます。各クラスタには、*y* データを含む点配

列が含まれています。配列をクラスタとしてバンドルするには Bundle 関数を使用し、そのクラスタを配列にするには Build Array 関数を使用します。指定した入力を含むクラスタの配列を作成する Build Cluster Array も使用できます。このデータタイプを受け入れるグラフの例については、examples\general\graphs\gengraph.llb 内の Waveform Graph VI の (Xo, dX, Y) マルチプロット 2 グラフを参照してください。

マルチプロット波形グラフは、*x* 値のクラスタの配列、Δ*x* 値、および *y* データの配列を受け入れます。これは、各プロットの x 軸の固有の開始点および増分を指定できるため、マルチプロット波形グラフのデータタイプの中で最も一般的です。このデータタイプを受け入れるグラフの例については、examples\general\graphs\gengraph.llb 内の Waveform Graph VI の (Xo, dX, Y) マルチプロット 1 グラフを参照してください。

シングルプロット XY グラフのデータタイプ

シングルプロット XY グラフは、*x* 配列と *y* 配列を含むクラスタを受け入れます。XY グラフは点の配列も受け入れます。ここで、点は *x* 値と *y* 値を含むクラスタです。

シングルプロット XY グラフのデータタイプの例については、examples\general\graph\gengraph.llb 内の XY Graph VI を参照してください。

マルチプロット XY グラフのデータタイプ

マルチプロット XY グラフはプロットの配列を受け入れます。ここで、このプロットは *x* 配列と *y* 配列を含むクラスタです。マルチプロット XY グラフは、プロットのクラスタの配列も受け入れます。ここで、このプロットは点の配列です。点は *x* 値と *y* 値を含むクラスタです。

マルチプロット XY グラフのデータタイプの例については、examples\general\graph\gengraph.llb 内の XY Graph VI を参照してください。

波形チャート

波形チャートは、1 つまたは複数のプロットを表示する特殊なタイプの数値表示器です。波形チャートの例については、examples\general\graphs\charts.llb を参照してください。

一度に 1 つまたは複数の値を同時にチャートに渡すことができます。波形グラフと同様に、LabVIEW はデータをグラフ上の点として解釈し、*x* = 0 で始まる *x* 指標を 1 つ増分します。チャートはこの入力をシングルプロットの新規データとして扱います。

チャートに複数の点を同時に渡すと、チャートの再描画回数が少なくなります。

複数のプロットのデータを波形チャートに渡すには、データをスカラ数値のクラスタ内にバンドルできます。ここで、各数値は各プロットの 1 つの点を表します。

一度の更新でプロットの複数の点を渡す場合は、数値のクラスタの配列をチャートに配線します。各数値は、各プロットの 1 つの *y* 値の点を表します。

表示するプロットの数を実行時まで決定できない場合、または一度の更新で複数のプロットの複数の点を渡す場合は、データの 2 次元配列をチャートに配線します。波形グラフと同様に、波形チャートはデフォルトで行を各プロットの新規データとして扱うようになっています。配列内の列を各プロットの新規データとして扱うには、2 次元配列データタイプをチャートに配線し、チャートを右クリックして、ショートカットメニューから**配列を転置**を選択します。

強度グラフおよびチャート

デカルト平面上に色のブロックを配置することによって、2 次元プロット上に 3 次元データを表示するには、強度グラフおよびチャートを使用します。たとえば、強度グラフとチャートを使用して、温度パターンや大きさが標高を表す地形などのパターンのあるデータを表示することができます。強度グラフおよびチャートは、数値の 2 次元配列を受け入れます。配列内の各数値は特定の色を表します。2 次元配列内の要素の指標が色のプロット位置を設定します。図 11-5 は強度チャート操作の概念を示しています。

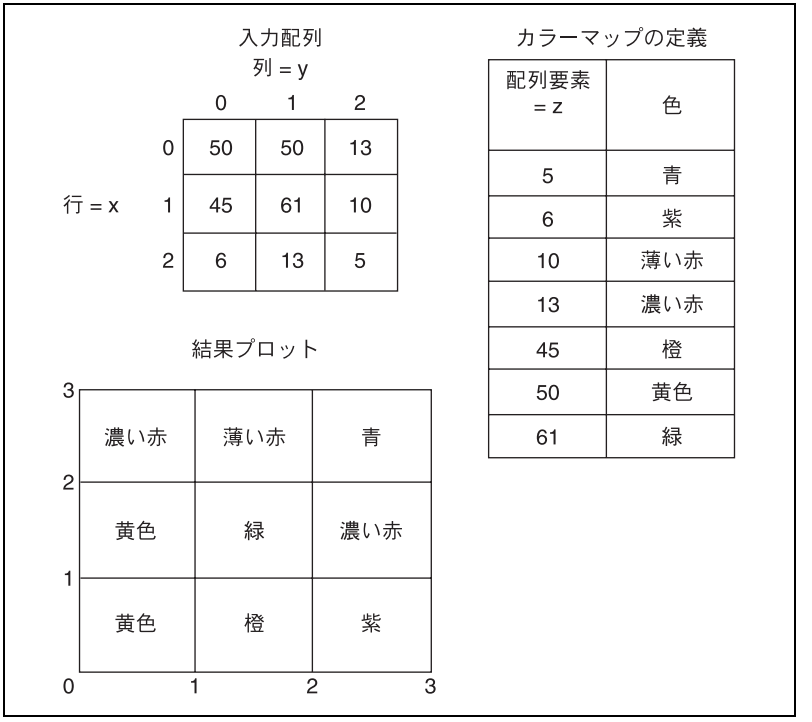


図 11-5　強度チャートのカラーマップ

データの行は、グラフまたはチャート上の新しい列として表示されます。行を画面上の行として表示する場合は、2 次元配列データタイプをグラフまたはチャートに配線し、グラフまたはチャートを右クリックして、ショートカットメニューから**配列を転置**を選択します。

配列の指標は、カラーブロックの左下頂点に相当します。カラーブロックは、配列の指標で定義された 2 点間の領域である単位領域を持ちます。強度グラフまたはチャートは、最大 256 の個別の色を表示できます。大量

のデータを強度チャート上に表示する場合は、十分なメモリがあることを確認してください。

強度チャート上にデータのブロックをプロットすると、デカルト平面の原点は最後のデータブロックの右側に移動します。チャートが新規データを処理するとき、その新規データは古いデータの右側に表示されます。チャートの表示が満杯になると、古いデータはチャートの左側にスクロールされて表示されなくなります。この動作はストリップチャートの動作に似ています。これらのチャートの詳細については、この章の「チャートの更新モード」のセクションを参照してください。

強度チャートおよびグラフの例については、`examples\general\graphs\intgraph.llb` を参照してください。

カラーマッピング

カラーランプ数値制御器にカラーを定義するのと同じ方法で、強度グラフおよびチャートに、対話式でカラーマッピングを設定できます。カラーランプの詳細については、第 4 章「フロントパネルを作成する」の「カラーランプ」のセクションを参照してください。

強度グラフとチャートにプログラムでカラーマッピングを設定するには、プロパティノードを 2 つの方法で使用します。通常は、プロパティノードに数値対色マッピングを指定します。この方法では、**Z Scale:Marker Values** プロパティを指定します。このプロパティは配列のクラスタで構成され、各クラスタには数値のリミット値とその値を表示するための対応するカラーが含まれています。この方法でカラーマッピングを指定する場合、上限を超えた色を指定するには **Z Scale:High Color** プロパティを使用し、下限を超えた色を指定するには **Z Scale:Low Color** プロパティを使用します。強度グラフおよびチャートは全部で 254 色に制限され、下限と上限を超えた色を合わせて合計 256 色になります。254 色を超える数の色を指定すると、強度グラフまたはチャートは、指定された色を補間して 254 色のカラーテーブルを作成します。

強度グラフ上にビットマップを表示する場合は、**Color Table** プロパティを使用してカラーテーブルを指定します。この方法では、最大 256 色の配列を指定できます。チャートに渡されるデータは、強度チャートのカラースケールに基づいて、このカラーテーブルの指標に割り当てられます。カラースケールの範囲が 0 ～ 100 の場合は、データ内の値 0 が指標 1 に、値 100 が指標 254 に割り当てられて、その間の値は 1 と 254 の間に補間されます。0 より小さい値はすべて下限を超えた色 (指標 0) に割り当てられ、100 を超える値はすべて上限を超えた色 (指標 255) に割り当てられます。

メモ　強度グラフまたはチャートで表示する色は、ご使用のビデオカードが表示できる数の厳密な色に制限されます。また、ご使用のディスプレイに割り当てられた色の数による制限もあります。

強度チャートオプション

強度チャートの任意の部分の多くは波形チャートと共通しています。チャートを右クリックしてショートカットメニューから**項目を表示**を選択すると、それらの部分を表示または非表示にできます。さらに、強度チャートは色を第 3 次元として扱うため、カラーランプ制御器に似たスケールが色に対する値の範囲とマッピングを定義します。

波形チャートと同様に、強度チャートは以前の更新の記録データ、つまりバッファを保持しています。チャートを右クリックし、ショートカットメニューから**チャート記録の長さ**を選択すると、このバッファを構成できます。強度チャートにおけるデフォルトのサイズは 128 データ点です。強度チャートの表示はメモリ主導です。たとえば、512 点の記録と 128 の *y* 値で単精度チャートを表示するには、512 ×128 × 4 バイト (単精度数のサイズ)、つまり 256 KB のメモリが必要です。

強度グラフオプション

強度グラフは強度チャートと同様に動作しますが、以前のデータを保持せず更新モードがないという点のみ異なります。新規データが強度グラフに渡されるたびに、古いデータは新規データに置き換えられます。

強度グラフには他のグラフと同様のカーソルがあります。各カーソルは、グラフ上の指定された点を示す *x* 値、*y* 値、および *z* 値を表示します。

デジタルグラフ

タイミングダイアグラムまたはロジックアナライザを操作する場合は特に、波形グラフを使用してデジタルデータを表示します。デジタルデータ集録の詳細については、『LabVIEW Measurements Manual』の Chapter 8「Digital I/O」を参照してください。

図 11-6 の波形グラフは、6 つの 8 ビット符号なし整数の配列を示しています。このグラフはバイナリ形式の整数を表し、各ビットはグラフ上のプロットを表します。

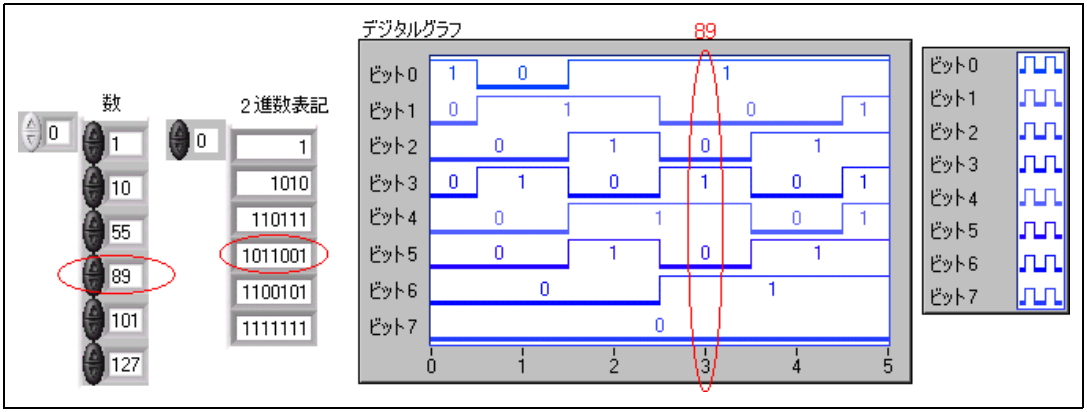


図 11-6　デジタル形式での整数のグラフ化

図 11-6 の**バイナリ表記**の第 1 ビット、つまり一番右のビットの列 (ここでは 101111) は、デジタルグラフ上の最初のプロットである Bit0 に対応します。2 番目のビット列 (011001) は 2 番目のプロットである Bit1 に対応します。以下同様に続きます。たとえば、数値 89 には 7 ビットのメモリが必要です。グラフィックによる数値 89 のバイナリ表現は、グラフ上のポイント 3 の位置に表示されます。

デジタルデータを表示する VI を作成するには、Bundle 関数を使用して、トリガ時間 (X0)、Δ*x*(dX)、数値配列、およびポート数を図 11-7 のように組み合わせます。

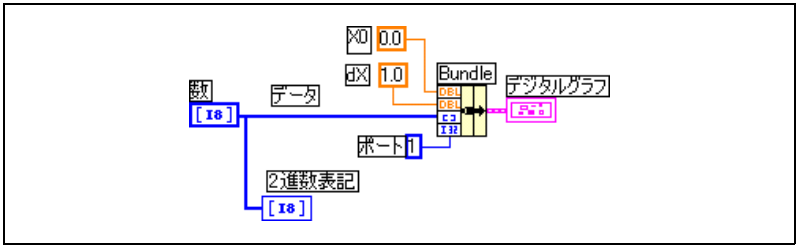


図 11-7　デジタルグラフでの Bundle 関数の使用

ポートは、1 つの整数として扱うデータ要素の数を指定します。8 ビットデータを 8 ビット整数として表現する場合、ポートの数は 1 つです。また、32 ビットデータを 32 ビット整数として表現する場合も、ポートの数は 1 つです。8 ビットデータのソースが 3 つある場合は、24 ビットを 1 つの整数として表現する必要があります。この場合は、図 11-8 のように、Interleave 関数を使用して 8 ビットデータをインターリーブし、ポート数を 3 に指定します。

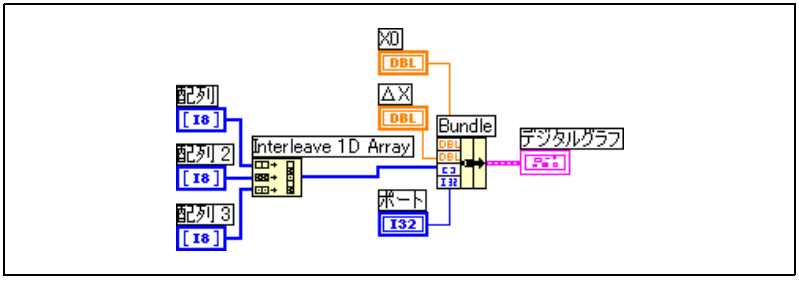


図 11-8　デジタルグラフでの Interleave 関数の使用

図 11-8 で、8 ビットデータの 3 つの配列をすべてグラフ化するには、**ポート**に「3」を入力します。8 ビットデータの 2 つの配列をグラフ化するには、**ポート**に「2」を入力します。8 ビットデータの 5 つの配列をグラフ化するには、**ポート**に「5」を入力します。この場合、40 ビットのうち 16 ビットには値が格納されません。

データをマスクする

図 11-6 の VI は、各プロットがデータ内の 1 つのビットを表現するグラフを生成します。グラフ上にデータを表示する前に、データ内のビットを選択したり、並べ替えたり、結合することもできます。ビットの選択、並べ替え、および結合をデータのマスクと呼びます。

マスクを使用すると、2 つ以上の異なるビットのプロットを結合し、1 つのプロットに表示することができます。8 ビット整数の配列の場合は、1 つのプロットに 8 ビットまで表示できます。16 ビット整数の配列の場合は、1 つのプロットに 16 ビットまで表示でき、以下同様に続きます。1 つのプロットに同じビットを 2 回以上プロットすることもできます。デジタルデータのマスクについては、付録 D「デジタルデータをマスクする」を参照してください。

3 次元グラフ

メモ　3D グラフ制御器は、Windows 対応の LabVIEW 開発システムおよびプロフェッショナル開発システムでのみ使用できます。

表面上の温度分布、時間／周波数共同領域解析、航空機の動きといった実世界の多くのデータセットでは、3 次元データを視覚化する必要があります。3 次元グラフを使用すると、3 次元データを視覚化でき、3 次元グラフのプロパティを変更することによってデータの表示方法を変更できます。

使用可能な 3 次元グラフは以下のとおりです。

- **3D Surface**：3 次元空間に曲面を描画します。この制御器をフロントパネル上にドロップすると、曲面を表すデータを受信するサブ VI に配線されます。
- **3D Parametric**：3 次元空間に複雑な曲面を描画します。この制御器をフロントパネル上にドロップすると、曲面を表すデータを受信するサブ VI に配線されます。
- **3D Curve**：3 次元空間に線を描画します。この制御器をフロントパネル上にドロップすると、線を表すデータを受信するサブ VI に配線されます。

曲線や曲面をプロットするには、**関数→グラフィック & サウンド→ 3D グラフプロパティ**パレットにある 3 次元グラフ VI とともに 3 次元グラフを使用します。曲線はグラフ上の個々の点から構成され、各点は *x*、*y*、および *z* 座標で示されます。この VI は、これらの点を線で結びます。曲線は、飛行機の飛行経路など、動いている物体の経路を視覚化するのに最適です。

3D グラフでは、ActiveX テクノロジと 3D 表現を処理する VI を使用します。**関数→グラフィック & サウンド→ 3D グラフプロパティ**パレットにある VI のプロパティ (基本、軸、グリッド、およびプロジェクションのプロパティを含む) を設定することによって、実行時の動作を変更できます。

曲面プロットは、*x*、*y*、および *z* のデータを使用してグラフ上に点をプロットします。その後、これらの点を結んでデータの 3 次元曲面画像を生成します。たとえば、曲面プロットを使用して地形図を作成できます。

3 次元グラフを選択すると、LabVIEW は 3D グラフ制御器が含まれたフロントパネル上に ActiveX コンテナを配置します。LabVIEW はまた、3D グラフ制御器のリファレンスをブロックダイアグラム上に配置します。LabVIEW は、3 つの 3 次元グラフ VI のいずれかにこのリファレンスを配線します。

波形データタイプ

波形データタイプは、波形のデータ、開始時刻、および Δt（時間分解能）を含む要素のクラスタです。波形を作成するには、**関数→波形**パレットにある Build Waveform 関数を使用します。波形の集録または解析に使用する多くの VI および関数は、デフォルトで波形データタイプを受け取り、波形データタイプを返します。波形グラフまたは波形チャートに波形データタイプを配線すると、波形のデータ、開始時刻、および Δx に基づいて、グラフまたはチャート上に自動的に波形がプロットされます。波形グラフまたは波形チャートに波形データタイプの配列を配線すると、グラフ

またはチャート上にすべての波形が自動的にプロットされます。波形データタイプの使用方法については、『LabVIEW Measurements Manual』の Chapter 5「Introduction to Data Acquisition in LabVIEW」を参照してください。

12

グラフィック & サウンド VI

VI の画像とサウンドを表示したり変更するには、**関数→グラフィック & サウンド**パレットにあるグラフィック & サウンド VI および関数を使用します。

詳細については

VI での画像とサウンドの使用方法の詳細については、「LabVIEW ヘルプ」を参照してください。

ピクチャ表示器を使用する

制御器→グラフ→制御器パレットにあるピクチャ表示器は、線、円、テキストなどの画像形状を含むことができるピクチャを表示する汎用の表示器です。ピクチャ表示器はピクセルレベルで制御できるので、ほとんどすべての画像オブジェクトを作成できます。

ピクチャ表示器は、ピクセルベースの座標系を持ち、原点 (0,0) は制御器の左上コーナーに位置するピクセルです。座標の水平 (x) 成分は右へ向かって増加し、座標の垂直 (y) 成分は下へ向かって増加します。

ピクチャが大きすぎてピクチャ表示器がそのピクチャを表示できない場合、LabVIEW はピクチャを切り取って、表示器の表示領域内に収まるピクセルだけを表示します。ピクチャ全体を表示するには、位置決めツールを使用して表示器をサイズ変更し、VI を再実行します。

ピクチャ表示器のピクチャ領域のサイズをプログラムで読み取ったり変更するには **Draw Area Size** プロパティを使用します。ピクチャ表示器全体のサイズを読み取るには、**Bounds** プロパティを使用します。ピクチャ表示器全体のサイズと、ピクチャ表示器の表示領域のサイズを決定するには、位置決めツールを使用して表示器を選択し、**編集→制御器を編集する**を選択して、制御器エディタの**制御器部品**ウィンドウを使用します。

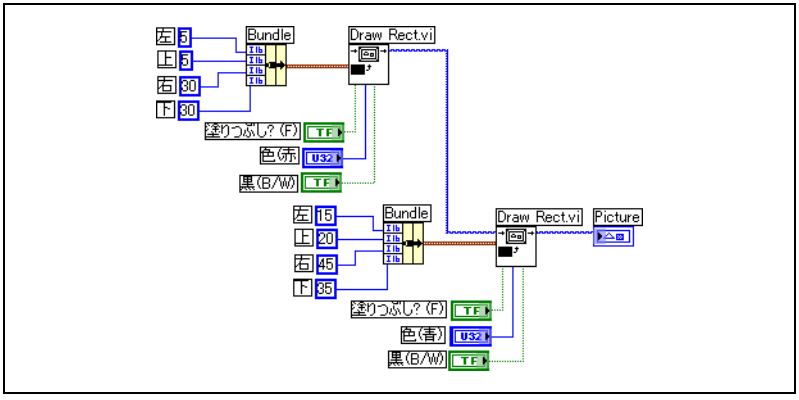
フロントパネル上にピクチャ表示器を配置すると、表示器は空白の四角形領域として表示され、左図のような、対応する端子がブロックダイアグラム上に表示されます。画像アプリケーションの代わりに、**関数→グラフィック & サウンド→画像関数**パレット上の画像関数 VI を使用して、LabVIEW 内の画像の機能の変更と追加を行います。

ピクチャ表示器内に画像を表示するには、画像関数 VI を使用して、一連の描画手順を指定する必要があります。各 VI は、描画手順を説明した入力値を受け取ります。VI は、これらの入力値に基づいて、表示のため、ピクチャ表示器に渡すこれらの手順を簡潔に説明したものを作成します。

画像関数 VI で作成したピクチャは、特定のピクチャ表示器または画像関数 VI の **picture** 入力にのみ接続できます。LabVIEW は、開いているフロントパネル上でピクチャ表示器を更新するときにピクチャを描画します。

各画像関数 VI は、その描画手順を、前の描画手順の **picture** 入力／出力に連結します。**picture** 入力が接続されていない場合、**picture** 出力は空白の四角形描画領域を返します。

次のブロックダイアグラムは、Draw Rect VI を使用して 2 つの重なり合う四角形を描画します。



black in B/W? ブール制御器は、モノクロのモニタに表示する場合にその四角形を黒または白のどちらで表示するかを示します。ピクチャ表示器を使用する一連の VI を作成して、VI をすべてのモニタ上で実行する場合は、**black in B/W?** 制御器を使用する必要があります。

画像プロット VI

ピクチャ表示器を使用して通常のタイプのグラフを作成するには、**関数→グラフィック & サウンド→画像プロット**パレットにある画像プロット VI を使用します。このグラフには極座標、波形グラフ表示、スミスプロット、およびグラフスケールが含まれています。

画像プロット VI は低レベル描画関数を使用して、データのグラフィカルな表示器を作成し、描画コードをカスタマイズして機能を追加します。
このグラフィカルな表示器は組み込み LabVIEW 制御器のような対話式では

ありませんが、それらを使用すると、現在の組み込み制御器ではできない方法で情報を表示できます。組み込み LabVIEW グラフとはわずかに異なる機能を持つプロットを作成するには、Plot Waveform VI を使用します。

Polar Plot VI をサブ VI として使用する

極グラフの隣接する特定の四分円またはグラフ全体を一度に描くには、Polar Plot VI を使用します。組み込み LabVIEW 波形グラフと同様に、そのコンポーネントの色を指定し、グリッドを含め、さらにスケールの範囲と形式を指定できます。

Polar Plot VI は、1 つの VI にさまざまな機能を与えます。したがって、その VI には入力端子のための複雑なクラスタが含まれています。デフォルト値とカスタム制御器を使用して、VI を簡単に作成することができます。ブロックダイアグラム上に独自のクラスタ入力を作成するのではなく、`examples\picture\demos.llb` 内の Polar Plot Demo VI からカスタム制御器をコピーし、それをフロントパネル上に配置します。

Waveform Plot VI をサブ VI として使用する

点、接続された線、およびバーを含む、さまざまなスタイルの波形を描くには、組み込み波形グラフの動作をエミュレートする Plot Waveform VI を使用します。組み込み LabVIEW 波形グラフと同様に、グリッドを含め、そのコンポーネントの色、スケールの範囲や形式を指定できます。

Waveform Plot VI により、1 つの VI にさまざまな機能を追加することができます。したがって、その VI には入力端子のための複雑なクラスタが含まれています。デフォルト値とカスタム制御器を使用して、VI を簡素化することができます。独自のクラスタ入力を作成するのではなく、`examples\picture\demos.llb` 内の Waveform and XY Plots VI からカスタム制御器をコピーし、それをフロントパネル上に配置します。

Plot XY VI と Plot Multi-XY VI は Plot WaveformVI と似ています。プロットの装飾的な外観を指定するには別の制御器を使用します。なぜなら、XY をプロットする VI は 3 つのプロットスタイル、すなわち 2 つの分散プロットスタイルと、個別の *x* 位置で線を描画してその *x* 値に対する *y* の最小値と最大値にマークを付ける 1 つのプロットスタイルを持っているからです。

Smith Plot VI をサブ VI として使用する

電気通信業界などの伝送ラインの動作を調べるには、Smith Plot VI を使用します。伝送ラインはエネルギーや信号を送る媒体です。伝送ラインはワイヤを指す場合や、信号を送る環境を指す場合があります。伝送ラインは送信中の信号に影響を与えます。この影響は伝送ラインのインピーダンスと呼ばれ、交流信号を減衰させたり、位相を変化させます。

伝送ラインのインピーダンスは、ラインの抵抗とリアクタンスの測定値です。通常、インピーダンス *z* は *z* = *r* + j*x* の形式の複素数としてリストされます。ここで、抵抗 (*r*) およびリアクタンス (*x*) はその要素です。

伝送ラインのインピーダンスを表示するには、Smith Plot VI を使用します。プロットは一定の抵抗とリアクタンスを表す円で構成されます。

与えられたインピーダンス *r* + j*x* は、*r* の円と *x* の円の交点の位置を定めることによってプロットできます。インピーダンスのプロット後、視覚効果として Smith Plot VI を使用し、インピーダンスを整合させて伝送ラインの反射係数を計算します。

Smith Plot VI により、1 つの VI にさまざまな機能を追加することができます。したがって、これらの VI には入力端子のための複雑なクラスタが含まれています。デフォルト値とカスタム制御器を使用すると、VI を簡素化することができます。独自のクラスタ入力を作成するのではなく、`examples\picture\demos.llb` 内のサンプルの Smith Plot VI からカスタム制御器をコピーしてフロントパネル上に配置します。

負荷インピーダンスをグラフ表示している場合、インピーダンスは *r* + j*x* の形式の複素数として表現できます。

スミスプロットの詳細データの損失を避けるには、Normalize Smith Plot VI を使用してデータを正規化します。Normalize Smith Plot VI で正規化したデータは Smith Plot VI に直接渡すことができます。通常、スミスプロットデータは、システムの特性インピーダンス (Z_0) に従ってサイズを調整します。

画像関数 VI

ピクチャ表示器内に形状を描画してテキストを入力するには、**関数→グラフィック & サウンド→画像関数**パレットにある画像関数 VI を使用します。点、線、形状、およびピクセルマップを描画できます。平坦化されていないデータのピクセルマップは 2 次元配列の色で、個々の値は色に相当します。

画像関数パレットの 1 行目には、点および線の描画に使用する VI があります。点は、ピクセルの xy 座標を表す 2 つの 16 ビット符号付き整数のクラスタです。描画時、ピクチャでは画像ペンの位置が記憶されます。

ほとんどの画像関数 VI では、原点 (0,0) を基準にした絶対座標で指定する必要があります。Draw Line VI と Move Pen VI を使用すると、絶対座標または相対座標を指定できます。相対座標は現在のペンの位置が基準となります。Move Pen VI を使用すると、描画せずにペンの位置を変更

できます。ペンの位置を変更できるのは、Draw Point VI、Move Pen VI、Draw LineVI、および Draw Multiple Lines VI のみです。

画像関数パレットの 2 行目には、塗りつぶし形状の描画に使用する VI があります。これらの VI はそれぞれ、ピクチャの四角形の領域内に形状を描画します。四角形は、左右上下のピクセルを表す 4 つの値のクラスタで指定します。Draw Arc VI を使用した場合、四角形は楕円のサイズを指定します。追加のパラメータは、描画する楕円の部分 (弧) を指定します。

画像関数パレットの 3 行目には、ピクチャ内でのテキスト描画に使用する VI があります。Get Text Rect VI はテキストを描画しません。代わりに、文字列の境界枠の四角形のサイズの計算に使用します。

画像関数パレットの 4 行目には、ピクチャ内でのピクセルマップの描画に使用する VI があります。2 次元配列のデータを指定するだけでなく、色に相当する配列値のカラーテーブルも指定します。2 次元配列はピクセルのグリッドに相当します。これらの VI は、データ配列内の個々の値をカラーテーブル配列内の指標として使用することによって、その値を色に変換します。

画像関数パレットの最後の行には、Empty Picture 定数があり、空のピクチャを使用するときや変更するときに使用します。

画像関数 VI を使用した色の作成と変更

多くの画像関数 VI には、形状およびテキストの色を変更する**色**入力端子があります。色を作成する最も簡単な方法は、**関数→数値→その他の数値定数**パレットにあるカラーボックス定数を使用し、その定数をクリックして色を選択することです。

