FIRSTCLASS® RAD 2.6
日本語版

リファレンスマニュアル

- もくじ -

1 インストール

● はじめに

動作環境

サーバー側

FirstClass サーバ 7.1 または 8.0 が動作している Windows 機。

Mac OS では動作しません。

クライアント側

インストールおよび FirstClass RAD アプリケーション開発時には、FirstClass クライアントおよび FirstClass デザイナーが必要となります。

FirstClass RAD アプリケーションの開発および利用は Macintosh 版クライアントおよび FirstClass デザイナーでも問題なく行うことができます。

制限事項（開発環境と実行環境）

サーバーの文字セットが日本語に設定してあるサーバーの場合、プログラムコードの編集に不具合が発生するため、開発用サーバーとして使用することができません。

また、文字セットを英語に設定してある場合には日本語のメッセージ交換に一部支障が出るため、使用中のサーバーを英語に設定することはできません。

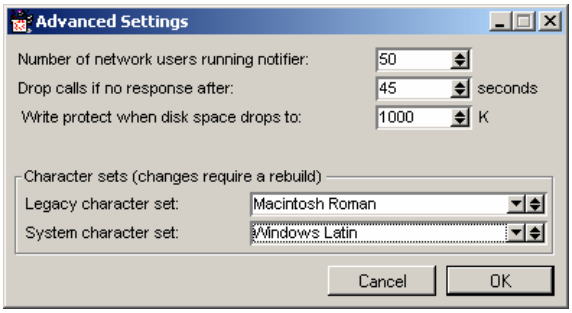
したがって、英語に設定した開発用マシンを用意し、そのマシンに FirstClass RAD をインストールしてアプリケーションの開発を行ってください。

開発が終了したら、そのビルドしたアプリケーションのみを日本語に設定した FirstClass サーバーにインストールしてご使用ください。

準備

開発用マシンに、FirstClass サーバーをインストールしてください。(詳細な手順は、「FirstClass サーバーリファレンスマニュアル」をご参照ください)

サーバーインストール後、FirstClass サーバーを閉じ、FirstClass Tools を起動してください。

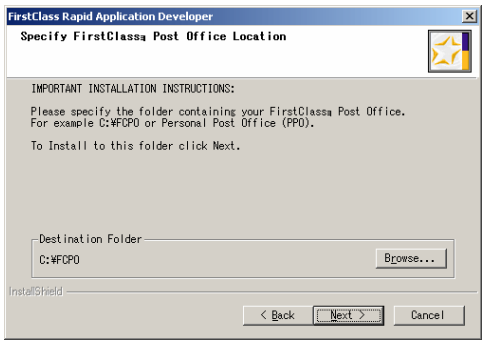


ツールバーの [Configure] > [Advanced Settings]を選択し、開いたウィンドウ内の[Legacy character set:] が Macintosh Roman、[System character set:]が Windows Latin になっていることを確認してください。

● インストール作業

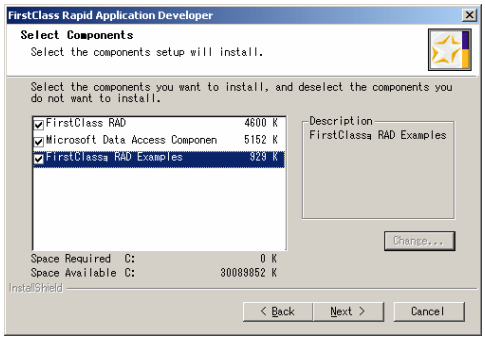
プログラムのインストール

FirstClass サーバーのウィンドウを閉じ、サーバー機上で FCRAD26.EXE をダブルクリックしてインストーラーを起動し、画面の指示に従ってください。



インストール先は、FirstClass サーバーをインストールしてある一次ボリュームの¥FCNS（バージョン 7.1 では¥FCPO）にしてください。

インストールするコンポーネントの選択



- FirstClass RAD
- Microsoft Data Access Component (MDAC)
- FirstClass RAD Examples

の三つにチェックを入れてください。

インストール中に、Microsoft Data Access Components 2.7 Setup の画面が出ますので、指示に従ってインストールを行ってください。

インストール後、サーバー機の再起動を行います。

● インストール後の設定

Windows Update の実行

インストールされた MDAC にセキュリティホールがある可能性がありますので、Windows Update を実行して必要な更新をインストールしてください。このときにサーバー機の再起動が必要となる場合があります。

RAD 日本語化ファイルのインストール

RAD26JDATA.EXE を実行し、解凍された FCRAD フォルダを、FirstClass サーバーがインストールしてある一次ボリュームの¥FCNS（7.1 では¥FCPO）フォルダ内に上書きコピーしてください。

サーバーの起動

FirstClass サーバーを起動してください。

日本語フォームデータのインストール

1. クライアントソフトを使用し、サーバーに admin でログインしてください。
2. [Multi-Site Setup]＞[FC Resource Registry]フォルダ内に Japanese Resources ファイルがあるかどうかを確認してください。
3. Japanese Resources ファイルがない場合には、English Resources ファイルを複製し、Japanese Resources に名前を変更してください。
4. Japanese Resources をダウンロードして、FirstClass デザイナーで開いてください。
5. CD-ROM 内の RAD.fc ファイルも FirstClass デザイナーで開き、RAD.fc 内のリソースを全て Japanese Resources 内にコピーして、FirstClass デザイナーを終了させてください。
6. 上記でフォームデータを入れた Japanese Resources ファイルを FC Resources Registry フォルダにアップロードし、古いほうの Japanese Resources ファイルを削除して下さい。

プログラム開発を行うユーザーの設定

1. クライアントソフトを使用し、サーバーに admin でログインしてください。
2. RAD プログラムを開発するユーザーを RAD Developer グループに所属させてください。(admin アカウントで開発を行う場合にも、admin を RAD Developer グループに所属させる必要があります)
3. さらに、デスクトップにある FirstClass RAD フォルダのエイリアスを上記ユーザーのデスクトップに置いてください。

2 FirstClass RAD の概要

● FirstClass RAD 日本語版

FirstClass RAD について

FirstClass Rapid Application Developer 日本語版 (FirstClass RAD) を利用すると、データベースへアクセスするためのインターフェースを含むイントラネットアプリケーションを使って、FirstClass クライアントの機能を強化することができます。

FirstClass RAD は、FirstClass サーバーをインストールした Windows 機にインストールしなければなりませんが、FirstClass RAD を動かすための FirstClass クライアントは Windows または Macintosh のいずれでも使うことができます。また、ユーザーも、Windows と Macintosh のどちらの FirstClass クライアントを使っても、管理者が作成した FirstClass RAD アプリケーションを動かすことができます。

FirstClass RAD アプリケーションを作成するには、FirstClass デザイナーを使って必要となるフォームをつくり、FirstClass RAD で、オブジェクトのためのイベントプロシージャを BASIC 言語でプログラミングします。

FirstClass RAD 名前付けの命名規則

フォームとフィールドや、変数、定数、サブルーチンといったコード要素に名前をつけるとき、以下の決まりにしたがう必要があります。

- 名前の先頭は文字で開始すること
- 先頭の文字以外は文字、数値、アンダースコア（ _ ）のみを使用すること
- 半角で 255 字以内であること
- OR、AND、MOD、NOT といった、BASIC の演算子や論理演算子等の予約語とは別の名前にすること

例

- Form
- frm
- FileOpen
- frmFileOpen
- Checkbox
- chk
- ReadOnly
- chkReadOnly

FirstClass RAD アプリケーションの作成

多少の違いはあるものの、すべての FirstClass RAD アプリケーションは、同じ基本的な段階を踏んでいくことで作成していきます。

FirstClass RAD プロジェクトについて

FirstClass RAD アプリケーションは、以下のものを含むプロジェクトとして管理・作成されます。

- フォーム
- コードモジュール
- データベースオブジェクト（作成するアプリケーションをデータベースに接続する場合）

FirstClass RAD プロジェクトを利用するには、FirstClass RAD プロジェクトマネージャーを使います。これにより、以下の情報にアクセスができるようになります。

- プロジェクト全体の情報
- プロジェクト内のフォームの情報
- データベースに接続するアプリケーションのための、データベース接続情報
- データベースに接続するアプリケーションのために利用するデータベーステーブルの情報

FirstClass RAD プロジェクトマネージャーを利用すると、テストのために、作成したアプリケーションを動かすこともできます。テストの度に再ビルドや再インストールをする必要がありません。

プロジェクトの作成

FirstClass RAD プロジェクトを作成するには、

1. FirstClass RAD をインストールしたサーバーに管理者としてログインします。本プロジェクトに必要なリソースを含む設定ファイル（「FirstClass デザイナー」で作成します）を利用します。
2. FirstClass RAD を起動します。
3. FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブを開き、［新規作成］をクリックします。
4. プロジェクト名を入力します。（半角で２３文字まで。スペース不可。）

プロジェクトを開く

プロジェクトを開くには、

1. FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブを開き、［開く］をクリックします。
2. プロジェクトを選びます。

プロジェクト属性の設定

プロジェクト属性はプロジェクト全体に適用されます。この属性を設定するには、

1. FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブを開き、［オプション］をクリックします。
2. プロジェクト属性フォームの各タブで、必要な情報をフィールドに入力してください。

プロジェクトにフォームを追加

プロジェクトにフォーム（あらかじめ「FirstClass デザイナー」で作成する必要があります。）を追加するには、

1. FirstClass RAD プロジェクトマネージャーの［フォーム］タブを開き、［新規作成］をクリックします。
2. フォームを選択してください。

フォームが開くと、属性が表示されます。

フォーム属性の設定

フォーム属性（バインドされたコラムのデータベース用のフォーム名、タイトル、テーブル）は、そのフォーム上にある個々のフィールドではなく、フォーム全体に適用されます。

フォーム属性を設定するには、

1. FirstClass RAD プロジェクトマネージャーの［フォーム］タブを開き、属性を設定するフォームを選択してください。
2. 対象となるフォームを選択し、［変更］をクリックしてください。
3. 属性フォームの［フォーム］タブ内にフォーム属性を入力してください。
4. 必要であれば、データベースのテーブルにそのフォームをバインドしてください。［追加］をクリックして、「テーブルの選択」でテーブルを選択します。

5. ［Tab］キーを押して変更を保存し、属性フォームを閉じてください。

フィールド属性の設定

フィールド属性は、フォーム上の個々のフィールドに適用されます。

フィールド属性（名前、およびバインドされたコラムのデータベース用のテーブルまたはコラム）を設定するには、

1. FirstClass RAD プロジェクトマネージャーの［フォーム］タブを開き、設定したいフォームを選択してください。
2. 対象となるフォームを選択し、［変更］をクリックしてください。
3. フォーム内の設定したいフィールドをクリックしてください。
4. 属性フォームの［フィールド］タブ内に、フィールド属性情報を入力してください。
5. 必要であれば、データベーステーブルにフィールドをバインドしてください。［フィールド］タブ内の「テーブル」で該当するテーブルを選択し、「コラム」でこのテーブルから該当するフィールドを選択してください。

プロジェクトからフォームを削除

FirstClass RAD プロジェクトからフォームを削除すると、そのフォームに割り当てられていた設定とコードもすべて削除されます。この操作は取り消すことができません。

プロジェクトからフォームを削除するには、

1. FirstClass RAD プロジェクトマネージャーの［フォーム］タブを開き、フォームを選択してください。
2. ［削除］をクリックしてください。

● FirstClass RAD とデータベースの連携

FirstClass RAD アプリケーションは、ODBC に対応したデータベースに接続することができます。（ODBC はリレーショナルデータベースにアクセスするための業界標準仕様です。）FirstClass RAD フォームは複数のデータベーステーブルおよび複数のデータベースに接続することができます。データベースは異なるサーバー機にあってもかまいません。常に同じソース（データベーステーブル内の同じコラム）からのデータを含むフィールドには、コラムにバインドされたデータベースアクセスを作成することができます。

より柔軟性をもたせるために、特定のコラムにバインドしないフィールドがある場合には、バインドしないデータベースアクセスを作ることができます。

ODBC データソースを作成

FirstClass RAD アプリケーションを ODBC 対応データベースに接続できるようにするためには、まず Windows で ODBC データソースを作成しなければなりません。

このためには、

1. Windows のコントロールパネルでデータソース（ODBC）を開いてください。
2. ODBC データソースアドミニストレータの［システム DSN］タブで、［追加］をクリックしてください。
3. 使用するデータベース用の ODBC ドライバを選択してください。
4. ［完了］をクリックしてください。
5. フィールドに必要項目を入力してドライバを設定してください。ヘルプが必要な場合は、Windows のオンラインヘルプかドライバメーカーのドキュメントを参照してください。

注意

ODBC データベースソースに接続する必要があるのは、各アプリケーションごとに一度だけです。

データソースをテストしたい場合、Microsoft Query、Microsoft Access、または Microsoft Excel のような ODBC クライアントでデータソースに接続できます。

データベース接続の追加

データベース接続を追加するには、

1. FirstClass RAD プロジェクトマネージャーの［接続］タブを開き、［新規作成］をクリックしてください。
2. 新しい接続フォームで接続情報を入力してください。

データベース接続の更新

データベース接続を更新するには、

1. FirstClass RAD プロジェクトマネージャーの［接続］タブを開き、更新する接続を選択してください。
2. ［変更］をクリックしてください。
3. 接続フォーム上で、必要に応じて情報を変更してください。

データベース接続の削除

FirstClass RAD プロジェクトからデータベース接続を削除すると、RAD 内の関連するテーブルもすべて削除されます。この操作は取り消すことができません。

データベース接続を削除するには、

1. FirstClass RAD プロジェクトマネージャーの［接続］タブを開き、削除する接続を選択してください。
2. ［削除］をクリックしてください。

データベーステーブル接続の追加

データベースでテーブルへの接続を追加する前に、データベース全体への接続を追加しなければなりません。

データベースでテーブルへの接続を追加するには、

1. FirstClass RAD プロジェクトマネージャーの［テーブル］タブを開き、［新規作成］をクリックしてください。
2. 新規テーブルフォームに、テーブルの情報を入力してください。

データベーステーブル接続を更新

データベーステーブル接続を更新するには、

1. FirstClass RAD プロジェクトマネージャーの［テーブル］タブを開き、更新するテーブルを選択してください。
2. ［変更］をクリックしてください。
3. テーブルフォーム上で、必要に応じて情報を変更して下さい。

データベーステーブル接続の削除

FirstClass RAD プロジェクトからデータベーステーブル接続を削除すると、プロジェクトフォーム内のそのテーブルへの関連付けも削除されます。この操作は取り消すことができません。

データベースでテーブルへの接続を削除するには、

1. FirstClass RAD プロジェクトマネージャーの［テーブル］タブを開き、削除するテーブルを選択してください。
2. ［削除］をクリックして下さい。

● プロジェクトのプログラミング

FirstClass RAD のプログラミングでは、コード、サブルーチン、（該当する場合）データベース接続の追加を行うことができます。

コード内変数のデータ型

作成するコード内の各変数には、変数が格納することのできるデータの種類を決定するデータ型が含まれます。例えば、文字列型には文字列が格納され、整数型にはすべての数が格納されます。

作成するコードで変数を使う前に、変数とそのデータ型を宣言しなければなりません。これにより、変数には常に予期されたデータ型の値のみが格納されるようになります。（パフォーマンス上の理由で、FirstClass RAD では、不明なデータ型を格納することができる変数には対応していません。）

データ型とデータベース

FirstClass RAD とデータベースはどちらも、データ構造に含まれる全データについて、明示的にデータ型を割り当てる必要があります。

FirstClass RAD アプリケーションをデータベースと接続する場合は、必ず対応するデータベースのデータ型と互換性のあるデータ型を使用しなければなりません。一般的な ODBC データベースのデータ型に相当する FirstClass RAD のデータ型は以下の通りです。