カラーボックス定数ではなく、計算結果によって色を作成するには、カラーボックスがどのように数値を使用して色を指定しているかを理解する必要があります。

32 ビット符号付き整数は色を表し、下位の 3 バイトが赤、緑、および青の要素を表します。青の範囲については、下位のバイトが青の要素を含んでいるので、下位のバイトの値のみが変化する 32 ビット整数の配列を作成します。灰色の範囲を作成するには、赤、緑、および青の各要素の値が等しい 32 ビット整数の配列を作成します。

画像形式 VI

ビットマップ (`.bmp`)、ポータブルネットワーク画像 (`.png`)、Joint Photographic Experts Group (`.jpg`) ファイルなどの複数の標準画像ファイル形式からデータを読み書きするには、**関数→グラフィック & サ**

ウンド→グラフィック形式パレットにあるグラフィック形式 VI を使用します。これらの VI を使用して、以下のタスクを実行します。

- ビットマップファイルからデータを読み取りピクチャ表示器に表示する。
- データを読み取り画像を操作する。
- 画像を書き込んで他のアプリケーションで表示する。

ビットマップデータは 2 次元配列で、色の濃さによって各点のデータタイプが異なります。たとえば、モノクロつまり 1 ビット画像の場合、各点はブール値です。4 ビットおよび 8 ビット画像の場合、各点はカラーテーブル内の指標です。24 ビット True Color 画像の場合、各点は赤、緑、および青 (RGB) の値を混合したものです。

グラフィックファイルの読み書きを行う VI は、単純で平坦化された形式のデータを利用します。これは、1 次元配列に格納されたデータを使い、画像ファイルをディスクに書き込む方法に似ています。これらの画像ファイルはピクセルマップで、概念はビットマップと同じです。**関数→グラフィック & サウンド→画像関数**パレットにある Draw Flattened Pixmap VI を使用して、この平坦化されたデータを直接表示します。このパレットには、1 ビット (単色)、4 ビット、8 ビット、および 24 ビットデータの 2 次元配列のピクセルマップを表示する VI があります。

関数→グラフィック & サウンド→グラフィック形式パレットにある Unflatten Pixmap VI と Flatten Pixmap VI を使用してデータを適切な形式に変換すると、そのデータを 2 次元配列として処理できます。

サウンド VI

サウンドファイルと関数をユーザの VI に統合するには、**関数→グラフィック & サウンド→サウンド**パレットにあるサウンド VI を使用します。この VI を使用すると、以下のタスクを実行できます。

- ユーザが特定の操作を実行したときに、録音された警告などのサウンドファイルを実行する VI を作成する。
- VI が実行を開始または完了したとき、または VI 内のある点に達したときにサウンドファイルを実行する VI を作成する。
- サウンドデータを集録するためのサウンド入力デバイスを構成する。サウンド入力 VI を使用して、サウンドデータを集録します。デバイスから入力される任意のサウンド情報を読み取ることもできます。
- 他のサウンド VI からのサウンドデータを受け入れるためのサウンド出力デバイスを構成する。デバイスを通るサウンド量を制御したり、サウンドを再生または一時停止したり、システムからサウンドを消すことができます。

13

ファイル I/O

ファイル I/O 操作により、ファイルとのデータ受け渡しができます。以下の操作を含むさまざまなファイル I/O 処理には、**関数→ファイル I/O** パレットにあるファイル I/O VI および関数を使用します。

- データファイルの開閉。
- ファイルに対するデータの読み書き。
- スプレッドシート形式のファイルに対する読み書き。
- ファイルおよびディレクトリの移動および名前変更。
- ファイル特性の変更。
- 構成ファイルの作成、変更、および読み取り。

通常の I/O 操作を実行するには、高レベル VI を使用します。各ファイル I/O 操作を個別に制御するには、低レベル VI および関数を使用します。

詳細については

ファイル I/O 操作の実行については、「LabVIEW ヘルプ」を参照してください。

ファイル I/O の基本

通常のファイル I/O 操作には、以下のプロセスが含まれます。

1. ファイルを作成するか開きます。パスを指定するか、ダイアログボックス内でファイルの場所を指定することによって、既存ファイルの保存場所または新規ファイルの作成場所を指定します。ファイルが開くと、refnum がそのファイルを表します。refnum の詳細については、第 4 章「フロントパネルを作成する」の「オブジェクトまたはアプリケーションへのリファレンス」のセクションを参照してください。
2. ファイルから読み取るか、ファイルに書き込みます。
3. ファイルを閉じます。

ほとんどのファイル I/O VI および関数は、1 回のファイル I/O 操作で 1 つのステップのみを実行します。ただし、通常のファイル I/O 操作用に設計された一部の高レベルファイル I/O VI は、3 つのステップをすべて実行します。これらの VI は低レベル関数ほど効率的であるとは限りませんが、使いやすさの面では優れています。

ファイル I/O 形式を選択する

使用するファイル I/O VI は、ファイルの形式によって異なります。ファイルに対するデータの読み書きは、テキスト、バイナリ、およびデータログの 3 つの形式で実行できます。使用する形式は、集録または作成の対象となるデータと、そのデータにアクセスするアプリケーションによって異なります。

使用する形式を決定するには、以下の基本的なガイドラインに従ってください。

- Microsoft Excel など、他のアプリケーションでもデータを使用できるようにする場合は、テキストファイルを使用します。これは、テキストファイルが最も一般的であり、最も移植性が高いためです。
- ランダムアクセスによる読み書きを行う必要がある場合や、速度の向上とディスク容量の節約が非常に重要な場合は、バイナリファイルを使用します。これは、バイナリファイルの方がテキストファイルよりもディスク容量および速度の面で効率が良いためです。
- 複雑なデータレコードやさまざまなデータタイプを LabVIEW で処理する場合は、データログファイルを使用します。これは、データへのアクセス元を LabVIEW に限定する場合や、複雑なデータ構造を保存する必要がある場合、データログファイルがデータの保存に最適であるためです。

テキストファイルを使用する場合

ディスク容量とファイル I/O 速度が重要でない場合や、ランダムアクセスによる読み書きを行う必要がない場合や、数値の精度が重要でない場合に、他のユーザまたはアプリケーションでもデータを使用できるようにするには、データに対してテキスト形式のファイルを使用します。

テキストファイルは、最も使いやすく最も共有しやすい形式です。テキストファイルの読み書きは、ほとんどすべてのコンピュータで実行できます。テキストベースのファイルの読み取りは、さまざまなテキストベースのプログラムで行うことができます。テキスト文字列は、ほとんどの計測制御アプリケーションで使用されます。

ワープロまたは表計算アプリケーションなど、他のアプリケーションからもデータにアクセスする場合は、テキストファイルにデータを保存します。テキスト形式でデータを保存するには、**関数→文字列**パレットにある文字列関数を使用して、すべてのデータをテキスト文字列に変換します。テキストファイルには、さまざまなデータタイプの情報を格納できます。

通常、データの ASCII 表現はデータ自体よりもサイズが大きいため、元のデータがテキスト形式でなくグラフやチャートなどのデータである場

合、テキストファイルはバイナリファイルやデータログファイルよりも多くのメモリを使用します。たとえば、−123.4567 という数値は、4 バイトの単精度浮動小数点数として保存できます。ただし、この数値の ASCII 表現は、1 文字あたり 1 バイトずつの合計 9 バイトを使用します。

また、テキストファイル内の数値データにランダムにアクセスするのは困難です。これは、文字列内の各文字が正確に 1 バイトの容量を使用していても、数値をテキストとして表現するために必要な容量が一定ではないためです。テキストファイル内の 9 番目の数値を検出するには、LabVIEW はまず、その前にある 8 つの数値を読み取り変換する必要があります。

テキストファイルに数値データを保存すると、精度が低下する場合があります。コンピュータは数値データをバイナリデータとして保存しますが、ユーザは一般的に数値データを 10 進表記法でテキストファイルに書き込みます。精度の低下は、テキストファイルからデータを読み取るときに発生する場合があります。バイナリファイルでは、精度の低下が問題となることはありません。

テキストファイルでのファイル I/O の使用例については、`examples\file\smplfile.llb` と `examples\file\sprdsht.llb` を参照してください。

バイナリファイルを使用する場合

バイナリファイルの保存によって使用されるディスク上のバイト数は一定です。たとえば、0 ～ 40 億の数値 (1、1000、1000000 など) をバイナリ形式で保存すると、1 つの数値あたり 4 バイトが使用されます。

数値データを保存したり、ファイル内の特定の数値にアクセスしたり、ファイル内の数値にランダムにアクセスするには、バイナリファイルを使用します。人間が読み取ることのできるテキストファイルとは異なり、バイナリファイルを読み取ることができるのはコンピュータのみです。バイナリファイルは、最も容量が小さく高速なデータ保存形式です。バイナリファイルでは複数のデータタイプを使用できますが、一般的ではありません。

バイナリファイルは効率的なデータ保存形式です。これは、使用するディスク容量が少なくてすむだけでなく、データの保存や取得時にデータをテキストに変換したり、テキストから変換したりする必要がないためです。バイナリファイルは、1 バイトのディスク容量で 256 個の値を表現できます。拡張精度数値や複素数の場合を除き、バイナリファイルには、データのバイト単位のイメージがメモリに保存されていたとおりに格納されています。データがこのように格納されていると変換が不要になるため、ファイルを速く読み取ることができます。LabVIEW がデータを格納する方法の詳細については、『LabVIEW Data Storage』アプリケーションノートを参照してください。

メモ　テキストファイルとバイナリファイルは、バイトストリームファイルとして知られています。これは、データを連続した文字またはバイトとして保存することを意味します。

バイナリファイルに対する倍精度浮動小数点数の配列の読み書きの例については、`examples\file\smplfile.llb` 内の Read Binary File VI と Write Binary File VI を参照してください。

データログファイルを使用する場合

データのアクセスおよび操作を LabVIEW でのみ行う場合や、複雑なデータ構造をすばやく簡単に保存する場合は、データログファイルを使用します。

データログファイルは、LabVIEW でのみ作成や読み取りが実行できるファイルです。バイナリファイルの場合、データ保存時、データタイプが類似している必要があります。データログファイルを使用すると、ファイルの 1 つのレコード内に複数の異なるデータタイプを保存できます。たとえば、1 つのデータログファイルに、文字列、数値、およびクラスタを保存できます。

データログファイルでは、スプレッドシートと同様に、同じ構造を持つ一連のレコードとしてデータが保存されます。各行がレコードを表します。データログファイル内の各レコードには、同じデータタイプを関連付ける必要があります。LabVIEW は、各レコードを保存するデータを含むクラスタとしてファイルに書き込みます。ただし、データログレコードの構成要素は、ファイルの作成時に指定した任意のデータタイプにすることができます。

たとえば、レコードのデータタイプが文字列および数値のクラスタであるデータログを作成できます。この場合、データログの各レコードは文字列および数値のクラスタになります。ただし、最初のレコードを (`"abc",1`) にして、2 番目のレコードを (`"xyz",7`) にすることができます。

データログファイルを使用すると、読み書きを高速化する操作がほとんど必要ありません。また、元のデータブロックをレコードとして読み取ることができ、ファイル内のそのデータブロックの前にあるレコードをすべて読み取る必要がないため、データの取得が簡単になります。データログファイルでは、レコード番号だけでレコードにアクセスできるため、ランダムアクセスが高速で簡単になります。LabVIEW は、データログファイルの作成時にレコード番号を各レコードに順番に割り当てます。

データログファイルには、フロントパネルとブロックダイアグラムからアクセスできます。フロントパネルからデータログファイルへのアクセスについては、この章の「フロントパネルのデータを記録する」のセクションを参照してください。

LabVIEW は、関連する VI を実行するたびにデータログファイルにレコードを書き込みます。LabVIEW がデータログファイルに書き込んだ後で、そのレコードを上書きすることはできません。データログファイルを読み取るときは、一度に 1 つまたは複数のレコードを読み取ることができます。

データログファイルの読み書きの例については、`examples\file\datalog.llb` を参照してください。

高レベルファイル I/O VI を使用する

以下のデータタイプの読み書きといった通常の I/O 操作を実行するには、**関数→ファイル I/O** パレットの一番上の行にある高レベルファイル I/O VI を使用します。

- テキストファイル内の文字
- テキストファイル内の行
- スプレッドシートテキストファイル内の単精度数値の 1 次元配列または 2 次元配列
- バイナリファイル内の単精度数値または符号付き 16 ビット整数の 1 次元配列または 2 次元配列

高レベル VI を使用してファイルに読み書きすることによって、プログラミングの労力と時間を節約できます。高レベル VI は、ファイルの開閉だけでなく、読み書きを実行します。VI が実行のたびに閉じる／開くの操作を行うため、高レベル VI をループ内に配置しないでください。複数の操作を実行しているあいだファイルを開いたままにしておく方法については、「ディスクストリーミング」のセクションを参照してください。

高レベル VI にはファイルパスを入力する必要があります。ファイルパスを接続しないと、読み書きの対象となるファイルの指定を要求するダイアログボックスが表示されます。エラーが発生すると、高レベル VI はエラーについて説明するダイアログボックスを表示します。ユーザは、実行を停止するか継続するかを選択できます。

図 13-1 は、高レベルの Write To Spreadsheet File VI を使用して Microsoft Excel のスプレッドシートファイルに数値を送る方法を示しています。この VI を実行すると、LabVIEW は、既存ファイルへのデータの書き込みまたは新規ファイルの作成をユーザに要求します。

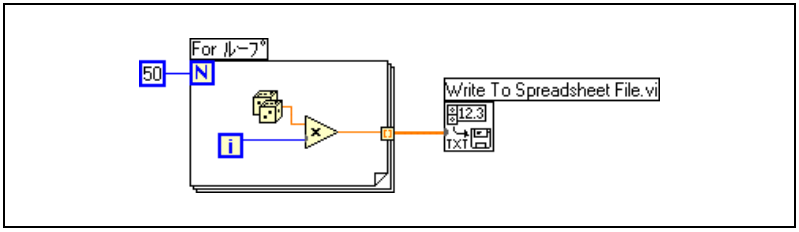


図 13-1　高レベル VI によるスプレッドシートへの書き込み

バイナリ形式のファイルに対して読み書きを行うには、**関数→ファイル I/O →バイナリファイル VI** パレットにあるバイナリファイル VI を使用します。データは、整数または単精度浮動小数点数を使うことができます。

低レベルおよび上級のファイル I/O VI および関数を使用する

各ファイル I/O 操作を個別に制御するには、**関数→ファイル I/O** パレットの 2 行目にある低レベルファイル I/O VI および関数と、**関数→ファイル I/O →上級ファイル関数**パレットにある上級ファイル I/O 関数を使用します。

ファイルを作成するか開き、ファイルに対するデータの読み書きを行い、ファイルを閉じるには、主要な低レベル関数を使用します。以下のタスクを実行するには、他の低レベル関数を使用します。

- ディレクトリの作成。
- ファイルの移動、コピー、または削除。
- ディレクトリの内容のリスト。
- ファイル特性の変更。
- パスの操作。

左図に示すパスは、ディスク上のファイルの場所を識別する LabVIEW データタイプです。パスは、ファイルが格納されているボリューム、ファイルシステムの最上位とそのファイル間のディレクトリ、およびそのファイルの名前を記述します。パス制御器またはパス表示器では、特定のプラットフォームの標準構文を使用してパスを入力または表示します。パス制御器および表示器の詳細については、第 4 章「フロントパネルを作成する」の「パス制御器および表示器」のセクションを参照してください。

図 13-2 は、低レベル VI および関数を使用して、Microsoft Excel のスプレッドシートファイルに数値を送る方法を示しています。この VI を実行すると、Open/Create/Replace VI が `numbers.xls` ファイルを開きます。Write File 関数は、数字列をファイルに書き込みます。Close 関数はファイルを閉じます。ファイルを閉じないと、ファイルがメモリ内に残っているため、他のアプリケーションまたは他のユーザはそのファイルにアクセスできません。

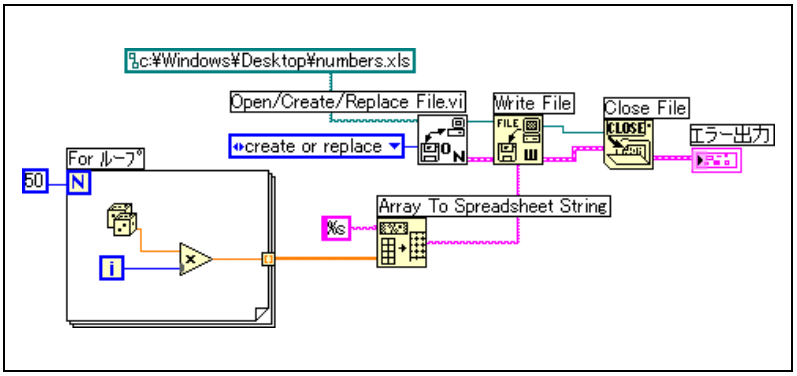


図 13-2　低レベル VI によるスプレッドシートへの書き込み

図 13-2 の VI と図 13-1 の VI を比較してください。これらの VI は同じタスクを実行します。図 13-2 では、Array To Spreadsheet String 関数を使用して、数値の配列を文字列としてフォーマットする必要があります。図 13-1 の Write To Spreadsheet File VI はファイルを開き、数値の配列を文字列に変換し、ファイルを閉じます。

低レベルファイル I/O VI および関数の使用例については、`examples\file\datalog.llb` 内の Write Datalog File Example VI を参照してください。

ディスクストリーミング

低レベルファイル I/O VI および関数を使用してディスクストリーミングを行うと、メモリリソースを節約できます。ディスクストリーミングは、ループ内などで複数の書き込みを実行している間、ファイルを開いたままにしておくためのテクニックです。高レベルの書き込み操作は使いやすい反面、実行するたびにファイルの開閉を行うため、オーバーヘッドが増えます。同じファイルを頻繁に開閉しないようにすると、VI の効率を向上させることができます。

ディスクストリーミングにより、ファイルの開閉時に関数がオペレーティングシステムと対話する回数が減ります。一般的なディスクストリーミング操作を作成するには、ループの前に Open/Create/Replace File VI

を配置し、ループの後に Close File 関数を配置します。これにより、ファイルの開閉によるオーバーヘッドを伴わずにループ内でファイルへの連続書き込みを実行できます。

ディスクストリーミングは、実行速度が重要な、大量のデータ集録に最適です。集録の実行中は、ファイルにデータを連続的に書き込むことができます。最適な結果を得るには、集録が完了するまで、解析 VI や解析関数などの他の VI や関数を実行しないでください。

テキストファイルとスプレッドシートファイルを作成する

テキストファイルまたはスプレッドシートファイルにデータを書き込むには、データを文字列に変換する必要があります。スプレッドシートファイルにデータを書き込むには、文字列をスプレッドシート文字列としてフォーマットする必要があります。スプレッドシート文字列は、タブなどの区切り文字を含んでいる文字列です。文字列のフォーマットについては、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「文字列をフォーマットする」のセクションを参照してください。

テキストを読み取ることができるほとんどのワープロアプリケーションではフォーマット済みテキストが不要なため、テキストファイルにテキストを書き込む場合はフォーマットする必要がありません。テキストファイルにテキスト文字列を書き込むには、Write Characters To File VI を使用します。この VI はファイルの開閉を自動的に行います。

グラフ、チャート、または集録データ内の一連の数値をスプレッドシート文字列に変換するには、Write To Spreadsheet File VI または Array To Spreadsheet String 関数を使用します。これらの VI および関数の使用方法については、この章の「高レベルファイル I/O VI を使用する」および「低レベルおよび上級のファイル I/O VI および関数を使用する」のセクションを参照してください。

ワープロアプリケーションでは、スプレッドシートファイル VI が処理できないさまざまなフォント、色、スタイル、およびサイズを使用してテキストがフォーマットされるため、ワープロアプリケーションからテキストを読み取るとエラーが発生する場合があります。

表計算アプリケーションまたはワープロアプリケーションに数値やテキストを書き込む場合は、**関数→文字列**パレットにある文字列関数と**関数→配列**パレットにある配列関数を使用してデータをフォーマットし、文字列を結合します。その後、ファイルにデータを書き込みます。これらの関数を使用してデータのフォーマットおよび結合を行う方法については、

第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」を参照してください。

データのフォーマットとファイルへの書き込み

文字列、数値、パス、およびブール値データをテキストとしてフォーマットし、フォーマットされたテキストをファイルに書き込むには、Format Into File 関数を使用します。多くの場合、Format Into String 関数を使用して文字列を個別にフォーマットし、Write Characters To File VI または Write File 関数を使用して結果の文字列を書き込む代わりに、この関数を使用できます。

Format Into File 関数を使用して、既存のデータをファイルに添付または上書きすることができます。refnum にファイルマークを設定するには、Seek 関数を使用します。Format into File 関数が結果文字列を書き込んだ後、ファイルマークはテキストの最後に設定されます。

文字列のフォーマットについては、第 9 章「文字列、配列、およびクラスタを使用してデータをグループ化する」の「文字列をフォーマットする」のセクションを参照してください。

ファイルからデータをスキャンする

ファイル内のテキストをスキャンして文字列、数値、パス、およびブール値を探し、そのテキストをデータタイプに変換するには、Scan From File 関数を使用します。多くの場合、Read File 関数または Read Characters From File VI を使用してファイルからデータを読み取り、Scan From String 関数を使用して結果の文字列をスキャンする代わりに、この関数を使用できます。

ファイル内のフォーマット済みのテキストを読み取るには、Scan From File 関数を使用します。スキャンを開始する個所にマークを設定するには、Seek 関数を使用します。

バイナリファイルを作成する

バイナリファイルを作成するには、一連の数値を集録し、その数値をファイルに書き込みます。**関数→ファイル I/O →バイナリファイル VI** パレットにある高レベルのバイナリファイル I/O VI を使用しない限り、数値をフォーマットする必要はありません。

数値の 1 次元配列および 2 次元配列をファイルに保存するには、Write To I16 VI と Write To SGL VI を使用します。まず、数値を 16 ビット整数または単精度浮動小数点数としてフォーマットする必要があります。作成したファイルを読み取るには、Read I16 VI と Read SGLVI を使用します。

倍精度浮動小数点数や 32 ビット符号なし整数などの異なるデータタイプの数値を書き込むには、低レベル関数か、**関数→ファイル I/O →上級ファイル関数**パレットにある上級ファイル関数を使用します。ファイルを読み取るときは、Read File 関数を使用して数値のデータタイプを指定します。

バイナリファイルに対する浮動小数点数の読み書きの例については、`examples\file\smplfile.llb` 内の Read Binary File VI と Write Binary File VI を参照してください。

データログファイルを作成する

フロントパネルのデータロギングを有効にするか、低レベル関数または**関数→ファイル I/O →上級ファイル関数**パレットにある上級ファイル関数を使用してデータの集録およびファイルへのデータの書き込みを行うことによって、データログファイルの作成および読み取りを実行できます。フロントパネルからのデータログファイルの作成およびアクセスについては、この章の「フロントパネルのデータを記録する」のセクションを参照してください。

データログファイル内のデータをフォーマットする必要はありません。ただし、データログファイルの読み書きを行うときは、データタイプを指定する必要があります。たとえば、温度の測定値とその温度の測定時刻と日付を集録した場合は、データログファイルにデータを書き込み、そのデータを 1 つの数値と 2 つの文字列のクラスタとして指定します。データログファイルへのデータの書き込みの例については、`examples\file\datalog.llb` 内の Simple Temp Datalogger VI を参照してください。

温度の測定値とその温度の測定時刻と日付が格納されているファイルを読み取る場合は、1 つの数値と 2 つの文字列のクラスタを読み取りを指定します。データログファイルの読み取りの例については、`examples\file\datalog.llb` 内の Simple Temp Datalog Reader VI を参照してください。

ファイルに波形を書き込む

ファイルに波形を送るには、**関数→波形→波形ファイル I/O** パレットにある Write Waveforms to File VI および Export Waveforms to Spreadsheet File VI を使用します。スプレッドシートファイル、テキストファイル、またはデータログファイルに波形を書き込むことができます。

作成した波形を VI 内でのみ使用する場合は、波形をデータログファイル (.log) として保存します。データロギングの詳細については、この章の「データログファイルを使用する場合」のセクションを参照してください。

図 13-3 の VI は複数の波形を集録し、グラフ上に表示し、Microsoft Excel のスプレッドシートファイルに書き込みます。

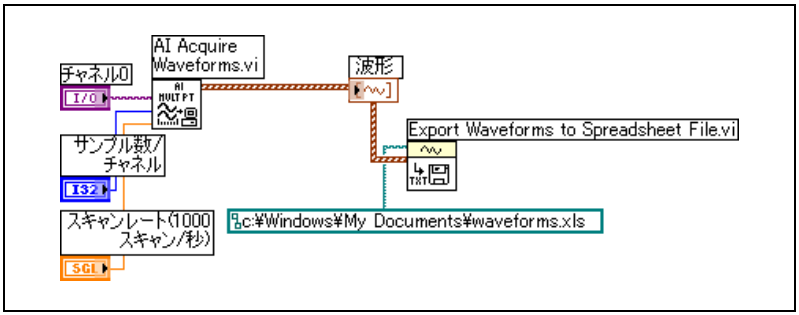


図 13-3　スプレッドシートファイルへの複数の波形の書き込み

ファイルから波形を読み取る

ファイルから波形を読み取るには、**関数→波形→波形ファイル I/O** パレットにある Read Waveform from File VI を使用します。1 つの波形を読み取った後、**関数→波形**パレットにある Build Waveform 関数を使用して波形データタイプの要素を追加または編集したり、**関数→波形**パレットにある Get Waveform Attribute 関数を使用して波形要素を抽出できます。

図 13-4 の VI はファイルから波形を読み取り、その波形の **dt** 要素を編集し、編集後の波形をグラフにプロットします。

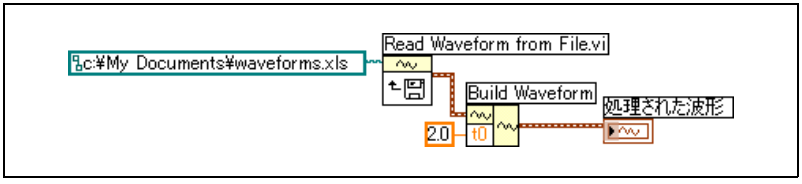


図 13-4　ファイルからの波形の読み取り

Read Waveform from File VI は、ファイルから複数の波形を読み取ることもできます。この VI は波形データタイプの配列を返します。この配列はマルチプロットグラフに表示できます。ファイル内の 1 つの波形にアクセスする場合は、図 13-5 のように、波形データタイプの配列に指標を付ける必要があります。配列への指標付けについては、第 9 章「文字列、

配列、およびクラスタを使用してデータをグループ化する」の「配列」のセクションを参照してください。この VI は、複数の波形が格納されているファイルにアクセスします。Index Array 関数はファイル内の最初および 3 番目の波形を読み取り、その波形を 2 つの別個の波形グラフ上にプロットします。グラフの詳細については、第 8 章「ループと Case ストラクチャ」を参照してください。

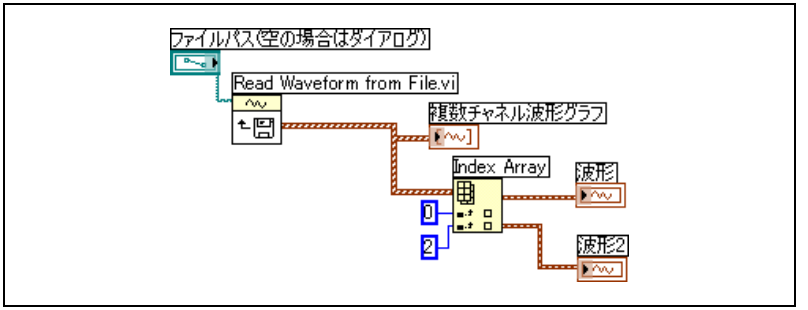


図 13-5　ファイルからの複数の波形の読み取り

フロースルーパラメータ

多くのファイル I/O VI および関数には、対応する入力パラメータと同じ値を返すフロースルーパラメータ (通常は refnum またはパス) があります。関数の実行順序を制御するには、このパラメータを使用します。最初に実行するノードのフロースルー出力を、次に実行するノードの対応する入力に接続することによって、人工データ依存を確立します。このフロースルーパラメータを使用しない場合は、シーケンスストラクチャを使用して、希望の順序でファイル I/O 操作が実行されるようにする必要があります。人工データ依存の詳細については、第 5 章「ブロックダイアグラムを作成する」の「データ依存と人工データ依存」のセクションを参照してください。

構成ファイルを作成する

標準的な Windows 構成ファイル (`.ini`) の読み取りおよび作成を行う場合や、これらの VI によって生成されるファイルを複数のプラットフォームで使用できるように、プラットフォーム固有のデータ (パスなど) をプラットフォームに依存しない形式で書き込む場合は、**関数→ファイル I/O →構成ファイル VI** パレットにある構成ファイル VI を使用します。

構成ファイル VI の使用例については、`examples\file\config.llb` を参照してください。

メモ　Windows 構成ファイルの標準の拡張子は .ini ですが、ファイルの内容が正しい形式であれば、構成ファイル VI はどのような拡張子を持つファイルでも処理します。内容の構成については、この章の「Windows 構成ファイルの形式」のセクションを参照してください。

構成ファイルを使用する

標準的な Windows 構成ファイルは、テキストファイルにデータを保存するための特殊な形式です。.ini ファイルは特殊な形式に従っているため、ファイル内のデータにプログラムで簡単にアクセスできます。

たとえば、次の内容を持つ構成ファイルを検討します。

```
[Data]
Value=7.2
```

図 13-6 のように、構成ファイル VI を使用するとこのデータを読み取ることができます。この VI は Read Key VI を使用して、Data という**セクション**から Value という**キー**を読み取ります。この VI は、ファイルが Windows 構成ファイルの形式であれば、ファイルがどのように変更されても動作します。

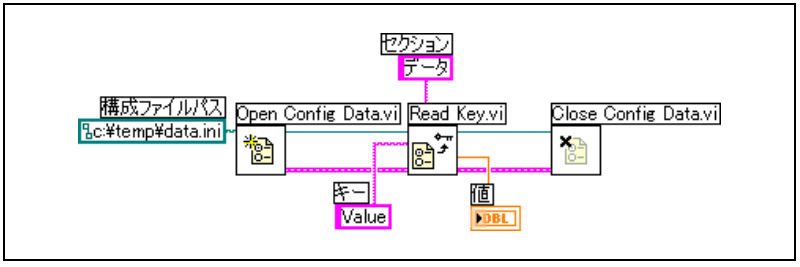


図 13-6　.ini ファイルからのデータの読み取り

Windows 構成ファイルの形式

Windows 構成ファイルは、名前付きのセクションに分けられたテキストファイルです。各セクション名は括弧で囲まれています。1 つのファイル内でセクション名が重複しないようにしてください。セクションには、等号 (=) で区切られたキーと値のペアが含まれています。各セクション内でキー名が重複しないようにしてください。キー名は構成の環境設定を表し、値名はその環境設定の設定値を表します。次の例は、ファイル内の項目の配置を示しています。

```
[Section 1]
key1=value
key2=value

[Section 2]
```

```
key1=value
key2=value
```

キーパラメータの値の部分には、構成ファイル VI を使用して以下のデータタイプを指定します。

- 文字列
- パス
- ブール
- 64 ビット倍精度浮動小数点数
- 32 ビット符号付き整数
- 32 ビット符号なし整数

構成ファイル VI は、未処理またはエスケープ文字列データの読み書きを実行できます。この VI は、未処理データを ASCII に変換せずに、そのデータをバイト単位で読み書きします。変換された文字列、つまりエスケープ文字列の場合、LabVIEW は、構成ファイル内の非表示テキスト文字を等価の 16 進エスケープコード (復帰文字では \0D) として保存します。また、LabVIEW は、構成ファイル内の円コード文字を 2 つの円コード (\ では \\) として保存します。Configuration File VI の **read raw string?** または **write raw string?** 入力を、未処理データの場合は TRUE に設定し、エスケープデータの場合は FALSE に設定します。

VI が構成ファイルに書き込むとき、スペース文字を含んでいる文字列データまたはパスデータの前後には引用符が付けられます。文字列に引用符が含まれている場合、LabVIEW は引用符を \" として保存します。テキストエディタを使用して構成ファイルの読み書きを行うと、LabVIEW によって引用符が \" に置換されていることがわかります。

LabVIEW は、プラットフォームに依存しない標準的な UNIX パス形式でパスデータを .ini ファイルに保存します。この VI は、構成ファイル内に保存されている絶対パス /c/temp/data.dat を次のように解釈します。

- **(Windows)** c:\temp\data.dat
- **(Macintosh)** c:temp:data.dat
- **(UNIX)** /c/temp/data.dat

この VI は、相対パス temp/data.dat を次のように解釈します。

- **(Windows)** temp\data.dat
- **(Macintosh)** :temp:data.dat
- **(UNIX)** temp/data.dat

フロントパネルのデータを記録する

他の VI やレポート内で使用するデータを記録するには、フロントパネルのデータロギングを使用します。たとえば、グラフからデータを記録し、そのデータを別の VI の別のグラフ内で使用できます。

VI を実行するたびに、フロントパネルのデータロギングによって別のデータログファイルにフロントパネルデータが保存されます。このファイルは、テキストが区切られた形式になっています。データは以下の方法で取り出すことができます。

- 対話形式でデータを取り出すには、データの記録元と同じ VI を使用します。
- プログラムでデータを取り出すには、VI をサブ VI として使用します。
- ファイル I/O VI および関数を使用します。

各 VI は、データログファイルの場所を記録するログファイルのバインディングを保持します。LabVIEW は、その場所に記録済みフロントパネルデータを保持します。ログファイルのバインディングとは、VI データの記録先となるデータログファイルと VI の関係です。

データログファイルには、VI の各実行時のタイムスタンプおよびデータを含んでいるレコードが格納されています。データログファイルにアクセスするときは、回収モードで VI を実行し、フロントパネル制御器を使用してデータを表示することによって、必要なレコードを選択します。回収モードで VI を実行すると、フロントパネルの一番上に数値制御器が表示されます。この制御器を使用するとレコード間を移動できます。この数値制御器の例については、図 13-7 を参照してください。

自動および対話形式でのフロントパネルのデータロギング

自動ロギングを有効にするには、**操作→ VI 終了後にログ**を選択します。VI のフロントパネルデータを初めて記録するときは、LabVIEW によってデータログファイルの名前の入力が要求されます。VI を実行するたびに LabVIEW によってデータが記録され、VI を再び実行するたびに新規レコードがデータログファイルに追加されます。LabVIEW がデータログファイルに書き込んだ後で、そのレコードを上書きすることはできません。

データを対話形式で記録するには、**操作→データロギング→ログ**を選択します。LabVIEW は、データログファイルにデータを即座に追加します。対話形式のデータロギングでは、データを記録する時刻を選択できます。自動的なデータロギングでは、VI を実行するたびにデータが記録されます。

メモ　波形チャートでは、フロントパネルのデータロギングによって一度に 1 つのデータ点のみが記録されます。チャート表示器に配列を接続した場合、データログファイルには、そのチャートに表示される配列のサブセットが格納されます。

記録済みのフロントパネルデータを対話形式で表示する

データを記録した後、**操作→データロギング→回収**を選択することによって、データを対話形式で表示できます。図 13-7 のようなデータ回収ツールバーが表示されます。

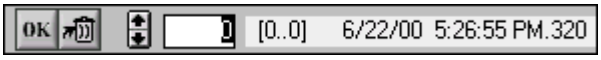


図 13-7　データ回収ツールバー

ハイライトされている数字は、表示中のデータレコードを示します。角括弧内の数字は、現在の VI について記録したレコードの範囲を示します。VI を実行するたびにレコードが記録されます。日付と時刻は、選択されているレコードを記録した日時を示します。次のレコードまたは前のレコードを表示するには、増分矢印または減分矢印をクリックします。キーボードの上矢印キーと下矢印キーを使用することもできます。

データ回収ツールバーだけでなくフロントパネルの外観も、ツールバーで選択したレコードに応じて変化します。たとえば、増分矢印をクリックして別のレコードに進むと、データを記録した時点におけるその特定のレコードのデータが制御器と表示器に表示されます。回収モードを終了し、表示していたデータログファイルを持つ VI に戻るには、**OK** ボタンをクリックします。

レコードを削除する

回収モード時に特定のレコードを削除できます。回収モードで個々のレコードを削除対象としてマークするには、そのレコードを表示し、**ごみ箱**ボタンをクリックします。**ごみ箱**ボタンをもう一度クリックすると、そのレコードは削除対象から外されます。

削除対象としてマークしたレコードをすべて削除するには、回収モード時に**操作→データロギング→データを排出**を選択します。

OK ボタンをクリックする前に**操作→データロギング→データを排出**を選択しないと、LabVIEW によって、マークしたレコードの削除を要求されます。

ログファイルのバインディングを消去する

フロントパネルデータの記録または回収時に使用するデータログファイルと VI を関連付けるには、ログファイルのバインディングを使用します。

複数のデータログファイルを 1 つの VI に関連付けることができます。これは、VI データのテストまたは比較に役立つことがあります。たとえば、VI を初めて実行したときに記録されたデータと、VI を 2 回目に実行したときに記録されたデータを比較できます。複数のデータログファイルと VI を関連付けるには、**操作→データロギング→ログファイルのバインディングの消去**を選択して、ログファイルのバインディングを消去する必要があります。その後、自動ロギングを有効にして VI を実行するか、データを記録することを対話形式で選択すると、LabVIEW によってデータログファイルの指定が要求されます。

ログファイルのバインディングを変更する

別のログファイルに対してフロントパネルデータの記録または回収を行うようにログファイルのバインディングを変更するには、**操作→データロギング→ログファイルのバインディングの変更**を選択します。LabVIEW により、別のログファイルの選択または新規ログファイルの作成が要求されます。別のデータを VI 内に取り込む場合や、VI から別のデータログファイルにデータを追加する場合は、ログファイルのバインディングを変更できます。

プログラムでフロントパネルデータを取り出す

サブ VI を使用するか、ファイル I/O VI および関数を使用して、記録済みデータを取り出すこともできます。

サブ VI を使用してフロントパネルデータを取り出す

サブ VI を右クリックし、ショートカットメニューから**データベースアクセスを可能にする**を選択すると、図 13-8 のように、サブ VI の周りに黄色いボックスが表示されます。

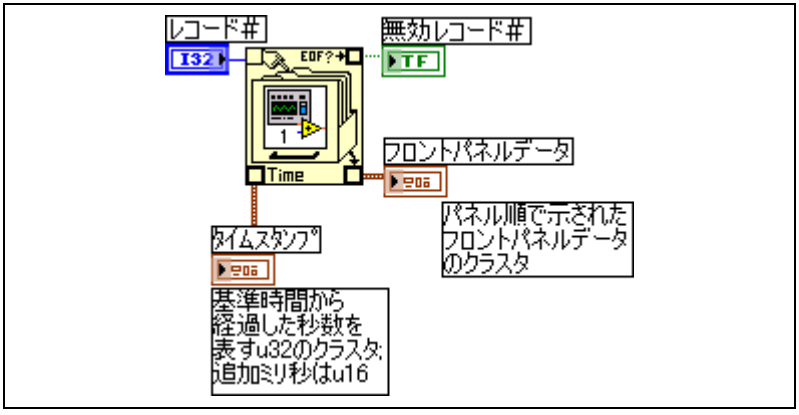


図 13-8　記録済みデータの取り出し

ファイルキャビネットのような形をした黄色いボックスには、データログファイルからデータにアクセスするための端子が含まれています。サブ VI に対するデータベースアクセスを可能にすると、サブ VI の入力および出力は実際には出力として働き、その記録済みデータを返します。**レコード #** は取り出すレコードを示し、**無効レコード #** はレコード番号が存在するかどうかを示します。**タイムスタンプ**はレコードの作成時刻であり、**フロントパネルデータ**はフロントパネルオブジェクトのクラスタです。フロントパネルオブジェクトのデータにアクセスするには、**フロントパネルデータ**クラスタを Unbundle 関数に接続します。

図 13-9 のように、サブ VI 上の対応する端子に直接接続することによって、特定の入力および出力の値を取り出すこともできます。

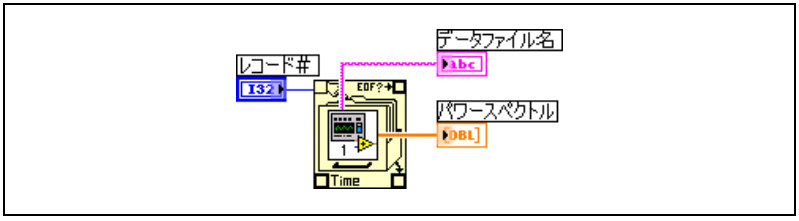


図 13-9　サブ VI の端子を経由した記録済みデータの取り出し

VI を実行すると、サブ VI は実行されません。その代わり、サブ VI のフロントパネルから VI のフロントパネルに記録済みデータがクラスタとして返されます。

レコードを指定する

サブ VI には *n* 個の記録済みレコードがあり、*–n* ～ *n* – 1 の任意の数値をサブ VI の**レコード #** 端子に接続できます。負でないレコード番号を使用し、最初の記録済みレコードを基準としてレコードにアクセスできます。0 は最初のレコードを表し、1 は 2 番目のレコードを表し、最後のレコードを表す *n*–1 まで以下同様に続きます。

負のレコード番号を使用し、最後の記録済みレコードを基準としてレコードにアクセスできます。–1 は最後のレコードを表し、–2 は最後から 2 番目のレコードを表し、最初のレコードを表す *–n* まで以下同様に続きます。*–n* ～ *n* – 1 の範囲外の数値を**レコード #** 端子に接続すると、**無効レコード #** 出力は TRUE になり、サブ VI はデータを取り出しません。

ファイル I/O 関数を使用してフロントパネルデータを取り出す

Read File 関数などのファイル I/O VI および関数を使用して、フロントパネルから記録済みデータを取り出すこともできます。フロントパネルデータログファイル内の各レコードのデータタイプにより、2 つのクラスタが作成されます。一方のクラスタにはタイムスタンプが含まれており、もう一方のクラスタにはフロントパネルデータが含まれています。タイムスタンプクラスタには、秒を表す符号なし 32 ビット整数と、LabVIEW の基準時刻である 1904 年 1 月 1 日午前 0 時から経過したミリ秒を表す符号なし 16 ビット整数が含まれています。

フロントパネルデータログファイルのレコードには、プログラムで作成したデータログファイルへのアクセスに使用するものと同じファイル I/O 関数を使用してアクセスします。図 13-10 のように、File Open 関数へのタイプ入力として**データログタイプ**を入力します。

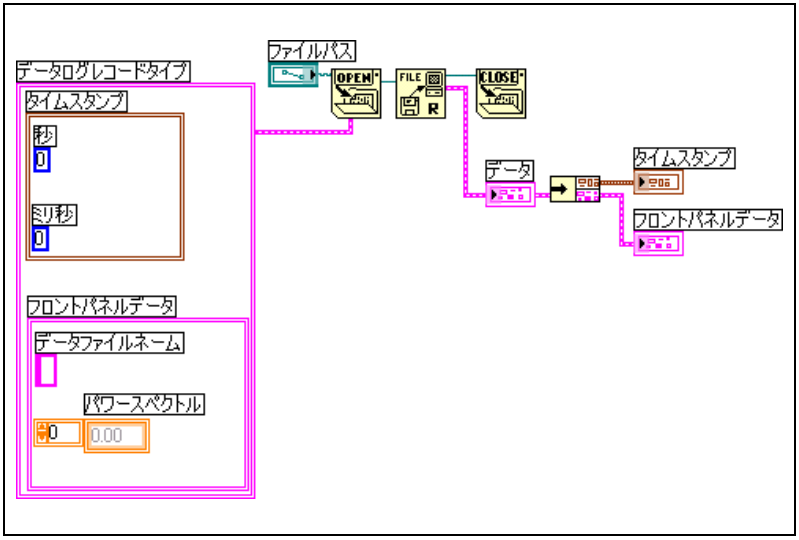


図 13-10　File Open 関数を使用した記録済みデータの取り出し

14

VI の文書化と印刷

LabVIEW を使用して、VI の文書化および印刷を行うことができます。

この章の「VI を文書化する」のセクションでは、開発の任意の段階でブロックダイアグラムやフロントパネルについての情報を VI の印刷文書に記録する方法が説明されています。

この章の「VI を印刷する」のセクションでは、VI の印刷オプションについて説明されています。VI についての情報の印刷に適したオプションと、VI によって生成されたデータおよび結果の報告に適したオプションがあります。いくつかの要素により、使用する印刷方法 (手作業とプログラムのどちらで印刷するかを含む)、レポート形式に必要なオプションの数、作成するスタンドアロンアプリケーションにおける機能の必要性、および VI を実行するプラットフォームが影響を受けます。

この章の「レポートを印刷する」のセクションでは、LabVIEW でレポートを印刷する方法が説明されています。

詳細については

VI の文書化と印刷については、「LabVIEW ヘルプ」参照してください。

VI を文書化する

LabVIEW を使用すると、開発作業の追跡、完成した VI の文書化、および VI のユーザに対する指示の作成を実行できます。文書は、LabVIEW 内で表示したり、印刷したり、HTML または RTF ファイルに保存できます。

メモ　作成したアプリケーション内では、どのような形式でも VI 文書を印刷できません。

印刷した文書を最も効果的に使用するには、VI およびオブジェクトの説明を作成し、VI のレビジョン履歴を設定します。

VI およびオブジェクトの説明を作成する

VI またはオブジェクトの目的を説明したり、その使用方法をユーザに指示するには、VI とそのオブジェクト (制御器や表示器など) の説明を作成します。この説明は、LabVIEW での表示や印刷が可能です。

VI の説明の作成、編集、および表示を行うには、**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**ドキュメント**を選択します。オブジェクトの説明の作成、編集、および表示を行うには、オブジェクトを右クリックし、ショートカットメニューから**説明とヒント**を選択します。ヒントラベルは、カーソルをオブジェクトの上に移動したときに表示される簡単な説明です。**説明とヒント**ダイアログボックスにヒントを入力しないと、ヒントラベルは表示されません。VI のアイコンやオブジェクトの上にカーソルを移動すると**ヘルプ**ウィンドウに VI やオブジェクトの説明が表示されます。**ヘルプ**ウィンドウを表示するには、**ヘルプ→ヘルプを表示**を選択します。VI やオブジェクトの説明は、生成する VI 文書にも表示されます。

VI のレビジョン履歴を設定する

レビジョン番号を含む VI の開発履歴を表示するには、それぞれの VI で**履歴**ウィンドウを使用します。**履歴**ウィンドウでは、VI に対する変更の記録および追跡を行います。**履歴**ウィンドウを表示するには、**ツール→ VI レビジョン履歴**を選択します。レビジョン履歴を印刷することもできます。レビジョン履歴の印刷については、この章の「文書を印刷する」のセクションを参照してください。

レビジョン番号

レビジョン番号を使用すると、VI に対する変更を簡単に追跡できます。レビジョン番号は 0 から始まり、VI を保存するたびに 1 ずつ増加します。ディスクに保存されている現在のレビジョン番号は、**履歴**ウィンドウに表示されます。VI のタイトルバーに現在のレビジョン番号を表示するには、**ツール→オプション**を選択し、一番上のプルダウンメニューから**レビジョン履歴**を選択し、**タイトルバーにレビジョン番号を表示する**チェックボックスをオンにします。

履歴ウィンドウのタイトルバーと VI のタイトルバーに表示される番号は、現在のレビジョン番号に 1 を加えた次のレビジョン番号です。履歴にコメントを追加すると、コメントのヘッダに次のレビジョン番号が含められます。**履歴**ウィンドウのみを変更した場合、VI を保存してもレビジョン番号は増加しません。

レビジョン番号は、**履歴**ウィンドウ内のコメントとは無関係です。コメント間のレビジョン番号の食い違いは、コメントを付けずに VI を保存したことを示します。

厳密に言えば履歴は開発時にのみ使用するため、実行ファイル作成の際、VI のブロックダイアグラムを削除すると、その履歴は LabVIEW によって自動的に削除されます。ブロックダイアグラムの削除については、第 7 章「VI およびサブ VI を作成する」の「VI を配布する」のセクション

を参照してください。**履歴**ウィンドウは、ランタイムバージョンの VI では使用できません。VI にブロックダイアグラムがない場合でも、**VI プロパティ**ダイアログボックスの**一般**ページにはレビジョン番号が表示されます。レビジョン履歴を消去し、レビジョン番号を 0 にリセットするには、**履歴**ウィンドウ内の**リセット**ボタンをクリックします。

文書を印刷する

VI 文書を印刷したり、HTML ファイルまたは RTF ファイルに保存するには、**ファイル→印刷**を選択します。文書の作成は、組み込み形式または、カスタムの形式を選択できます。作成する文書には、以下の項目を含めることができます。

- コネクタペーンとアイコン
- フロントパネルとブロックダイアグラム
- 制御器、表示器、およびデータタイプ
- VI 階層
- サブ VI のリスト
- レビジョン履歴

HTML ファイルまたは RTF ファイルに保存する

VI 文書を HTML ファイルまたは RTF ファイルとして保存できます。HTML ファイルと RTF ファイルはほとんどのワープロアプリケーションにインポートでき、これらのファイルを使用するとコンパイルされたヘルプファイルを作成できます。また、LabVIEW によって生成された HTML ファイルを使用して VI 文書をウェブに表示することもできます。HTML ファイルと RTF ファイルを使用してヘルプファイルを作成する方法については、この章の「独自のヘルプファイルを作成する」のセクションを参照してください。プログラムによる HTML ファイルと RTF ファイルの作成については、第 16 章「プログラム的に VI を 制御する」を参照してください。

RTF ファイルに文書を保存する場合は、オンラインヘルプファイルを作成するか、あるいはワープロ用ファイルを作成するかを指定します。ヘルプファイル形式では、LabVIEW はグラフィックを外部のビットマップファイルに保存します。ワープロファイル形式では、LabVIEW はグラフィックを文書内に埋め込みます。HTML ファイルでは、LabVIEW はすべてのグラフィックを JPEG 形式または PNG 形式で外部に保存します。

HTML ファイル用のグラフィック形式を選択する

HTML ファイルに文書を保存する場合は、グラフィックファイルの形式と色の濃さを選択できます。

JPEG 形式はグラフィックの圧縮率が高い反面、グラフィックの細部が損なわれることがあります。この形式は写真に最も適しています。線画、フロントパネル、およびブロックダイアグラムの場合、JPEG 形式で圧縮するとグラフィックがぼやけたり、色むらが発生することがあります。JPEG グラフィックは常に 24 ビットグラフィックです。色の濃さを低く (モノクロなど) 選択した場合、グラフィックは、指定した濃さで保存されますが 24 ビットグラフィックのままです。

PNG 形式でも、JPEG 形式ほどではありませんが、高圧縮率でグラフィックが圧縮されます。ただし、PNG 圧縮では細部が損なわれません。また、PNG 形式では、1 ビット、4 ビット、8 ビット、および 24 ビットのグラフィックがサポートされています。ビットを下げると、結果のグラフィックは JPEG 形式よりも圧縮率が大幅に高くなります。PNG 形式は Graphics Interchange Format (GIF) 形式に取って代わります。PNG 形式は JPEG や GIF よりも優れていますが、ほとんどの Web ブラウザではサポートが遅れています。

GIF 形式もグラフィックの圧縮性能が優れており、しかも大半のウェブブラウザでサポートされています。ライセンス上の問題があるため、LabVIEW ではグラフィックを圧縮された GIF ファイルとして保存できませんが、将来的にサポートする可能性もあります。LabVIEW が保存する未圧縮の GIF ファイルを圧縮した GIF ファイルに変換するには、グラフィック形式コンバータを使用してください。また、グラフィック形式コンバータを使用して、PNG グラフィックを GIF グラフィックに変換することもできます。PNG グラフィックは元のグラフィックを損なわずに再現するため、PNG グラフィックを変換元にしてください。.gif 拡張子の付いた GIF グラフィックを参照するように VI 文書の保存先の HTML ファイルを変更します。

グラフィックファイルの命名規約

外部グラフィックとともに HTML ファイルまたは RTF ファイルを保存する場合、LabVIEW は、制御器および表示器の端子を同じ名前のグラフィックファイルに保存します。VI に同じタイプの端子が複数ある場合、LabVIEW は各端子を含んでいる 1 つのグラフィックファイルを作成します。たとえば、VI に 32 ビット符号付き整数の入力が 3 つある場合、LabVIEW は 1 つの `ci32.jpg` ファイルを作成します。

独自のヘルプファイルを作成する

LabVIEW によって生成された HTML または RTF ファイルを使用して、独自のコンパイルされたヘルプファイルを作成することができます。**(Windows)** LabVIEW が生成した個別の HTML ファイルを HTML Help ファイルにコンパイルすることができます。

LabVIEW が生成した RTF ファイルを **(Windows)** WinHelp、**(Macintosh)** QuickHelp、または **(UNIX)** HyperHelp ファイルにコンパイルすることができます。

ファイル→ VI プロパティを選択し、**カテゴリ**プルダウンメニューから**ドキュメント**を選択して、VI から HTML ファイルまたはコンパイル済みのヘルプファイルへのリンクを作成することができます。

VI を印刷する

LabVIEW では、主に以下の方法を使用して VI を印刷できます。

- アクティブウィンドウの内容を印刷するには、**ファイル→ウィンドウの印刷**を選択します。
- フロントパネル、ブロックダイアグラム、サブ VI、制御器、VI 履歴などの情報を含む、VI についてのより総合的な情報を印刷するには、**ファイル→印刷**を選択します。この方法による VI の印刷については、この章の「文書を印刷する」のセクションを参照してください。
- VI によって生成された情報の文書によるレポートを印刷するか、または HTML レポートを保存するには、レポート生成 VI を使用します。
- 任意の VI ウィンドウまたは VI 文書をプログラムで随時印刷するには、VI サーバを使用します。この方法による VI の印刷については、第 16 章「プログラム的に VI を 制御する」を参照してください。

アクティブウィンドウを印刷する

アクティブなフロントパネルまたはブロックダイアグラムのウィンドウの内容を最小限のプロンプトで印刷するには、**ファイル→ウィンドウの印刷**を選択します。LabVIEW はアクティブウィンドウの作業領域を印刷しますが、アクティブウィンドウのサイズによる制限はありません。タイトルバー、メニューバー、ツールバー、またはスクロールバーは印刷されません。

ファイル→ウィンドウの印刷を選択した場合や、プログラム的に印刷する場合、LabVIEW による VI の印刷方法を設定するには、**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**印刷オプション**を選択します。プログラムによる印刷については、この章の「プログラム的に印刷する」のセクションを参照してください。

レポートを印刷する

VI によって生成された情報の文書または HTML レポートを印刷するには、**関数→レポート**生成パレットにあるレポート生成 VI を使用します。Easy Text Report VI を使用して基本的なレポートを作成するか、他のレポート生成 VI を使用してより複雑なレポートを生成します。

メモ　HTML ベースのレポート生成は、LabVIEW 開発システムおよびプロフェッショナル開発システムでのみ使用できます。

レポート生成 VI を使用すると、以下のタスクを実行できます。

- レポートのヘッダおよびフッタの設定。
- テキストのフォント、サイズ、スタイル、および色の設定。
- 余白およびタブの設定。
- レポートのどの行またはページにどの情報を印刷するかの決定。
- レポートの向き (縦向きまたは横向き) の設定。
- 他のファイルにあるテキストのレポートへの組み込み。
- 既存のレポートからの情報の消去。
- レポートの自動印刷または HTML レポートの自動保存。
- レポートへのテキスト、グラフィック、または表の追加。
- 印刷後のレポートの破棄。

プログラム的に印刷する

ファイル→ウィンドウの印刷を選択したときや**ファイル→印刷**を選択したときに表示される**印刷**ダイアログボックスを使用して対話形式で印刷するのではなく、プログラムで VI を印刷するには、以下の方法に従ってください。

- VI の実行が終了するたびにそのフロントパネルを自動的に印刷するように VI を設定します。
- VI を印刷するサブ VI を作成します。
- 任意の VI ウィンドウまたは VI 文書をプログラムで随時印刷するには、VI サーバを使用します。この方法による VI の印刷については、第 16 章「プログラム的に VI を 制御する」を参照してください。

終了時に印刷する

VI の実行終了時に VI を印刷するには、**操作→ VI 終了後に印刷**を選択します。**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**印刷オプション**を選択し、**VI が実行を完了するたびに、自動的にパネルを印刷**チェックボックスをオンにすることもできます。

このオプションを選択すると、LabVIEW は、VI の実行が終了するたびにそのフロントパネルの内容を印刷します。VI をサブ VI として使用した場合、LabVIEW は、そのサブ VI の実行が終了したときと、そのサブ VI が呼び出し側 VI に戻る前に印刷します。

サブ VI を使用して終了時に選択的に印刷する

VI の実行が終了するたびに VI を印刷するのではなく、ユーザがボタンをクリックしたときや、ある状況 (テストの失敗など) が発生したときにのみ印刷が必要となる場合があります。また、場合によっては、プリントアウトの形式をより正確に制御したり、制御器のサブセットのみを印刷する必要があります。このような場合は、終了時に印刷するように設定したサブ VI を使用できます。

サブ VI を作成し、希望の印刷方法に合わせてフロントパネルをフォーマットします。VI 内で**操作→ VI 終了後に印刷**を選択する代わりに、サブ VI 内でこれを選択します。印刷するときは、サブ VI を呼び出し、データをサブ VI に渡して印刷します。

その他の印刷方法

LabVIEW の標準的な印刷方法がニーズに合わない場合は、以下に挙げるその他の方法を使用してください。

- データを 1 行ずつ印刷します。シリアルポートにラインプリンタを接続している場合は、シリアル互換性の関数を使用してプリンタにテキストを送信します。通常、これを行うには、プリンタのコマンド言語についての知識が多少必要です。
- Microsoft Excel など、他のアプリケーションにデータをエクスポートし、そのデータをファイルに保存し、そのアプリケーションから印刷します。
- **(Windows および UNIX)** System Exec VI を使用します。
- **(Macintosh)** AESend Print Document VI を使用します。
- **(Windows)** ActiveX オートメーションを使用して別のアプリケーションでデータを印刷します。ActiveX の詳細については、第 18 章「ActiveX」を参照してください。

15

VI をカスタマイズする

アプリケーションのニーズに応じて動作するように VI とサブ VI を構成できます。たとえば、ユーザ入力が必要なサブ VI として VI を使用する場合は、その VI を呼び出すたびにフロントパネルが表示されるように VI を構成します。

VI はさまざまな方法で構成できます。その VI 内で構成するか、あるいは VI サーバを使用してプログラムで構成できます。VI サーバを使用して複数の VI の動作を同様に構成する方法については、第 16 章「プログラム的に VI を 制御する」を参照してください。

詳細については

VI のカスタマイズの詳細については、「LabVIEW ヘルプ」を参照してください。

VI の外観と動作を設定する

VI の外観と動作を設定するには、**ファイル→ VI プロパティ**を選択します。ダイアログボックスの一番上にある**カテゴリ**プルダウンメニューを使用して、さまざまなオプションのカテゴリから選択します。オプションには次のカテゴリがあります。