- 整数型 integer, int, smallint, tinyint, boolean, bit, yes/no
- 長整数型 integer, int, smallint, tinyint, boolean, bit, yes/no
- 単精度浮動小数点型 float, double, single, double precision, decimal, numeric
- 倍精度浮動小数点型 float, double, single, double precision, decimal, numeric
- 通貨型 monetary, money, currency, smallmoney
- 文字列型 char, character, varchar, variable character, text, memo
- 日付/時刻型 date, time, date/time, datetime, smalldatetime

コードの追加

FirstClass RAD プロジェクトのプログラミングには、以下のものが含まれます。

- プロジェクト全体で使用されるコードの追加
- フォーム用のコードの追加
- フィールド用（オブジェクト用）のコードの追加

プロジェクト用のコードを追加

FirstClass RAD プロジェクトには多くの場合グローバルなコードが含まれます。（つまり、このコードは複数のフォーム、フォーム上の複数のフィールド、あるいはプロジェクト内のあらゆる所から呼び出されます。）このコードは、以下のような形をとることができます。

- サブルーチン ... 特定された複数の処理を実行するコードです。ユーザーは、コードが必要となる処理の度にそのコードを繰り返し記述する代わりに、サブルーチンを呼び出すことができます。
- 宣言 ... アプリケーションをプログラミングする際に使用される変数、定数、もしくはユーザー定義の型のことです。
- グローバル変数 ... コード内で、ユーザーが入力した値のような特定の値を格納します。グローバル変数は、使用する前に宣言しなければなりません。
- 定数 ... 値を変更することができない点を除いて、変数とほぼ同様です。
- ユーザー定義型（構造体） ... 単一のデータ構造に関連付けされた複数の変数の組み合わせです。

グローバルコードはコードモジュールに格納します。これにより、ユーザーはコードを何度も書かずに一度だけ書けば、そのコードが呼び出された時に、コードへのいかなる変更も自動的に反映されます。

初期状態では、１つのプロジェクトには１つのコードモジュールが含まれており、これをモジュール１（Module1）といいます。このモジュールには、「宣言（Declaration）」セクションと、作成するアプリケーションのエントリーポイントである「メイン（Main）」サブルーチンが含まれます。アプリケーションは、「メイン（Main）」サブルーチンに追加したコードの第１行目から実行が開始されます。ユーザーは、目的に必要なコードモジュールをさらに追加することができます。

フォーム用のコードを追加

特定のフォームに適用されるコードを追加するには、

1. 属性フォームの［フォーム］タブを開き、コードを追加するフォームを選択してください。
2. ［変更］をクリックして、フォームとその属性フォームを開いてください。
3. フォームの背景をクリックして、属性フォームの［フォーム］タブ内の［コードの編集］をクリックしてください。（ショートカット：フォームの背景をダブルクリックしてください。）
4. ［イベント］欄で、プログラミングしたいイベントプロシージャを選択してください。 (フォーム用コードの編集では、[オブジェクト]欄が「Form」になっている必要があります。)
5. 必要に応じてコードを更新してください。
6. ［Tab］キーを押して変更を保存し、コードウィンドウを閉じてください。

フィールド用のコードを追加

フォーム上の特定のフィールドに適用されるコードを追加するには、

1. 属性フォームの［フォーム］タブを開き、フォームを選択してください。
2. ［変更］をクリックしてフォームとその属性フォームを開いてください。
3. フィールドをクリックしてから属性フォームの[フィールド]タブを開き、［コードの編集］をクリックしてください。
4. ［イベント］欄でプログラミングしたいイベントプロシージャを選択してください。 (フィールド用コードの編集では、[オブジェクト欄]が編集したいフィールドになっている必要があります。)
5. 必要に応じてコードを更新してください。
6. ［Tab］キーを押して変更を保存し、コードウィンドウを閉じてください。

注意

このフィールド用に追加のイベントプロシージャをプログラミングすることができます。また、［オブジェクト］で追加のフィールドをプログラミングすることができます。

コードモジュールの追加

コードモジュールを追加するには、

1. FirstClass RAD プロジェクトマネージャーの［モジュール］タブを開き、［新規作成］をクリックしてください。
2. モジュール名を入力してください。（半角で２３文字まで。スペース不可。）

モジュールウィンドウが開きます。

コードモジュールの更新

コードモジュールを追加、編集、または削除するには、

1. FirstClass RAD プロジェクトマネージャーの［モジュール］タブを開き、更新するモジュールを選択してください。
2. ［変更］をクリックしてください。
3. 必要に応じてコードを更新してください。
4. ［Tab］キーを押して変更を保存し、コードウィンドウを閉じてください。

コードモジュールの名前変更

コードモジュールの名前を変更するには、

1. FirstClass RAD プロジェクトマネージャーの［モジュール］タブを開き、名前を変更するモジュールを選択してください。
2. ［名前変更］をクリックしてください。
3. 新しいモジュール名を入力してください。

コードモジュールの削除

FirstClass RAD プロジェクトからコードモジュールを削除すると、そのモジュール内にあるサブルーチンと宣言もすべて削除されます。この操作は取り消すことができません。

プロジェクトからコードモジュールを削除するには、

1. FirstClass RAD プロジェクトマネージャーの［モジュール］タブを開き、削除するモジュールを選択してください。
2. ［削除］をクリックしてください。

宣言の更新

コードモジュールの宣言を追加、更新、削除するには、

1. コードモジュールのウィンドウを開いてください。
2. ［サブルーチン］欄で、更新する宣言を選択してください。
3. 必要に応じてコードを更新してください。
4. ［タブ］キーを押して変更を保存し、コードウィンドウを閉じてください。

サブルーチンの追加

コードモジュールにサブルーチンを追加するには、

1. コードモジュールのウィンドウを開いてください。
2. ［新規サブルーチン］ボタン（ウィンドウの右上にある白紙のボタン）をクリックしてください。
3. 追加するサブルーチンの名前を入力して［OK］ボタンを押してください。

サブルーチンの更新

サブルーチンでコードを追加、編集、または削除するには、

1. 該当するコードモジュールのウィンドウを開いてください。
2. ［サブルーチン］欄で、更新するサブルーチンを選択してください。
3. 必要に応じてコードを更新してください。
4. ［タブ］キーを押して変更を保存し、コードウィンドウを閉じてください。

サブルーチンの削除

コードモジュールからサブルーチンを削除すると、その中に含まれるコードもすべて削除されます。この操作は取り消すことができません。

コードモジュールからサブルーチンを削除するには、

1. このコードモジュールのウィンドウを開いて下さい。
2. ［サブルーチン］欄で削除するサブルーチンを選択して下さい。
3. ［サブルーチンの削除］ボタン（ウィンドウの右上にある×印のついたボタン）をクリックしてください。

● アプリケーションのテスト

作成した FirstClass RAD 内でアプリケーションをビルド前に実行して、エラーをチェックすることができます。これにより、テストのためにアプリケーションをひんぱんにリビルドおよび再インストールする必要がなくなります。開発のさまざまな段階で（例えば、全てのフォームを追加した直後など）アプリケーションを実行することができ、その時点で記述したコードの量に応じた応答がされます。

作成したプロジェクトをテストするには、

1. FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブを開き、［実行］をクリックしてください。 エラーメッセージが表示される場合は、［OK］をクリックしてコードモジュール内のコードをチェックしてください。
2. アプリケーションのテストを中止する時には、［中止］をクリックしてください。

● アプリケーションのビルド

プロジェクトの開発とテストが完了したら、そのプロジェクトからアプリケーションをビルドすることができます。このためには、

1. FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブを開き、［ビルド］をクリックしてください。 コンパイルされたアプリケーションファイル（fcx）が¥FCPO¥RAD フォルダに出力されます。このファイルには、FirstClass にインストールされた後にアプリケーションを実行するために必要なものがすべて含まれています。

   注意

   以前にこのアプリケーションをビルドしたことがある場合、新しい FCX ファイルが以前のファイルに上書きされます。

2. サーバーを再起動して、作成した fcx ファイルをロードしてください。

● FirstClass にアプリケーションをインストール

FirstClass RAD アプリケーションをビルドしたら、FirstClass にインストールして、他のユーザーが利用できるようにする必要があります。作成したアプリケーションをインストールするには、

1. アプリケーション用のステーショナリを作成します。 この作業は、アプリケーションをアイコンから起動できるようにする場合のみ必要です。
2. 作成したアプリケーションで利用するリソース（フォーム、アイコンなど）を配布します。 リソースを配布する詳細な方法は、FirstClass デザイナーを参照してください。

アプリケーションのステーショナリを作成

ステーショナリを作成して、アイコンからアプリケーションを起動できるようにするには、

1. 管理者でログインし、FirstClass RAD プロジェクトの設定ファイルを使用します。
2. アプリケーションのアイコンを置きたい会議室を開いてください。
3. メニューバーから［管理］>［新規テンプレート］>［RAD インストールフォーム］を選択してください。
4. RAD インストールフォームを入力してください。
5. ステーショナリを選択した状態で、メニューバーから［ファイル］>［プロパティ］（Windows）または［情報を見る］（Macintosh）を選択してください。
6. ［保護］を解除してください。
7. ステーショナリの名前を入力し、お好みのアイコンを選んでください。
8. ［OK］をクリックしてください。
9. 再度、［プロパティ］（Windows）または［情報を見る］（Macintosh）を選択して、［保護］を有効にしてください。

注意

複数のインスタンスをインストールすることができます。他の会議室を開いて、上述の操作を繰り返してください。

3 言語リファレンス

● BASIC コマンド

BatchAdmin (コマンド, バッファサイズ)

コマンド

サーバーに実行させる、一行のバッチ管理コマンドです。

バッファサイズ（オプション）

返答を保存するバッファのバイト数を整数値で表します。例えば、4096 は４K バイトです。このバッファは、サーバー機で利用できるメモリによって制限されます。

Beep

FirstClass クライアントでシステムビープ音を鳴らします。

注意

FirstClass クライアントは、アプリケーションがフォームを表示するまでシステムビープ音を鳴らすことができません。

構文

Beep

例

```
Sub Click()
   Dim i as integer
   For i = 1 to 3              'ビープ音３回
   Beep Next i
End Sub
```

Call

サブルーチンを呼び出します。

これはオプションのコマンドです。このコマンドを利用せず直接サブルーチンを呼び出すことができます。

構文

Call subroutine
Call サブルーチン名(引数 1, 引数 2...)
サブルーチン名

引数（引数１など）をこのコマンドに追加することができます。

例

```
Sub Main()
    Call MyStartupSub(StartupString)
    Call MyMainProgramSub(1, "John", TRUE)
    Call MyEndSub
End Sub
```

Console

サーバーコンソールに出力内容（出力式）を表示します。

このコマンドは、FirstClass クライアントの出力ウィンドウなしにサーバーコンソールに直接出力される点以外は、Print コマンドと同じ機能です。サーバーログファイルにも出力したい場合には、Report コマンドを使用してください。

構文

Console [出力式, 出力式]
Console [出力式, 出力式]

この構文に関する説明は、Print コマンドをご覧ください。

例

```
Console "私のアプリケーションへようこそ"
Console "こんにちは"; FCUserName
Console " 'i' の値は";i
```

Const

ユーザー定義の定数を宣言します。

構文

Const 定数名 = 数式
Const 定数名 As データ型 = 数式

定数に任意の型の数式を割り当てることができます。もちろんデータ型を指定することもできます。もしデータ型を指定しない場合には、指定した数式のデータ型が使用されます。

例

```
Sub Main()
      Const MYSTRING = "Hello world"
End Sub
```

CopyFile

FirstClass サーバー上のファイルを１つの場所から別の場所にコピーします。このコマンドは、ある場所から別の場所へのコピー命令をホスト OS から出す場合に使用します。

構文

CopyFile(コピー元パス名, コピー先パス名, TRUE/FALSE)

３番目のパラメータは、既存のファイルを上書きするかどうかを決定します。

例

```
CopyFile ("C:¥FCSERVER¥FCS.OLD","C:¥FCSERVER¥FCS2.OLD",FALSE)
```

Debug

開発環境からプログラムを実行する場合にのみコンソールにテキストを出力します。

Debug コマンドは、ビルドされたアプリケーションの実行時ではなく開発環境からの実行時に動作するという点をのぞいて、Print コマンドと同じように動作します。これにより、アプリケーションをビルドしてリリースした後で、デバッグ情報がユーザーの目に触れることのないようにすることができます。

構文

Debug 出力する文字列

例

```
Sub Main()
      Print "この文字はつねに出力されます"
      Debug "この文字は開発環境から実行した場合にのみ表示されます"
End Sub
```

Dim

変数を宣言し、メモリ領域を割り当てます。

構文

```
Dim 変数名 As 変数型
Dim 変数名(最大配列数) As 変数型
Dim 変数名(最小配列数 To 最大配列数) As 変数型
```

Dim 変数名 As String * 文字列長
Dim 変数名 1 As 変数型, 変数名 2 As 変数型...

変数の名前（変数名）は、数字（0-9）もしくはアンダースコア（ _ ）で始まらない限り、いかなる英数字も有効です。アンダースコアと数字は、先頭以外の他の箇所では使用できます。

変数のデータ型（変数型）を特定しなければなりません。有効な変数型は、整数型、長整数型、単精度浮動小数点型、倍精度浮動小数点型、日付/時刻型、通貨型、文字列型、ユーザーが定義した型です。

変数の配列（...配列数）は６１次元、１００万要素未満まで作成することができます。また、文字変数の固定長文字列（文字列長）を宣言することもできます。文字列長の引数は、有効な数字を利用して最大文字列長を指定することができます。

例

```
Dim i As Integer
Dim MyArray(10) As Integer              '10 要素数（0-9）の整数文字列を宣言
Dim MyArray(0 To 9, 0 To 9) As Double           '２次元の配列を宣言
Dim str2 As String * 100                '100 文字の固定長文字列を宣言
```

Do...Loop

条件式が TRUE である間、もしくは TRUE になるまで、一連のサブコマンドを実行します。

構文

Do While 条件 命令...Loop
Do Until 条件 命令...Loop

Do 命令...Loop While 条件
Do 命令...Loop Until 条件

ループが最後のコマンドまで達すると、ループの条件が再評価され、ループが繰り返されるか、もしくはループ後の最初の行の処理が続けられます。Do...Loop には４つの種類があります (Do While...Loop, Do Until...Loop, Do...Loop While, Do...Loop Until)。おのおのは異なる役目を持っています。２つの While と Until ループのおもな違いは、条件を指定する位置にあります。Do...Loop...condition コマンドは、条件が繰り返しをしないよう指定しても、少なくとも１度は必ずサブコマンドを実行します。Do While/Until condition...Loop コマンドは、条件がループを繰り返すよう指定した場合のみ、サブコマンドを実行します。While を使った Loop は、条件が TRUE の間繰り返されます。Until を使った Loop は、条件が TRUE になるまで繰り返されます。

例

```
Do While Score < 10
    Print Score
    Loop
Do
    Print Score
    Loop Until Score > 10
```

どちらのループも同じ結果になります。

DoEvents

保留されているユーザー入力を処理するために、現在実行中のサブルーチンの処理を中断します。

DoEvents を使って、アプリケーションがループしている間にユーザー入力に応答するアプリケーションを開発できます。また、データまたはフォーム上の値の更新を監視することもできます。

構文

DoEvents (開始時間, 終了時間)

例

```
Sub Click ()
    Dim StartTime as integer
    LastTime as integer
    Dim IconNumber as integer

    Randomize
    StartTime = Timer

    Do While (Timer < StartTime + 60)          '６０秒間ループを実行
        If Timer > LastTime + 1 Then           '毎秒、このコードを実行
        IconNumber = Int ( (9 * Rnd) +1)           '９つのアイコンから１つをランダムに生成
        DisplayIcon(IconNumber)
    End If
    DoEvents '**DoEvents を呼び出して、未処理の入力を処理**
    Loop
End Sub
```

End

構造体やプロシージャを終了させます。

このコマンドは、Sub、Function、Select、Type などの構造体の終了を示すために必要です。If コマンドがすべて１行にある場合を除いて、If 構造体の終了を示すのにも必要です。単独で使われる End コマンドは、アプリケーションの実行を中止します。

構文

End
End Sub
End Function
End If
End Select
End Type

例

```
If StartupString = "Box" Then
    Print "Take the box to the curb"
    End
End If
```

Erase

配列の値をクリアにし、再初期化します。

数値配列の全要素はゼロに設定されます。文字型配列の全要素は空の文字列（""）に設定されます。日付型の配列はゼロ日に設定されます。ユーザー定義型の配列の場合は、その型の各部が個別の変数として扱われ、その型にしたがって消去されます。

構文

Erase 配列
Erase 配列 1, 配列 2...