- **一般**：VI の現在の保存先のパス、VI のレビジョン番号、レビジョン履歴、および VI の最終保存後の変更内容を表示します。このページを使用してアイコンを編集することもできます。
- **ドキュメント**：このページを使用して、VI の説明を追加したり、ヘルプファイルのトピックにリンクします。文書オプションについては、第 14 章「VI の文書化と印刷」の「VI を文書化する」のセクションを参照してください。
- **セキュリティ**：このページを使用して、VI をロックしたり、パスワードで保護します。
- **ウィンドウの外観**：このページを使用して、さまざまなウィンドウの外観を設定します。
- **ウィンドウサイズ**：このページを使用して、ウィンドウのサイズを設定します。

- **実行**：このページを使用して、VI の実行方法を構成します。たとえば、VI を開くとすぐに実行するように構成したり、サブ VI として呼び出されると一時停止するように構成することができます。また、さまざまな優先順位で実行するように VI を構成できます。たとえば、他の操作が完了するのを待たずに VI を実行する必要がある場合、時間重視 (最高) の優先順位で実行するように VI を構成します。マルチスレッド VI の作成については、『Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability』アプリケーションノートを参照してください。

メニューをカスタマイズする

作成するすべての VI に対してカスタムメニューを作成したり、VI のメニューバーの表示／非表示を構成できます。メニューバーの表示／非表示を切り替えるには、**ファイル→ VI プロパティ**を選択し、**カテゴリ**プルダウンメニューから**ウィンドウの外観**ページを選択し、**カスタマイズする**ボタンをクリックし、**メニューバーを表示する**チェックボックスをオンまたはオフにします。

メニューの構成には、メニューの作成と、メニュー項目の選択をする際にその選択に対応するブロックダイアグラムコードを書くという作業が含まれます。

メモ　カスタムメニューは VI の実行中にのみ表示されます。

メニューを作成する

VI の編集時はスタティックに、VI の実行時はプログラムで、カスタムメニューの作成またはデフォルトの LabVIEW メニューの変更を行うことができます。**編集→ランタイムメニュー**を選択し、**メニュー編集**ダイアログボックス内でメニューを作成すると、LabVIEW によってランタイムメニューファイル (`.rtm`) が作成されます。このファイルを使用すると、デフォルトメニューバーではなくカスタムメニューバーが使用できるようになります。`.rtm` ファイルを作成して保存する際に、VI と `.rtm` ファイルの相対パスを同一にしておく必要があります。

カスタムの `.rtm` ファイルと VI を関連付けるには、**メニュー編集**ダイアログボックスを使用します。VI を実行すると、VI はその `.rtm` ファイルからメニューを読み込みます。**メニュー編集**ダイアログボックスを使用して、LabVIEW がデフォルトメニューで提供するメニュー項目であるアプリケーション項目、またはユーザが追加するメニュー項目であるユーザ項目を持つカスタムメニューを作成することができます。LabVIEW は、アプ

リケーション項目の動作を定義しますが、ユーザ項目の動作はユーザがブロックダイアグラムで制御します。ユーザメニュー項目の選択の詳細については、本章の「メニュー選択処理」のセクションを参照してください。

メニュー編集ダイアログボックスを使用して、VI の編集時にメニューをカスタマイズします。**関数→アプリケーション制御→メニュー**パレットにあるメニュー関数を使用して、実行時にメニューをプログラム的にカスタマイズします。これらの関数を使用して、ユーザ項目の属性を挿入、削除、変更することができます。アプリケーション項目を追加または削除するだけで、LabVIEW がアプリケーション項目の動作および状態を定義します。

メニュー選択処理

カスタムメニューを作成するときは、大文字と小文字の区別がない固有の文字列識別子 (であるタグ) を各メニュー項目に割り当てます。ユーザがメニュー項目を選択したときは、Get Menu Selection 関数を使用して、そのタグをプログラムで取り出します。LabVIEW により、各メニュー項目のタグの値に基づいて、各メニュー項目のブロックダイアグラム上にハンドラが提供されます。ハンドラは While ループと Case ストラクチャの組み合わせです。これを使用すると、どのメニューが選択されているかを調べて適切なコードを実行します。

カスタムメニューを作成した後、そのメニュー内の各項目を実行または処理する Case ストラクチャをブロックダイアグラムで作成します。このプロセスはメニュー選択処理と呼ばれます。LabVIEW は、すべてのアプリケーション項目を自動的に処理します。

図 15-1 では、ユーザが選択したメニュー項目を Get Menu Selection 関数が読み込んで Case ストラクチャに渡し、Case ストラクチャでメニュー項目が実行されます。

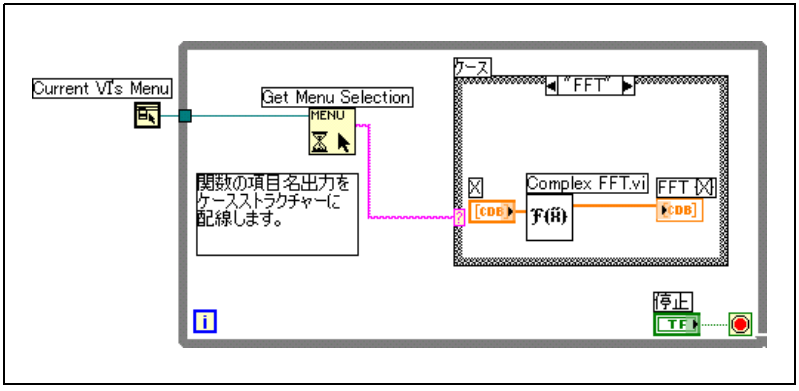


図 15-1　メニュー処理を使用したブロックダイアグラム

あるメニュー項目の処理に長時間かかることがわかっている場合は、Get Menu Selection 関数の**ブロックメニュー**入力にブール制御器を接続し、そのブール制御器を TRUE に設定してメニューバーを無効にします。これにより、LabVIEW がメニュー項目を処理している間はユーザがメニュー上で他の項目を選択できなくなります。LabVIEW によるメニュー項目の処理が完了した後、Enable Menu Tracking 関数に TRUE 値を接続してメニューバーを有効にしてください。

16

プログラム的に VI を制御する

ブロックダイアグラム、ActiveX テクノロジ、および TCP プロトコルによって VI サーバにアクセスして、VI や他の LabVIEW のインスタンスと通信できます。このようにして、プログラム的に VI および LabVIEW を制御できます。VI サーバの操作は、ローカルコンピュータ上やネットワークを介してリモートで実行できます。

詳細については

プログラムで VI を制御する方法の詳細については、「LabVIEW ヘルプ」を参照してください。

VI サーバの機能

VI サーバを使用すると、以下のプログラムによる操作を実行できます。

- リモートで VI を呼び出します。
- ウェブ上で LabVIEW の他のインスタンスから呼び出せる VI をエクスポートするサーバとして、LabVIEW のインスタンスを構成します。たとえば、遠隔地でデータを集録して記録するデータ集録アプリケーションを持っていれば、いつでもローカルコンピュータからそのデータをサンプリングできます。LabVIEW 環境設定を変更してウェブ上で VI にアクセスできるようにすることによって、最新のデータの転送がサブ VI 呼び出しと同じように簡単になります。ネットワークの細部は VI サーバによって処理されます。また、VI サーバはプラットフォームにまたがって実行可能なので、異なるプラットフォーム上でクライアントとサーバを実行できます。
- VI および LabVIEW のプロパティを編集します。たとえば、ダイナミックに VI ウィンドウの位置を設定したり、フロントパネルの特定の場所が表示されるようにスクロールできます。また、プログラム的にディスクにすべての変更内容を保存できます。
- **ファイル→ VI プロパティ**ダイアログボックスを使用して各 VI について手作業で更新せずに、複数の VI のプロパティを更新します。

- バージョン番号や改版などの LabVIEW のインスタンスの情報を取り出します。また、LabVIEW を実行しているプラットフォームなどの環境情報も取り出すことができます。
- VI を開くときにすべてのサブ VI をロードするのではなく、他の VI で呼び出す必要がある場合に、ダイナミックにそれらの VI をメモリにロードします。
- アプリケーションのプラグインアーキテクチャを作成することにより、カスタマに配布した後でもアプリケーションに機能を追加できるようになります。たとえば、データをフィルタ処理する VI のセットがあり、その VI すべてが同じパラメータを使用するとします。これらの VI が 1 つのプラグインディレクトリからダイナミックにロードされるように設計したアプリケーションに、これらの VI のセットの一部だけを組み込んだ状態で出荷し、プラグインディレクトリに新しいフィルタ処理 VI をロードするだけでユーザのフィルタ処理オプションをさらに拡張できます。

VI サーバアプリケーションを作成する

VI サーバアプリケーションのプログラミングモデルは refnum をベースにしています。refnum はファイル I/O、ネットワーク接続、および LabVIEW の他のオブジェクトでも使用されます。refnum の詳細については、第 4 章「フロントパネルを作成する」の「オブジェクトまたはアプリケーションへのリファレンス」のセクションを参照してください。

通常、まず LabVIEW のインスタンスまたは VI の refnum を開きます。次に、他の VI に対するパラメータとして refnum を使用します。VI はプロパティを取得 (読み取り) または設定 (書き込み) し、メソッドを実行して、参照されている VI をダイナミックにロードします。最後に、refnum を閉じ、参照されている VI をメモリから解放します。

VI サーバアプリケーションを作成する場合は、**関数→アプリケーション制御**パレットにある以下の関数およびノードを使用します。

- **Open Application Reference**：ローカルアプリケーションまたはサーバを介してアクセスしているリモートアプリケーションの refnum を開いたり、refnum を開いて LabVIEW のリモートインスタンスにアクセスします。ローカルコンピュータまたはリモートコンピュータ上の VI にアクセスするには、Open VI Reference 関数を使用します。
- **Property Node**：VI やアプリケーションのプロパティを取得または設定します。プロパティの詳細については、この章の「プロパティノード」のセクションを参照してください。

- **Invoke Node**：VI やアプリケーション上のメソッドを呼び出します。メソッドの詳細については、この章の「インボークノード (Invoke Node)」のセクションを参照してください。
- **Call By Reference Node**：参照されている VI をダイナミックにロードします。
- **Close LV Object Reference**：VI サーバを使用してアクセスした VI またはアプリケーションの refnum を閉じます。

アプリケーションリファレンスと VI リファレンス

VI サーバ機能へのアクセスは、アプリケーションオブジェクトと VI オブジェクトの 2 つの主要クラスのオブジェクトのリファレンスを通じて行います。これらのオブジェクトのいずれかに対するリファレンスを作成すると、そのオブジェクト上で操作を実行する VI や関数にそのリファレンスを渡すことができます。

アプリケーション refnum は LabVIEW のローカルまたはリモートのインスタンスを参照します。アプリケーションプロパティおよびメソッドを使用して、LabVIEW の環境設定を変更したり、システム情報を返すことができます。VI の refnum は LabVIEW のインスタンスの VI を参照します。

LabVIEW のインスタンスの refnum を使用すると、LabVIEW を実行しているプラットフォーム、バージョン番号、現在メモリにロードされているすべての VI のリストなどの LabVIEW の環境についての情報を取り出すことができます。現在のユーザ名や LabVIEW の他のインスタンスにエクスポートした VI のリストなどの情報を設定することもできます。

VI の refnum を取得すると、メモリに VI がロードされます。refnum の取得後、その refnum を閉じるまでメモリに VI が常駐します。開いている 1 つの VI について複数の refnum が同時に存在する場合は、その VI のすべての refnum を閉じるまで、その VI はメモリに常駐します。VI の refnum を使用すると、フロントパネルウィンドウの位置などのダイナミックなプロパティだけでなく、**ファイル→ VI プロパティ**ダイアログボックスで操作できるその VI のすべてのプロパティを更新できます。また、プログラムで VI を印刷、別の場所に保存、およびその文字列をエクスポートおよびインポートして他の言語に変換することもできます。

アプリケーションと VI の設定値を操作する

VI サーバを使用したアプリケーションや VI の設定値の取得および設定は、プロパティノードやインボークノードを使用して行います。プロパティノードやインボークノードを使用しないと、多くのアプリケーションや VI の設定値を取得したり設定することができません。

アプリケーションクラスと VI クラスのプロパティおよびメソッドの使用例については、`examples\viserver` を参照してください。

プロパティノード

アプリケーションや VI のさまざまなプロパティを取得または設定するには、プロパティノードを使用します。プロパティノードからプロパティを選択するには、操作ツールでプロパティ端子をクリックするか、そのノードの白い領域を右クリックし、ショートカットメニューから**プロパティ**を選択します。

1 つのノードで、複数のプロパティの読み取りまたは書き込みができます。プロパティノードをサイズ変更して新しい端子を追加するには、位置決めツールを使用します。小さな矢印がプロパティの右側にあるものはプロパティの読み取り、左側にあるものはプロパティの書き込みになります。プロパティの動作を変更するには、そのプロパティを右クリックし、ショートカットメニューから**読み取りに変更**または**書き込みに変更**を選択します。

ノードは上から下の順に実行されます。プロパティノードは実行前にエラーが発生すると実行されません。エラーが発生したかどうかを常に確認してください。プロパティにてエラーが発生すると、LabVIEW は残りのプロパティを無視し、エラーを返します。**エラー出力**クラスタには、どのプロパティによってエラーが発生したのかを示す情報が含まれています。

プロパティノードの自動リンク

フロントパネルオブジェクトからプロパティノードを作成する (オブジェクトを右クリックしてショートカットメニューから**作成→プロパティノード**を選択) 場合、LabVIEW は、フロントパネルオブジェクトに自動的にリンクされたブロックダイアグラムにプロパティノードを作成します。これらのプロパティノードは自動的に作成元のオブジェクトにリンクされるので、**refnum** 入力はありません。また、プロパティノードをフロントパネルオブジェクトの端子や制御器 refnum に配線する必要はありません。制御器 refnum の詳細については、この章の「フロントパネルオブジェクトを制御する」のセクションを参照してください。

インボークノード (Invoke Node)

アプリケーションや VI に対して動作 (メソッド) を実行するにはインボークノードを使用します。プロパティノードとは異なり、1 つのインボークノードではアプリケーションや VI に対して 1 つのメソッドしか実行できません。メソッドを選択するには、操作ツールでメソッド端子をクリックするか、そのノードの白い領域を右クリックしてショートカットメニューから**メソッド**を選択します。

メソッド名は常にインボークノードのパラメータリストの最初の端子名になります。メソッドによって値が返される場合は、メソッド端子には返される値が表示されます。それ以外の場合、メソッド端子に値は表示されません。

インボークノードによって上から下にパラメータがリストされ、メソッド名が上に、オプションのパラメータが下にグレーで表示されます。

アプリケーションクラスのプロパティとメソッドを操作する

LabVIEW のローカルまたはリモートのインスタンス上でプロパティを取得または設定したり、LabVIEW 上でメソッドを実行したり、またはその両方を行うことができます。図 16-1 は、ローカルコンピュータ上のメモリに入っているすべての VI をフロントパネル上の文字列配列に表示する方法を示しています。

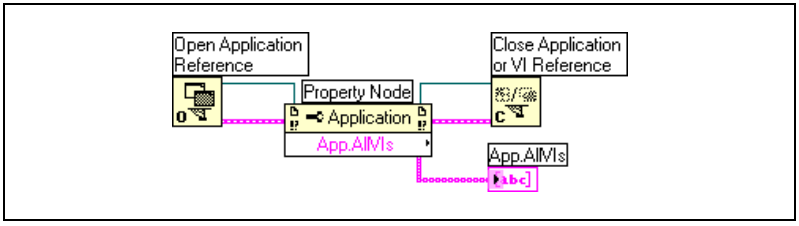


図 16-1　ローカルコンピュータ上のメモリ内の VI をすべて表示する

リモートコンピュータ上のメモリにある VI を検索するには、図 16-2 のように Open Application Reference 関数の**マシン名**入力に文字列制御器を配線し、IP アドレスやドメイン名を入力します。また、図 16-1 で使用されている **All Vis in Memory** プロパティは LabVIEW のローカルインスタンスにのみ適用されるので、**Exported VIs in Memory** プロパティを選択する必要があります。

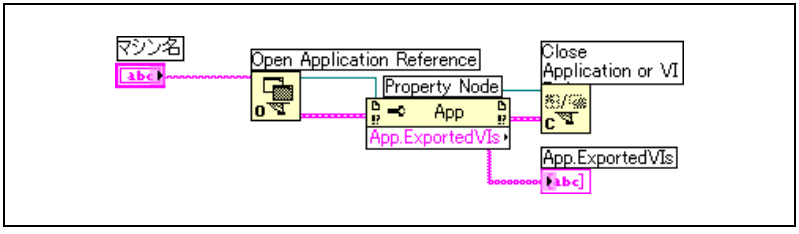


図 16-2　リモートコンピュータ上のメモリ内の VI をすべて表示する

VI クラスのプロパティとメソッドを操作する

VI のプロパティの取得や設定、VI 上でのメソッドの実行、またはその両方を実行できます。図 16-3 のように、LabVIEW はインボークノードを使用して VI のフロントパネルオブジェクトをそれぞれのデフォルト値にもう一度初期化します。フロントパネルが開き、プロパティノードを使用してデフォルト値が表示されます。

VI のプロパティやメソッドにアクセスするには、Open VI Reference 関数を使用してアクセスする VI の refnum を作成する必要があります。VI 上のメソッドを呼び出すにはインボークノードを使用します。

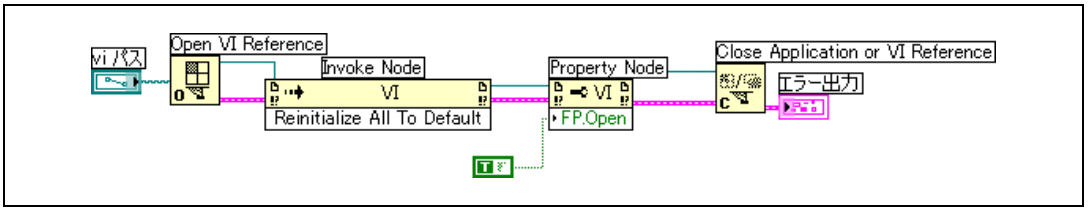


図 16-3　VI クラスのプロパティとインボークノードを使用する

Open VI Reference 関数をインボークノードに配線すると、すべての VI クラスメソッドにアクセスできます。

プロパティノードはインボークノードと同様に動作します。プロパティノードに VI refnum を配線すると、すべての VI クラスプロパティにアクセスできます。

アプリケーションと VI クラスのプロパティおよびメソッドを操作する

VI によっては、アプリケーションと VI クラスの両方のプロパティまたはメソッドにアクセスする必要があります。図 16-4 のように、アプリケーションおよび VI クラスの refnum を別々に開いたり、閉じたりする必要があります。

図 16-4 は、ローカルコンピュータ上のメモリに入っている VI のパスをフロントパネルにある文字列の表示器の配列に表示する方法を示しています。メモリ内のすべての VI を見つけるには、アプリケーションクラスプロパティにアクセスする必要があります。これらの各 VI へのパスを調べるには、VI クラスのプロパティにアクセスする必要があります。For ループの実行回数はメモリ内の VI 数によって決まります。メモリ内の各 VI に対して VI refnum が必要なので、Open VI Reference 関数および Close LV Object Reference 関数を For ループ内部に入れる必要があります。アプリケーション refnum は、For ループがすべての VI のパス検索を終了した後で閉じてください。

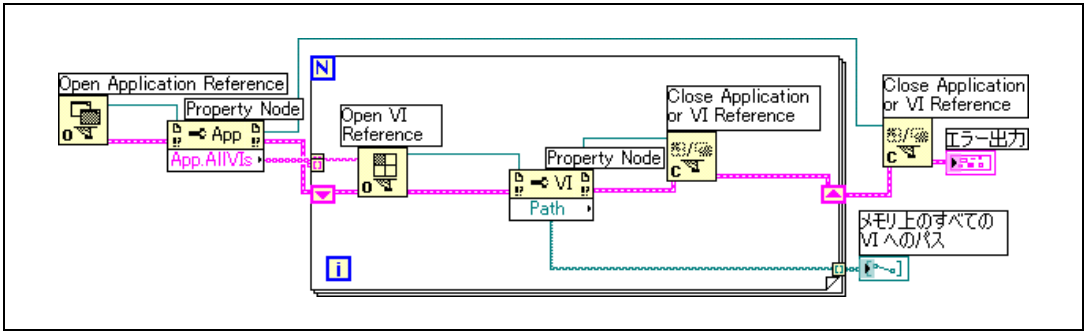


図 16-4　アプリケーションおよび VI クラスのプロパティとメソッドを使用する

VI のダイナミックなロードと呼び出し

スタティックにリンクされたサブ VI を呼び出さずに、ダイナミックに VI を呼び出すことができます。スタティックにリンクされたサブ VI は呼び出し側 VI のブロックダイアグラム上に直接配置するものです。このサブ VI は呼び出し側 VI のロードと同時にロードされます。

スタティックにリンクされるサブ VI とは異なり、ダイナミックにロードされるサブ VI は呼び出し側 VI がそのサブ VI の呼び出しを実行するまでロードされません。サブ VI は呼び出し側 VI で必要になるまでロードされないので、大量の呼び出し側 VI がある場合、サブ VI をダイナミックにロードすることによって時間とメモリを節約できます。さらに、操作が終了するとサブ VI をメモリから解放できます。

Call By Reference Node と厳密に類別化された VI Refnum

VI をダイナミックに呼び出すには、Call By Reference Node を使用します。

Call By Reference Node には厳密に類別化された VI refnum が必要です。厳密に類別化された VI refnum によって、呼び出す VI のコネクタペーンが識別されます。厳密に類別化された VI refnum は、永久的な VI への関連を確立したり、VI の名前や位置などの他の情報を持つわけではありません。Call By Reference Node の入力と出力は、他の VI に配線する場合と同じように配線できます。

図 16-5 は、Call By Reference Node を使用して Frequency Response VI をダイナミックに呼び出す方法を示しています。Call By Reference Node には Open VI Reference 関数と Close LV Object Reference 関数を使用する必要があります。これは、プロパティノードやインボークノードを使用する場合と似ています。ただし、Call By Reference Node では、厳密に類別化された VI refnum を Open VI

Reference 関数の**タイプ識別子 VI Refnum** 入力に配線する必要があります。まず、**制御器→ Refnum → VI Refnum** を選択し、そのフロントパネルに貼り付けます。次に、VI Refnum を右クリックし、ショートカットメニューから **VI サーバクラスを選択→参照**を選択します。**開く VI を選択してください**という名称のダイアログボックスで VI を選択します。

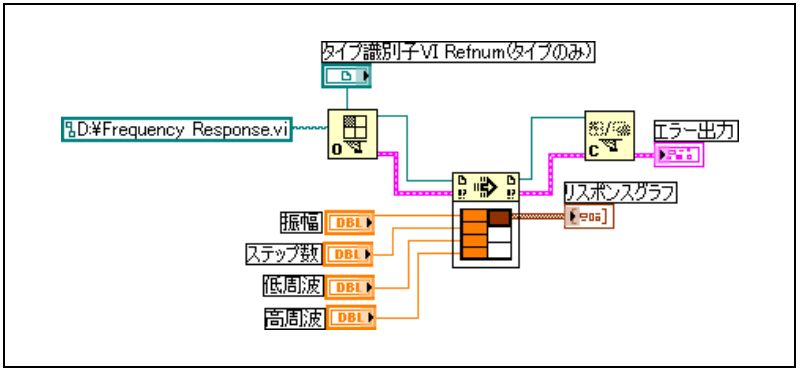


図 16-5　Call By Reference Node を使用する

厳密に類別化された refnum として指定された VI は、コネクタペーン情報のみを提供します。このように refnum と VI 間には固定された関連性がありませんので、これだけではその VI 全ての情報は得られません。Call By Reference Node 関数は、選択する VI の refnum を取得することで VI コネクタペーンの情報を取得し、その他の情報を得るために Open VI Reference 関数の **VI パス**入力で指定された VI パスから情報を引き出します。この方法で Call By Reference Node に必要とする情報を提供することができます。

リモートコンピュータ上での VI の編集と実行

アプリケーションと VI refnum の両方で重要な点は、それらのネットワークにおける透過性です。つまり、オブジェクトの refnum は、リモートコンピュータでもユーザのコンピュータで開く場合と同じ方法で開くことができます。

いくつかの制約はありますが、リモートオブジェクトの refnum を開くと、ローカルオブジェクトとほとんど同じ方法でリモートオブジェクトを処理できます。リモートのオブジェクトでの操作については、VI サーバがネットワーク上での操作に関する情報を送信し、結果を返します。アプリケーションは操作がリモートであるかローカルであるかにかかわらず、見た目には同じように機能します。

フロントパネルオブジェクトを制御する

制御器→ Refnum および**制御器→旧バージョンの制御器→ Refnum** パレットにある制御器 refnum を使用して、フロントパネルオブジェクトのリファレンスを他の VI に渡します。サブ VI に制御器 refnum を渡すと、プロパティノードおよびインボークノードを使用して、プロパティの読み書き、参照されるフロントパネルオブジェクトのメソッドの呼び出しを実行できます。イベントを使用して、フロントパネルオブジェクトからプログラム的にブロックダイアグラムの動作を制御する方法については、第 8 章「ループと Case ストラクチャ」の「Case ストラクチャとシーケンスストラクチャ」のセクションを参照してください。

厳密に類別化された制御器 refnum と非厳密に類別化された制御器 refnum

厳密に類別化された制御器 refnum は、同じ種類のデータの制御器 refnum のみを受け入れます。たとえば、厳密に類別化された制御器 refnum のタイプが 32 ビット整数のスライドである場合は、制御器 refnum 端子に 32 ビット整数、8 ビット整数、または倍精度スカラのスライドを配線できます。ただし、32 ビット整数クラスタのスライドは配線できません。

制御器から作成する制御器 refnum はデフォルトで、厳密に類別化された制御器 refnum となります。フロントパネル上の制御器 refnum の左下の隅の赤い星は、その制御器 refnum が、厳密に類別化された制御器 refnum であることを示します。ブロックダイアグラムでは、制御器 refnum 端子に配線されたプロパティノードやインボークノード上に `(strict)` が表示されます。これは、この制御器 refnum が、厳密に類別化された制御器 refnum であることを示します。

メモ　ラッチ式機械的動作は厳密に類別化された制御器 refnum と互換性がないので、ラッチ式機械的動作を行うブール制御器では非厳密に類別化された制御器 refnum が生成されます。

非厳密に類別化された制御器 refnum は、より広範囲のデータのタイプを受け入れます。たとえば、非厳密に類別化された制御器 refnum がスライドである場合、制御器 refnum 端子には 32 ビット整数スライド、単精度スライド、または 32 ビット整数スライドのクラスタのいずれかを配線できます。非厳密に類別化された制御器 refnum のタイプが制御器である場合、すべてのタイプの制御器の制御器 refnum を制御器 refnum 端子に配線できます。

メモ　非厳密に類別化された制御器 refnum 端子にプロパティノードを配線する場合、値によってバリアントデータが生成されます。バリアントデータは使用前に変換する必要がある場合があります。チャートの History Data プロパティは、チャートの refnum が厳密に類別化されている場合にのみ使用できます。バリアントデータの詳細については、第 5 章「ブロックダイアグラムを作成する」の「バリアントデータを処理する」のセクションを参照してください。

17

LabVIEW のネットワーク動作

VI は、他のアプリケーションやリモートコンピュータで実行中の他のプロセスとも通信 (ネットワーク動作) を行うことができます。以下のタスクを実行するには、LabVIEW のネットワーク動作機能を使用します。

- ナショナルインスツルメンツの DataSocket テクノロジを使用して、ネットワーク上で実行中の他の VI とライブデータを共有します。
- ウェブ上にフロントパネル画像および VI ドキュメントをパブリッシュします。
- TCP、UDP、DDE、AppleEvents、PPCToolbox などの低レベルのプロトコルによって、他のアプリケーションや VI と通信する VI を作成します。

詳細については

LabVIEW のネットワーク動作の詳細については、「LabVIEW ヘルプ」を参照してください。

ファイル I/O、VI サーバ、ActiveX、およびネットワーク動作から選択する

アプリケーションによっては、ネットワーク化が最良のソリューションにならない場合もあります。他の VI やアプリケーションが読み込めるデータを含んでいるファイルを作成する場合は、ファイル I/O の VI および関数を使用します。ファイル I/O VI および関数の使用方法の詳細については、第 13 章「ファイル I/O」を参照してください。

他の VI を制御する場合は、VI サーバを使用します。ローカルおよびリモートコンピュータ上にある VI および他の LabVIEW アプリケーションの制御方法の詳細については、第 16 章「プログラム的に VI を 制御する」を参照してください。

(Windows) Excel スプレッドシートへの波形グラフの埋め込みなどマイクロソフト社のアプリケーションの機能にアクセスする場合は、ActiveX VI および関数を使用します。ActiveX が有効なアプリケーションへのア

クセス、および他の ActiveX アプリケーションの LabVIEW へのアクセスを許可する方法の詳細については、第 18 章「ActiveX」を参照してください。

ネットワークのクライアントおよびサーバとしての LabVIEW

LabVIEW をクライアントとして使用すると、別のアプリケーションの機能を使用したり、データをサブスクライブ (読み取り) でき、またサーバとして使用すると LabVIEW の機能を他のアプリケーションで使用可能にすることができます。VI サーバを使用してローカルおよびリモートコンピュータ上の VI を制御する方法の詳細については、第 16 章「プログラム的に VI を 制御する」を参照してください。プロパティノードを使用してプロパティにアクセスしたり、インボークノードを使用してメソッドを呼び出すことによって、VI を制御します。

プロパティにアクセスしたり他のアプリケーションのメソッドを呼び出すには、プロパティやメソッドへのアクセスに使用するネットワークプロトコルをあらかじめ確立しておく必要があります。使用できるプロトコルには、HTTP や TCP/IP などがあります。選択するプロトコルはアプリケーションによって異なります。たとえば、HTTP プロトコルはウェブへのパブリッシュに適していますが、他の VI が作成したデータを受信する VI を作成する場合は使用できません。これを行うには、TCP プロトコルを使用します。

LabVIEW がサポートしている通信プロトコルの詳細については、この章の「低レベル通信アプリケーション」のセクションを参照してください。

(Windows) LabVIEW を ActiveX サーバまたはクライアントとして使用する ActiveX テクノロジの使用方法の詳細については、第 18 章「ActiveX」を参照してください。

DataSocket テクノロジを使用する

ウェブ上またはローカルコンピュータ上の他の VI やナショナルインスツルメンツの ComponentWorks などのアプリケーションとの間でライブデータを共有するには、ナショナルインスツルメンツの DataSocket テクノロジを使用します。DataSocket によって確立されている通信プロトコルがまとめられ、測定や自動化が可能になります。これは、ウェブブラウザによって、異なるインターネットテクノロジがまとめられることによく似ています。

DataSocket テクノロジにより、**DataSocket 接続**ダイアログボックスを通じてフロントパネルから複数の入出力メカニズムにアクセスできます。**DataSocket 接続**ダイアログボックスを表示するには、フロントパネルオブジェクトを右クリックし、ショートカットメニューから**データ操作→ DataSocket 接続**を選択します。URL の指定によって、データのパブリッシュ (書き込み) またはサブスクライブ (読み込み) を行います。これはウェブブラウザで URL を指定する方法とよく似ています。

たとえば、フロントパネル上の温度表示器のデータをウェブ上の他のコンピュータと共有する場合、**DataSocket 接続**ダイアログボックスで URL を指定することによって温度計のデータをパブリッシュします。他のコンピュータ上のユーザは自分のフロントパネル上に温度計を配置し、**DataSocket 接続**ダイアログボックスで URL を選択することによって、そのデータをサブスクライブします。フロントパネル上での DataSocket テクノロジの使用方法の詳細については、この章の「フロントパネル上で DataSocket を使用する」のセクションを参照してください。

DataSocket テクノロジの詳細については、『Integrating the Internet into Your Measurement System』ホワイトペーパーを参照してください。このホワイトペーパーは、インストール CD の `manuals` ディレクトリまたはナショナルインスツルメンツのウェブサイト `ni.com` から PDF 形式で提供されています。

URL を指定する

URL では、`dstp`、`ftp`、`file` などの通信プロトコルを使用してデータが転送されます。URL で使用するプロトコルは、パブリッシュするデータのタイプとネットワークがどのように構成されているかによって異なります。

DataSocket を使用してデータをパブリッシュしたりデータをサブスクライブする場合は、以下のプロトコルを使用できます。

- **DataSocket 転送プロトコル (`dstp`)**：DataSocket 接続のネイティブプロトコルです。このプロトコルを使用する場合、VI は DataSocket サーバと通信します。データには名前の付いたタグを付ける必要がありますが、これは URL に追加されます。DataSocket 接続では名前付きのタグを使用して、DataSocket サーバ上の特定のデータ項目をアドレス指定します。このプロトコルを使用するには、DataSocket サーバを実行する必要があります。
- **(Windows) OLE for Process Control (`opc`)**：自動製造工程によって生成されるデータのようなリアルタイムの生産データを共有するように特別に設計されています。このプロトコルを使用するには、OPC サーバを実行する必要があります。

- **(Windows) `logos`**：ネットワークとローカルコンピュータ間のデータ転送に使用されるナショナルインスツルメンツ社独自のテクノロジです。
- **ファイル転送プロトコル (`ftp`)**：このプロトコルを使用すると、データの読み込み元のファイルを指定できます。
- **`file`**：このプロトコルを使用すると、データが含まれているローカルまたはネットワークのファイルをリンクできます。

表 17-1 は、各プロトコル URL の例を示しています。

表 17-1　DataSocket URL の例

URL	例
dstp	dstp://servername.com/numeric、numeric はデータの名前付きのタグです。
opc	opc:\National Instruments.OPCTest\item1 opc:\\machine\National Instruments.OPCModbus\Modbus Demo Box.4:0 opc:\\machine\National Instruments.OPCModbus\Modbus Demo Box.4:0?updaterate=100&deadband=0.7
logos	logos://computer_name/process/data_item_name
ftp	ftp://ftp.ni.com/datasocket/ping.wav
file	file:ping.wav file:c:\mydata\ping.wav file:\\machine\mydata\ping.wav

ライブデータを共有するには、dstp、opc、および logos の URL を使用します。これは、これらのプロトコルがリモートおよびローカルの制御器および表示器を更新できるからです。ファイルからデータを読み取るには、ftp および file の URL を使用します。これは、これらのプロトコルがリモートおよびローカルの制御器および表示器を更新できないからです。

DataSocket 接続の使用例については、examples/comm/datasktx.llb ファイルを参照してください。

DataSocket でサポートされているデータ形式

DataSocket を使用すると、以下のデータをパブリッシュしたりサブスクライブできます。

- **未処理テキスト**：文字列を文字列表示器に渡す場合は未処理テキストを使用します。
- **タブ付きテキスト**：スプレッドシート内でデータを配列としてする。パブリッシュする場合は、タブ付きテキストを使用します。LabVIEW によってタブ付きデータがデータ配列に変換されます。
- **wav データ**：VI または関数にサウンドをパブリッシュする場合は、.wav データを使用します。
- **バリアントデータ**：ComponentWorks 制御器など他のアプリケーションからデータをサブスクライブするにはバリアントデータを使用します。

フロントパネル上で DataSocket を使用する

フロントパネルオブジェクト内のデータをパブリッシュしたりサブスクライブするには、フロントパネル DataSocket 接続を使用します。フロントパネルオブジェクトのデータを他のユーザと共有する場合はデータをパブリッシュします。パブリッシュされているデータを取り出してフロントパネル上に表示する場合はデータをサブスクライブします。

DataSocket 接続は、DataSocket 接続をブロックダイアグラムプログラムを使用しないで直接フロントパネルから使用できる点で、ウェブサーバや ActiveX 接続とは異なります。各フロントパネル制御器や表示器では、それぞれの DataSocket 接続を介してデータのパブリッシュやサブスクライブが可能です。フロントパネル DataSocket 接続ではデータのみパブリッシュし、フロントパネル制御器のグラフィックはパブリッシュしません。したがって、DataSocket 接続を介してサブスクライブする VI は独自のデータ操作を実行できます。

フロントパネル上でフロントパネル制御器の値を直接設定してからデータをパブリッシュしたり、ブロックダイアグラムを作成して VI や関数の出力を表示器に配線し、その表示器からデータをパブリッシュできます。制御器および表示器とともに DataSocket 接続を使用する通常のシナリオは以下のとおりです。

- 他のユーザが制御器や表示器を介してサブスクライブできるように、制御器を操作してデータをパブリッシュするには、フロントパネル制御器から値をパブリッシュします。たとえば、フロントパネルに温度を調整するノブを配置すると、他のコンピュータのユーザはそのデータをサブスクライブしてサブ VI や関数に配線された制御器で使用したり、データを表示器に表示できます。制御器によって DataSocket

接続からデータをサブスクライブしながら、その VI 上のフロントパネル制御器を操作することはできません。

- 他のユーザがデータをサブスクライブして自分のフロントパネルにある制御器または表示器にパブリッシュしたり、サブ VI や関数の入力に配線されている制御器で結果をデータとして使用できるようにするには、フロントパネル表示器に表示されている値をパブリッシュします。たとえば、平均温度を計算してフロントパネルにある温度計に平均温度を表示する VI は、温度データをパブリッシュできます。
- 他の VI のフロントパネルにある制御器や表示器に表示されているデータを自分の VI のフロントパネル表示器に表示するには、フロントパネル表示器からの値をサブスクライブします。
- 他の VI のフロントパネルにある制御器や表示器に表示されているデータを自分の VI のフロントパネル制御器に表示するには、フロントパネル制御器からの値をサブスクライブします。制御器でデータをサブスクライブする場合、サブ VI や関数の入力に制御器を配線すると VI 内でそのデータを使用できます。
- 他のユーザの VI のフロントパネルから自分の VI のフロントパネルにある制御器を操作できるようにするには、自分の VI のフロントパネル制御器をパブリッシュおよびサブスクライブできるようにします。VI を実行すると、VI のフロントパネル制御器によって、DataSocket 接続を介して他の VI またはアプリケーションからパブリッシュされる現在の値が自分の VI のフロントパネル制御器に取り出されます。他のユーザがフロントパネルにある制御器の値を変更すると、DataSocket 接続によって自分の VI のフロントパネル制御器に新しい値がパブリッシュされます。自分のフロントパネル制御器の値を操作すると、自分の VI が他のユーザのフロントパネルに値をパブリッシュします。

データをサブスクライブしているフロントパネルオブジェクトは、データをパブリッシュするオブジェクトと同じものである必要はありません。ただし、数値データタイプは同じである必要があります。数値データの場合は、データタイプが異なると強制的に変換されます。たとえば、VI でタイプが整数のデジタル表示器を用意して、別の VI の温度計が生成した浮動小数点数のデータを読み取ることができますが、この際、データタイプが整数に変換されます。

フロントパネルの DataSocket 接続の主な目的はライブデータの共有です。ローカルファイル、FTP サーバおよび Web サーバにデータを読み取るには、**関数→通信→ DataSocket** パレットにある DataSocket Read 関数、**関数→ファイル I/O** パレットにあるファイル I/O VI および関数、または**関数→アプリケーション制御**パレットにあるアプリケーション制御 VI および関数を使用します。ナショナルインスツルメンツでは、ライブ

フロントパネルを更新する場合は、未処理テキストおよびバリアントデータのみをパブリッシュすることをお勧めします。

ブロックダイアグラムでライブデータの読み書きを行う

ブロックダイアグラムから、**関数→通信→ DataSocket** パレットにある DataSocket 関数を使用してプログラムでデータの読み取りまたは書き込みができます。

DataSocket 接続を介してプログラムでライブデータを書き込むには、DataSocket Write 関数を使用します。図 17-1 は、数値を書き込む方法を示しています。

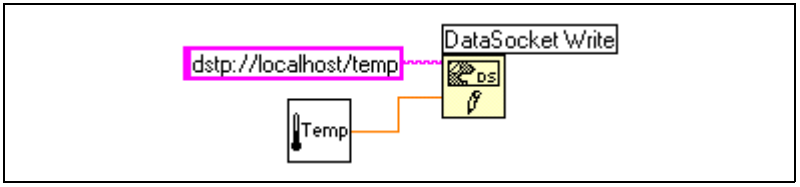


図 17-1　DataSocket Write を使用してデータをパブリッシュする

DataSocket Write 関数は多形性なので、ほとんどのデータタイプを**データ**入力に配線できます。多形性 VI や関数の詳細については、第 5 章「ブロックダイアグラムを作成する」の「多形性 VI および関数」のセクションを参照してください。

DataSocket 接続からプログラムでライブデータを読み取るには、DataSocket Read 関数を使用します。図 17-2 は、データを読み取って倍精度浮動小数点数に変換する方法を示しています。

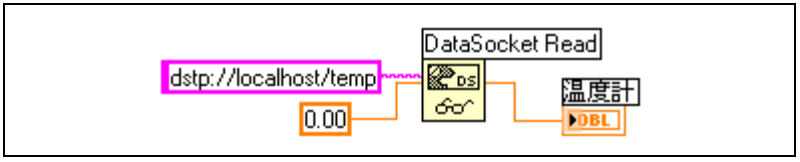


図 17-2　DataSocket Read を使用して 1 つの値を読み取る

DataSocket Read の**タイプ**入力に制御器または定数を配線して、ライブデータを特定のデータタイプに変換します。タイプを指定しないと、DataSocket Read の**データ**出力はバリアントデータを返します。このバリアントデータは、**関数→ DataSocket →バリアント**パレットにある Variant to Data 関数で操作する必要があります。場合によっては、バリアントデータから LabVIEW データに変換する必要があります。

DataSocket とバリアントデータ

他のアプリケーションからデータをサブスクライブする場合など、プログラムでデータを読み取る VI や他のアプリケーションがデータを元のデータタイプに戻すことができないことがあります。また、場合によっては、データタイプで許容されないタイムスタンプや警告などの属性をデータに追加する必要もあります。

このような場合は、**関数→ DataSocket →バリアント**パレットにある To Variant 関数を使用して、DataSocket 接続に書き込むデータをプログラムでバリアントデータに変換します。図 17-3 は、温度の読み取り値を連続して集録し、そのデータをバリアントデータに変換し、タイムスタンプを属性としてデータに追加するブロックダイアグラムを示しています。

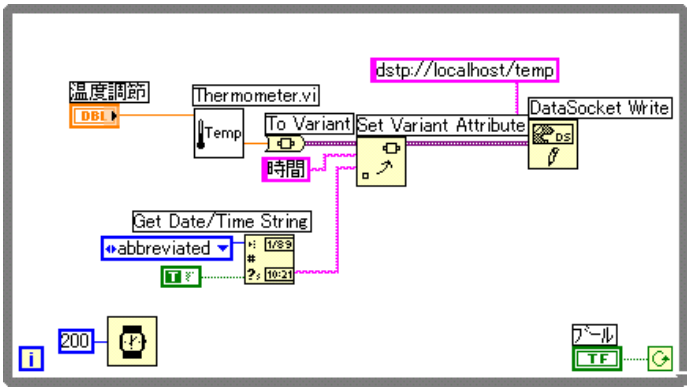


図 17-3　ライブ温度データをバリアントデータに変換する

他の VI がライブデータを読み込む場合、その VI はバリアントデータを操作可能なデータタイプに変換する必要があります。図 17-4 は、DataSocket 接続から温度データを連続して読み取り、バリアントデータを温度読み取り値に変換し、各読み取り値に対応するタイムスタンプ属性を取得し、フロントパネルに温度とタイムスタンプを表示するブロックダイアグラムを示しています。

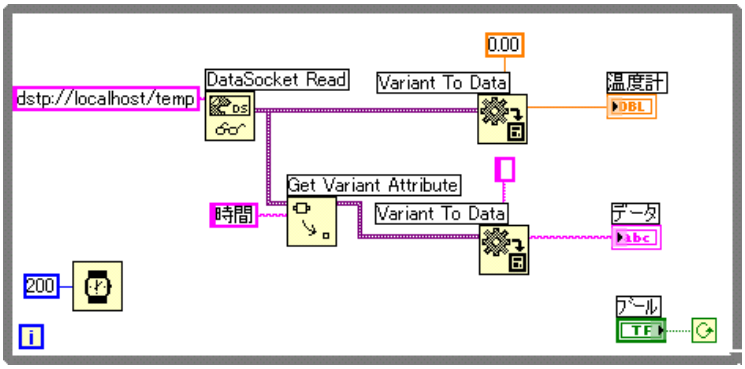


図 17-4　ライブバリアントデータを温度データに変換する

ウェブ上に VI をパブリッシュする

HTML ドキュメントを作成したり、フロントパネル画像をウェブ上にパブリッシュしたり、VI をウェブページに埋め込むには、LabVIEW ウェブサーバを使用します。パブリッシュされているフロントパネルへのブラウザアクセスを制御したり、ウェブ上に表示する VI を構成することができます。

メモ　ウェブ上の VI を制御したり、ウェブ上にパブリッシュする VI にセキュリティ機能を追加するには、LabVIEW エンタープライズコネクティビティツールセットを使用します。このツールセットの詳細については、ナショナルインスツルメンツのウェブサイト `ni.com` または `ni.com/jp` を参照してください。

ウェブサーバのオプション

ツール→オプションを選択し、一番上のプルダウンメニューから**ウェブサーバ**項目のいずれかを選択して、以下のオプションを設定します。

- ルートディレクトリおよびログファイルを設定します。
- ウェブサーバを使用可能にします。
- VI フロントパネルへのブラウザのアクセスを制御します。
- どの VI フロントパネルをウェブ上に表示可能にするかを構成します。

VI をウェブ上にパブリッシュするには、**ウェブサーバ : 構成**ページ (**LabVIEW オプション**ダイアログボックス) のウェブサーバを使用可能にする必要があります。後述のウェブパブリッシュツールを選択してウェブサーバを使用可能にすることもできます。VI をパブリッシュする前に VI をメモリにロードしておく必要があります。

HTML ドキュメントを作成する

以下のタスクを実行するには、**ツール→ウェブパブリッシュツール**を選択してウェブパブリッシュツールを使用します。

- HTML ドキュメントを作成します。
- HTML ドキュメントにフロントパネルの静止画像または動画を埋め込みます。現在、動画をサポートしているのは Netscape ブラウザのみです。
- クライアントがリモートで表示および制御できる VI を埋め込みます。
- 埋め込まれたフロントパネル画像の上下にテキストを追加します。
- 画像または埋め込まれた VI の周りに枠を配置します。
- ドキュメントをプレビューします。
- ディスクにドキュメントを保存します。
- ウェブサーバを使用可能にして、ウェブ上に HTML ドキュメントとフロントパネル画像をパブリッシュします。

フロントパネル画像をパブリッシュする

現在メモリ内にある VI のフロントパネルの静止画像を返すには、ウェブブラウザまたは HTML ドキュメントで `.snap` URL を使用します。URL のクエリパラメータによって VI 名と画像の属性を指定します。たとえば、*`VIName.vi`* が表示する VI の場合、`http://`*`web.server.address`*`/.snap?`*`VIName.vi`* となります。

現在メモリ内にある VI のフロントパネルの動画を返すには、`.monitor` URL を使用します。URL のクエリパラメータに VI 名、動画の属性、および画像の属性を指定します。たとえば、*`VIName.vi`* が表示する VI の場合、`http://`*`web.server.address`*`/.monitor?`*`VIName.vi`* となります。

フロントパネルの画像形式

ウェブサーバでは、フロントパネルの画像を JPEG および PNG の画像形式で生成できます。

JPEG 形式はグラフィックの圧縮率が高い反面、グラフィックの細部が損なわれることがあります。この形式は写真に最も適しています。線画、フロントパネル、およびブロックダイアグラムの場合、JPEG 形式で圧縮するとグラフィックがぼやけたり、色むらが発生することがあります。PNG 形式でも、JPEG 形式ほどではありませんが、高圧縮率でグラフィックが圧縮されます。ただし、PNG 圧縮では細部が損なわれません。

フロントパネルをリモートで参照および制御する

LabVIEW の組み込み式ウェブサーバに接続することによって、LabVIEW から、またはウェブブラウザから VI フロントパネルをリモートで参照および制御することができます。フロントパネルをクライアントからリモートで開く場合、ウェブサーバはフロントパネルをクライアントに送信しますが、ブロックダイアグラムとすべてのサブ VI はサーバコンピュータに残ります。ブロックダイアグラムがサーバ上で実行していることを除いて、VI をクライアント上で実行している場合と同様にフロントパネルと対話することができます。この機能を使用して、フロントパネル全体をパブリッシュしたり、リモートアプリケーションを安全で簡単に、しかも迅速に制御することができます。

メモ　VI 全体を制御するには、LabVIEW ウェブサーバを使用してください。DataSocket サーバを使用して、VI の 1 つのフロントパネル制御器に対してデータを読み書きします。DataSocket サーバの使用方法の詳細については、この章の「DataSocket テクノロジを使用する」のセクションを参照してください。

クライアント用のサーバを構成する

クライアントが LabVIEW またはウェブブラウザを使用してリモートでフロントパネルを参照および制御するためには、サーバコンピュータのユーザはまずサーバを構成する必要があります。**ツール→オプション**を選択して、上部のプルダウンメニューから**ウェブサーバ**ページを選択し、ウェブサーバを構成します。これらのページを使用して、サーバに対するブラウザのアクセスを制御し、リモートで参照可能なフロントパネルを指定します。また、これらのページを使用して、特定のリモートクライアントが VI を制御できる時間の制限値を設定することもできます。

ウェブサーバを使用すると、複数のクライアントが 1 つのフロントパネルに同時に接続できますが、フロントパネルを制御できるのは一度に 1 クライアントのみです。サーバコンピュータのユーザは、いつでも VI を制御することができます。コントローラがフロントパネル上の値を変更すると、すべてのクライアントのフロントパネルに変更が適用されます。ただし、クライアントのフロントパネルにはすべての変更は反映されません。通常、クライアントのフロントパネルには、フロントパネルオブジェクトのディスプレイに加えられた変更は反映されず、フロントパネルオブジェクトの実際の値に加えられた変更が反映されます。たとえば、コントローラがチャートのマッピングモードまたはスケールのマーカ間隔を変更したり、コントローラがチャートのスクロールバーを表示 / 非表示にすると、これらの変更はコントローラのフロントパネルにのみ反映されます。

リモートパネルライセンス

サーバに接続する可能性のある複数のクライアントをサポートするには、ライセンスを構成する必要があります。デフォルトでは、LabVIEW のリモートフロントパネルのライセンスは、1 クライアントに対してのみフロントパネルの参照および制御を許可します。ライセンスに許可された数を超えると、クライアントは、`labview\www` の `LicenseErrorMessage.txt` で指定した情報を含むメッセージを受け取ります。このメッセージには、サーバ管理者やリモートパネルのライセンスの更新担当者の連絡先に関する情報を記載したい場合があると考えられます。このファイルに何も入力しないと、クライアントのコンピュータに接続要求が拒否されたという内容のデフォルトのエラーメッセージが表示されます。

メモ　任意の LabVIEW 開発システムを使用して、クライアントがリモートで表示および制御可能な VI を作成することができますが、これらの VI をリモートで参照および制御する機能をサポートしているのは LabVIEW 開発システムおよびプロフェッショナル開発システムのみです。

LabVIEW またはウェブブラウザからフロントパネルを参照および制御する

クライアントは、LabVIEW またはウェブブラウザからフロントパネルをリモートで参照および制御することができます。

メモ　フロントパネルをリモートで参照および制御するには、まず参照および制御する VI のあるサーバコンピュータでウェブサーバを使用可能にする必要があります。

フロントパネルを LabVIEW で参照および制御する

LabVIEW をクライアントとして使用してリモートフロントパネルを参照するには、新規 VI を開いて**操作→リモートパネルに接続する**を選択し、**リモートパネルに接続**ダイアログボックスを表示します。このダイアログボックスを使用して、サーバのインターネットアドレスと参照する VI を指定します。デフォルトでは、リモート VI のフロントパネルはオブザーバモードになっています。VI をリクエストする際に、**リモートパネルに接続**ダイアログボックスの**制御をリクエスト**チェックボックスにチェックを入れて、制御器をリクエストすることができます。VI がコンピュータに表示されたら、フロントパネルのいずれかの部分を右クリックして、ショートカットメニューから**制御をリクエスト**を選択します。フロントパネルウィンドウの下部にあるステータスバーをクリックしてもこのメニューにアクセスすることができます。他のクライアントが制御していなければ、フロントパネルが制御可能であることを示すメッセージが表示されます。

他のクライアントが VI を制御している場合、サーバは、他のクライアントが制御を止めるまで要求を待ち行列に入れます。サーバコンピュータのユーザのみが**ツール→リモートパネル接続マネージャ**を選択してクライアントの待ち行列リストをモニタすることができます。

クライアントから参照および制御するすべての VI がメモリ内にある必要があります。要求された VI がメモリ内にある場合、サーバは VI のフロントパネルを要求側のクライアントに送信します。VI がメモリにない場合には、**リモートパネル接続**ダイアログボックスの**接続状況**のセクションにエラーメッセージが表示されます。

ウェブサーバからフロントパネルを参照および制御する

ウェブブラウザを使用して、LabVIEW がインストールされていないクライアントからリモートでフロントパネルを参照および制御するには、LabVIEW Run-Time Engine をインストールする必要があります。LabVIEW Run-Time Engine には、ブラウザのプラグインディレクトリにインストールされる LabVIEW ブラウザプラグインパッケージが含まれます。LabVIEW の CD には、LabVIEW Run-Time Engine のインストーラが含まれています。

クライアントは LabVIEW ランタイムエンジンをインストールし、サーバコンピュータのユーザはクライアントに参照および制御させる VI を参照する <OBJECT> タグを含む HTML ファイルを作成します。このタグには、VI に対する URL リファレンスと VI を LabVIEW ブラウザプラグインに渡すようにウェブブラウザに指示する情報が含まれています。クライアントは、ウェブブラウザウィンドウの上部のアドレスまたは URL フィールドにウェブサーバのウェブアドレスを入力して、ウェブサーバに移動します。プラグインは、ウェブブラウザウィンドウにフロントパネルを表示し、クライアントがリモートフロントパネルと通信できるようにウェブサーバと通信します。クライアントは、ウェブブラウザのリモートフロントパネルウィンドウの下部の**制御をリクエスト**を選択するか、またはフロントパネル内で右クリックしてショートカットメニューから**制御をリクエスト**を選択して制御を要求します。

メモ　ナショナルインスツルメンツでは、ウェブブラウザでフロントパネルを参照および制御する場合には、Netscape 4.7 以降、または Internet Explorer 5.0 以降をご使用になることをお勧めします。

リモートフロントパネルの参照および制御でサポートされていない機能

ウェブブラウザの制約により、フロントパネルの次元や位置の操作を試みるユーザインタフェースアプリケーションは、フロントパネルがウェブページの一部として表示される場合に正しく動作しません。ウェブサーバと LabVIEW ブラウザのプラグインは、複雑なユーザインタフェースアプリケーションとの整合性を保とうとしますが、特にダイアログボックスやサブ VI ウィンドウを表示するアプリケーションについては、ウェブブラウザで使用した場合に正常に動作しない可能性があります。ナショナルインスツルメンツでは、このようなアプリケーションをウェブブラウザで使用するためにエクスポートしないことをお勧めします。

データ量の多い VI は通常はリモートでの参照および制御用にエクスポートしないでください。たとえば、複数のチャートを含むフロントパネルはネットワークの負荷を増やすため、フロントパネルの更新に時間がかかる場合があります。また、While ループが使用されているのに wait 関数がない VI の場合、バックグラウンドタスクの実行時間が制限されるため、リモートで参照または制御する場合にフロントパネルが応答しない可能性があります。

また、VI によっては、リモートコンピュータから実行したときとローカルで実行したときで動作が異なる場合もあります。フロントパネルに埋め込まれた ActiveX コントロールは、LabVIEW とほぼ無関係に描画および操作するため、リモートクライアントには表示されません。VI が標準のファイルダイアログボックスを表示した場合、リモートでファイルシステムを検索することができないために、コントローラはエラーを受け取ります。また、パス制御器の**参照**ボタンはリモートパネルでは使用できない状態になります。

フロントパネルをリモートで参照しているクライアントは、接続しているフロントパネルが作成したアプリケーションのものかどうかによって異なる動作をする可能性があります。特に、作成したアプリケーションのフロントパネルの場合、クライアントがフロントパネルに接続する前にフロントパネルに加えられたプログラム的な変更はクライアントコンピュータには反映されません。たとえば、クライアントがそのフロントパネルに接続する前にプロパティノードによって制御器のキャプションが変更された場合、クライアントは変更されたキャプションではなく変更前のキャプションを参照することになります。

フロントパネル制御器のプロパティをポーリングすることによって特定のユーザインタフェース効果を実現するブロックダイアグラムの場合、リモートコンピュータから VI を制御するとパフォーマンスが低下する可能性があります。Wait for Front Panel Activity 関数を使用すると、このような VI のパフォーマンスを向上することができます。

低レベル通信アプリケーション

LabVIEW では、コンピュータ間の通信に使用できる低レベルプロトコルがいくつかサポートされています。

プロトコルはそれぞれ異なりますが、特にリモートアプリケーションのネットワーク位置を参照する方法が異なっています。各プロトコルは一般的に他のプロトコルと互換性がありません。たとえば、Macintosh と Windows の間で通信する場合、両方のプラットフォームで動作する TCP などのプロトコルを使用する必要があります。

TCP と UDP

転送制御プロトコル (Transmission Control Protocol：TCP) とユーザデータグラムプロトコル (User Datagram Protocol：UDP) は、LabVIEW によってサポートされているすべてのプラットフォームで使用可能です。TCP は信頼性のある接続ベースのプロトコルです。TCP にはエラー検出機能があり、データは重複することなく正しい順序で確実に届きます。これらの理由から、通常 TCP はネットワークアプリケーションに最適とされています。

UDP は TCP よりも高いパフォーマンスを提供でき、接続を必要としませんが、配信が保証されていません。通常は、配信の保証が重要でない場合にアプリケーションで UDP を使用します。あるアプリケーションで送信先に頻繁にデータを転送しているため、データの一部のセグメントが失われても問題にならない場合などです。LabVIEW アプリケーションでの TCP および UDP の使用方法の詳細については、『Using LabVIEW with TCP/IP and UDP』アプリケーションノートを参照してください。

DDE (Windows)

動的データ交換 (DDE) は TCP よりも上位で動作し、クライアントとサーバ間でコマンドやデータを交換します。DDE は Microsoft Excel などの標準で既成のアプリケーションとの通信に適しています。LabVIEW アプリケーションでの DDE の使用方法の詳細については、『Using DDE in LabVIEW』アプリケーションノートを参照してください。

AppleEvents および PPC Toolbox (Macintosh)

AppleEvents は、最も一般的な Macintosh 専用の通信形式です。DDE と同様に、AppleEvents はアクションを要求するメッセージを送信したり、他の Macintosh アプリケーションからの情報を返したりする場合に使用します。