例

```
Sub Main()
    Dim MyArray(1 to 10) As Integer
    Dim I As Integer
     Print " My Array を指定して出力します。"
        For I = 1 to 10
            MyArray(i)=i
            Print MyArray(i)
        Next i
        Erase MyArray                'MyArray 配列のすべての値をゼロにする
        Print "MyArray は抹消され、次のものが含まれます。:"
             For i = 1 to 10                'すべてのゼロを出力
        Print MyArray(i)
```

```
        Next i
End Sub
```

Exit

プロシージャ、もしくは構造体のループを終了します。

プロシージャの場合は、プロシージャの呼び出しの次の行へ処理が移動します。ループ構造体の場合は、ループに続く行へ処理が移動します。

構文

```
Exit
Exit Sub
Exit Function
Exit Loop
Exit For
```

For...Next BASIC コマンドの例をご覧ください。

For..next

カウンタ変数が増える度に、一連のコマンドを実行します。

構文

```
For カウンタ変数 = 初期値 To 終了値
サブコマンド
Next
```

```
For カウンタ変数 = 初期値 To 終了値
コマンド
Next カウンタ変数
```

```
For カウンタ変数 = 初期値 to 終了値 Step 増分値
サブコマンド
Next
```

```
For カウンタ変数 = 初期値 To 終了値
サブコマンド
条件
Exit For

サブコマンド
Next
```

For...Next ループが繰り返される度に、カウンタ変数が増加します。初期値と終了値を指定して、ループのカウンタの初期値と終了値を指定することができます。デフォルトの増分値は１ですが、Step キーワードを追加して異なる増分値を指定することがで

きます。増分値には有効な数値を指定することができます。増分値が負の値の場合、始めから終わりまで逆方向にカウントされます。これは、初期値が終了値より大きい場合にのみ適用されます。BASIC の Exit コマンドを利用すれば、ある条件に合致したときにループ処理を事前に中断することもできます。

例

```
Dim i As Integer
For i = 1 to 10
    If i > 5 Then
        Exit For            'これは前もってループを中断する処理です。.'
    End If
        Print "カウント中", i
    Next i
Next i
```

Function

プロシージャを、呼び出された際に値を返すことのできる関数として定義します。

Function（関数）の名前はその関数の中からの変数として扱われ、関数が返されたときにその値が返されます。Function はどのようなデータ型でもよく、他のプロシージャもしくは内部関数を呼び出して、結果を処理することができます。

構文

Function 関数名(関数へのパラメータ As データ型) as 戻り値のデータ型

例

```
Function MultiplyByTen(X as Integer)
as Integer
    MultiplyByTen=MultiplyByTen*10
End Function
Sub Main
    Print"8 ×10 は"& MultiplyByTen(8)
End Sub
```

GiveTime

アプリケーションの処理を一時中断して、サーバーを操作できるようにします。

すべてのアプリケーションには、ほとんど同じ処理時間が与えられています。割り当てられた時間がすべて必要ではないことがわかっているアプリケーションでは、GiveTime コマンドを利用して、残りの時間を他のサーバー処理およびアプリケーションに与えることができます。すべてのデータベースとフィールドの操作、およびいく

つかのコマンドは自分でサーバーへの再入を開始できます。この場合には、GiveTime コマンドを呼び出す必要はありません。

これは、ときどきアイドリングを行う（例えば、設定時間ごとにポーリングを行う）アプリケーションにとって便利なコマンドです。

構文

GiveTime

```
starttime = Timer
Do While TRUE
    If Timer < starttime + 5 Then
    Give Time
Else
    Print "５秒経過しました。..."
    starttime = Timer
End If
Loop
```

Gosub...Return

ラベルが付いたプロシージャのサブルーチンへジャンプして、GoSub 以下の行へ戻ります。

注意

このコマンドの使用はお勧めしません。下位互換性を保つために用意されているだけです。代わりに、Sub や Function のような、ひんぱんに処理されるルーチンを扱うサブルーチンを使用してください。処理はラベルで始める行にジャンプします。ラベルには（０－９の数字からなる）行番号、もしくは（数字で始まらないかぎり）英数字の文字列を使うことができます。

構文

```
Gosub ラベル名
...
Return
```

ラベルは、コロン（:）の後につづけて、同じ行に非ブランク文字を最初に入れなければなりません。

例

```
    For I = 1 to 10
        Gosub MySubroutine
        Goto MyEnd
```

Goto

ラベルへジャンプします。ラベルで始まる行へ処理がジャンプします。ラベルには（0－9の数字からなる）行番号、もしくは（数字で始まらないかぎり）英数字の文字列を使うことができます。

構文

Goto ラベル名

ラベルは、コロン（:）の後につづけて、同じ行に非ブランク文字を最初に入れなければなりません。

例

```
Sub Main()
    Dim num As Integer
    num = 5
    Goto 20
        Print "これは印刷されません。"
    20 If num = 5 then
Goto numis5
```

If...Then

条件にもとづいて処理を分岐させます。

このコマンドは、必ず TRUE もしくは FALSE となる条件が必要です。TRUE でない数値条件は FALSE とみなされます。条件が TRUE の場合、このステートメントにつづくコマンドが実行され、その後、End If に続く行の処理が続けられます。

構文

If 条件 Then コマンド 1 Else コマンド 2
If 条件 Then
コマンド
End If

If 条件 1 Then
コマンド
ElseIf 条件 2 Then
コマンド
Else
コマンド
End If

追加の条件を指定するために、一組の If...Then 構造をもつ ElseIf...Then 節をいくつ作ってもかまいません。

例

```
If FCUserid = "guest" Then
    Print "ようこそゲストユーザーさん"
ElseIf FCUserid = "admin" Then
    Print "ようこそ管理者さん"
Else
    Print "ようこそ"              'ユーザーがゲストでも管理者でもない場合
End If
```

Let

前もって行われる変数への代入です。変数に値を代入します。これは BACIS 言語のオプションです。

構文

Let 変数名 = 値

例

```
Let X=1
X=1
```

Load

フォームのフィールドとデータのリソースをロードします。

このコマンドは、アプリケーションで利用できるフォームのフィールドとデータのリソースを作成しますが、フォームを表示することはできません。フォームを表示するには Show form メソッドを使わなければなりません。

構文

Load フォーム名

例

```
Sub Main()
    Load MyForm
    If MyForm.MyTable.Column(1) = TRUE Then
        MyForm.Title = "My Form's Title"
        MyForm.Show
    End If
End Sub
```

MsgBox

モーダルメッセージボックス（一度開いたら、閉じるまで他の操作ができないメッセージボックス）を表示します。

これはモーダルメッセージボックスなので、ユーザーがメッセージボックス内のボタンをクリックして応答するまで、アプリケーションの処理が一時中断されます。そのとき、MsgBox につづく行で処理が続行されます。

MsgBox Response 機能を使って、どのボタンをユーザーがクリックしたのかをアプリケーションに伝えることができます。

構文

MsgBox("メッセージボックスの文字")
MsgBox("メッセージボックスの文字", メッセージボックスのボタン）
MsgBox("メッセージボックスの文字", メッセージボックスのボタン, "メッセージボックスのタイトル")

メッセージボックスの文字は、半角 59 文字で切り捨てられます。メッセージボックスに表示される文字列を指定します（メッセージボックスの文字）。また、メッセージボックスにタイトルを追加し（メッセージボックスのタイトル）、メッセージボックスに必要なボタンを指定することもできます（メッセージボックスのボタン）。メッセージボックスのボタンに有効な値は、

- fcOKOnly - OK ボタンのみ
- fcOKCancel - OK ボタンと Cancel ボタン
- fcYesNo - Yes ボタンと No ボタン
- fcYesNoCancel - Yes ボタン、 No ボタン、Cancel ボタン.

メッセージボックスのタイトルを指定しないと何も表示されません。メッセージボックスのボタンを指定しない場合は、OK ボタンが表示されます。

例

MsgBox ("変更を保存しますか?", FCYesNoCancel, "保存")

On Error

エラーの処理を可能または不可能にします。

構文

On Error Goto ラベル名
On Error Resume
On Error Resume Next
On Error Goto 0

このコマンドはつぎのコマンドといっしょに使われます。

・Goto ラベル名

重大なエラーが起きた場合、ラベルのある行に処理がジャンプします。このラベルには、（０－９の数字からなる）行番号か、もしくは（数字で始まらないかぎり）英数字の文字を使うことができます。ラベルは、コロン（:）の後につづけて、同じ行に非ブランク文字を最初に入れなければなりません。

・Resume

アプリケーションは、エラーの原因となった行の再実行を、実行が成功するまで試みます。アプリケーションが無限ループする可能性がありますので、このコマンドの使用にはご注意ください。

・Resume Next

重大なエラーを含むエラーの原因となった行の後から処理を再開します。エラーの原因となった行はスキップされ、重大なエラーが起きても処理は停止しません。

・Goto 0

この地点から先は、エラー処理を無効にします。

例

```
Sub Main()
    Dim i As Integer
    On Error Goto ErrHandler
    i=1/0               'ゼロ除算エラー
        Print "エラー処理により、ここからアプリケーションの実行が再開されます。
"
Exit Sub
```

Option

このアプリケーション用に、BASIC のインタプリターのオプションを設定します。

構文

Option Base 値

現在、Base というオプションのみ設定できます。このオプションを使うと、Dim コマンドで宣言した最小配列値の第１要素の初期値を指定することができます。値は（初期値の）０もしくは１が指定可能です。Option Base は、コードモジュールの宣言をするプロシージャでのみ使用でき、アプリケーションにつき一度だけ呼び出すことができます。

例

```
Option Base 0
...
Dim MyArray(10) As Integer          '0 から 9 まで領域を整数型で確保
Option Base 1
...
Dim MyArray(10) As Integer          '1 から 10 まで領域を整数型で確保
```

PlaySound

FirstClass クライアントのサウンドリソースを再生します。FirstClass サウンドリソースのリソース ID を指定しなければなりません。

構文

PlaySound(リソース ID)

Print

FirstClass クライアントの出力ウィンドウに出力内容（出力式）を書き出します。

構文

```
Print
Print 出力式
Print 出力式, 出力式
Print 出力式; 出力式
Print,
Print;
```

１つの Print コマンドで出力式をいくらでも出力することができます。式として文字列を指定するためには、ダブルクオーテーションで文字列を囲んでください("文字列")。このコマンドといっしょに使うことができる文字は次の通りです。

・コンマ（,）で式をつなげる

それぞれの式は、前の式に続くプリントゾーン内に表示されます。プリントゾーンは半角１４文字あります。

・セミコロン（;）で式をつなげる

それぞれの式は、前の式から間隔をあけないで表示されます。

・行の終わりにコンマ（,）またはセミコロン（;）をつける

通常各 Print コマンドが出力されると、その行末に出力される CR が表示されません。その後につづく Print コマンドの出力が同じ行に表示されます。

出力式のない Print コマンドでは、CR が出力されます。

例

```
Print "私のアプリケーションへようこそ"
Print 1, 2, 3, 4, 5              '離れたプリントゾーンにそれぞれの数を出力
Print "この２行は";
Print "１行に表示されます。"
```

Randomize

乱数ジェネレータを初期化します。

このコマンドを利用しないと、Rnd 関数を実行する度に同じ乱数系列が返されます。

構文

Randomize [シード値]

シード引数を使って、乱数系列が生成される元となる数値キーを指定することができます。プログラムが実行される度に同じシード値を使った場合は、同じ乱数系列が生成されます。シード引数がない場合には、Randomize コマンドは Timer 関数を使って乱数系列を生成します。

例

```
Dim i As Integer
Randomize
Print "５つの乱数を生成"
For i = 1 to 25
```

ReDim

配列の要素数を変更します。

このコマンドを既存の配列に利用すると、古い列の次元が新しいものに置き換わります。

構文

ReDim 変数名 As 変数型
ReDim Preserve 変数名 As 変数型
ReDim 変数名 (最大配列数) As 変数型
ReDim 変数名 (最小配列数 To 最大配列数) As 変数型
ReDim 変数名 1 As 変数型, 変数名 2 As 変数型...

この構文の説明は、Dim コマンドをご覧ください。

配列内の既存の値を保持するには、Preserve キーワードを含めてください。このキーワードを含めない場合は、Erase コマンドが使われたのと同じように、配列が消去されます。

Rem

テキスト行をコメントアウトします。行全体が無視されます。

構文

Rem コメント

'コメント

例

```
Rem これはコメントです。          'これはコメントです。
```

Report

出力（出力式）をすべてサーバーのログファイルに書き出します。

このコマンドは、出力対象が FirstClass クライアントの出力ウィンドウではなく、ログファイルおよびサーバーコンソールである点を除けば、Print コマンドと同機能です。

サーバーコンソールに出力を表示するだけなら、Console コマンドを使用してください。

構文

```
Report
Report 出力式
Report 出力式, 出力式
Report 出力式; 出力式
Report,
Report;
```

この構文の説明は、Print コマンドをご覧ください。

例

```
Report "アプリケーションは開始された日時は、:"; Now
```

RowCount

これはステートメントで利用可能な行数を返すステートメント属性です。

警告

これはすべての ODBC ドライバで対応しているわけはなく、ODBC 規格の必須条件ではありません。

RunApp

ビルドし、ロードしたアプリケーションを新しいアプリケーションの環境変数領域で実行します。

構文

RunApp("アプリケーション名")
RunApp("アプリケーション名, "スタートアップ文字列")

このコマンドではアプリケーション名をいっしょに指定してください。また、有効な文字列式をスタートアップ文字列として指定することもできます。この文字列は起動するアプリケーションに渡され、そのアプリケーションでスタートアップ文字列値として使用されます。

例

```
RunApp("Calc")                    '電卓アプリケーションを起動
RunApp("Order", "buy12")                    'スタートアップ文字列を"buy 12"として、購入用アプリケーションを起動
```

Select Case

プログラムフローの式による条件分岐を生成します。

構文

```
Select Case 条件比較式
        [Case [Is | # To # ] 条件比較式 [コマンド]]
        [Case Else] [コマンド]]
End Select
```

Select Case 選択の構造体によって、簡易的な方法で式もしくは条件の命令を、一連の実行可能な条件比較式もしくは「case（場合）」と比較することができます。

Case 以下が TRUE の場合、Case の後につづくコマンドが実行されます。選択構造体の各 Case は、それぞれに応じて実行されます。各 Selecet Case 選択構造体は、End Select キーワードで終了しなければなりません。

Is と To キーワードを使うと、Case コマンド内での比較を行うことができます。Is は比較演算子を利用することができ、To は条件に使う値の範囲を検証することができます。

Sleep

指定した時間が経過するまでプログラムの実行を休止します。

ステーショナリで起動するアプリケーションの場合、プログラムがサーバーからロードされないままにするか、もしくはフォームを閉じてプログラムを停止することがで

きます。サーバーアプリケーションでは、Sleep コマンドは非常に有効で、プログラムを呼び出すためにサーバーの CPU を大量に消費しないようにしつつ、ときどきコードを実行したいような場合に利用できます。

構文

Sleep(秒|日付)

パラメータには、プログラムが再開されるまでの休止時間を秒数で指定する整数か、もしくは、指定した日時になるまでプログラムが再開されないようにする日付のどちらかを利用できます。

例

```
Print "このプログラムは５秒で自動消滅します！"
Sleep (5)
End
```

Sort

昇順または降順で一次元配列内の要素を並べ替えます。

構文

Sort(配列)

オプションで fcAscending および fcDescending パラメータを使用すると、Sort コマンドは整数型、長整数型、日付/時刻型、文字列型、単精度浮動小数点型、倍精度浮動小数点型、通貨型の配列を、昇順または降順に並べ替えることができます。

Sub

サブプロシージャの名前と引数を宣言します。

構文

Sub 名前 ([ByRef | ByVal] [引数 1],[ByRef | ByVal] 引数 2]...)

名前には、BASIC で有効な識別子であればどのような文字でも利用できますが、アンダースコア（_)もしくは 0 から 9 までの数字で始めることはできません。

Sub はいくつの引数でも受け取ることができます。Sub の引数は、BASIC で有効な識別子なら何でも利用可能で、データ型も問いません。

ByRef キーワードを使用すると、引数は参照によって渡されます。ByVal キーワードを使用すると、引数は値によって渡されます。キーワードが指定されない場合の標準動作では、引数は値で渡されます。

例

```
Sub DisplayMyUserInfo()
    Print "私のユーザーID は "; FCUserID
End sub
```

Type

ユーザ定義のデータ型を宣言します。

ユーザ定義のデータ型を利用すると、関連するいくつかの変数を単一のデータ構造体に結合することができるようになり、データをまとめるのに役立ちます。

コードモジュールの宣言セクションでユーザ定義のデータ型を定義すると、データ型はアプリケーション全体で表示可能になり、アプリケーションの生存期間中は有効になります。プロシージャで定義すると、データ型はプロシージャのみで表示可能で、プロシージャの生存期間中しか有効になりません。

構文

```
Type データ型名
    変数名 As データ型
    [変数名 As データ型...]
End Type
```

例

```
Type CustType                '宣言で定義
    First as String
    Last as String
    SSNum as String *9               '文字列は 9 文字まで
    CustomerID as Integer
End Type
```

Unload

フォームを閉じ、データ環境を閉鎖します。

構文

```
Unload フォーム名
```

Unload コマンドは、文字列式のフォーム名で指定されたフォームを閉じます。閉じられている状態では、フォームは Unload イベントプロシージャを実行し、データ環境内のオープンテーブルを閉鎖します。

例

```
Sub Click()
        Unload MyForm
End Sub
```

While…Wend

条件式が TRUE と判断されている間、一連のコマンドを実行します。.

While...Wend ループ構造体は Do...Loop 構造体と似ていますが、Do...Loop 構造体より機能が制限されています。While...Wend 構造体は下位互換性を保つために用意されています。

構文

```
While 条件
    commands
Wend
```

● BASIC の演算子

&

２つの文字列（文字列 1 と文字列 2）を連結します。

文字列の連結では、最初の文字列の後に次の文字列を加えて、これを結果の文字列として返します。

構文

文字列 1 & 文字列 2

例

```
Sub Main()
    Dim s1 as String, s2 as String, result as String
    s1 = "Hello" : s2 = "world"
    result = s1 & s2
        Print result            'Hello world という結果が与えられる。
        Print result            'Hello world と出力。
End Sub
```

*

２つの数値（数値 1 と数値 2）を乗算します。

構文

数値 1 と数値 2 には、有効な数値式を使用することができます。

例

```
Sub Main()
        Print "10 × 4 は", 10 * 4
End Sub
```

+

２つの数値（数値 1 と数値 2）を加算するか、もしくは２つの文字列を連結します。（ただし、連結には＆演算子の利用をお勧めします）

構文

数値 1 + 数値 2

数値 1 と数値 2 には、有効な数値式を使用することができます。

例

```
Sub Main()
      Print "10 + 4 は", 10 + 4
End Sub
```

-

２つの数値（数値 1 と数値 2）を減算するか、もしくは数値（数値）を負にします。

減算の構文

数値 1 - 数値 2

数値 1 と数値 2 には、有効な数値式を使用することができます。

減算の例

```
Sub Main()
     Print "10 - 4  は:", 10 - 4
End Sub
```

負の構文

-数値

負の例

```
Sub Main()
     Print "99 を負にすると:", -99
     Print "-99 を負にすると:", -(-99)
End Sub
```

/

２つの数値（数値 1 と数値 2）を除算します。計算結果を浮動小数点数値で表示したい場合に使用します。

構文

数値 1 / 数値 2

数値 1 と数値 2 には、有効な数値式を使用することができます。

例

```
Sub Main()
     Print"10÷4 は、", 10/4              '2.5 を出力
End Sub
```

¥

2 つの数値（数値 1 と数値 2）を整数除算します。

整数除算は、結果を切り捨てて、余りのない整数値で表します。

構文

数値 1 ¥ 数値 2

数値 1 と数値 2 には、有効な数値式を使用することができます。

例

```
Sub Main()
    Print "10 を 4 で整数除算すると", 10¥4              '2 を出力
End Sub
```

< (より少ない)

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

数値 1 が数値 2 より小さい場合は TRUE が返されます。大きい場合は FALSE が返されます。

構文

数値 1 < 数値 2

例

```
If num < 5 Then
    Print "num は 5 より小さい値です。"
```

<= (等しいか、より小さい)

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

数値 1 が数値 2 と等しいか、数値 2 より小さい場合は TRUE が返されます。大きい場合は FALSE が返されます。

構文

数値 1 <= 数値 2

例

```
If num <= 5 Then
    Print "num は 5 と等しいか 5 より小さい値です。"
```

<> (等しくない)

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

2 つの値が異なる場合は TRUE が返されます。同じ場合は FALSE が返されます。

構文

数値 1 <> 数値 2

例

```
Sub Main()
    Dim num as Integer
    num = 5
    If num <> 5 Then
        Print "num の値は 5 と等しくありません。"
Else
    Print "num の値は 5 と等しいです。"
    End If
End Sub
```

=

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

2 つの値が同じ場合は TRUE が返されます。異なる場合は FALSE が返されます。

構文

数値 1 = 数値 2

例

```
Sub Main()
    Dim num as Integer
    num = 5
If num = 5 Then
     Print "num の値は５に等しいです。"
```

> (より大きい)

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

数値 1 が数値 2 より大きい場合は TRUE が返されます。小さい場合は FALSE が返されます。

構文

数値 1 > 数値 2

例

```
Sub Main()
    Dim num as Integer
    num = 5
```

```
    If num > 5 Then
        Print "num は 5 より大きい値です。"
    Else
        num は 5 より小さいか、5 に等しい値です。
    End If
End Sub
```

>= (より大きいか、等しい)

同じもしくは類似のデータ型である 2 つの数値（数値 1 と数値 2）の値を比較します。

数値 1 が数値 2 より大きいか等しい場合は TRUE が返されます。小さい場合は FALSE が返されます。

構文

数値 1>= 数値 2

例

```
If num >= 5 Then
    Print "num は 5 より大きいか、5 に等しい値です。"
```

AND

2 つの数値（数値 1 と数値 2）が TRUE がどうか判断します。

AND の結果は TRUE となるためには、両方の式が TRUE でなければなりません。

構文

数値 1 AND 数値 2

例

```
Sub Main()
Dim e1 as Integer, e2 as Integer e1 = TRUE: e2 = 5< 34
    If e1 AND e2 Then
        Print "両方の式が TRUE です。"
    Else
        Print "どちらかの式が TRUE ではありません。"
    End If
End Sub
```

MOD

2 つの数値（数値 1 と数値 2）を除算した余りを返し、整数で表します。

構文

数値 1 MOD 数値

例

```
Sub Main()
    Print "10 を 4 で割った余りは:", 10 MOD 4          '2 を出力
End Sub
```

NOT

式が FALSE かどうか判断します。

構文

NOT 式

例

```
Sub Main()
    Dim e1 as Integer
    e1 = TRUE
    If NOT e Then
        Print "式 e は FALSE でした。"
        'TRUE ではない=FALSE
    Else
        Print "式 e は TRUE でした。"
    End If
End Sub
```

OR

2 つの式（式 1 と式 2）の一方または両方が TRUE であるかどうか判断します。

どちらかひとつの式が TRUE であれば、OR の結果は TRUE になります。

構文

式 1 OR 式 2

例

```
Sub Main()
    Dim e1 as Integer, e2 as Integer
e1 = TRUE : e2 = 5 < 34
    If e1 OR e2 Then
Print "どちらか一方または両方の式が TRUE です。"
    Else
    Print "式 1 と 式 2 の両方とも TRUE とは判断されません。"
```

```
    End If
End Sub
```

XOR

２つの数値（数値 1 と数値 2）のどちらか一方のみが TRUE であるか判断します。

XOR の結果が TRUE になるには、一方の式が TRUE で、もう一方の式が FALSE でなければなりません。

構文

式 1 XOR 式 2

例

```
Sub Main()
    Dim e1 as Integer, e2 as Integer
    e1 = TRUE: e2 = 5 < 34
If e1 XOR e2 Then
        Print "式 1 と式 2 のどちらか一方のみが TRUE です。"
    Else
        Print "式 1 と式 2 の両方とも TRUE か、両方とも FALSE です。"
    End If
End Sub
```

^

１番目の数値（数値 1）を、２番目の数値（数値 2）で累乗します。

構文

数値 1^数値 2

例

```
Sub Main()
    Print "10 の二乗は", 10^2
    Print "4 の三乗は", 4^3
    Print "6.4 の 1.5 乗は", 6.4^1.5
End Sub
```

● BASIC 変数のデータ型

通貨型（Currency）

金額のデータを含む変数に使われます。このデータ型は固定小数点で、小数点の左１５桁、右４桁まで対応しています。

これは金額計算にすぐれたデータ型です。単精度浮動小数点型と倍精度浮動小数点型では、四捨五入による小さな誤差が発生することがあるからです。

代入できる値

-922337203685477.5808 から 922337203685477.5807 までです。

日付型（Date）

日付のデータを含む変数に使われ、1/1/2000 または January 1, 2000 のような有効なデータ形式でなければなりません。時刻のデータも同様に使うことができます。

注意

日付に数値型のデータを使うこともできます。この場合、小数点の左側の値は、1899 年 12 月 30 日午前 0 時からの日数を表します。小数点の右側の値は時刻を１日を 1 としたときの率で表します（例えば、午後 12 時は.5 と表されます）。

代入できる値

January 1, 100 から December 31, 9999 までです。

DBConnection

データベース接続用のオブジェクトを作成します。

DBConnection には、独自の属性とメソッドのセットがあります。

DBStatement

データベースに命令をするオブジェクトを作成します。

DBStatement には、独自の属性とメソッドのセットがあります。

倍精度浮動小数点型（Double）

どのような数値も含むことができる変数に使用されます。数値には、分数や小数点以下の大きい数値が含まれます。このデータ型は整数型や長整数型と比べ、効率的ではありませんがあらゆる種類の数値が扱えます。

単精度型にも、倍精度型と同じ制約とサイズ制限があります。この２つのデータ型は交互に使うことができます。FirstClass RAD では、もともと倍精度型より小さい単精度型のデータを倍精度型として扱うことで、以前からある BASIC コードに対応しています。

代入できる値

-1.79769313486231E308 から 1.79769313486231E308 までです。

Email

電子メール用のオブジェクトを作成します。

Email には、独自の属性とメソッドのセットがあります。

File

ファイル用のオブジェクトを作成します。

File には、独自の属性とメソッドのセットがあります。

整数型（Integer）

整数型のデータに使用します。詳細は長整数型（Long）をご覧ください。

範囲

-2,147,483,648 から 2,147,483,647 までです。

長整数型（Long）

小数を含まない変数に使用します。このデータ型は小数を含むことができる数値型のデータより小さくて効率的です。

整数型にも、長整数型と同じ制約とサイズ制限があります。この２つのデータ型は交互に使うことができます。FirstClass RAD では、もともと長整数型より小さい整数型のデータを長整数型として扱うことで、以前からある BASIC コードに対応しています。

代入できる値

-2,147,483,648 から 2,147,483,647 までです。

文字列型（String）

テキスト文字列を含む変数に使用します。

デフォルトでは、文字の長さを変えることができます。しかし、固定長の文字列型変数を使用して、特定の長さに強制することができます。制限よりも短い文字が許可さ

れ、パディングはされません。文字の長さが制限よりも長い場合のみ、制限が行われます。

文字型変数を固定長にするためには、String キーワードにアスタリスク（*）と最大文字数を表す整数を追加してください。最大文字数より長い文字列型変数は切り捨てられます。

代入できる値

0 から およそ 65,500 文字までです。

例

Dim LastName As String * 7

単精度浮動小数点型（Single）

単精度型のデータに利用されます。詳細は倍精度浮動小数点型（Double）をご覧ください。

範囲

1.7E +/- 308 (15 桁)です。

● 列属性

ColNum

テーブル内で列の順序を示す数値を取り出します。

この属性はランタイム時のみ読み込むことができます。

構文

テーブル名.列(列 id).ColNum

Data

バイナリーデータの列の値を取り出すか、割り当てます。データ属性にはバイナリーデータが含まれ、たいていの場合、文字列として扱われます。

構文

テーブル名.列(列 id).Data

Datatype

列のデータ型を取り出します。データ型には、列のデータ型を指定する定数の整数値が含まれます。

この属性はランタイム時のみ読み込むことができます。

構文

テーブル名.列(列 id).Datatype [定数]

定数に入れることのできる値は次のとおりです。

- fcString - 文字またはテキストデータ
- fcInteger -整数型の数値データ
- fcLong - 長整数型の数値データ
- fcSingle - 単精度浮動小数点型の数値データ
- fcDouble - 倍精度浮動小数点型の数値データ
- fcCurrency - 通貨型データ
- fcDate - 日付/時刻型データ
- fcBinary - バイナリーデータ

Name

列のデータベース識別子を取り出します。

この属性はランタイム時のみ読み込むことができます。

構文

テーブル名.列(列 id).Name

Nullable

列が null であるかどうかを返します。

この属性はランタイム時のみ読み込むことができます。

構文

テーブル名.列(列 id).Nullable

Precision

列またはパラメータのデータ型が使用する列（またはパラメータ）の最大桁数を取り出します。この属性はランタイム時のみ読み込むことができます。

注意

文字型のデータでは、データの文字数の長さになります。バイナリーデータでは、データのバイト数の長さで定義されます。時刻、タイムスタンプ、時間の間隔をあらわす全てのデータでは、そのデータを表す文字の文字数になります。

構文

テーブル名.列(列 id).Precision

Scale

列の小数点から右側の最大桁数を取り出します。

この属性はランタイム時のみ読み込むことができます。

注意

浮動小数点の列およびパラメータの概数では、小数点より右側の桁数が固定されていないため、この値は定義されません。日時や、秒数をあらわすデータを含む時間の間隔を表すデータでは、小数点の桁数は、秒を表すデータの小数点から右側の桁数として定義されます。

SQL_DECIMAL データ型と SQL_NUMERIC データ型では、最大桁数は通常最大精度と同じになります。しかし、最大桁数に個別の制限を設けているデータソースもあります。

構文

テーブル名.列(列 id).Scale

Text

現在の行にある列の文字列を取り出すか、割り当てます。Text にはデータ情報も含まれます。

Text と Value に規定値が決められている場合はその値が入ります。

構文

テーブル名.列(列 id).Text

Value

現在の行にある列の数値を取り出すか、割り当てます。Value にはデータ情報も含まれます。

Text と Value に規定値が決められている場合はその数値が入ります。

構文

テーブル名.列(列 id).Value

● 接続属性

AutoCommit

自動コミットモードと手動コミットモードのどちらを使用するか指定します。

構文

接続名.AutoCommit [= 定数]

定数として使用できる値は次の通りです。

・SQL_AUTOCOMMIT_OFF

接続は手動コミットモードを使用します。アプリケーションは、Commit メソッドもしくは Rollback メソッドで、明示的にトランザクションをコミットもしくはロールバックしなければなりません。

・SQL_AUTOCOMMIT_ON

ドライバは自動コミットモードを使用します。各コマンドは実行直後にコミットされます。

・SQL_AUTOCOMMIT_DEFAULT.