プログラム間通信ツールボックス (Program-to-Program Communication Toolbox：PPC Toolbox) は、Macintosh アプリケーション間でデータを送受信する低レベルの形式です。PPC Toolbox は AppleEvents よりも高いパフォーマンスを提供します。これは、情報の転送にかかるオーバヘッドが少ないからです。ただし、PPC Toolbox は転送できる情報のタイプを定義していないので、ほとんどのアプリケーションがサポートしていません。PPC Toolbox は PPC Toolbox をサポートしているアプリケーション間で大量の情報を送信する場合の最適な方法です。LabVIEW アプリケーションでの AppleEvents と PPC Toolbox の使用方法については、『Using Apple Events and the PPC Toolbox to Communicate with LabVIEW Applications on the Macintosh』アプリケーションノートを参照してください。

パイプ VI (UNIX)

UNIX の名前付きパイプに対して開閉および読み書きを行うには、パイプ VI を使用します。LabVIEW と関連性のないプロセス間の通信を行うには、名前付きパイプを使用します。

システムレベルのコマンドを実行する (Windows および UNIX)

他の Windows アプリケーションや、UNIX コマンドラインアプリケーションを VI 内部から実行したり起動するには、System Exec VI を使用します。System Exec VI を使用すると、起動するアプリケーションでサポートされているパラメータを含んでいるシステムレベルのコマンドラインを実行できます。

18

ActiveX

LabVIEW などの Windows アプリケーションは、ActiveX オートメーションを使用すると、他の Windows アプリケーションがアクセスできる公開オブジェクトのセット、コマンド、および関数を提供できます。LabVIEW を ActiveX のクライアントとして使用すると、ActiveX が有効な他のアプリケーションに関連付けられているオブジェクト、プロパティ、メソッド、およびイベントにアクセスできます。また、LabVIEW は ActiveX サーバとしても機能するため、ActiveX が有効な他のアプリケーションから LabVIEW のオブジェクト、プロパティ、およびメソッドにアクセスできます。

本書では、ActiveX はマイクロソフト社の ActiveX テクノロジと OLE テクノロジを意味します。このテクノロジは Windows でのみ使用できます。ActiveX の詳細については、Microsoft Developer's Network documentation の Kraig Brockschmidt 著、第 2 版『Inside OLE』、および Don Box 著、『Essential COM』を参照してください。

詳細については

ActiveX テクノロジの使用方法の詳細については、「LabVIEW ヘルプ」を参照してください。

ActiveX のオブジェクト、プロパティ、メソッド、およびイベント

ActiveX 有効のアプリケーションには、他のアプリケーションからアクセス可能な、公開されているプロパティやメソッドを持つオブジェクトが含まれています。ボタン、ウィンドウ、ピクチャ、ドキュメント、ダイアログボックスなどのオブジェクトはユーザに見えますが、アプリケーションオブジェクトなどユーザに見えない場合もあります。アプリケーションに関連付けられているオブジェクトにアクセスし、プロパティを設定するか、そのオブジェクトのメソッドを呼び出すことによって、アプリケーションにアクセスします。

オブジェクト、プロパティ、およびメソッドの詳細については、第 16 章「プログラム的に VI を 制御する」の「アプリケーションと VI の設定値を操作する」のセクションを参照してください。

イベントとは、マウスをクリックする、キーを押す、メモリ容量不足になるなど、オブジェクト上で行う動作またはそれに伴った状況を意味します。オブジェクトにこれらのアクションが発生すると、オブジェクトはイベント固有のデータとともにイベントを送信し、ActiveX コンテナに警告します。

LabVIEW での ActiveX イベントの使用例および ActiveX のイベント VI の使用例については、`examples\comm\axevent.llb` の ActiveX Event Template VI と List ActiveX Events VI、および `examples\comm` ディレクトリの FamilyTree VI と MultiEvents VI を参照してください。

ActiveX VI、関数、制御器、および表示器

ActiveX 有効の他のアプリケーションに関連付けられているオブジェクト、プロパティ、メソッド、およびイベントにアクセスするには、以下の VI、関数、制御器、および表示器を使用します。

- ActiveX オブジェクトへのリファレンスを作成するには、**制御器→ ActiveX** パレットにあるオートメーション refnum 制御器を使用します。フロントパネル上にあるこの制御器を右クリックして、アクセスするタイプライブラリからオブジェクトを選択します。
- **関数→通信→ ActiveX** パレットにある Automation Open 関数を使用して、ActiveX オブジェクトを開きます。
- **制御器→ ActiveX** パレットにある ActiveX コンテナを使用して、フロントパネル上の ActiveX オブジェクトにアクセスして表示します。この制御器を右クリックして、アクセスするオブジェクトを選択します。
- **関数→通信→ ActiveX** パレットにあるプロパティノードを使用して、ActiveX オブジェクトに関連付けられたプロパティを取得（読み取り）および設定（書き込み）します。
- **関数→通信→ ActiveX** パレットにあるインボークノードを使用して、ActiveX オブジェクトに関連付けられたメソッドを呼び出します。
- **関数→通信→ ActiveX → ActiveX イベント**パレットにある ActiveX イベント VI を使用して、フロントパネルの ActiveX コンテナに配置した ActiveX オブジェクトに発生するイベントを管理します。
- データをサブ VI に受け渡しするには、**制御器→ ActiveX** パレットにあるバリアント制御器および表示器を使用します。ActiveX バリアント制御器を使用するとデータを表示できますが、変更はできません。バリアントデータの詳細については、第 5 章「ブロックダイアグラムを作成する」の「バリアントデータを処理する」のセクションを参照してください。

ActiveX クライアントとしての LabVIEW

ActiveX 有効の他のアプリケーションに関連付けられているオブジェクトに LabVIEW がアクセスすると、LabVIEW は ActiveX のクライアントとして動作します。LabVIEW を ActiveX のクライアントとして使用する方法は以下のとおりです。

- Microsoft Excel などのアプリケーションを開き、ドキュメントを作成し、そのドキュメントにデータを追加します。
- フロントパネル上に Microsoft Word ドキュメントや Excel スプレッドシートなどのドキュメントを埋め込みます。
- 他のアプリケーションに使用されているヘルプファイルを呼び出す**ヘルプ**ボタンなど他のオブジェクトをフロントパネルに配置します。
- 別のアプリケーションで作成した ActiveX コントロールにリンクします。

LabVIEW はオートメーション refnum 制御器と ActiveX コンテナを使用して ActiveX オブジェクトにアクセスします。どちらもフロントパネルオブジェクトです。ActiveX オブジェクトを選択するには、オートメーション refnum 制御器を使用します。ボタンやドキュメントなど、表示可能な ActiveX オブジェクトを選択してフロントパネル上に配置するには、ActiveX コンテナを使用します。両方のオブジェクトともブロックダイアグラム上にオートメーション refnum 制御器として表示されます。

ActiveX 有効のアプリケーションにアクセスする

ActiveX 有効のアプリケーションにアクセスするには、ブロックダイアグラム上でオートメーション refnum 制御器を使用してアプリケーションに対するリファレンスを作成します。制御器を Automation Open 関数に配線します。呼び出しアプリケーションが開きます。そのオブジェクトに関連付けられているプロパティを選択してアクセスするには、プロパティノードを使用します。そのオブジェクトに関連付けられているメソッドを選択してアクセスするには、インボークノードを使用します。
Automation Close 関数を使用してオブジェクトへのリファレンスを閉じます。リファレンスを閉じるとメモリからオブジェクトが削除されます。

たとえば、Microsoft Excel を開く VI を作成することによって、画面に Excel を表示したり、ワークブックやスプレッドシートを作成したり、LabVIEW で表を作成したり、Excel スプレッドシートにその表を書き込んだりすることができます。

Excel のクライアントとして LabVIEW を使用する方法の例については、`examples\comm\ExcelExamples.llb` の Write Table To XL VI を参照してください。

メモ　ActiveX カスタムインタフェースを含むアプリケーションは アイコンで示されます。カスタムインタフェースのオブジェクトを選択するには、アイコンをクリックします。カスタムインタフェースの詳細については、この章の「カスタム ActiveX オートメーションインタフェースのサポート」のセクションを参照してください。

フロントパネルに ActiveX オブジェクトを挿入する

フロントパネルに ActiveX コントロールを挿入するには、ActiveX コンテナを使用して ActiveX オブジェクトまたはドキュメントを選択し、そのオブジェクトに関連付けられているプロパティやメソッドにアクセスします。ActiveX プロパティブラウザまたはプロパティページを使用して、ActiveX オブジェクトのプロパティを設定したり、プロパティノードを使用してプロパティをプログラム的に設定することができます。ActiveX プロパティの設定の詳細については、この章の「ActiveX プロパティを設定する」のセクションを参照してください。

そのオブジェクトに関連付けられているメソッドを呼び出すには、インボークノードを使用します。

たとえば、ActiveX コンテナを使用して Microsoft Web Browser オブジェクトにアクセスしたり、メソッドの Navigate クラスを選択したり、URL メソッドを選択したり、URL を指定することによって、フロントパネル上にウェブページを表示できます。

ActiveX コンテナを使用すると、ブロックダイアグラム上のオートメーション refnum 制御器を Automation Open 関数に配線したり、Automation Close 関数を使用してオブジェクトへのリファレンスを閉じる必要がありません。ActiveX コンテナによって呼び出しアプリケーションが LabVIEW に埋め込まれているので、インボークノードやプロパティノードに直接配線できます。ただし、ActiveX コンテナに他のオートメーション refnum 制御器を返すプロパティやメソッドが含まれている場合は、それらの他の refnum 制御器を閉じる必要があります。

ActiveX プロパティを設定する

ActiveX オブジェクトまたはドキュメントを呼び出しまたは挿入した後、ActiveX プロパティブラウザ、プロパティページ、およびプロパティノードを使用して、そのオブジェクトまたはドキュメントに関連付けられたプロパティを設定することができます。

ActiveX プロパティブラウザ

ActiveX プロパティブラウザを使用して、ActiveX オブジェクトまたは ActiveX コンテナ内のドキュメントに関連付けられたすべてのプロパティを表示および設定することができます。ActiveX プロパティブラウザ

にアクセスするには、フロントパネルのコンテナにあるオブジェクトまたはドキュメントを右クリックして、ショートカットメニューから**プロパティブラウザ**を選択します。また、**ツール→上級→ ActiveX プロパティブラウザ**を選択することもできます。ActiveX プロパティブラウザを使用すると、ActiveX コントロールのプロパティを対話式で簡単に設定することができますが、ブラウザを使用してプログラム的にプロパティを設定することはできません。また、ActiveX プロパティブラウザは、コンテナに入っている ActiveX オブジェクトにのみ使用できます。ActiveX プロパティブラウザは、実行モードでは使用できません。

ActiveX プロパティページ

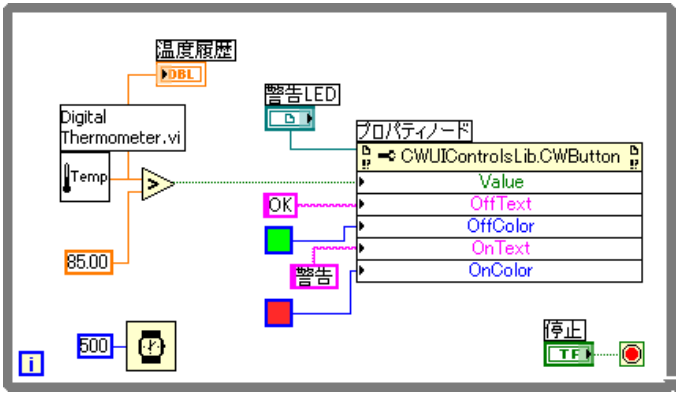
多くの ActiveX オブジェクトには、別のタブのオブジェクトに関連付けられたプロパティを構成するプロパティページが含まれます。ActiveX プロパティページにアクセスするには、フロントパネルのコンテナにあるオブジェクトまたはドキュメントを右クリックして、ショートカットメニューからオブジェクト名を選択します。

ActiveX プロパティブラウザと同様に、ActiveX プロパティページを使用して、ActiveX コントロールのプロパティを対話式で設定することができますが、プロパティをプログラム的に設定することはできません。また、プロパティページは、コンテナに入っている ActiveX オブジェクトにのみ使用できます。プロパティページは、すべての ActiveX オブジェクトに使用できるわけではありません。

プロパティノード

プロパティノードを使用して、ActiveX プロパティをプログラム的に設定します。たとえば、ActiveX コントロールを使用して、ユーザに対して温度が限度値を超えたときに警告するには、プロパティノードを使用してオブジェクトの値プロパティを設定して限度値を指定します。

以下の例では、温度が華氏 85 度以上になったときに、Component Works Control Library の一部である CWButton ActiveX オブジェクトの値プロパティを変更します。



この場合、CWButton は LED としての役割を果たし、色が変わって、温度が限度値を超えた場合（CWButton オブジェクトの ON の状態）に警告を表示します。

メモ　この例では、CWButton の OffText、OffColor、OnText、および OnColor プロパティはプログラム的に設定する必要はないので、ActiveX プロパティブラウザまたはプロパティページを使用して、これらのプロパティを設定することができます。ActiveX プロパティブラウザの詳細についてはこの章の「ActiveX プロパティブラウザ」、プロパティページの詳細についてはこの章の「ActiveX プロパティページ」をそれぞれ参照してください。

ActiveX サーバとしての LabVIEW

他のアプリケーションからの ActiveX 呼び出しによって、LabVIEW のアプリケーション、VI、および制御器のプロパティとメソッドを使用できます。Microsoft Excel などの ActiveX 有効の他のアプリケーションが LabVIEW からプロパティ、メソッド、および個々の VI を要求する場合、LabVIEW は ActiveX サーバとして機能します。

たとえば、Excel のスプレッドシートに VI グラフを埋め込むと、スプレッドシートから VI 入力にデータを入力して VI を実行できます。VI を実行すると、データによってグラフがプロットされます。

Excel スプレッドシートで LabVIEW のプロパティおよびメソッドを使用する方法の例については、`examples\comm\freqresp.xls` を参照してください。

カスタム ActiveX オートメーションインタフェースのサポート

LabVIEW を使用して ActiveX サーバからプロパティおよびメソッドにアクセスする ActiveX クライアントを作成する場合、サーバによって表示されるカスタムインタフェースにアクセスすることができます。ActiveX サーバの作成者は、これらのカスタムインタフェースのプロパティおよびメソッドのパラメータのデータタイプが Automation (IDispatch) であることを確認する必要があります。カスタムインタフェースへのアクセスの詳細については、サーバ開発プログラミング環境のドキュメントを参照してください。

定数を使用して ActiveXVI のパラメータを設定する

ActiveX ノードのパラメータには、有効な値の離散リストを取り込むものがあります。これらのパラメータ値を設定する場合は、リング定数から該当する名前を選択します。ActiveX VI の作成時にリング定数にアクセスするには、データ値を受け取るノードのパラメータを右クリックして、ショートカットメニューから**定数を作成**を選択します。リング定数で選択できる定数は、ノードに渡される refnum によって異なります。図 18-1 と図 18-2 は、リング定数および数値定数を使用してパラメータ値を設定する方法の例を示しています。

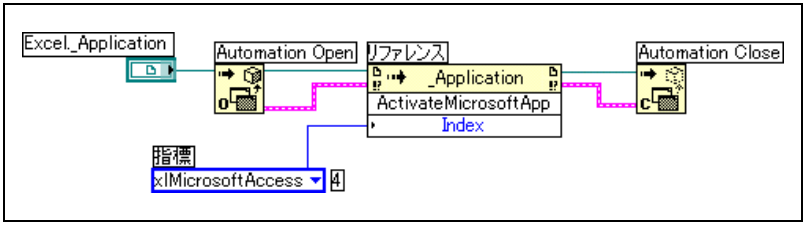


図 18-1　リング定数でデータ値を設定する

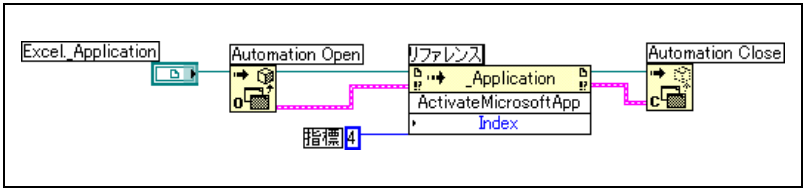


図 18-2　数値定数でデータ値を設定する

データ値を受け取るパラメータには、パラメータ名の左側に小さな矢印が表示されます。対応する数値データ値を表示するには、リング定数を右クリックしてショートカットメニューから**項目を表示→デジタル表示**を選択します。

図 18-1 と図 18-2 の VI は両方とも、Microsoft Excel アプリケーションにアクセスしてメソッドを実行します。**ActivateMicrosoftApp** メソッドの**指標**パラメータにはいくつかのオプションがあります。**MicrosoftWord**、**MicrosoftPowerPoint**、**MicrosoftMail**、**MicrosoftAccess**、**MicrosoftFoxPro**、**MicrosoftProject**、および **MicrosoftSchedulePlus** がそのオプションです。

図 18-1 の **MicrosoftAccess** オプションに対応する**指標**パラメータの数値を識別するには、リング定数のプルダウンメニューから **MicrosoftAccess** オプションを選択します。現在選択されているオプションの数値がリング定数のとなりのボックスに表示されます。リング定数を使用する代わりに、図 18-2 のように数値定数にオプションの数値を入力することもできます。

19

テキストベースのプログラミング言語からのコード呼び出し

Call Library Function ノードを使用すると、LabVIEW で最も標準的な共有ライブラリを呼び出すことができます。コードインターフェースノード (CodeInterface Node : CIN) を使用して、LabVIEW で C コードを呼び出すこともできます。

テキストベースのプログラミング言語からコードを呼び出す方法の詳細については、『Using External Code in LabVIEW』のマニュアルを参照してください。

詳細については

テキストベースのプログラミング言語からコードを呼び出す方法の詳細については、「LabVIEW ヘルプ」を参照してください。

Call Library Function ノード

最も標準的な共有ライブラリまたは DLL を呼び出すには、Call Library Function ノードを使用します。この関数を使用すると、LabVIEW にインタフェースを作成し、既存のライブラリや LabVIEW 用に特別に作成された新規ライブラリを呼び出すことができます。ナショナルインスツルメンツでは、Call Library Function ノードを使用して外部コードに対するインタフェースを作成することをお勧めします。

コードインタフェースノード

C で記述されたソースコードを呼び出す別の方法として CIN を使用する方法があります。通常、CIN よりも Call Library Function ノードの方が簡単に使用できます。

20

フォーミュラと方程式

LabVIEW で複雑な方程式を使用する場合、ブロックダイアグラム上でさまざまな演算関数を組み合わせて配線する必要はありません。慣れている数学的環境で方程式を構築し、その方程式をアプリケーションに統合することができます。

LabVIEW 環境で数学演算を実行するにはフォーミュラノードおよび Expression Node を使用します。さらに高度な機能が必要な場合は、数学アプリケーションである HiQ や MATLAB にリンクして方程式を作成できます。HiQ および MATLAB は、実際の数学、科学、および工学上の問題を整理して視覚的にとらえる場合に役立つソフトウェアパッケージです。

詳細については

方程式および構文を使用して使用可能な関数や演算子を活用する方法と発生する可能性のあるエラーの説明については、「LabVIEW ヘルプ」を参照してください。

LabVIEW で方程式を使用する方法

フォーミュラノード、Expression Node、HiQ Script ノード、および MATLAB スクリプトノードを使用すると、ブロックダイアグラムで数学演算を実行できます。

メモ スクリプトノードを使用するには、コンピュータに HiQ または MATLAB をインストールする必要があります。LabVIEW は ActiveX テクノロジを使用してスクリプトを HiQ または MATLAB に渡して実行するためです。LabVIEW は、Windows でのみ使用可能な ActiveX テクノロジを使用してスクリプトノードを実行します。このため、スクリプトノードは Windows でのみ使用可能です。

ナショナルインスツルメンツでは、HiQ を LabVIEW と同梱して出荷しているので、追加の費用をかけなくてもこのソフトウェアをインストールして方程式の処理機能をより高めることができます。

スクリプトノードはフォーミュラノードと似ていますが、スクリプトノードを使用すると既存の HiQ や MATLAB スクリプトを ASCII 形式で LabVIEW にインポートしたり、そのスクリプトを LabVIEW で実行でき

ます。フォーミュラノードと一緒に使用することによって、ノードとのデータの受け渡しが可能になります。

フォーミュラノード

フォーミュラノードはテキストベースの便利なノードで、ブロックダイアグラム上で数学演算を実行できます。外部コードやアプリケーションにアクセスする必要がなく、方程式を作成するために低レベルの演算関数を配線する必要もありません。フォーミュラノードでは、テキストベースの方程式表現だけでなく、テキストベースの if ステートメント、while ループ、for ループ、および do ループも使用できます。これらは C 言語のプログラマによく知られているものです。これらのプログラミング要素は C 言語でのプログラムに使用されるものと似ていますが、同一ではありません。

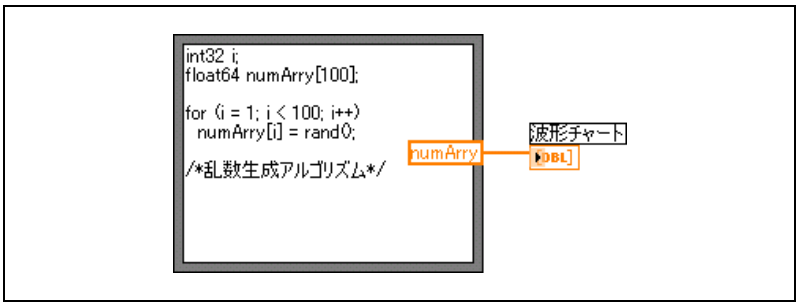
方程式に変数が多くある場合や方程式が複雑な場合、または既存のテキストベースのコードを使用する場合に、フォーミュラノードが役立ちます。既存のテキストベースのコードは、グラフィカルに再作成するのではなく、コピーしてフォーミュラノードに貼り付けます。

フォーミュラノードは、類別化チェックを使用して、配列指標が数値データであり、ビット演算に対するオペランドが整数データであることを確認します。また、フォーミュラノードは、配列指標が範囲内であることも確認します。配列については、範囲外の値はデフォルトでゼロ、範囲外の割り当てはデフォルトで nop に設定され、演算が行われないことを示します。

フォーミュラノードは、自動タイプ変換も行います。

フォーミュラノードを使用する

関数→ストラクチャおよび**関数→数学→フォーミュラ**パレット上にあるフォーミュラノードは、For ループ、While ループ、Case ストラクチャ、およびシーケンスストラクチャのようなサイズ変更が可能なボックスです。次の例のように、フォーミュラノードはサブダイアグラムを含んでいるのではなく、セミコロンで区切られた、C 言語に似たステートメントを含んでいます。C 言語の場合と同じように、コメントをスラッシュとアスタリスクのペアで囲んで (/* コメント */) 追加できます。



フォーミュラノードの使用例については、`examples\general\structs.llb` の Equations VI を参照してください。

フォーミュラノードの変数

変数を使用して作業する場合、以下の点に注意してください。

- 1 つのフォーミュラノード内の変数または方程式の数に制限はありません。
- 複数の入力値または出力値どうしが同じ名前を使用することはできませんが、出力と入力を同じ名前にすることはできます。
- フォーミュラノードの枠を右クリックして、ショートカットメニューから**入力端子を追加**を選択することによって入力変数を宣言します。フォーミュラノード内では入力変数を宣言できません。
- フォーミュラノードの枠を右クリックして、ショートカットメニューから**出力端子を追加**を選択することによって出力変数を宣言します。出力変数名は、入力変数名またはフォーミュラノード内で宣言した変数名と一致する必要があります。
- 変数を右クリックし、ショートカットメニューから**入力に変更**または**出力に変更**を選択すると、変数を入力または出力に変更できます。
- フォーミュラノード内部で変数を宣言する場合、入力配線または出力配線に関連付けなくても使用できます。
- すべての入力端子は配線する必要があります。
- 変数を浮動小数点値のスカラにすることができます。浮動小数点値のスカラの精度はコンピュータの構成によって異なります。また、変数として整数や数値配列も使用できます。
- 変数に単位を付けることはできません。

数式ノード

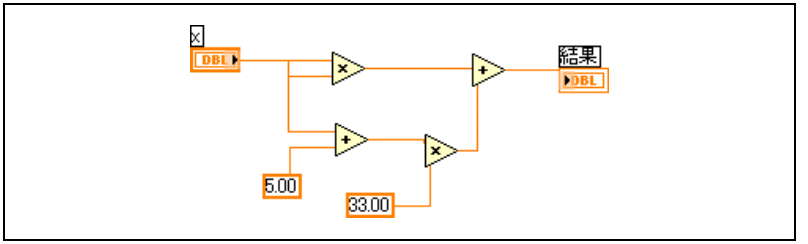
1 つの変数を含んでいる数式や方程式を計算するには数式ノードを使用します。数式ノードは方程式に変数が 1 つのみの場合に有効ですが、それ以外の場合には複雑になります。

数式ノードは変数の値として入力端子に渡される値を使用します。出力端子は計算値を返します。

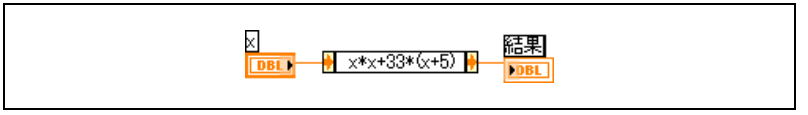
例として、次に示す簡単な方程式を検討してみます。

$$x \times x + 33 \times (x + 5)$$

次の図のブロックダイアグラムでは、数値関数を使用してこの方程式を表現します。



次の図のように数式ノードを使用すると、ブロックダイアグラムがより簡単になります。



数式ノードの多形性

数式ノードの入力端子は、その端子に配線されている制御器や定数と同じデータタイプになります。出力端子は入力端子と同じデータタイプになります。入力のデータタイプは複素数でないスカラ数値、複素数でないスカラ数値の配列、または複素数でないスカラ数値のクラスタにすることができます。配列やクラスタを使用することによって、数式ノードは入力配列またはクラスタの各要素に方程式を適用できます。

LabVIEW で HiQ を使用する

HiQ は高性能で対話形式の問題解決環境であり、これによって実際の科学および工学の問題の解析、視覚化、および文書化を行います。HiQ ではバーチャルノートブックインタフェースを使用し、そのプログラミング言語は HiQ スクリプトと呼ばれています。このノートブックにはページ、セクション、およびタブがあり、ここにテキスト、数値、グラフなどのオブジェクトを配置したり配列することができます。

LabVIEW から HiQ を制御し、さまざまな形式の 2 つのアプリケーション間でデータを転送するには、**関数→通信→ HiQ** パレットにある VI を使用します。これらの VI を使用するには、HiQ をインストールする必要があります。これらの HiQ VI を使用して HiQ を起動し、HiQ ノートブック (Notebook) にアクセスして操作することによって、データを解析して視覚化し、ノートブック内で HiQ スクリプトを実行できます。

数学機能、関数リファレンス、構文の情報など、HiQ の機能の詳細については、「HiQ ヘルプ」を参照してください。

HiQ VI のセットアップ例と使用例については、`examples\comm\hiq` を参照してください。

HiQ スクリプトノードと MATLAB スクリプトノード

ブロックダイアグラム上に HiQ スクリプトおよび MATLAB スクリプトをロードして編集するには、**関数→数学→フォーミュラ**パレットにある HiQ スクリプトノードと MATLAB スクリプトノードを使用します。これによって、LabVIEW は高度な数学的機能を扱うことができるようになります。

スクリプトノードを使用するには、HiQ または MATLAB をインストールしておく必要があります。HiQ または MATLAB で記述された既存のスクリプトがある場合は、それをスクリプトノードにインポートできます。

スクリプトノード端子を入力または出力としてスクリプト内の変数に割り当てると、HiQ または MATLAB と LabVIEW の間で値をやり取りできます。方程式が記述された方法によって、端子の関数を決めることができます。たとえば、スクリプトに割り当てステートメント *X* = *i* + 3 が含まれている場合、*i* を入力端子として割り当てることによってスクリプトノードが *X* を計算する方法を制御できます。さらに、*X* を出力端子に割り当てるとスクリプト計算の最終的な結果を取り出すことができます。

既存のスクリプトがない場合は、ブロックダイアグラム上にスクリプトノードを配置し、HiQ または MATLAB の構文を使用してスクリプトを作成できます。LabVIEW は、スクリプトを実行するプログラムであるスクリプトサーバエンジンと通信します。LabVIEW は業界標準のプロトコルを使用して、通信したりスクリプトサーバエンジンを制御します。スクリプトサーバエンジンは HiQ や MATLAB と一緒にインストールされます。

HiQ および MATLAB スクリプト言語の性質により、作成した端子のデータタイプをスクリプトノードによって決めることはできません。各スクリプトノード端子に LabVIEW データタイプを割り当てる必要があります。LabVIEW のスクリプトノードは、HiQ や MATLAB がサポートしているデータタイプを識別します。表 20-1 は、LabVIEW データタイプとそれに対応する HiQ および MATLAB のデータタイプを示しています。

表 20-1　LabVIEW、HiQ、および MATLAB のデータタイプ

LabVIEW データタイプ	HiQ データタイプ	MATLAB データタイプ
	整数	—
	実数	実数
	テキスト	—
	整数ベクトル	—
	実数ベクトル	実数ベクトル
	整数行列	—
	実数行列	実数行列
	複素数	複素数
	複素ベクトル	複素ベクトル
	複素行列	複素行列

HiQ スクリプトと MATLAB スクリプトについてのプログラム上の提案

以下のプログラミング手法を使用すると、スクリプトのデバッグ作業が簡単になります。

- スクリプトを作成し、LabVIEW にスクリプトをインポートする前に、テストとデバッグを兼ねて HiQ または MATLAB 内で作成したスクリプトを実行します。

- HiQ スクリプトノードを右クリックしてショートカットメニューから**サーバを編集**を選択して、ネイティブ HiQ スクリプトウィンドウで編集、デバッグ、コンパイル、および実行を行います。
- データタイプを確認します。新しい入力端子または出力端子を作成する場合は、端子のデータタイプが正しいことを確認します。HiQ および MATLAB の両方で、計算時に変数のタイプが変わることがあるので、Error In 関数と Error Out 関数を使用してこれを監視します。
- 入出力端子の制御器や表示器を作成して、スクリプトノードが LabVIEW と HiQ または MATLAB の間でやり取りする値を監視できるようにします。これによって、必要に応じてスクリプトノードのどこで間違って計算しているのかを突き止めることができます。
- エラーチェックのパラメータを情報のデバッグに利用します。スクリプトノードに**エラー出力**端子用の表示器を作成し、エラー情報を実行時に表示できるようにします。フォーミュラノードではコンパイル時のエラーが表示されます。

LabVIEW アプリケーションに必要な HiQ サポートファイル

HiQ VI の呼び出しが含まれている LabVIEW アプリケーションを作成したら、そのアプリケーションを実行する対象のコンピュータを決める必要があります。

LabVIEW コードで HiQ スクリプトノードを呼び出すと HiQ が起動します。HiQ スクリプトが含まれている LabVIEW アプリケーションを配布する場合は、対象のマシンに HiQ がインストールされている必要があります。

次の表は、LabVIEW および HiQ の使用および配布方法に基づいた必須ファイルを示しています。

配布するファイル	対象マシン上で現在使用できるソフトウェア	インストールが必要なもの
HiQ スクリプトノードを持つ LabVIEW アプリケーション VI	LabVIEW (任意のバージョン) がインストール済み、HiQ は未インストール	LabVIEW と同梱のライセンス付き HiQ
実行可能ファイル	LabVIEW と HiQ の両方が未インストール	HiQ Reader によって、他のマシンに HiQ Professional Notebooks を表示してスクリプトを実行できます。LabVIEW 開発システムまたはプロフェッショナル開発システムの CD から、HiQReader を無料で配布できます。HiQ Reader のダウンロードについては、ナショナルインスツルメンツのウェブサイト ni.com または ni.com/jp を参照してください。

A

LabVIEW の構成

この付録では、LabVIEW のファイルシステムのストラクチャとファイルの保存に望ましい場所について説明します。

LabVIEW のディレクトリストラクチャの構成

このセクションでは、Windows、Macintosh、および UNIX での LabVIEW ファイルシステムのストラクチャについて説明します。LabVIEW では、プラットフォームの可用性に応じて、GPIB、DAQ、VISA、IVI、モーションコントロール、および IMAQ ハードウェア用のドライバソフトウェアがインストールされます。ハードウェアの構成の詳細については、『LabVIEW Measurements Manual』の Chapter 3「Installing and Configuring Your Measurement Hardware」を参照してください。

ソフトウェアに添付されている『LabVIEW リリースノート』に説明があるように、インストールを終了すると、LabVIEW ディレクトリには以下のグループが含まれます。

ライブラリ

- `user.lib`：ユーザが作成する制御器および VI を保存します。LabVIEW では、制御器は**制御器→ユーザ制御器**パレット上に表示され、VI は**関数→ユーザライブラリ**パレット上に表示されます。LabVIEW をアップグレードまたは削除しても、このディレクトリは変更されません。
- `vi.lib`：GPIB、解析、データ集録 (DAQ) などの組み込み VI のライブラリが含まれています。LabVIEW では、これらの VI は**関数**パレットにある関連グループに表示されます。ユーザのファイルを `vi.lib` に保存しないでください。アップグレード時に LabVIEW によってファイルが上書きされる場合があります。
- `instr.lib`：PXI、VXI、GPIB、シリアル計測器、およびコンピュータベースの計測器の制御に使用される計測器ドライバが含まれています。ナショナルインスツルメンツの計測器ドライバをインストールする場合は、計測器ドライバをこのディレクトリに配置します。それらは LabVIEW によって**計測器 I/O →計測器ドライバ**パレットに追加されます。

ストラクチャとサポート

- menus：LabVIEW が**制御器**および**関数**パレットのストラクチャの構成に使用するファイルが含まれています。
- resource：LabVIEW アプリケーションの追加サポートファイルが含まれています。ユーザのファイルをこのディレクトリに保存しないでください。アップグレード時に LabVIEW によってファイルが上書きされる場合があります。
- project：LabVIEW の**ツール**メニューの項目になるファイルが含まれています。
- templates：通常の VI のテンプレートが含まれています。
- www：ウェブサーバを介してアクセスできる HTML ファイルを保存します。

実習と手順

- activity：「LabVIEW チュートリアル」の作業を行うために使用する VI が含まれています。activity\solution ディレクトリには、各作業で実行される VI が含まれています。
- examples：VI のサンプルが含まれています。サンプルを検索するには、**ヘルプ→サンプルの検索**を選択します。

マニュアル

- manuals：PDF 形式のドキュメントが含まれています。このフォルダにはヘルプファイルは含まれていません。**ヘルプ→印刷版マニュアルを表示**を選択して、PDF にアクセスします。
- help：ヘルプファイルが含まれています。**ヘルプ→ヘルプを表示**を選択して、「LabVIEW ヘルプ」にアクセスできます。

その他のファイル

- serpdrv：Windows および UNIX 上のシリアルポートにアクセスするサポートファイルです。このファイルは、これらのプラットフォーム上のシリアルポートを使用するアプリケーションと一緒に配布します。

Macintosh

Macintosh 版は、前述のディレクトリの他に、LabVIEW アプリケーションのサポートファイルを含んでいる共有ライブラリフォルダを使用します。

ファイル保存の推奨場所

vi.lib および resource ディレクトリは、LabVIEW システム専用として LabVIEW にインストールされています。これらのディレクトリにユーザのファイルを保存しないでください。

ユーザのファイルは以下のディレクトリに保存できます。

- user.lib：**関数→ユーザライブラリ**パレットに表示させる、常時使用する VI を保存します。vi.lib で用意されている関数の追加として使用します。

メモ　サブ VI を変更することなくプロジェクト間で移植可能な場合にのみ、user.lib ディレクトリにサブ VI を保存します。user.lib 内の VI パスは絶対的なパスです。その他の場所に保存するサブ VI パスは呼び出し側 VI を基準とした相対パスです。そのため、特別な場合に VI を変更するために user.lib から VI をコピーしても、user.lib にあるサブ VI へのパスは変わりません。

- instr.lib：**計測器 I/O →計測器ドライバ**パレットに表示する計測器ドライバ VI を保存します。
- project：LabVIEW の機能を拡張するために使用する VI を保存します。このディレクトリに保存する VI は**ツール**メニューに表示されます。
- www：ウェブサーバを介してアクセスできる HTML ファイルを保存します。
- help：**ヘルプ**メニューで使用可能にする VI、PDF、.hlp ファイルを保存します。

また、ユーザが作成する LabVIEW ファイルを保存するディレクトリは、ハードドライブ上のどこに作成してもかまいません。ディレクトリ作成の詳細については、第 7 章「VI およびサブ VI を作成する」の「VI を保存する」のセクションを参照してください。

B

多形性関数

関数の多形性には度合いがあります（どの入力も多形性でない関数、入力の一部が多形性である関数、または入力のすべてが多形性である関数）。関数の入力には、数値またはブール値を受け入れるものがあります。数値や文字列を受け入れるものもあります。また、スカラ数値だけでなく、数値配列、数値クラスタ、数値クラスタの配列などを受け入れるものもあります。また、1 次元配列しか受け入れないものもありますが、配列要素はどのタイプでもかまいません。複素数値を含むすべてのタイプのデータを受け入れる関数もあります。多形性単位の作成と使用の詳細については、『Polymorphic Units in LabVIEW』アプリケーションノートを参照してください。

詳細については

多形性関数の詳細については、「LabVIEW ヘルプ」を参照してください。

数値変換

数値表記法を他の数値表記法に変換できます。複数の異なる表記法の数値入力を関数に配線すると、関数は通常、精度の大きい方の形式で出力を返します。この関数は、実行前に小さい表記法を一番精度の大きい形式に強制し、LabVIEW は変換される端子に強制ドットを付けます。

Divide、Sine、Cosine などの関数には、常に浮動小数点を出力するものもあります。入力側に整数を接続すると、これらの関数は整数を倍精度浮動小数点数に変換してから計算を実行します。

浮動小数点のスカラ値には通常、倍精度浮動小数点数の使用が最適です。単精度浮動小数点数を使用してもほとんど実行時間は短縮できず、オーバーフローがかなり発生しやすくなります。たとえば解析ライブラリでは、倍精度浮動小数点数を使用します。必要な場合に限り、拡張精度浮動小数点数を使用します。拡張精度の計算のパフォーマンスと精度はプラットフォームによって変わります。浮動小数点のオーバーフローの詳細については、第 6 章の「VI の実行とデバッグ」の「不定データまたは予想外のデータ」のセクションを参照してください。

整数の場合は通常、32 ビット符号付き整数の使用が最適です。

数値表記の異なる接続先に出力を配線する場合、LabVIEW は以下の規則に従ってデータを変換します。

- **符号付きまたは符号なし整数から浮動小数点数**：倍長整数から単精度浮動小数点数に変換する場合を除いて、変換は完全に一致します。この場合、LabVIEW は精度を 32 ビットから 24 ビットに下げます。
- **浮動小数点数から符号付きまたは符号なし整数**：LabVIEW は範囲外の値を整数の最小値または最大値にします。For ループの繰り返し端子などの整数オブジェクトのほとんどは、浮動小数点数を四捨五入します。LabVIEW では、0.5 の指数部分を丸めて最も近い偶数にします。たとえば、6.5 は 7 ではなく 6 になります。
- **整数から整数**：LabVIEW は範囲外の値を整数の最小値や最大値にしません。ソースが接続先よりも小さい場合、LabVIEW は符号付きのソースの符号を拡張し、符号なしソースの余分なビットに 0 を入れます。ソースが接続先より大きい場合、LabVIEW はその値の最下位ビットのみをコピーします。

数値関数の多形性

演算関数は数値入力データを使用します。関数の説明に記述されている例外を除いて、出力には入力と同じ数値表記法が使用され、複数の入力が異なる表記法である場合、出力は入力の数値表記法の中で最大幅のものになります。

演算関数は、数値、数値の配列、数値のクラスタ、数値のクラスタの配列、複素数などで実行できます。使用可能な入力タイプの正式な再帰定義は次のとおりです。

数値タイプ = 数値スカラ OR 配列 (数値タイプ)
OR クラスタ (数値タイプ)

数値スカラは浮動小数点数、整数、または複素数浮動小数点数となり得ます。LabVIEW では配列の配列は使用できません。

配列にはサイズおよび次元数の制限がありません。クラスタの要素数にも制限はありません。関数の出力タイプは入力タイプと同じ数値表記法です。入力が 1 つである関数については、その関数は配列またはクラスタの各要素の演算を行います。

2 つの入力を持つ関数については、以下の入力の組み合わせを使用できます。

- **同一**：入力が両方とも同じストラクチャを持ち、出力が入力と同じストラクチャを持ちます。
- **1 つのスカラ**：一方の入力が数値スカラで、他方が配列またはクラスタの場合、出力は配列またはクラスタになります。

- **配列**：1 つの入力が数値配列で、他方がその数値配列の数値タイプの場合、出力は配列になります。

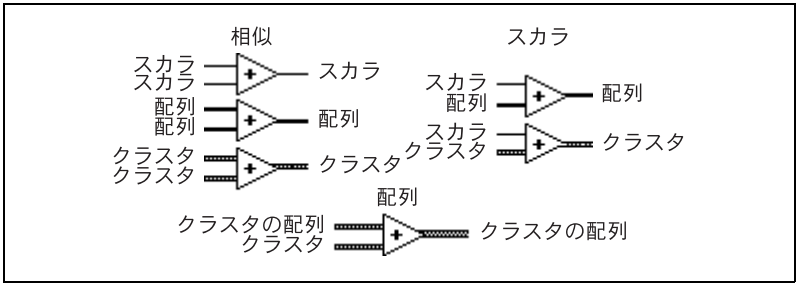
同一の入力の場合、LabVIEW はストラクチャの各要素について関数を実行します。たとえば、LabVIEW では 2 つの配列要素の加算は要素ごとに実行します。この場合、両方の配列が同じ次元である必要があります。要素数の異なる配列の加算も可能ですが、この場合の出力は最も少ない入力と同じ要素数の配列になります。クラスタは同じ要素数を持ち、各要素が同じタイプである必要があります。

multiply 関数を使用して行列を乗算することはできません。2 つの行列で multiply 関数を使用すると、LabVIEW は 1 行目の最初の数値どうしのように対応する要素ごとに乗算を行い、以下同様に処理を続けます。

スカラおよび配列またはクラスタを含む演算の場合、LabVIEW はスカラおよびストラクチャの各要素について関数を実行します。たとえば、配列の次元にかかわらず、LabVIEW は配列のすべての要素から同じ数値を減算できます。

ある数値タイプおよびそのタイプの配列を含む演算の場合、LabVIEW は各配列の要素について関数を実行します。たとえば、グラフはポイントの配列で、ポイントは *x* および *y* の 2 つの数値タイプのクラスタです。グラフを *x* 方向に 5 単位、*y* 方向に 8 単位移動するには、そのグラフに点 (5,8) を加えます。

次の例は Add 関数での可能な多形の組み合わせを示します。



ブール関数の多形性

論理関数はブールまたは数値入力データのいずれかを使用します。入力が数値の場合、LabVIEW はビット単位の演算を実行します。入力が整数の場合、出力は同じ表記法になります。入力が浮動小数点数の場合、LabVIEW は倍長整数に四捨五入するので出力も倍長整数になります。

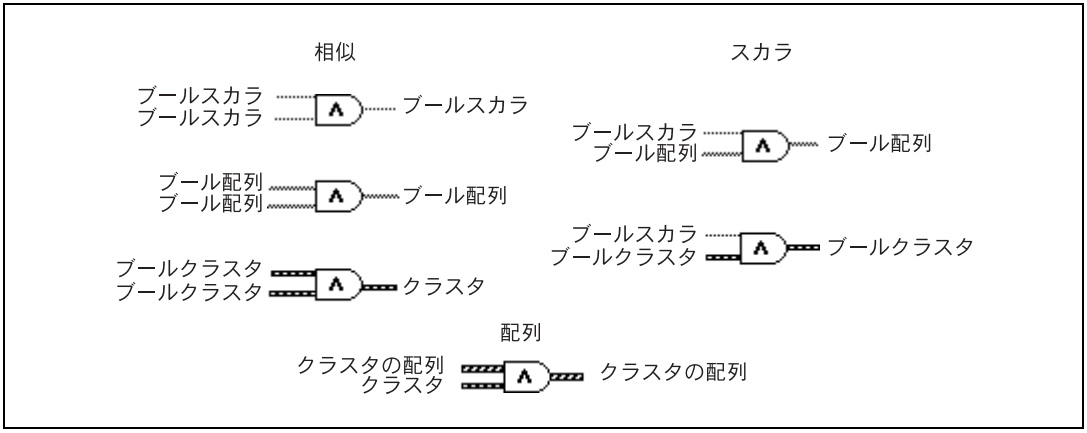
論理関数は、数値の配列またはブール値の配列、数値またはブール値のクラスタ、数値またはブール値のクラスタの配列などで実行できます。

使用可能な入力タイプの正式な再帰定義は次のとおりです。

論理タイプ = ブールスカラ OR 数値スカラ
OR 配列 (論理タイプ) OR クラスタ (論理タイプ)

複素数および配列の配列は使用できないので除外します。

2 つの入力がある論理関数では、演算関数と同じ入力の組み合わせを持つことができます。ただし、論理関数には基本操作が 2 つのブール値の間または 2 つの数値の間に限られるという制約があります。たとえば、ブール値と数値の間で AND は使用できません。次の例に AND 関数に対するブール値の組み合わせをいくつか示します。



配列関数の多形性

ほとんどの配列関数は、すべてのタイプの *n* 次元配列を受け入れます。ただし、関数の説明にある配線ダイアグラムでは、数値配列をデフォルトデータタイプとして示します。

文字列関数の多形性

String Length、To Upper Case、To Lower Case、Reverse String、および Rotate String は、文字列、クラスタ、文字列の配列、およびクラスタの配列を受け入れます。To Upper Case および To Lower Case は数値、数値のクラスタ、および数値の配列も受け入れ、それらは文字の ASCII コードとみなされます。幅および精度の入力はスカラである必要があります。

文字列変換関数の多形性

Path To String および String To Path 関数は多形性です。つまり、スカラ値、スカラの配列、スカラのクラスタ、スカラのクラスタの配列などを入力できます。出力は入力と同じ構成ですが、新しいタイプを含みます。

その他の文字列→数値変換関数の多形性

To Decimal、To Hex、To Octal、To Engineering、To Fractional、および To Exponential はクラスタおよび数値の配列を受け入れ、文字列のクラスタおよび文字列の配列を生成します。From Decimal、From Hex、From Octal、および From Exponential/Fract/Sci は文字列のクラスタや配列を受け入れ、数値のクラスタや配列を生成します。幅および精度の入力はスカラである必要があります。

クラスタ関数の多形性

Bundle および Unbundle 関数は、オブジェクトを入力または出力端子に配線するまで、それらの各端子のデータタイプを表示しません。配線すると、これらの端子は対応するフロントパネル制御器や表示器端子のデータタイプと同じように表示されます。

比較関数の多形性

Equal?、Not Equal?、および Select 関数は、入力が同じタイプであればどのようなタイプでも使用できます。

Greater or Equal?、Less or Equal?、Less?、Greater?、Max & Min および In Range? 関数は、入力が同じタイプであれば、複素数、パス、または refnum を除くすべてのタイプを使用できます。数値、文字列、ブール、文字列の配列、数値のクラスタ、文字列のクラスタなどを比較できます。ただし、数値と文字列の比較、文字列とブール値の比較などはできません。

値を 0 と比較する関数は、数値スカラ、クラスタ、および数値の配列を受け入れます。これらの関数は、入力と同じデータストラクチャのブール値を出力します。

Not A Number/Path/Refnum 関数は、値を 0 と比較する関数と同じ入力タイプを受け入れます。この関数は、パスや refnum も受け入れます。Not A Number/Path/Refnum 関数は、入力と同じデータストラクチャのブール値を出力します。

Decimal Digit?、Hex Digit?、Octal Digit?、Printable?、および White Space? 関数は、スカラ文字列やスカラ数値入力、文字列または非複素数のクラスタ、文字列または非複素数の配列などを受け入れます。出力は、入力と同じデータストラクチャのブール値から成ります。

Empty String/Path? 関数は、パス、スカラ文字列、文字列のクラスタ、文字列の配列などを受け入れます。出力は、入力と同じデータストラクチャのブール値から成ります。

Equal?、Not Equal?、Not A Number/Path/Refnum?、Empty String/Path?、および Select 関数ではパスや Refnum を使用できますが、他の比較関数はパスや Refnum を入力として受け入れません。

配列やクラスタを使用する比較関数は通常、ブール配列や同じストラクチャのクラスタを出力します。関数を右クリックして**基礎群の比較**を選択します。この場合、関数から 1 つのブール値が出力されます。関数は最初の要素セットの比較によって基礎群を比較し、最初の要素どうしが異なる場合は出力し、最初の要素どうしが等しい場合は 2 番目の要素セットを比較し、以下同様に処理を続けます。

対数関数の多形性

対数関数では数値入力データを使用できます。入力が整数の場合、出力は倍精度浮動小数点数になります。それ以外の場合、出力は入力と同じ数値表記法になります。

これらの関数は、数値、数値の配列、数値のクラスタ、数値のクラスタの配列、複素数などで実行できます。使用可能な入力タイプの正式な再帰定義は次のとおりです。

数値タイプ = 数値スカラ OR 配列 (数値タイプ)
OR クラスタ (数値タイプ)

配列の配列は使用できないので除外します。

配列にはサイズおよび次元数の制限はありません。クラスタの要素数にも制限はありません。出力タイプは入力と同じ数値表記法であり、関数はクラスタまたは配列の各要素に対して実行します。2 入力対数関数の詳細については、この章の「数値関数の多形性」のセクションを参照してください。以下は 2 入力対数関数の場合に使用可能な入力タイプの組み合わせを示します。

- **同一**：入力が両方とも同じストラクチャを持ち、出力が入力と同じストラクチャを持ちます。
- **1 つのスカラ**：一方の入力が数値スカラで、他方が数値配列またはクラスタの場合、出力は配列またはクラスタになります。

C

比較関数

ブール値、文字列、数値、配列、およびクラスタを比較するには、**関数→比較**パレットにある比較関数を使用します。ほとんどの比較関数は、1 つの入力をテストするか 2 つの入力を比較し、ブール値を返します。

詳細については

比較関数の詳細については、「LabVIEW ヘルプ」を参照してください。

ブール値を比較する

比較関数では、ブール値 TRUE の方がブール値 FALSE よりも大きいと見なされます。

文字列を比較する

LabVIEW では、ASCII 文字の数値表現に基づいて文字列が比較されます。たとえば、a (10 進数の 97) は A (65) よりも大きく、A は数字の 0 (48) よりも大きく、0 はスペース文字 (32) よりも大きいと見なされます。LabVIEW は、不一致が発生するまで文字列の先頭から 1 文字ずつ比較し、その時点で比較は終了します。たとえば、文字列 abcd と abef の場合、LabVIEW は e の値よりも小さい c が見つかるまで評価します。また、文字が存在する場合の方が、存在しない場合よりも大きいと見なされます。したがって、文字列 abcd は abc よりも長いので、前者の方が大きいと見なされます。

Decimal Digit? 関数や Printable? 関数など、文字列のカテゴリをテストする関数は、文字列の最初の文字だけを評価します。

数値を比較する

比較関数は、数値を同じ形式に変換してから比較します。NaN(Not a Number) 値を持つ 1 つまたは 2 つの入力との比較では、不一致を示す値が返されます。NaN 値の詳細については、第 6 章「VI の実行とデバッグ」の「不定データまたは予想外のデータ」のセクションを参照してください。

配列とクラスタを比較する

一部の比較関数には、データの配列またはクラスタを比較するための 2 つのモードがあります。基礎群の比較モードでは、2 つの配列またはクラスタを比較する場合、1 つのスカラ値が返されます。要素の比較モードでは、要素が個別に比較され、ブール値の配列またはクラスタが返されます。

基礎群の比較モードでは、文字列比較操作と配列比較操作はまったく同じプロセスに従います。つまり、文字列は ASCII 文字の配列と見なされます。

一部の比較関数は基礎群の比較モードでのみ動作するため、ショートカットメニューのオプションが表示されません。

配列

複数次元の配列を比較する場合は、関数に入力される各配列の次元数が同じとなる必要があります。基礎群の比較モードまたは要素の比較モードがない比較関数は、文字列の比較と同じ方法で配列を比較します。つまり、不一致が発生するまで最初の要素から要素を 1 つずつ比較します。

要素の比較モード

要素の比較モードでは、比較関数は入力配列と同じ次元のブール値配列を出力します。出力配列の各次元は、その次元の 2 つの入力配列のうち小さい方と同じサイズになります。各次元 (行、列、またはページなど) に沿って、関数は各入力配列内の対応する要素の値を比較し、対応するブール値を出力配列として生成します。

基礎群の比較モード

基礎群の比較モードでは、比較関数は 1 つのブール結果を出力します。LabVIEW は、入力配列内の各要素の対応値において、後ろの値を前の値の補助的なものと見なします。LabVIEW は、以下のステップを実行して比較の結果を確定します。

- LabVIEW は、各入力配列に対応する要素の比較を配列の先頭から開始します。
- 対応する要素が等しくない場合、LabVIEW は停止し、この比較の結果は比較関数の出力として使用されます。
- 対応する要素が等しい場合、LabVIEW は次の値のペアを処理し、不一致を検出するか、どちらかの入力配列の最後に達するまで比較を続けます。
- 入力配列内のすべての値が等しいのに、片方の配列の最後に余分な要素がある場合、長い方の配列は短い方の配列よりも大きいと見なされ

ます。たとえば、配列 [1,2,3,2] は配列 [1,2,3] よりも大きいと見なされます。

クラスタ

比較するクラスタには同じ数の要素が含まれていて、クラスタ内の各要素は互換性を持つタイプである必要があります。また、要素は同じクラスタ順位である必要があります。たとえば、倍精度数値と文字列を含むクラスタは、倍長整数数値と文字列を含むクラスタと比較できます。

要素の比較モード

要素の比較モードでは、比較関数が出力するクラスタには、入力クラスタ内の要素一つ一つに対応するブール要素が含まれています。

基礎群の比較モード

基礎群の比較モードでは、比較関数は 1 つのブール結果を出力します。LabVIEW は不一致が検出されるまで対応する要素を比較し、不一致が検出された時点で結果を確定します。2 つのクラスタが等しいと見なされるのは、すべての要素が等しい場合だけです。

ソート済みのデータを含んでいる 2 つのレコードを比較する場合は、クラスタに対して基礎群の比較モードを使用します。ここで、クラスタ内の後にある要素は、クラスタ内の前にある要素の補助的なものと見なされます。たとえば、2 つの文字列 (ラストネームの次にファーストネーム) を含んでいるクラスタでは、ラストネームのフィールドが一致した場合にのみファーストネームのフィールドが比較されます。

D

デジタルデータをマスクする

図 D-1 のように、ブロックダイアグラム上でデジタル波形と整数の 2 次元配列を組み合わせることによってマスクを作成します。

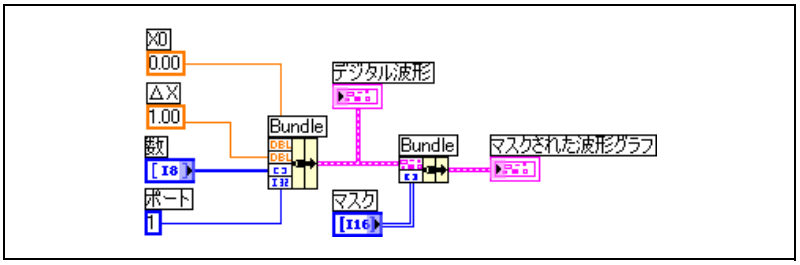


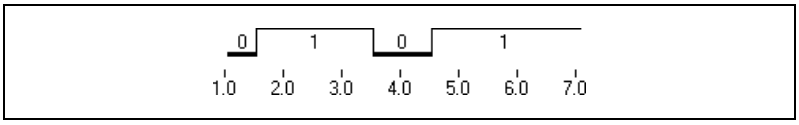
図 D-1　デジタルデータをマスクする

2 次元の**マスク**配列の各行に値を指定することによってビットを組み合わせます。

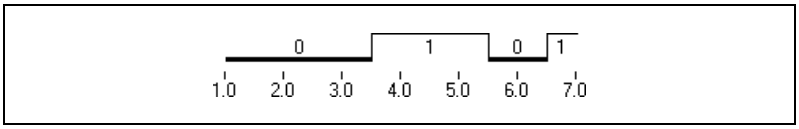
たとえば、1、2、7、32、55、82、127 の 7 つの 8 ビット数値から構成される 1 次元配列について考えてみます。次の表にこれらの数値の 2 進数表記を示します。

	1	2	7	32	55	82	127
ビット 0	1	0	1	0	1	0	1
ビット 1	0	1	1	0	1	1	1
ビット 2	0	0	1	0	1	0	1
ビット 3	0	0	0	0	0	0	1
ビット 4	0	0	0	0	1	1	1
ビット 5	0	0	0	1	1	0	1
ビット 6	0	0	0	0	0	1	1
ビット 7	0	0	0	0	0	0	0

マスクを使用すると、グラフに 1 つのビットをプロットし、他のビットを同じプロットに重ねることができます。たとえば、以下のプロットに示すように、ビット 1 には 0、1、1、0、1、1、1 の値が含まれています。



ビット 5 には、以下のプロットに示すように、0、0、0、1、1、0、1 の値が含まれています。



マスクは 2 次元配列です。配列の各行はデジタルグラフのプロットを表します。プロット上にビット 1 およびビット 5 を表示するには、図 D-2 のように配列の 1 つの要素に「1」を入力し、別の要素に「5」を入力します。配列の他のすべての要素に「-1」を入力します。これらの要素はビットをプロットしません。

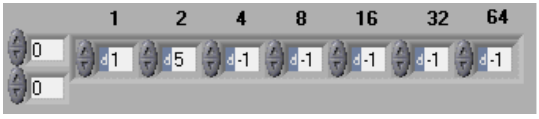
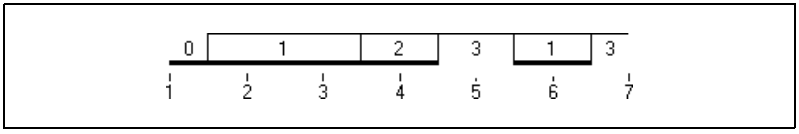


図 D-2　マスク配列制御器の例

この例では、配列の各要素は 2 の累乗として識別されます (2^0 ～ 2^6)。これらの数値を使用して、配列のどの要素がどのビットをプロットするのかを設定します。この例では、要素 1 がビット 1 をグラフにプロットするので、ビット内の値 1 はプロットに 1 を割り当て、値 0 は 0 をプロットに割り当てます。要素 2 はビット 5 をグラフにプロットするので、ビット内の値 1 は 2 をプロットに割り当て、値 0 は 0 をプロットに割り当てます。

図 D-2 の配列は、デジタル波形グラフの次のプロットになります。



このプロットは以下のことを示します。

- ビット 1 およびビット 5 の最初の要素には値 0 が含まれています。
- ビット 1 の 2 番目と 3 番目の要素には値 1 が含まれています。ビット 5 の 2 番目と 3 番目の要素には値 0 が含まれています。
- ビット 1 の 4 番目の要素には値 0 が含まれ、ビット 5 の 4 番目の要素には 1 が含まれています。
- ビット 1 およびビット 5 の 5 番目の要素には値 1 が含まれています。
- ビット 1 の 6 番目の要素には値 1 が含まれ、ビット 5 の 6 番目の要素には 0 が含まれています。
- ビット 1 およびビット 5 の 7 番目の要素には、値 1 が含まれています。