自動コミットモードを有効にします。

定数に値が指定されない場合、デフォルト設定はお使いの ODBC の設定によって決まります。手動コミットモードから自動コミットモードへ変更すると、その接続上のあらゆるオープントランザクションがコミットされます。

AccessMode

ODBC 経由で接続するアクセスモードを指定する整数値を取り出すか、割り当てます。

構文

接続名.AccessMode [= 定数]

定数として使用できる値は次の通りです。

・SQL_MODE_READ_WRITE

接続は、読み取りまたは書き込み要求の処理を許可されます。

・SQL_MODE_READ_ONLY

接続は、読み取り専用でない要求を処理しないよう指示されます。このような要求を受け取ったデータソースとODBCドライバの動作はODBCドライバ固有の動作によって定義されます。

・SQL_MODE_DEFAULT.

読み取り-書き込みのアクセスモードです。

この属性のデフォルト値は SQL_MODE_READ_WRITE.です。

CurrentQualifier

並行処理制御属性を指定します。この属性は、データソースが使用する SQL コマンドの修飾子を含む文字列です。

構文

接続名.CurrentQualifier [= 修飾子]

データベースによっては、アクセスするデータベースを修飾子文字に指定できるものがあります。もしくは、ファイルベースのデータソースによっては、テーブルファイルが格納されるディレクトリを指定できるものがあります。どちらでも指定可能です。

ODBCCursors

接続がどのように ODBC のカーソルライブラリを使用するかを指定します。

構文

接続名.ODBCCursors [= 定数]

定数として使用できる値は次の通りです。

・SQL_CUR_USE_IF_NEEDED

必要な場合のみ、接続が ODBC カーソルライブラリを使用します。ODBC データソースが Move メソッドと Find メソッドに対応していない場合、ODBC ドライバマネージャーはカーソルをエミューレートしようと試みます。

・SQL_CUR_USE_ODBC

接続は、ODBC カーソルライブラリを使用します。

・SQL_CUR_USE_DRIVER

接続はカーソルと、スクロール可能なドライバを使用します。Move メソッドと Find メソッドはデータソースのカーソルモードを使用して動かされます。

・SQL_CUR_USE_DEFAULT.

ドライバ側のカーソルを使用します。

この属性のデフォルト値は SQL_CUR_USE_DRIVER です。

LoginTimeout

ログイン要求が完了するのを待つ秒数を示す整数値を指定します。デフォルトはデータソースドライバによって異なり、常にゼロ以外の値です。LoginTimeout に 0 が指定された場合はタイムアウトが無効になり、ログインが試みられている間はずっと接続待ちの状態になります。

構文

接続名.LoginTimeout [= ログインタイムアウト秒数]

接続のログインタイムアウトのデフォルト秒数は SQL_LOGIN_TIMEOUT_DEFAULT の値になります。

OptTraceFile

ODBC のトレースファイル名を指定します。

構文

接続名.OptTraceFile [= ファイル名]

ファイル名には有効なファイルシステムパスを表す任意の有効な文字列を指定できます。

OptTrace

ODBC ドライバマネージャーがトレースを実行するかどうかを決める整数値を指定します。

構文

接続名.OptTrace [= 定数]

定数として使用できる値は次の通りです。

- SQL_OPT_TRACE_OFF - トレースを実行しない
- SQL_OPT_TRACE_ON - トレースを実行する
- SQL_OPT_TRACE_DEFAULT - トレースを実行しない

アプリケーションは OptTraceFile 属性でトレースファイルを指定します。ファイルがすでに存在している場合、ODBC ドライバマネージャーはファイルを追加します。存

在していない場合はファイルを作成します。トレースファイルが指定されなければ、ODBC ドライバマネージャーは¥SQL.LOG.ファイルに書き込みます。

この属性のデフォルト値は SQL_OPT_TRACE_OFF です。これは、ODBC トレースが起動しないことを示します。

PacketSize

ネットワークのパケットサイズをバイト単位で指定する整数値です。

注意

多くのデータソースでは、この属性に対応していません。

構文

接続名.PacketSize [= サイズ]

TranslateDLL

ODBC のトランスレーション機能を持つ DLL 名を指定します。

構文

接続名.TranslateDLL [= DLL 名]

DLL 名にはトランスレーション機能を持つ DLL ファイルへの有効なファイルシステムパスである任意の有効な文字列を指定できます。トランスレーション機能を持つ DLL を使うと、文字セットのトランスレーションなどのトランスレーション機能を追加で実行することができます。

QuoteChar

データソース特有の引用文字を取り出します。

この属性はランタイム時のみ読み出すことができます。

構文

接続名.QuoteChar

TxnIsolation

ODBC のトランザクション分離レベルを示す整数値を取り出すか、割り当てます。

次の定義と動作が ODBC では規定されています。

・ダーティーリード

トランザクション１は行を修正し、トランザクション２はトランザクション１が変更をコミットする前に修正された行を読み込みます。トランザクション１が変更をロールバックすると、トランザクション２は絶対に存在するはずのなかった行を読み込むことになります。

・非繰り返し読み込み

トランザクション１は行を読み込み、トランザクション２はその行を更新もしくは削除して、この変更をコミットします。トランザクション１が行を再読み込みすると、異なる値を受け取るか、またはその行がすでに存在しないことを返します。

・ファントム

トランザクション１は検索基準を満たした行セットを読み込みます。トランザクション２はその検索基準に合致する行を挿入します。トランザクション１がその行を読み込むクエリーを再実行すると、異なる行セットを受け取ります。

構文

接続名.TxnIsolation [= 定数]

定数として使用できる値は次の通りです。

・SQL_TXN_READ_UNCOMMITTED

ダーティーリード、非繰り返し読み込み、ファントムが可能です。

・SQL_TXN_READ_COMMITTED

ダーティーリードはできませんが、非繰り返し読み込みとファントムは可能です。

・SQL_TXN_REPEATABLE_READ

ダーティーリードと非繰り返し読み込みはできませんが、ファントムは可能です。

・SQL_TXN_SERIALIZABLE

トランザクションが直列化できます。ダーティーリード、非繰り返し読み込み、ファントムはできません。

・SQL_TXN_VERSIONING.

トランザクションが直列化できますが、SQL_TXN_SERIALIZABLE.よりも高度な並行処理制御が可能です。ダーティーリードはできません。

● 接続メソッド

BrowseConnection

ODBC ドライバまたはデータソースに対して呼び出しを行い、接続に必要な引数を引き出します。

接続するための接続文字列属性内に情報が返され、初めて ODBC データベースに接続する際に役立ちます。この同じ情報には、接続を確立するために必要なデータについても記述されています。FirstClass RAD で使われる標準の接続メソッドは OpenConnection で、これは標準のデータソースに対して３つの引数を指定します。

構文

接続名.BrowseConnection(文字列)

ほとんどの ODBC データソースには、文字列として最大で３つの引数（データソース名、ユーザーID、パスワード）が必要です。Oracle と他社のデータベースでは、文字列として少なくとも４つの引数（データソース名、ユーザーID、パスワード、接続文字列）が必要になることがあります。

例

```
Dim cnct as dbConnection
cnct.BrowseConnection ("DSN=ORACLE")                'Oracle データソース
Print cnct.ConnectionString
UID:Login=?,PWD:Password=?;ConnectString: Connection String=?;
'どのパラメータが必要かを示す文字列がドライバから返される
REM
'どのパラメータが必要かを示す文字列の書式がドライバから返される
(KEYWORD:DESCRIPTION=VALUE)
```

CloseConnection

ODBC データソースへの接続を終了します。オープン状態の接続をすべて自動的に閉じます。

構文

接続名.CloseConnection

例

OpenConnection 接続メソッドの例をご覧ください。

Commit

データソースのトランザクションをコミットします。

注意

このメソッドは、トランザクションに対応しているデータソースにのみ有効です。Commit は、AutoComit 属性が有効になっている接続に対しては有効な操作ではありません。

構文

接続名.Commit

例

```
Sub Main()

    Dim cnct as DBConnection
    cnct.OpenConnection("MyDSN")            ' "MyDSN"データソースを開く
    cnct.AutoCommit = SQL_AUTOCOMMIT_OFF
'トランザクションを開始
    'ステートメントを作成して SQL を実行
    ' ...
If (UserClickedCancel = TRUE)
    cnct.Rollback           '変更をロールバック
Else
    cnct.Commit             '変更をコミット
End if
    cnct.CloseConnection            '接続を閉じる
End Sub
```

DriverConnection

まだ作成されていない ODBC 接続情報でデータソースを確立します。これは、BrowseConnection が返す情報です。

注意

（コードではなく、テーブルと FirstClass RAD オブジェクトへの接続を使用して）バインドした列を持つアプリケーションを異なるデータベースでビルドする場合、および Oracle に移行させる場合には、次の作業をしなければなりません。

1. 手動で開始するために、接続選択リストを設定します。
2. DriverConnection のコード例（下記参照）と類似のコードを実行して、お使いの Sub Main()内でデータベースに接続します。
3. Oracle データベースのテーブルと列の名前を、これまでお使いのプロジェクト内の名前と一致させてください。（そうしなければ、移行がうまくいきません）

構文

接続名.DriverConnection(接続文字列)

接続文字列には、ODBC ドライバの接続文字列を指定します。

例

次のコードは、プログラム内で Oracle データソースへの接続をあらかじめ確立し、ステートメントを配置し、そのデータベースに対してクエリーを実行するのに使用できます。

```
Dim cnct As dbConnection
cnct.DriverConnection("DSN=ORACLE;UID=admin;PWD=frisky;ConnectString=MyOr
acleServerString")
```

OpenConnection

ODBC データソース接続を開きます。

注意

このメソッドは、ステートメントを開いてデータベース操作を実行する前に、任意の DBCConnection 上で実行しなければなりません。

構文

接続名.OpenConnection(データソース名[,ログイン名] [,パスワード])

データソース名の引数には、ODBC データソースアドミニストレータで設定したような、有効なデータソース名を指定する任意の文字列を使うことができます。オプションのログイン名とパスワードの引数には、データソースへのログイン名とパスワードを指定する任意の文字列を使うことができます。

例

```
Sub Main()
    Dim cnct As DBConnection
    cnct.OpenConnection("MyDSN")
'"MyDSN"データソースを開く
' ステートメントを作成して SQL を実行..
' ...
    cnct.CloseConnection
'接続を閉じる
End Sub
```

Rollback

データソース上のトランザクションをロールバックします。

注意

このメソッドは、トランザクションに対応したデータソースのみに有効です。ロールバックは、AutoCmmit 属性が有効になっている接続に対しては有効な操作ではありません。

構文

接続名.Rollback

例

コード例は Commit 接続メソッドをご覧ください。

● 電子メール属性

Body

電子メールのメッセージに本文を設定します。

この属性はオプションで、送信する電子メールのメッセージに本文が必要ない場合には不要です。

属性

電子メール変数.Body [= 文字列式]

文字列式には任意のテキスト値を使用でき、システム上で有効なユーザーには制限されません。

例

```
Sub Main()
    Dim MyMessage As Email
    ...
    MyMessage.Body = "FCRAD のデモです。使用しているディスク容量が非常に大きくなっています。"
    MyMessage.Send
End Sub
```

From

電子メールのメッセージに送信者情報を設定します。

この属性は必ず設定しないと、メールは配信されません。

構文

電子メール変数.From [= 文字列式]

文字列式には任意のテキスト値を使用でき、システム上で有効なユーザーには制限されません。

例

```
Sub Main()
    Dim MyMessage As Email
    ...
    MyMessage.From = "管理者"
    MyMessage.Send
End Sub
```

Subject

電子メールのメッセージに件名のテキストを設定します。

この属性はオプションで、送信する電子メッセージに件名が必要ない場合には不要です。

構文

電子メール変数.Subject [= 文字列式]

文字列式には任意のテキスト値を使用できます。

例

```
Sub Main()
    Dim MyMessage As Email
    ...
    MyMessage.Subject = "データベースのテスト"
    MyMessage.Send
End Sub
```

To

電子メールのメッセージの宛先を設定します。

注意

To 属性は FirstClass メールリストに対応していますが、ゲートウェイの使用には今のところ対応していません。ユーザ名が有効でない場合、電子メールのメッセージはエラーになります。この属性を必ず設定しなければ、メッセージは配信されません。

構文

電子メール変数.To [= ユーザー名]

宛先となる受信者への送信権限があるかぎり、宛先のユーザー名には任意のユーザーもしくは会議室を指定できます。

例

```
Sub Main()
    Dim MyMessage As Email
    ...
    MyMessage.To = "春田"
    MyMessage.Send
End Sub
```

● 電子メールメソッド

Send

電子メールのメッセージを送ります。

構文

電子メール変数.Send

例

```
Sub Main()
    Dim MyMessage As Email
    ...
    MyMessage.From = "Administrator"
    MyMessage.Subject = "ようこそ!"
    For i = 0 to 3
        MyMessage.To = Users(i)
        MyMessage.Body = "ようこそ " & Users(i) & "あなたのアカウントが " _ & "
作成されました。"
        MyMessage.Send
Next i
End Sub
```

● フォーム属性

Hidden

フォームを表示もしくは非表示にします。このフォーム属性は、フォームの現在の状態を決めるのにも利用できます。フォームが表示されている場合、Hidden 属性は FALSE になります。非表示の場合は、TRUE になります。

構文

[フォーム名].Hidden [= TRUE|FALSE 値]

例

```
Sub Click()
    If MyForm.Hidden = TRUE Then
        MyForm.Title = "フォーム名"
        MyForm.Show
    End If
End Sub
```

MouseX,MouseY

フォーム（保護設定のかかっているものも含む）上で、最後にクリックされた位置の XY 座標を取り出します。

構文

フォーム名.MouseX
フォーム名.MouseY

例

```
Sub Main()
    MyForm.MouseX
    MyForm.MouseY
End Sub
```

Title

フォームウィンドウのタイトルバーにタイトル文字を表示します。

構文

[フォーム名].Title [=文字列式]

タイトルには、任意の有効な文字列式を指定することができます。

例

```
Sub Click()
    MyForm.Title = "My form のタイトル"
    MyForm.Show
End Sub
```

● フォームイベント

Activate

フォーム上でイベントが発生します。

フォームがアクティブになるとイベントが発生します。フォーカスを受けたフォームがアクティブになります。

Click

フォーム上でクリックするとイベントが発生します。

フォーム上もしくはフォーム上の保護されたフィールド上でクリックをすると、イベントが発生します。

Deactivate

フォーム上のイベントが終了します。

フォームが非アクティブになるとイベントが発生します。フォーカスが他のフォームに外れたフォームが非アクティブになります。

DoubleClick

フォーム上でダブルクリックするとイベントが発生します。

フォーム上もしくはフォーム上の保護されたフィールド上でダブルクリックをすると、イベントが発生します

Load

イベントをロードします。

フォームが初めてロードされるか表示されると、イベントが発生します。フォームは Load コマンドでロードするか、Show メソッドで表示します。

Unload

イベントをアンロードします。

フォーム上の閉じるボタンをクリックするか、もしくは Unload コマンドを利用してプログラム的にウィンドウを閉じると、イベントが発生します。

● フォームメソッド

DisableFormEvents

つぎのフォームイベントを無効にします。

- Click
- DoubleClick
- Activate
- Deactivate

また、つぎのフィールドイベントを無効にします。

- Click (ボタンを除く)
- DoubleClick
- GotFocus
- LostFocus.