プロットに表示される数値は以下の方程式から導出します。

$$\frac{\begin{array}{c}〔\text{ビット値（1または0）}〕\times〔2〕^{\text{要素番号}} \\ +〔\text{その他のビット値（1または0）}〕\times〔2〕^{\text{その他の要素番号}} \\ \vdots \\ +〔\text{nビット値（1または0）}〕\times〔2〕^{\text{n要素番号}}\end{array}}{\text{プロットのポイント値}}$$

図 D-3 では、4 番目 (8) と 6 番目 (32) の要素がそれぞれビット 5 とビット 1 をプロットします。

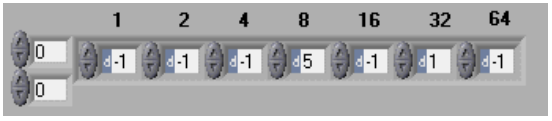
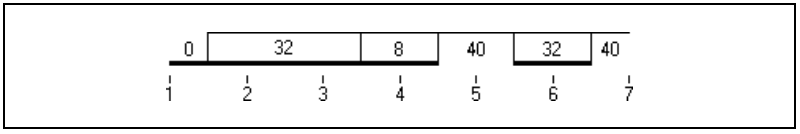


図 D-3　異なる要素で同じビットを指定する

図 D-3 の配列は結果的に次に示すプロットになります。プロットは似ていますが、数値が異なっていることに注意してください。



マスクの各行はグラフ上のプロットに対応しています。マスクに新しい行を追加して、図 D-4 のようにデジタルグラフ上に新しいプロットを追加します。

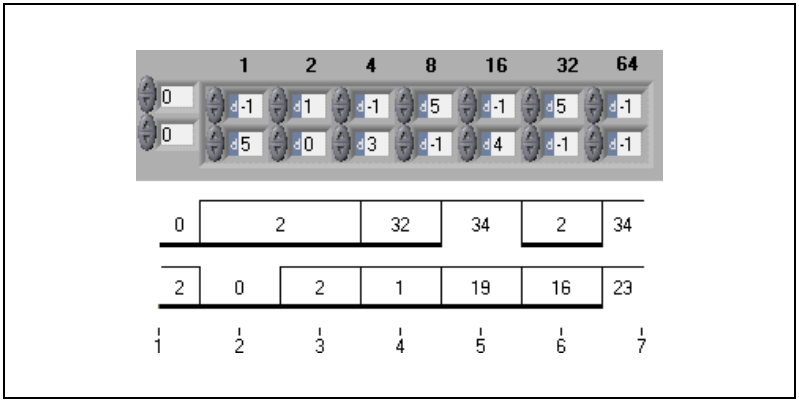


図 D-4　デジタル波形グラフ上にマスクされたデータのプロットを 2 つ作成する

E

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

接頭語	意味	値
m-	ミリ	10^{-3}
k-	キロ	10^{3}
M-	メガ	10^{6}

数字または記号

Δ　デルタ。偏差。Δ*x* は、ある指標から次の指標まで *x* の変化値を示します。

π　パイ (Pi)。

∞　無限。

1D　1 次元。

1 次元　たとえば、1 行の要素のみを持つ配列の場合のように、1 つの次元を持つこと。

2D　2 次元。

2 次元　たとえば、複数の行と列を持つ配列の場合のように、2 つの次元を持つこと。

3D カーブ　特殊なパラメトリックプロット (*x*(*t*)、*y*(*t*)、*z*(*t*))。ここで、パラメータ *t* は指定された間隔で実行されます。

A

A　アンペア。

AC　交流。

ASCII　情報交換用米国標準コード (American Standard Code for Information Interchange)。

C

CIN — 「コードインタフェースノード (CIN)」の項を参照。

D

DAQ — 「データ集録」の項を参照。

DAQ チャネル ウィザード — DAQ のアナログチャネルおよびデジタルチャネルの命名および構成を手引きするユーティリティ。Measurement & Automation Explorer のデータ設定 **(Windows)**、または DAQ チャネルウィザード **(Macintosh)** で使用できます。

DDE — 「ダイナミックデータ 交換」の項を参照。

DLL — ダイナミックリンクライブラリ (Dynamic Link Library)。

F

FIFO — 先入れ先出し (First-in-first-out) のメモリバッファ。最初に格納されたデータは最初に受け入れ側に送られます。

For ループ — サブダイアグラムを一定回数実行するループストラクチャ。以下のようなテキストベースのコードに相当します。`For i = 0 to n - 1, do...`

G

GPIB — 「汎用インタフェース バス (GPIB:General Purpose Interface Bus)」の項を参照。

H

hex — 16 進数。基数を 16 とする数。

I

IEEE — 米国電子電気技術者協会 (Institute for Electrical and Electronic Engineers)。

`Inf` — 無限を表す浮動小数点のデジタル表示値。

I/O　入出力。通信チャネル、オペレータ入力装置、またはデータ集録インタフェースおよびデータ制御インタフェースを使用して、コンピュータシステムとの間で行うデータの転送。

IP　インターネットプロトコル。

L

LabVIEW　Laboratory Virtual Instrument Engineering Workbench (ラボラトリ仮想計測器エンジニアリングワークベンチ)。LabVIEW は、テキスト行ではなくアイコンを使用してプログラムを作成するグラフィカルプログラミング言語です。

LED　発光ダイオード (Light-emitting diode)。

LLB　VI ライブラリ。

M

Measurement & Automation Explorer　ナショナルインスツルメンツの Windows 用のハードウェア構成および診断環境。

N

NaN　「数字ではない」ことを表す浮動小数点のデジタル表示値。通常は、log(−1) などの不定の演算結果です。

NI-DAQ　ナショナルインスツルメンツの DAQ ハードウェアと同梱されている包括的なドライバソフトウェア。

O

OLE　オブジェクトのリンクと埋め込み (Object Linking and Embedding)。

P

PPC　プログラム間の通信 (Program-to-program communication)。

PXI　計測器用の PCI の拡張 (PCI eXtensions for Instrumentation)。モジュール形式、コンピュータベースの計測器プラットフォーム。

R

refnum　リファレンス番号。開いたファイルに対し LabVIEW によって関連付けられた識別子。開いているファイルに対して関数または VI に操作を実行させる場合に refnum を使用します。

S

SCXI　信号調節拡張機構 (Signal Conditioning eXtensions for Instrumentation)。ナショナルインスツルメンツのセンサ付近の外部シャーシ内の条件付き低レベル信号用製品ライン。ノイズの多い環境の高レベル信号のみが DAQ デバイスに送信されます。

T

TCP　Transmission Control Protocol。

U

UDP　User Datagram Protocol。

V

VI　「バーチャル インスツルメンツ (VI：仮想計測器)」の項を参照。

VISA　「バーチャルインスツルメンツソフトウェアアーキテクチャ (仮想計測器ソフトウェアアーキテクチャ)」の項を参照。

VI クラス　VI のプロパティおよびメソッドにアクセスできる仮想計測器に対するリファレンス。

VI サーバ　プログラムによってローカルにもリモートにも VI や LabVIEW を制御するメカニズム。

VI ライブラリ　特定の用途に関連する VI 群を含む特殊ファイル。

VXI　VME eXtensions for Instrumentation (バス)。

W

While ループ	一定の条件が満たされるまでコードの一部を繰り返し実行するループストラクチャ。

あ

アイコン	ブロックダイアグラム上のノードの図式的な表示。
アクティブウィンドウ	現在ユーザの入力が可能なウィンドウで、通常は一番上に表示されているウィンドウです。アクティブウィンドウのタイトルバーはハイライトされます。ウィンドウをアクティブにするには、アクティブにするウィンドウをクリックするか、**ウィンドウ**メニューから選択します。
アプリケーションソフトウェア	LabVIEW 開発システムを使用して作成され、LabVIEW ランタイムシステム環境で実行されるアプリケーション。
位置決めツール	オブジェクトを移動したりサイズを変更するツール。
イベント	アナログまたはデジタル信号の状態。
イベントデータノード	イベントストラクチャの左右に添付される、そのケースが処理するために構成した利用可能なデータを示すノード。ある 1 つのケースが複数のイベントを処理するように構成した場合、処理されたすべてのイベントタイプに共通するデータのみを使用することができます。
色付けツール	前景色および背景色を設定するためのツール。
エラー出力	VI から出るエラーストラクチャ。
エラーストラクチャ	ブール値ステータス表示器、数値コード表示器、および文字列ソース表示器で構成されています。
エラー入力	VI に入るエラーストラクチャ。
エラーメッセージ	ソフトウェアやハードウェアの動作不能または受け入れられないデータ入力が行われたことを示します。
オートスケーリング	スケールがプロットした値の範囲に合わせて調整される機能。グラフのスケールでは、この機能によりスケールの最大値と最小値が決まります。
オブジェクト	制御器、表示器、ノード、ワイヤ、取り込んだピクチャなど、フロントパネルまたはブロックダイアグラム上の項目の総称。
オブジェクトショートカットメニューツール	オブジェクトのショートカットメニューにアクセスするためのツール。

か

階層ウィンドウ VI およびサブ VI の階層を図で表示するウィンドウ。

外部トリガ A/D 変換などのイベントをトリガする外部ソースからの電圧パルス。

カウント端子 For ループの端子で、この値によって For ループがサブダイアグラムを実行する回数が決まります。

カラーコピーツール 色付けツールでペーストするために色をコピーします。

空の配列 要素が 0 で、データタイプが定義されている配列。たとえば、データ表示ウィンドウに数値制御器があるにも関わらずどの要素にも値が定義されていない配列は、空の数値配列です。

関数 内蔵されている実行要素で、テキストベースのプログラミング言語の演算子、関数、またはステートメントに相当します。

関数パレット VI、関数、ブロックダイアグラムストラクチャ、および定数を含んでいるパレット。

競合状態 並列に実行される 2 つ以上のコードが、グローバル変数またはローカル変数で代表される同じ共有リソースの値を変更する場合に発生します。

強制 データ要素の数値表記法を変更するため、LabVIEW が実行する自動変換。

強制ドット 端子上のドットで、配線された 2 個の端子のうちの一方が LabVIEW によって他方の端子のデータタイプに合わせて変換されたことを示します。

行列 2 次元の配列。

クラスタ 数値、ブール値、文字列、配列、またはクラスタなど、順序付けられてはいるが指標付けされていない任意データタイプの集合。要素は、すべて制御器、またはすべて表示器のいずれかである必要があります。

グラフ 1 つ以上のプロットの 2 次元表示。グラフではブロックとしてデータを受け取り、プロットします。

グラフ制御器 デカルト平面上にデータを表示するフロントパネルオブジェクト。

繰り返し端子 現時点までに完了した繰り返し回数が入っている For ループまたは While ループの端子。

グリフ 小さなピクチャまたはアイコン。

グループ　ユーザが定義した入力チャネルや出力チャネル、またはポートの集合。グループには、アナログ入力、アナログ出力、デジタル入力、デジタル出力、またはカウンタ／タイマチャネルなどが含まれます。1 つのグループには 1 種類のチャネルのみが含まれています。作成後は、グループの参照にタスク ID 番号を使用します。一度に最大で 16 個のグループを定義できます。

グループを解消するには、空のチャネル配列およびグループ番号をそのグループ構成 VI に渡します。グループのメンバを変更する場合にグループを消去する必要はありません。タスクがアクティブになっているグループを再構成すると、LabVIEW はタスクを消去し、警告を返します。グループの再構成後、LabVIEW はそのタスクを再開しません。

グローバル変数　ブロックダイアグラム上の複数の VI 間でデータにアクセスし、データのやり取りを行います。

クローン化　制御器や他のオブジェクトをコピーすること。<Ctrl> キーを押したまま、コピーする制御器やオブジェクトをクリックし、新しい位置までコピーをドラッグします。

(Macintosh)<Option> キーを押します。**(Sun)**<Meta> キーを押します。**(Linux)**<Alt> キーを押します。

(UNIX) 中央のマウスボタンでオブジェクトをクリックし、そのコピーを新しい位置にドラッグすることもできます。

計測器ドライバ　プログラム可能な計測器を制御する VI。

ケース　ケースストラクチャの一つのサブダイアグラム。

ケースストラクチャ　ケースストラクチャへの入力に基づいてそのサブダイアグラムのいずれかを実行する条件分岐制御ストラクチャです。制御フロー言語では、IF、THEN、ELSE、および CASE ステートメントの組み合わせになります。

現在の VI　フロントパネル、ブロックダイアグラム、またはアイコンエディタがアクティブウィンドウになっている VI。

構成ユーティリティ　Windows の場合は「Measurement & Automation Explorer」、Macintosh の場合には「NI-DAQ」の項を参照。

構文　特定のプログラミング言語においてステートメントが遵守する必要がある一連のルール。

コードインタフェースノード (CIN)　CIN。ブロックダイアグラムの特殊ノードで、テキストベースのコードを VI にリンクする場合に使用します。

コネクタ　入力端子と出力端子を含んでいる VI または関数ノードの一部。ノードとのデータのやり取りはコネクタを経由して行われます。

コネクタペーン　フロントパネルウィンドウまたはブロックダイアグラムウィンドウの右上コーナーにあり、VI 端子のパターンを表示する領域。これによって VI に配線できる入力と出力が定義されます。

壊れた VI　エラーのため実行ができない VI。壊れた**実行**ボタンに壊れた矢印が表示されます。

壊れた**実行**ボタン　エラーのため VI が実行できない場合に**実行**ボタンから置き換わるボタン。

コンパイル　高レベルコードをコンピュータが実行できるコードに変換するプロセス。VI を作成または修正後初めて実行する前に、LabVIEW では VI が自動的にコンパイルされます。

さ

最上位 VI　VI 階層の最上位にある VI。サブ VI から VI を区別する名称。

サイズ変更ハンドル　ポイントのサイズ変更を行う、オブジェクトのコーナーにある斜めのハンドル。

サブ VI　他の VI のブロックダイアグラムで使用される VI。サブルーチンに相当します。

サブダイアグラム　ストラクチャの枠内にあるブロックダイアグラム。

サンプル　アナログまたはデジタルの 1 つの入力または出力データ点。

シーケンスストラクチャ　数値順にサブダイアグラムを実行するプログラム制御ストラクチャ。データに依存しないノードを指定した順に強制実行するときに使用します。

シーケンスローカル　シーケンスストラクチャのフレーム間でデータのやり取りを行う端子。

四角形　4 つの 16 ビット整数値を含んでいるクラスタ。最初の 2 つの値は左上のコーナーの垂直および水平座標を示します。最後の 2 つの値は右下のコーナーの垂直および水平座標を示します。

実行のハイライト　VI のデータフローを示すため VI の実行を図式的に表示するデバッグテクニック。

自動指標付け	ループストラクチャがその境界で配列を分解したり組み立てたりする機能。自動指標付けをオンにした状態で配列がループに入ると、ループは自動的にそれを 1 次元配列から抽出したスカラや 2 次元配列から抽出した 1 次元配列などに分解します。配列がループを出るとき、ループはその逆の手順でデータを配列に組み立てます。
シフトレジスタ	ループストラクチャのオプションのメカニズムで、ループの一つの繰り返しから次の繰り返しへ変数の値を渡します。シフトレジスタは、テキストベースのプログラミング言語におけるスタティック変数に似ています。
条件端子	VI が次の繰り返しを実行するかどうかを決めるブール値を含んでいる While ループの端子。
ショートカットメニュー	オブジェクトを右クリックすることによってアクセスできるメニュー。メニューオプションはそのオブジェクトに固有のものです。
人工的データ依存	データフロープログラミング言語において、データの値ではなくデータの到着によってノードの実行がトリガされる状態。
スイープチャート	オシロスコープの動作を模倣した数値制御器。スコープチャートに似ています。異なるのは、ディスプレイが線で区切られ、古いデータと新しいデータが分離される点です。
数値制御器とブール表示器	数値データの操作および表示に使用するフロントパネルオブジェクト。
スカラ	スケール上の一点で表すことができる数字。配列とは異なり、スカラは 1 つの値です。スカラのブール値とクラスタはそれぞれのデータタイプの明示的な単一のインスタンスです。
スクロールツール	ウィンドウ内の移動に使用するツール。
スケール	チャート、グラフ、および数値制御器や表示器の一部で、測定単位を示すために一定の間隔で一連のマークまたは点を付けた部分。
スコープチャート	オシロスコープの動作を模倣した数値制御器。
ストラクチャ	シーケンスストラクチャ、ケースストラクチャ、For ループ、While ループなどのプログラム制御要素。
ストリップチャート	紙テープに記録するチャートレコーダを模倣した数値プロット表示器で、データをプロットするたびにスクロールします。
スライダ	スライド制御器および表示器の可動部分。

制御器　対話式で VI にデータを入力したり、プログラムによってサブ VI にデータを入力するためのノブ、押しボタン、ダイアルなどのフロントパネルオブジェクト。

制御器パレット　フロントパネル制御器、表示器、装飾体オブジェクトを含むパレット。

制御フロー　命令の順序によって実行順序が決まるプログラミング体系。テキストベース言語の多くは制御フロー言語です。

整数　自然数、負の自然数、または 0 のいずれか。

絶対座標　画像表示器の原点 (0,0) を基準とした画像座標。

絶対パス　ファイルシステムの最上位を基準として場所を示す、ファイルやディレクトリのパス。

操作ツール　制御器にデータを入力したり、制御器を操作するためのツール。

相対座標　現在のペンの位置を基準にしたピクチャ座標。

た

ダイナミックデータ交換　DDE。ユーザの介入や監視なしにアプリケーション間のデータの移動を行う方法。

タイプ定義　複数の VI が使用できるカスタムオブジェクトのマスタコピー。

多形性　異なる表現、タイプ、またはストラクチャのデータに応じてノードが自動的に調整できる機能。

端子　データのやり取りが行われるノード上のオブジェクトまたは領域。

チェックボックス　選択や選択解除ができるダイアログボックス内の小さな正方形。チェックボックスは、通常ユーザによる設定が可能な複数のオプションに関連付けられています。複数のチェックボックスを選択することができます。

チャート　1 つまたは複数のプロットの 2 次元表示です。表示には、ユーザが定義する最大限度まで過去のデータが保持されます。チャートはデータを受け取り、表示を点または配列ごとに更新し、表示目的でバッファに過去の点を一定数保持します。「スコープチャート」、「ストリップチャート」、および「スイープチャート」の項も参照。

チャネル　ピンまたはワイヤで、適用するものに接続したり、これらからアナログまたはデジタル信号を読み取ります。アナログ信号はシングルエンドと差動のどちらかです。デジタル信号の場合は、チャネルをグループ化してポートを形成します。通常、ポートは 4 個または 8 個のデジタルチャネルで構成されています。

チャネル名　DAQ チャネルウィザードでチャネル構成に指定する固有の名前。

ツール　特定の操作を実行する特殊カーソル。

ツールバー　VI の実行やデバッグに使用するコマンドボタンが並んでいるバー。

ツールパレット　フロントパネルおよびブロックダイアグラムオブジェクトの編集およびデバッグに使用するツールで構成されているパレット。

ディザリング　アナログ入力信号にガウスノイズを加えること。ディザリングを適用した後、入力データを平均化することによって効果的に、さらに 0.5 ビットだけ分解能を増やすことができます。

定数　「ユニバーサル定数」および「ユーザ定義定数」の項を参照。

ディレクトリ　ファイルを扱いやすいグループに分けるストラクチャ。ディレクトリとは、ファイルの位置を示すアドレスのようなもので、中にはファイルやファイルのサブディレクトリが入っています。

データ依存　データフロープログラミング言語で、別のノードからデータを受け取るまでノードを実行できない状態。「人工的データ依存」の項も参照。

データ集録　DAQ。データを集録するプロセス。通常、A/D またはデジタル入力プラグインデバイスから集録します。

データタイプ　情報の形式。LabVIEW では、数値、配列、文字列、ブール値、パス、refnum、列挙型、波形、クラスタといったデータタイプをほとんどの VI および関数に使用できます。

データフロー　実行可能なノードで構成されるプログラミング体系。ノードは必要な入力データをすべて受け取った場合にのみ実行され、実行されると自動的に出力を作成します。LabVIEW はデータフローシステムです。

データロギング　一般的に、データを集録し、同時にそれをディスクファイルに格納することです。LabVIEW のファイル入出力 VI および関数はデータロギングできます。

データログファイル　ファイル作成時に指定した 1 つの任意データタイプの一続きのレコードとしてデータを格納するファイル。データログファイルのレコードはすべて 1 つのタイプである必要がありますが、そのタイプが複合的であっても支障ありません。たとえば、各レコードを文字列、数値、配列を含むクラスタとして指定できます。

デバイス　1 つの構成要素としてアドレス指定でき、実環境の入出力ポイントを制御および監視する計測器またはコントローラ。多くの場合、デバイスはいずれかのタイプの通信ネットワークを介してホストコンピュータに接続されます。プラグインデバイスの場合もあります。

データ集録 (DAQ) アプリケーションでは、DAQ デバイスはコンピュータ内にあるか、コンピュータのパラレルポートに直接接続されています。プラグインボード、PCMCIA カード、DAQPad-1200 のようなコンピュータのパラレルポートに接続するデバイスは、すべて DAQ デバイスの例です。多機能データ集録モジュールの SCXI-1200 を除き、SCXI モジュールはデバイスとは異なります。

デフォルト　あらかじめ設定されている値。値を指定しない場合に、多くの VI 入力にデフォルト値が使用されます。

ドライバ　DAQ デバイスなどの、特定のハードウェアデバイスを制御するソフトウェア。

ドライブ　a ～ z までの文字にコロン (:) が付いたもので、論理ディスクドライブを示します。

ドラッグ　オブジェクトの選択、移動、コピー、または削除を行うために、画面上のカーソルを使用すること。

トンネル　ストラクチャ上のデータ入力端子または出力端子。

な

ノード　プログラム実行要素。ノードは、テキストベースのプログラミング言語のステートメント、演算子、関数、およびサブルーチンに似ています。ブロックダイアグラムでは、関数、ストラクチャ、およびサブ VI が含まれます。

は

バーチャルインスツルメンツ (VI：仮想計測器)　実際の計測器の外観や機能を模倣した LabVIEW のプログラム。

バーチャルインスツルメンツソフトウェアアーキテクチャ (仮想計測器ソフトウェアアーキテクチャ)	VISA。GPIB、VXI、RS-232、その他のタイプの計測器を制御する単独インタフェースライブラリ。
バイトストリームファイル	一連の ASCII 文字またはバイトでデータを格納するファイル。
配列	一定の順序で並べられ、指標付けされている同一タイプのデータ要素。
波形	特定のサンプリングレートで読み取った複数の電圧の読み取り値。
波形チャート	一定の速度でデータポイントをプロットする表示器。
バッファ	集録されたデータや生成されたデータの一時格納場所。
パネルウィンドウ	フロントパネル、ツールバー、およびアイコンと、コネクタペーンを含む VI ウィンドウ。
パレット	使用可能なオプションを表すアイコン表示。
範囲	測定、受信、または送信される数量の上限と下限の間の領域。範囲の上限値と下限値を示すことによって表現します。
ハンドル	ブロックメモリのポインタを指すポインタで、配列と文字列の参照を管理します。文字列の配列は、文字列に対するハンドルが入っているメモリブロックを参照するハンドルです。
汎用インタフェースバス (GPIB:General Purpose Interface Bus)	GPIB は、HP-IB と同義語です。電子計測器制御にコンピュータで使用される標準バス。ANSI/IEEE 規格集 488-1978、および 488.1-1987、488.2-1992 に既定されているため、IEEE488 バスとも呼ばれています。
凡例	チャートまたはグラフが所有するオブジェクトで、そのチャートまたはグラフにプロット名とプロット方式を表示します。
ピクセル	デジタル化されたピクチャの最小単位。
ピクセルマップ	画像を格納する標準フォーマットで、各ピクセルを色の値で表します。ビットマップは、ピクセルマップを白黒で表示したものです。
ピクチャ	ピクチャ表示器によって画像の作成に使用される一連の図式的な手順。
ピクチャ表示器	線、円、テキスト、その他の画像形状を含むことがあるピクチャを表示する汎用表示器。

表現	数値データタイプのサブタイプ。符号付きと符号なしバイト整数、ワード整数、倍長整数のほか、単精度、倍精度、拡張精度の浮動小数点数があります。
表示器	グラフや LED などの出力を表示するフロントパネルオブジェクト。
ヒントラベル	端子名を識別する小さな黄色のテキストバナーで、配線する端子の識別を容易にします。
ファイル refnum	「refnum」の項を参照。
フィルタ処理	測定する信号から不要な信号を除去する信号処理の一種。
ブール制御器とブール表示器	ブール (TRUE または FALSE) データの操作および表示に使用するフロントパネルオブジェクト。
フォーミュラノード	テキストとして入力された式を実行するノード。式が長く、ブロックダイアグラムの形式で作成するのが面倒な場合に特に効果的です。
フリーラベル	フロントパネルまたはブロックダイアグラム上のラベルで、他のどのオブジェクトにも属さないもの。
プルダウンメニュー	メニューバーからアクセスするメニュー。通常、プルダウンメニューのオプションは一般的なものです。
ブレークポイント	デバッグに使用される、実行の一時停止。
ブレークポイントツール	VI、ノード、またはワイヤにブレークポイントを設定するためのツール。
フレーム	シーケンスストラクチャのサブダイアグラム。
プログラムで印刷	実行後、VI フロントパネルを自動的に印刷する機能。
ブロックダイアグラム	プログラムまたはアルゴリズムを図式的に説明または表記したもの。ブロックダイアグラムは、ノードという実行可能なアイコンと、ノード間でデータをやり取りするワイヤで構成されています。ブロックダイアグラムは VI のソースコードです。ブロックダイアグラムは VI のブロックダイアグラムウィンドウにあります。
プロット	グラフまたはチャートのいずれかに表示される、データ配列の図式的な表示。
プロパティノード	VI またはアプリケーションのプロパティを設定したり、検出します。
プローブ	VI の中間値をチェックするデバッグ機能。

プローブツール　ワイヤ上にプローブを作成するツール。

フロントパネル　VI の対話式ユーザインタフェース。フロントパネルの外観はオシロスコープやマルチメータなどの実際の計測器を模倣したものです。

平坦化されたデータ　文字列に変換された何らかのタイプのデータで、通常はファイルに書き込むためのものです。

ベクトル　1 次配列。

ヘルプウィンドウ　VI または関数の端子名および位置、制御器および表示器の説明、ユニバーサル定数の値、制御器属性の説明およびデータタイプを表示する、LabVIEW の特殊なウィンドウ。

変換　データ要素のタイプを変更します。

ポイント　水平および垂直座標を表す 2 つの 16 ビット整数を含んでいるクラスタ。

ま

マルチスレッドアプリケーション　複数の異なる実行スレッドを別々に実行するアプリケーション。マルチプロセッサコンピュータでは、異なるスレッドを別個のプロセッサ上で同時に実行できます。

メソッド　オブジェクトがメッセージを受け取ったときに実行される手順。メソッドは常にクラスに関連付けられています。

メニューバー　アプリケーションのメインメニューの名前を一覧表示する水平バー。メニューバーは、ウィンドウのタイトルバーの下に表示されます。メニューおよびコマンドの中には多くのアプリケーションに共通するものもありますが、各アプリケーションのメニューバーはそれぞれ異なります。

メモリバッファ　「バッファ」の項を参照。

文字列　テキストとしての値の表現。

文字列制御器および表示器　テキストの操作および表示に使用するフロントパネルオブジェクト。

や

ユーザ定義定数　設定した値を送り出すブロックダイアグラムオブジェクト。

ユニバーサル定数　特定の ASCII 文字や標準の数値定数を送出する、編集不可能なブロックダイアグラムオブジェクト。π など。

ら

ライブラリ　「VI ライブラリ」の項を参照。

ラベリングツール　ラベルを作成し、テキストウィンドウにテキストを入力するためのツール。

ラベル　フロントパネルやブロックダイアグラム上で、オブジェクトや領域に名前を付けたり説明を加えたりするために使用するテキストオブジェクト。

離散　独立した変数が不連続の値を持つこと。通常、時間がこれに該当します。

リストボックス　コマンドに対する選択肢をすべて一覧表示するダイアログボックス内のボックス。たとえば、ディスク上のファイル名の一覧表示などがあります。

リング制御器　32 ビット整数を一連のテキストラベルやグラフィックに結び付ける特殊な数値制御器。0 から始まり順次増分します。

ローカル変数　VI のフロントパネル上の制御器または表示器に読み取りや書き込みができるようにする変数。

わ

ワイヤ　ノード間のデータパス。

ワイヤ屈折点　2 本のワイヤセグメントの接合点。

ワイヤスタブ　配線ツールを VI または関数ノード上に移動したときに未配線の端子部分に表示される短いワイヤ。

ワイヤセグメント　水平または垂直な 1 本のワイヤ。

ワイヤ接合点　3 本以上のワイヤセグメントの結合点。

ワイヤツール　端子間のデータパスを定義します。

ワイヤブランチ　ある接合点から別の接合点まで、ある端子から次の接合点まで、端子間に接合点がない場合は端子から端子までの、すべてのワイヤセグメントを含むワイヤの部分。

索引

F

G

H

I

J

L

M

う

え

お

か

き

け

こ

さ

し

す

せ

そ

た

ち

つ

て

と

な

に

ね

の

は

ひ

ふ

へ

ほ

ま

む

め

も

ゆ

よ

ら

り

る

れ

ろ

わ