このフォームメソッドは、利用頻度の高いシステム上で動かすアプリケーションに有効です。アプリケーションで上記リストにあるイベントを利用しない場合、そのフォームイベントを無効にすることで、クライアント／サーバー間のオーバーヘッドを減少させることができます。

構文

フォーム名.DisableFormEvents

EnableFormEvents

DisableFormEvents フォームメソッドで無効にされたフォームイベントとフィールドイベントを有効にします。

構文

フォーム名.EnableFormEvents

Hide

表示していたフォームを非表示にします。

Hide メソッドで非表示にされたフォームはアンロードされません。つまり、このフォームは現在の属性と値が保持されたまま存在し続けており、このフォームを表示すると、Hide フォームメソッドを実行する前に設定されていた属性を保ったまま再表示されます。アプリケーションが終了すると、非表示のフォームは自動的にアンロードされます。

構文

フォーム名.Hide

Show

モードレスフォームを表示します。Show メソッドは、フォームがロードされていなかった場合、そのフォームとデータ環境をロードしてからフォームを表示します。

構文

フォーム名.Show

Swap

フォームの内容を非表示にして、本来のフォームのウィンドウ内に 2 番目のウィンドウを表示します。

構文

フォーム名.Swap(スワップフォーム)

スワップフォーム引数には、有効なフォーム名である任意の文字列式を指定することができます。このメソッドによって、プログラマーは、フィールドの表示・非表示を切り替えることなく、ウィンドウ内でインターフェースを大きく変えることができます。

スワップフォーム引数は文字列として指定し、フォームオブジェクトへの直接参照を指定しないよう確認してください。スワップフォーム引数がフォームオブジェクトへの直接参照であった場合には、フォームのデフォルトの属性である Title が、スワップのフォーム名として使用されます。

例

```
Form1.Swap("Form2)          '正解!
Form1.Swap(Form2)           '不正解! Form2 のタイトル名は文字列として渡してください。
```

● フィールドイベント

Change

フィールドの値が変更される時に起こるイベントを呼び出します。

例

```
Sub Change()
...
End Sub
```

Click

ユーザーが保護されていないフィールドをクリックした時に起こるイベントを呼び出します。

例

```
Sub Click()
...
End Sub
```

DoubleClick

ユーザーが保護されていないフィールドをダブルクリックした時に起こるイベントを呼び出します。

GotFocus

フィールドがアクティブになると（フォーカスを受けると）、イベントが発生します。

フィールドは、クリックされるか、Tab キーで選択されるか、SetForcus メソッドでプログラム的にフォーカスを受けることができます。

LostFocus

フィールドがアクティブでなくなると（フォーカスを失うと）、イベントが発生します。

他のフィールドがフォーカスを得ると、フィールドはフォーカスを失います。

● フィールドメソッド

AddItem

すでにあるリスト形式のフィールドに項目を追加します。

構文

[フォーム名].フィールド名.AddItem(文字列)

AddItem を使うと、ドロップダウンリストに新しい項目を自動的に追加することができ、リストの利用が簡単になります。AddItem では、文字列パラメータで指定された項目を検出してリストに追加します。AddItem は、「選択リスト（入力可）」と「選択リスト（入力不可）」にのみ使用することができ、その List（リスト）属性が変更されます。

DblClick

ユーザーがフィールドをダブルクリックした時に起こるイベントを呼び出します。

ダブルクリックでイベントを実行したい時にはいつでも、DoubleClick メソッドにコードを記入することができます。

例

```
Sub DblClick()

...
End Sub
```

RemoveItem

すでにあるリスト形式のフィールドから項目を削除します。

構文

[フォーム名].フィールド名.RemoveItem(文字列)

RemoveItem を使うと、ドロップダウンリストからアイテムを自動的に削除することができ、リストの利用が簡単になります。RemoveItem では、文字列パラメータで指定されたアイテムを検出してリストから削除し、他のすべての項目はそのままにします。RemoveItem は「選択リスト（入力可）」と「選択リスト（入力不可）」にのみ使用することができ、その List（リスト）属性が変更されます。

例

```
RemoveItem (an item in the list)
```

SelLength

テキストフィールド内で選択されたテキストの文字数を設定します。

構文

[フォーム名].フィールド名.SelLength(数値式)

テキストを選択する開始位置が SelStart で設定されていない場合、テキストの選択は先頭文字から開始されます。数値式がテキストバッファの文字数より大きい場合、SelLength はフィールド内の全テキストを選択します。数値式が 0 に設定されている場合、カーソルの位置は SelStart（SelStart が最初に呼び出されていない場合には 0 文字目）で指定された場所になり、テキストは選択されません。

例

```
Sub Click()
    ...
    StringField1002.SelLength(25)          'StringField 1002 内の最初の 25 文字を選択
    StringField1002.SelLength(0)           '0 は非選択。カーソル位置は SelStart で決定
    StringField1002.SelLength(5)           '7 番目から 12 番目の文字を選択
    ...
End Sub
```

SelStart

テキスト選択のコードで開始位置を設定します。

構文

[フォーム名].フィールド名.SelStart(開始位置)

開始位置は数値式で番号を入れます。

SelStart を使って、SelLength でテキスト選択を開始する文字位置を指定します。
SelStart(0)とすると、テキストフィールドの先頭から選択が始まるように設定されます。

例

```
Sub Click()
    ...
    StringField1002.SelStart(5)            'テキスト選択の開始位置を設定
    StringField1002.SelStart(7)            'テキスト選択の開始位置を設定
    ...
End Sub
```

SetFocus

フィールドにアプリケーションのフォーカスを設定します。

フォーカスしたいフィールドのあるフォームが表示されているのにアクティブでない場合、そのフォームがアクティブになり、フォーカスが設定されます。フォーカスを受けるフィールドが非表示または保護設定になっている場合は、このメソッドは無効になります。

構文

[フォーム名].フィールド名.SetFocus

例

```
Sub Click()
    StringField1002.SetFocus            'StringField 1002 にフォーカスを設定
End Sub
```

● フィールドの種類

ボタン

チェックボックス

0（オフ）、1（オン）、または 2（グレー）のデフォルトの Value 属性を持っています。

チェックボックスを 2 箇所以上に表示させたい場合には、FirstClass デザイナーで作成してください。

コマンドボタン

FirstClass デザイナーでコマンドを設定して FirstClass RAD のボタンに使用するか、またはボタンの Click メソッドに機能を割り当てることができます。

Text（テキスト）属性によって、ボタン上に表示される文字をプログラムで制御することができます。

ラジオボタン

クリックなどのアクションによって、ボタンが押されたときにプログラムの命令を追加することができます。ラジオボタンには Value 属性はありません。

どのラジオボタンが選択されたかを知るには、ラジオボタンフィールドを含むラジオグループフィールドの Value 属性を使用します。

任意のラジオグループ内の各ラジオボタンフィールドには、FirstClass デザイナーで異なる値を設定するようにしてください。

ラジオグループ

ラジオグループ内で利用可能なボタンのうち、現在選択されているラジオボタンの値が含まれます。ラジオグループフィールドは、デフォルトの Value 属性を持っています。

ボタンの値は FirstClass デザイナーで設定しなければなりません。また、正しく機能させるために一意の値にしてください。

URL

リンクを開く単一の URL を FirstClass デザイナーで事前に設定してください。
FirstClass RAD でプログラム的に修正することはできません。

注意

うまくいかない場合に備えて、コマンドボタンとボタンの Click メソッドの LaunchURL コマンドを使って、同じ動作ができるようにしてください。

日付セレクタ

日付データ型でフィールドに入力または表示される日付を含む Value 属性を持っています。

編集可能テキスト

フィールドに入力されたテキストが入る Text（テキスト）属性を持っています。

選択リスト（入力可）

リスト内に、FirstClass デザイナーで指定した場合と全く同じように表示される項目が入る List（リスト）属性があります。

各項目はセミコロンで区切られます。このリストは、リストから選択された項目の文字列を含む Text（テキスト）属性を持っています。文字列ではなく数値を返すには、「選択リスト（入力不可）」を使用してください。リストにない値をユーザーが入力できないようにするには、FirstClass デザイナーかプログラムによって、編集可（Editable）属性を無効にして、このリストを編集不可モードに設定することができます。

拡張リスト

FirstClass デザイナーで設定する場合のような、サブフィールド Type と同じ属性を持っています。サブフィールド Type が「テキスト」なら、どのサプフィールドも Text（テキスト）属性となります。

拡張リストには配列としてアクセスします。たとえば、 lstExpanding(0)="Hello"とすると、拡張リストの一番目の属性をリテラルストリング（引用符などで囲まれた文字列）に設定します。Reset メソッドを使って、フィールドの全内容をクリアし、Index 属性を利用して、リスト内のどの項目がフォーカスされているか確認することができます。

ファイルビューワー

バイナリーデータを含むさまざまなデータが入ります。

ファイルビューワーフィールドは、利用可能なビューワーがある場合にデータを表示しますが、表示できないバイナリーデータも同様に格納することができます。ファイルビュワーフィールドを使って、例えば、データベースから画像データを表示したり、ユーザーがプログラムにデータをアップロードしたりできます。「ファイル名」「ファイルサイズ」「ファイルタイプ」「ファイル作成者」「ファイルフラグ」などの特定の属性を利用して、現在ファイルビューワーフィールド内で開いているファイルの情報を得ることができます。

固定リスト

FirstClass デザイナーで設定するのと同様なサブフィールド Type の属性を持っています。

サブフィールド Type が「テキスト」なら、どのサブフィールドも Text（テキスト）属性となります。

固定リスト属性には固定リストが配列であるかのようにアクセスします。したがって、lstFixed(0)="Hello"とすると、固定リストの一番目の属性をリテラルストリング（引用符などで囲まれた文字列）に設定します。Reset メソッドを使って、フィールドの全内容をクリアし、Index 属性を利用して、リスト内のどの項目がフォーカスされているか確認することができます。固定リストの大きさは最初に設定した大きさのまま変わらず、項目が追加されると大きくなる拡張リストとは異なります。

画像

アイコン

フィールド内に表示されるアイコンのアイコン ID を含む Value 属性を規定します。

直線

特別な属性はありません。

楕円

特別な属性はありません。

画像

画像のリソース ID と等しい Value 属性を持っています。

画像リソースは、設定ファイル内かサーバーの FC Resource Registry 内に存在しないと、正しく表示されません。

四角形

特別な属性はありません。

角丸四角形

特別な属性はありません。

グループボックス

特別な属性はありません。

ガイドテキスト

フィールド内に表示されるテキストを含む Text（テキスト）属性を持っています。

マーキー

マーキーフィールド内の文字を含む Text（テキスト）属性を持っています。

数値

フィールドに入力する数値を含む Value 属性を持っています。

このフィールドでは数値の入力のみ可能で、テキストは入力できません。

進行状況バー

設定するかまたは取り出すことができる Value 属性を持っており、バー上で進行状況を表します。

この値の範囲は FirstClass で設定します。デフォルトは 0 から 100 です。

アイコン付きテキスト

フィールドのテキスト部分を含む Text（テキスト）属性と、フィールドに関連付けられたアイコンのリソース ID を含む Icon（アイコン ID）属性の両方を持っています。

期間

フィールドに表示される期間を表す数値を含む Value（値）属性を持っています。

● フィールド属性

Admin

フィールドの管理者属性を取り出すかまたは割り当てます。この属性により、非表示もしくは保護設定されているフィールドを管理者が表示または修正することができます。

構文

[フォーム名].フィールド名.Admin [= TRUE/FALSE 値]

Bold（太字）

フィールドのテキストを太字で表示します。

構文

[フォーム名].フィールド名.Bold [= TRUE/FALSE 値]

Border（枠線）

フィールドの周囲に枠を表示します。このフィールド属性が設定されると、ボタンのみがフィールドのクリックイベントを発生させることができます。

構文

[フォーム名].フィールド名.Border [= TRUE/FALSE 値]

Bottom（下揃え）

フィールドのテキストを、フィールドの下辺に揃えて表示します。

構文

[フォーム名].フィールド名.Bottom [= TRUE/FALSE 値]

Caption（名前）

テキストフィールドでないフィールドで、表題のテキストを取り出すか、割り当てます。この属性はボタンフィールドの名前を設定するのによく使用されます。

構文

[フォーム名].フィールド名.Caption [=文字列式]

Center（中寄せ）

フィールドのテキストを、フィールドの上辺と下辺の中間に配置して表示します。

構文

[フォーム名].フィールド名.Center [= TRUE|FALSE 値]

Color（色）

フィールドのテキストを、デフォルトのテキスト色で表示します。

構文

[フォーム名].フィールド名.Color [= TRUE|FALSE 値]

Condense（圧縮）

フィールドのテキストを圧縮した状態で表示します。

注意

この属性は Windows 用クライアントには対応してません。Macintosh 用クライアントでのみ利用できます。

構文

[フォーム名].フィールド名.Condense [= TRUE|FALSE 値]

CreationDate

ファイルビューワーで、ファイル作成日属性を取り出します。

構文

[フォーム名].フィールド名.CreationDate

Data

フィールドのバイナリーデータを取り出すか、割り当てます。たいていの場合、バイナリーデータは文字列として扱うことができます。

この属性は、画像フィールド、もしくはファイルビューワーなどのフィールドで利用します。

構文

[フォーム名].フィールド名.Data

DataSize

ファイルビューワーフィールドのデータサイズ属性を取り出します。

構文

[フォーム名].フィールド名.DataSize

DataType

フィールドのデータ型を指定する整数の定数値を取り出します。

このフィールド属性は、ランタイム時のみ読み取ることができます。

構文

[フォーム名].フィールド名.DataType [定数]

利用できる定数値は以下の通りです。

- FCString - 文字またはテキストデータ
- FCInteger - 整数型の数値データ
- FCLong - 長整数型の数値データ
- FCSingle - 単精度浮動小数点型の数値データ
- FCDouble - 倍精度浮動小数点型の数値データ
- FCCurrency - 通貨型のデータ
- FCDate - 日付/時刻型のデータ

Editable（編集可）

フィールドの編集可属性を取り出すか、割り当てます。この属性を設定すると、ユーザーがフィールドの値を編集、修正することができます。

構文

[フォーム名].フィールド名.Editable [= TRUE|FALSE 値]

Extend（拡張）

フィールドのテキストを拡張した状態で表示します。

この属性は Windows 用クライアントには対応してません。Macintosh 用クライアントでのみ利用できます。

構文

[フォーム名].フィールド名.Extend [=TRUE|FALSE 値]

FileCreator

ファイルビューワーフィールドで、Macintosh ファイルの creator 属性を取り出します。

構文

[フォーム名].フィールド名.FileCreator

FileFlags

ファイルビューワーフィールドで、Macintosh ファイルのファイルフラグ属性を取り出します。

構文

[フォーム名].フィールド名.FileFlags

FileName

ファイルビューワーフィールドで、ファイル名属性を取り出します。

構文

[フォーム名].フィールド名.Filename

FileSize

ファイルビューワーフィールドで、ファイルサイズ属性を取り出します。

構文

[フォーム名].フィールド名.FileSize

FileType

ファイルビューワーフィールドで、Macintosh ファイルの type 属性を取り出します。

構文

[フォーム名].フィールド名.FileType

FullJust

フィールドが完全に整えられているかどうかを取り出すか、割り当てます。

構文

[フォーム名].フィールド名.FullJust [=TRUE|FALSE]

Gray（灰色）

フィールドのテキストを、デフォルト色（黒）ではなく灰色で表示します。

この属性は Protected（保護）設定と同時に使用して、現在保護設定になっている編集可能なフィールドを指定するようにしてください。

構文

[フォーム名].フィールド名.Gray [=TRUE|FALSE value]

Hidden（非表示）

フィールドを非表示にします。

注意

この属性が設定されると、どのユーザーもそのフィールドを見たり操作したりすることができなくなります。ユーザーがフィールドを利用できないため、データの修正、イベントの受け取り、イベントプロシージャの処理をユーザーインターフェースから行うことはできません。ただし、フィールド属性、値、イベントには、コード内からアクセスできます。

構文

[フォーム名].フィールド名.Hidden [= TRUE|FALSE]

Icon（アイコン ID）

現在表示しているアイコンのリソース ID を取り出すか、割り当てます。

リソース ID は、ユーザーの設定ファイル、クライアント自体、または、サーバーの FC Resource Registry に格納されているアイコンのリソース ID と一致しなければなりません。一致しない場合は何も表示されません。

注意

アイコンの変更ではクライアントからサーバーへデータを送る必要がありませんので、どのようなアイコンのリソース ID 変更もすばやく反映されます。

構文

[フォーム名].フィールド名.Icon [= リソース ID]

Index

固定リストまたは拡張リストで、現在選択されているフィールドの順序を表す数値を取り出すか、割り当てます。

構文

[フォーム名].フィールド名.Index [= 数値]

リスト内の一番目の項目は Index が 0 になります。リスト中の項目にフォーカスが設定されると、リストの Index 属性はフィールドの順序を表す値を設定します。Index 属性は正の整数値で直接設定することも可能です。

Italic（斜体）

フィールドのテキストを斜体で表示します。

構文

[フォーム名].フィールド名.Italic [= TRUE|FALSE 値]

Key

拡張リストもしくは固定リスト内の項目に関連付けられた、非表示の Key 値を取り出すか、割り当てます。

注意

Key 属性は、いかなるデータ型の値でも割り当てることができますが、ランタイム時にいったん割り当てると、そのリストの Key 属性のデータ型は変更できません。

構文

[フォーム名].フィールド名.Key [=任意の値]

List（リスト）

リストフィールドの List 属性を取り出すか、割り当てます。

構文

[フォーム名].フィールド名.List [=文字列式]

リスト属性は、セミコロンで区切られた文字列のリストとして格納されます。列挙データリストのフィールドでリストを使用した場合、リスト内の各文字列に値が自動的に割り当てられます。自動生成される番号は 0 で始まり、リスト内の項目が 1 つ増えるごとに 1 増分された値が割り当てられます。自分で列挙データリスト内のリスト項目に番号を割り当てることもできます。

例

「猫;犬;魚」と入力すると、「猫」「犬」「魚」という別々の 3 つの文字列（1 行につき 1 項目）としてリストに表示されます。

自動的に生成される番号は、次のようになります。

猫　　'0 が返される
犬　　'1 が返される
魚　　'2 が返される

自分で番号を割り当てる場合は、次のようになります。

猫=100;犬=23;魚=7
猫　　'100 が返される

犬　　'23 が返される
魚　　'7 が返される

ListCount

拡張リストもしくは固定リストのフィールドで、現在の項目番号を返します。

構文

[フォーム名].フィールド名.ListCount

ModificationDate

ファイルビューワーフィールドで、ファイルの修正日属性を取り出します。

構文

[フォーム名].フィールド名.ModificationDate

Name

フィールド名を取り出すか、割り当てます。

注意

コードの記述を十分慎重に行っていない場合、実行中のプログラムでフィールド名を変更すると、予期しない動作が起こることがあります。

構文

[フォーム名].フィールド名.Name[= フィールド名]

フィールド名は、文字列式フォーマットで指定します。

NoOpenSpace

フィールドが拡張しても、フォームが拡張しないようにします。

構文

[フォーム名].フィールド名.NoOpenSpace [= TRUE|FALSE 値]

Outline（中抜き）

フィールドのテキストを、中抜きで表示します。

注意

この属性は Windows 用クライアントには対応していません。Macintosh 用クライアントでのみ利用できます。

構文

[フォーム名].フィールド名.Outline [=TRUE|FALSE 値]

Password（パスワード）

すべてのテキスト文字をアスタリスク（*）で表示します。テキストの値は入力したとおりになりますが、このフィールドへの入力、出力はともにアスタリスクで表示されます。

構文

[フォーム名].フィールド名.Password [= TRUE|FALSE 値]

Protected（保護）

すべてのユーザーがフィールドを操作できないようになります。データの修正、イベントの受け取り、イベントプロシージャの処理が無効になります。ただし、フィールド属性の値とイベントにはコード内からアクセスできます。

構文

[ファイル名].フィールド名.Protected [=TRUE|FALSE 値]

ResourceSize

ファイルビューワーのフィールドで、Macintosh のリソースフォークのサイズ属性を取り出します。

構文

[フォーム名].フィールド名.ResourceSize

RJust（右寄せ）

フィールドのテキストを右寄せで表示します。

Selectable（選択可）

フィールドを選択可能にします。

構文

[フォーム名].フィールド名.Selectable [=TRUE|FALSE 値]

Selected（選択状態）

フィールドにフォーカスを設定し、選択されている状態にします。

注意

この属性が設定できない場合は、SetFocus メソッドを使用してください。

構文

[フォーム名].フィールド名.Selected [=TRUE|FALSE 値]

Shadow（影文字）

フィールドのテキストを影文字で表示します。

注意

この属性は Windows 用クライアントには対応していません。Macintosh 用クライアントでのみ利用できます。

構文

[フォーム名].フィールド名.Shadow [=TRUE|FALSE 値]

Text（テキスト）

テキスト値である文字列をフィールドから取り出すか、フィールドに割り当てます。Text（テキスト）属性と Value 属性はフィールドのデフォルト属性です。この属性はテキストのフィールドや選択リスト（入力可）などの文字列型のフィールドでのみ利用可能です。

構文

[フォーム名].フィールド名.Text [=文字列式]

Transparent（透過）

フィールドの背景を透過にして表示します。画像やアイコンなど背景として使えるフィールドでは、このフィールドを使って画像を表示できます。

構文

[フォーム名].フィールド名.Transparent [TRUE|FALSE]

Underline（下線）

フィールドのテキストに下線を付けます。

構文

[フォーム名].フィールド名.Underline [=TRUE|FALSE 値]

Value

フィールドの数値を取り出すか、割り当てます。

この属性は、チェックボックスや列挙リストなど、数値を扱うフィールドにのみ利用可能です。

構文

[フォーム名].フィールド名.Value [=TRUE|FALSE 値]

Value 属性は、データ情報を取り出すこともできます。Text（テキスト）属性と Value 属性はフィールドのデフォルトの属性です。

Wrap（折り返し）

フィールドのテキストを折り返して表示します。この属性は、複数行を表示できるテキストボックスに重要なものです。

テキストフィールドで改行を入力して表示するには、この属性を設定しなければなりません。

構文

[フォーム名].フィールド名.Wrap [TRUE|FALSE 値]

● ファイル属性

Archive

Windows ファイルのアーカイブ属性を取り出すか、割り当てます。この属性は Macintosh ファイルのダウンロードでは無視されます。

構文

ファイル変数.Archive

CreationDate

ファイル作成日を取り出すか、割り当てます。

作成日の割り当ては、ファイルのダウンロードにのみ適用されます。

構文

ファイル変数.CreationDate

Data

ファイルデータを取り出すか、割り当てます。

Data 属性にはバイナリーデータが含まれ、たいていの場合、文字列として扱うことができます。Data 属性は Download メソッドと Read メソッドとを同時に合わせて使用します。

構文

ファイル変数.Data

DataSize

ファイルのデータフォークのサイズをバイト単位で取り出します。

構文

ファイル変数.DataSize

FileCreator

Macintosh ファイルの creator タグを取り出すか、割り当てます。

FileCreator 属性は Windows のファイル形式では無視されます。

構文

ファイル変数.FileCreator [整数値|文字列]

この属性には、長整数型の値もしくは Macintosh ファイルの creater タグを表す 4 桁の文字（例えば、MyFile.FileCreator = "ttxt"）を指定することができます。

FileFlags

Macintosh ファイルのファイルフラグを取り出すか、割り当てます。

FileFlags 属性は Windows ファイル形式では無視されます。

構文

ファイル変数.FileFlags

Filename

ファイルのファイル名を取り出すか、割り当てます。

構文

ファイル変数.Filename

FileSize

ファイルサイズをバイト単位で取り出します。

リソースフォークのサイズが少しでもあれば、返されるファイルサイズに算入されます。

構文

ファイル変数.FileSize

FileType

Macintosh ファイルの type タグを取り出すか、割り当てます。

この属性は Windows ファイル形式では無視されます。

構文

ファイル変数.FileType [= 文字列]

この属性には、長整数型の値もしくは Macintosh ファイルの type タグ（例えば、MyFile.FileType = "TEXT"）を指定することができます。

Hidden

Windows ファイルの hidden 属性を取り出すか、割り当てます。

この属性は Macintosh ファイルのダウンロードでは無視されます。

構文

ファイル変数.Hidden

ModificationDate

ファイルの最終修正日を取り出すか、割り当てます。修正日の割り当ては、ファイルのダウンロードにのみ適用されます。

構文

ファイル変数.ModificationDate

Normal

Windows ファイルの通常ファイルであるかどうかを取り出すか、割り当てます。

この属性のファイルには、読み取り専用属性、システム属性、隠しファイル属性、アーカイブ属性設定はありません。

構文

ファイル変数.Normal

ReadOnly

Windows ファイルの読み取り専用属性を取り出すか、割り当てます。

この属性は、Macintosh ファイルのダウンロードでは無視されます。

構文

ファイル変数.ReadOnly

ResourceSize

ファイルのリソースフォークのサイズをバイト単位で取り出します。

構文

ファイル変数.ResourceSize

SubDir

ファイルエントリーがサブディレクトリであるかどうかを表します。FindFirst と FindNext メソッドから返されるディレクトリ情報にアクセスするのに使用されます。この属性は読み取り専用です。

TRUE または FALSE フラグにより、アイテムがサブディレクトリであるかどうかを表します。サブディレクトリであれば TRUE が返され、ファイルもしくは他のファイルシステムオブジェクトであれば FALSE が返されます。

構文

ファイル変数.SubDir

System

Windows ファイルのシステムファイル（*.dll）かどうかを取り出すか、割り当てます。TRUE または FALSE フラグによって、ファイルに OS のシステム属性が含まれているかを表します。

この属性は、Macintosh ファイルのダウンロードでは無視されます。

構文

filevariable.System

● ファイルメソッド

CloseFile

OpenFile メソッドで開いたファイルを閉じます。

注意

グローバル変数として宣言された場合、ファイルはプログラム終了時に閉じられます。ローカル変数として宣言された場合、ファイルはプログラムがそのサブルーチンを抜けたときに閉じられます。

構文

ファイル変数.CloseFile

例

Openfile メソッドの例をご覧ください。

Download

クライアントでファイルのダウンロードを開始します。

（ファイルのダウンロードのダイアログが表示されたときの）デフォルトのファイル名を特定するには、そのファイルの Filename 属性を指定してください。

ダウンロードできるのは、ファイルシステム内のファイルか、文字またはデータ変数から作ることのできるファイルです。ファイルシステム内のファイルをダウンロードするには、ダウンロードしたいファイルを OpenFile メソッドで読み取ってから、Download メソッドを呼び出してください。文字またはデータ変数から作られたファイルをダウンロードするには、現在開かれていないファイルの Data 属性を指定して、Download メソッドを呼び出してください。

構文

ファイル変数.Download

例

```
MyFile.Data = "テストファイルのデータ"

MyFile.Filename = "Test.txt"

MyFile.Download
```

EOF

ファイルの末尾（EOF）のフラグの現在の状態を返します。EOF フラグはファイルがその末尾まで読み取られたか、それとも末尾以降まで読み取られたかを返します。

ファイル変数.EOF

例

Read ファイルメソッドの例をご覧ください。

FindFirst, FindNext

構文

ファイル変数.FindFirst(ファイルスペック), ファイル変数.FindNext

FindFirst メソッドは、ファイルスペック引数（有効な OS のファイルを指定する任意の文字で、ワイルドカードも指定可）と一致する最初のファイルから情報を取り出します。FindNext メソッドは、FindFirst で指定されたファイルスペック引数と一致する次のファイル（とその後につづくファイル）を検出します。

一致するファイルがそれ以上見つからない場合には、ファイル変数の FileName 属性は空文字（""）になります。

FindFirst と FindNext を使う場合は、OpenFile でファイル変数を開かないでください。

例

MyFile.FindFirst ("C:¥Winnt¥*.exe)

MyFile.FindNext

OpenFile

読み取り、書き込み、ダウンロードのいずれはまたはすべてを行うために、ファイルを開くか、作成するか、または送信します。

構文

ファイル変数.OpenFile(パス,open 定数)

パスは、サーバー上で開かれてないファイルのパスを指定します。open 定数では、ファイルをどのように開くべきかを指定し、次の値から選んで指定することができます。

- fc.Read - 読み取り用にファイルを開きます。できない場合にはエラーを表示します。
- fc.Write - 書き込み用にファイルを開きます。できない場合はエラーを表示します。
- fc.Append -新規データが書き込まれる前に EOF マーカーを削除しないで追加するためにファイルを開きます。
  ファイルが存在しない場合には、新しいファイルを作成します。
- fcUpdate - 読み取りと書き込み用にファイルを開きます。できない場合にはエラーを表示します。
- fcReadWrite - 読み取りと書き込み用に空のファイルを開きます。
  ファイルがすでに存在していたら、その内容を削除します。存在していなければ、ファイルを作成します。

例

MyFile.OpenFile ("C:¥TEST.TXT", fcWrite)

MyFile.OpenFile ("C:¥TEST.TXT", fcRead)

Position

読み取りと書き込みのために、ファイルのカーソルを移動します。

構文

ファイル変数.Position(始点[,オフセット])

Positon メソッドは、サーバーのファイルシステム内にあるファイルのカーソル位置を動かします。始点引数は開始位置を示し、オフセット引数はその開始位置から移動するバイト数を示します（＋の値であれば前方に移動し、－の値であれば後方に移動します）。始点引数は、次のうちのいずれかでなければなりません。

- fcBegin - ファイルのカーソルはファイルの先頭を始点とします。
- fcEnd - ファイルのカーソルはファイルの末尾を始点とします。
- fcCurr - ファイルのカーソルは現在位置（不変）を始点とします。

例

MyFile.Position (fcBegin)　　'カーソルをファイルの先頭に配置

MyFile.Position (fcEnd, -10)　　'カーソルをファイルの末尾から 10 文字戻ったところに配置

Read

サーバー上のファイルのバイナリー読み込みを試みます。

構文

ファイル変数.Read(バイト数)

読み込みデータはファイルのデータ属性内に格納されます。指定されたバイト数が読み込まれる前にファイルの末尾に達した場合、Read メソッドはその最大バイト数を格納します。

例

MyFile.Read(100)

Print MyFile.Data

ReadLine

開いているテキストファイルから現在の行末まで読み込んで、その結果を文字列で返します。これによりテキストの解析がはるかに簡単になり、テキストベースのデータを RAD 変数に従来より早く読み込むことができます。

構文

ReadLine (読み込まれる最大バイト数)

例

```
Dim MyFile as File
    MyFile.OpenFile ("c:¥autoexec.bat", fcRead)
        print "お使いの autoexec.bat ファイルの先頭行は" &
    MyFile.ReadLine(1024)
    MyFile.CloseFile
```

Write

サーバー上のファイルにバイナリーデータもしくはテキストデータを書き込みます。

ファイルを OpenFile メソッドで書き込み用に開かないと、Write メソッドはエラーを表示します。

構文

ファイル変数.Write(ファイルデータ)

例

```
MyFile.Write            'この文字列がファイルに書き込まれる
```

● 内部関数

Abs

数式の絶対値を計算します。

構文

Abs(数式)

Abs は数式の符号なし絶対値を返します。例えば、Abs(-21)は２１を返し、Abs(21)は２１を返します。Abs 関数からの戻り値は、引数と同じデータ型になります。

例

```
Sub Main()
    Print "0.3456 の絶対値は",
Abs(.3456)
    Print "-0.3456 の絶対値は",
Abs(.3456)
End Sub
```

Asc

文字列式の先頭文字の ANSI 文字コードを返します。

構文

Asc(文字列式)

文字列式の引数には任意の有効な文字列を指定できますが、文字列式の先頭文字のみが判断されます。その後につづく文字はすべて無視されます。

例

Chr 内部関数のコード例をご覧ください。

Atn

数値のアークタンジェント（逆正接）を計算します。

Atn の戻り値はラジアンで表わされます。Atn の戻り値のデータ型は倍精度浮動小数点型です。

構文

Atn(数値式)

Chr

ANSI 文字コードの引数で表された文字を含む文字列を返します。

構文

Chr(数値式)

例

```
Sub Main()
    Dim i as Integer
    Dim c as String
    For i = 1 to 255
    c = Chr(i)
        Print "文字は", c
End Sub
```

Cos

数値のコサインを計算します。

Cos の戻り値はラジアンで表されます。Cos の戻り値のデータ型は倍精度浮動小数点型です。

構文

Cos(数値式)

DateSerial

年、月、日を引数として、日付の値を返します。

無効な日付が指定された場合は、システムがエラーを表示します。

構文

DateSerial(年, 月, 日)

この関数の引数は、つぎの整数型の値の範囲内の数値であればすべて受け付けます。

- 年 - 100 から 9999 まで
- 月 - 1 から 12 まで
- 日 - 1 から 31 まで

例

```
Sub Main ()
    Dim myyear As Integer
    Dim mymonth As Integer
    Dim myday As Integer
    Dim mybirthday As Date
```

```
    myyear = 1971 : mymonth = 7 : myday = 9
        Print "私の誕生日は", mybirthday
End Sub
```

DateValue

文字の引数で与えられた日付の値を返します。

構文

DateValue(文字列式)

この関数の引数には、有効な日付を表す任意の文字列式を指定できます。無効な文字列が指定された場合は、システムがエラーを表示します。

例

```
Sub Main()
    Dim mydatestr As String
    Dim mybirthday As Date
    mydatestr = "7/9/71"
    mybirthday = DateValue (mydatestr)
        Print "私の誕生日は", mybirthday
End Sub
```

Day

任意の有効な日付型の式に対して、月の何日かを表す数（1-31）を返します。

構文

Day(日付式)

例

```
Sub Main()
    Dim d As Date
    Dim daynum As Integer
    d = Today
    daynum = Day(d)
    Select Case daynum
        Case 1, 21, 31
            Print "今日の日付は英語で"; daynum; "st です。"
        Case 2, 22
            Print "今日の日付は英語で"; daynum; "nd です。"
        Case 3, 23
            Print "今日の日付は英語で"; daynum; "rd です。"
```

```
        Case Else
    End Select
End Sub
```

Exp

数値型の引数に対して、e を底とする自然対数を返します。

戻り値は浮動小数点になります。

構文

Exp(数値式)

この関数の引数には、任意の有効な数値式を指定できます。

Fix

数字の端数を削除し、整数型（または長整数型）に切り捨てた値を返します。

これは Int 関数と同じ機能です。

構文

Fix(数値式)

引数には任意の有効な数値式を指定できます。

例

```
Sub Main()
    Dim mynum As Double
    mynum = 3.1415
        Print "mynum の整数部分は"; Fix (mynum)
        Print "そして小数部分は"; mynum - Fix(mynum)
End Sub
```

Hex

数値を 16 進数で表した値を含む文字列を返します。

構文

Hex(数値式)

数値式の引数は、変換前に整数に丸められます。

例

```
Sub Main()
    Dim i As Integer
```

```
    Dim hexval As String
    For i = 0 to 256
        hexval = Hex(i)
            Print "１０進数では", i
            Print "１６進数では", hexval
    Next i
End Sub
```

Hour

任意の有効な日付式から時間を表す整数（0-23）を返します。

構文

Hour(日付式)

例

```
Sub Main()
    ...
    Dim secnum As Integer, minnum As Integer,
    hournum As Integer
    ...
    secnum = Second(d)
        Print "現在の秒は "; secnum
End Sub
```

IIf (immediate if)

式を評価し、考えられる２つの値から１つを返します。

IIf はプログラムフローを指定するのではなく、２つの値から１つを返す場合にのみ利用できます。IIf の戻り値のデータ型は返される値と同じです。

構文

IIf(評価する式, TRUE の場合の戻り値, FALSE の場合の戻り値)

TRUE の場合の戻り値と FALSE の場合の戻り値には、任意のデータ型を指定できます。これらの引数を同じデータ型にする必要はありません。

例

```
Sub Main()
    Dim i As Integer
    For i = 1 to 10
        Print "i の値は";
IIf(i<=5, "より小さいか等しい", "より大きい"); "5"
    Next i
End Sub
```

InStr

ある文字列（検索元の文字列）に、指定した文字列（検索する文字列）が最初にあらわれた場合の位置を返します。

構文

InStr([開始位置,]) 検索する文字列, 検索元の文字列 [比較のタイプ])

開始位置の引数のオプションは、検索元の文字列で検索する開始位置を設定するために使用する数値です。省略された場合、デフォルトの開始位置は 1 になります。比較のタイプの引数のオプションは、比較の際に大文字と小文字の区別を行うかどうかを決めるために使用します。

比較のタイプが 0 の場合は、大文字と小文字を区別して比較が行われ、例えば、a と A は区別されます。比較のタイプが 1 の場合は、大文字と小文字が区別されずに比較され、例えば、a と A は同じものとみなされます。比較のタイプが省略された場合、デフォルトの動作では大文字と小文字を区別します。比較にタイプを使用する場合は、開始位置を指定する必要があります。

この関数は、検索する文字列が検索元の文字列内で見つかると、最初にその文字が見つかった文字位置を示す整数を返します。検索する文字列が見つからない場合には、InStr は 0 を返します。検索する文字列の長さが 0 の場合は、start が返されます。

例

```
Sub Main()
    Dim s As String, loc As Integer
    s = "いろはにほへと　ちりぬるを　わかよたれそ　つねならむ　うゐのおくやまけふこえて　あさきゆめみし　ゑひもせす"
    loc = InStr(s, "ゆめ")
    If loc<>0 Then
        Print "「ゆめ」という文字は、つぎの文字位置に見つかりました。"; loc
    Else
        Print "「ゆめ」という文字は、検索元の文字列には見つかりませんでした。"
    End If
End Sub
```

Int

数字の端数を削除し、整数型（または長整数型）に切り捨てて返します。

構文

Int(数値式)

引数には任意の有効な数値式を指定することができます。

例

```
Sub Main()
    Dim mynum As Double
    mynum = 3.1415
        Print "mynum'の整数部分は"; Int(mynum)
        Print "そして、小数部分は"; mynum - Int(mynum)
End Sub
```

IsArray

与えられた引数が配列であるか判定します。与えられた引数が配列であるかどうかによって TRUE もしくは FALSE を返します。

構文

IsArray(変数名)

例

```
Sub Main()
    Dim i As Integer
    Dim arr(10) As Integer
    If IsArray(arr) Then
        For i = LBound(arr) To UBound(arr)
            arr(i) = i
        Next i
    End If
End Sub
```

IsDate

与えられた引数が有効な日付であるか判断します。与えられた引数が有効な日付であるどうかによって TRUE もしくは FALSE を返します。

構文

IsDate (日付式)

日付式の引数には文字列型、日付型、もしくは任意の数値型のデータを指定可能です。

例

```
Sub Main()
    Dim d As Date
    Dim s As String
    s = "1/2/97"
    If IsDate(s) Then
```

```
        d=s
        Print "有効な日付形式です。"
    Else
        Print "無効な日付形式です。"
    End If
End Sub
```

IsNumeric

与えられた引数が数値型であるか判断します。与えられた引数が有効な数値型であるかどうかによって TRUE もしくは FALSE を返します。

構文

IsNumeric(式)

LaunchURL

FirstClass クライアントで URL (http, ftp, fcp, mailto...) にリンクします。

構文

LaunchURL(URL 文字列)

リンクしたい URL を文字列としてコマンドに与えると、登録済みのブラウザ、メールクライアント、もしくは利用可能な他のアプリケーションからリンクすることができます。

例

```
Sub Main()
    LaunchURL ("http://www.fcm.co.jp/")
End Sub
```

LBound

使用できる最小のインデックスを表す整数を返します。

変数が配列でない場合は、エラーが発生します。

構文

LBound(配列 変数)

例

```
Sub Main()
    Dim i As Integer
    Dim arr(10) As Integer
    If IsArray(arr) Then
```

```
        For i = LBound(arr) To UBound(arr)
            arr(i) = i
        Next i
    End If
End Sub
```

LCase

文字列の引数をすべて小文字に変換して返します。

構文

LCase(文字列式)

例

```
Sub Main()
    Dim s As String
    s = LCase ("hEllo wOrld")
    Print s
'"hello world"と表示
    Print LCase("THIS IS NOW LOWERCASE")
'"this is now lowercase"と表示
End Sub
```

Left

文字列の引数の最も左側にある文字から始まる指定した文字数の文字列を返します。

構文

Left(文字列式, 数値式)

数値式の引数には、０以上で任意の有効な値を指定できます。数値式の長さが文字列式より大きいか、もしくは等しい場合は、すべての文字式が返されます。

例

```
Sub Main()
    Dim s As String, Ioc As Integer
    s = "the quick fox jumps over the dog"
    Ioc = InStr(s, " ")              'スペースが最初に現れる位置を見つける
    If Ioc<>0 Then
        Print "文章の最初の単語は"; Left(s, Ioc - 1)
        Print "残りの文章は"; Right(s, Len(s) - Ioc)
    Else
        print "無効な文章です(スペースがありません)."
```

```
    End If
End Sub
```

Len

文字列の引数の長さを整数で返すか、または変数を作るのに必要なバイト数を返します。

構文

Len (文字列式|変数名)

引数の文字列式には、任煮の有効な文字列式を指定することができます。

例

```
Sub Main()
    Dim s As String, length As Integer
    Dim x As Long, y As Double
    s = "the quick brown fox jumps over the lazy dog"
    length = Len(s)
    ...
        Print "倍精度小数点型のサイズは"; Len(y); "バイト"
End Sub
```

Log

自然対数の数を計算します。

構文

Log(数値型式)

LTrim

文字列の先頭にあるスペースおよびタブを削除した文字を返します。

構文

LTrim(文字列式)

引数の文字列式には任意の有効な文字列を指定できます。

例

```
  Sub Main()
      Dim s As String
      s = "              テスト文字列  "
          Print "(" & LTrim(s) & ")"
```

```
        ' (テスト文字列        )と出力
    End Sub
```

Mid

文字列の中からサブ文字列を返します。

構文

Mid(文字列式, 開始位置 [,長さ])

文字列式の引数には、任意の有効な文字列を指定できます。開始位置の引数は、サブ文字列を取り出す文字の位置を指定します。長さの引数のオプションは、返される文字数を指定します。長さの引数を省略した場合、Mid は開始位置から文字の末尾まですべての文字を返します。長さが、残りの文字列の文字数より長い場合には、開始位置から文字の末尾までの文字がすべて返されます。

例

```
  Sub Main()
        Dim s As String
        Dim loc As Integer, loc2 As Integer
        s = "the quick fox runs fast"
        loc = InStr(1, s, " ")
        loc2 = InStr(loc + 1, s, " ")
              Print "トークン:"; Mid(s, loc + 1, loc2 - loc - 1);" ' "
  End Sub
```

Minute

任意の有効な日付式から、分を表す整数（0-59）を返します。

構文

Minute(日付式)

例

Second 内部関数のコード例をご覧ください。

Month

任意の有効な日付式から、月を表す数を返します。

構文

Month(日付式)

例

```
Sub Main()
      Dim d As Date
```

```
    Dim monthnum As Integer
    d = Today
    monthnum = Month(d)
        Select Case monthnum
            Case 1
                Print "今日は"; monthnum; "月です。"
            ...
            Case Else
                Print "今日は "; monthnum; " 月です。"
        End Select
End Sub
```

MsgBoxResponse

BASIC の MsgBox コマンドに対してユーザーが最後に入力した値を返します。

MsgBoxResponse は次の定数値のうち１つを返します。

- fcOK - ユーザーが OK ボタンをクリック
- fcCancel - ユーザーがキャンセルボタンをクリック
- fcYes - ユーザーが Yes ボタンをクリック
- fcNo - ユーザーが No ボタンをクリック

構文

MsgBoxResponse

例

コード例は、Sgn 内部関数をご覧ください。

Now

サーバーのシステム時計から現在の時刻と日付を含む日付の値を返します。

構文

Now

例

```
Sub Main()
    Dim d As date
    d = Now
    Print "現在の日付と時刻は: ", d
End Sub
```

Oct

数字を 8 進数で表した値を含む文字列を返します。

構文

Oct(数値式)

数値式の引数は、返還前に整数に丸められます。

例

コード例は Hex 内部関数をご覧ください。

Replace

検索する文字列を検索元の文字列からすべて検索します。検索する文字が見つかる度に、その文字を定義した代替の文字で置換します。

構文

Replace(検索元の文字列, 検索する文字列, 置換する文字列)

検索元の文字列、検索する文字列、置換する文字列には、任意の有効な文字列を指定することができます。

例

```
Sub Main()
    Dim s As string
    s = "one, two, three, four, five"
    s = Replace(s, ",", ";")
    ' s は"one; two; three; four; five"となる。
    Print s
End Sub
```

Right

文字列の引数の最も右側にある文字から指定した文字数分の文字列を返します。

構文

Right(文字列式, 数値式)

数値式には、0 以上で、任意の有効な整数を指定することができます。数値式が文字列の長さより長いか、もしくは等しい場合、すべての文字が返されます。

例

コード例は Left 内部関数をご覧ください。

Rnd

0 以上 1 未満の分数の乱数を返します。

注意

Rnｄ関数が乱数系列を発生するようにシード値を与えるには、BASIC の Randomize コマンドを使用してください。

構文

Rnd

例

コード例は Randmize コマンドをご覧ください。

Round

小数部分の数を削除し、整数（または長整数）として四捨五入した値を返します。小数部分の数が 0.5 以上であれば、もっとも近い値に繰り上げられます。小数部分の数が 0.5 未満の場合は切り下げられます。

構文

Round(数値式)

引数には任意の有効な数値式を指定できます。

例

```
Sub Main()
    Dim mynum As Double
    Dim i as integer
    For I = 1 to 20
        mynum = i/10
            Print "mynum の値は"; mynum
            Print "四捨五入した mynum の値は";
Round(mynum)
    Next i
End Sub
```

RTrim

文字列の末尾にあるスペースおよびタブを削除した文字を返します。

構文

RTrim(文字列式)

引数の文字列式には任意の有効な文字列を指定できます。

例

```
Sub Main()
    Dim s As String
    s = " test string
        Print "(" & RTrim(s) & ")"
        'prints ( test string)
End Sub
```

Second

任意の有効な日付式から秒を表す整数値を返します。

構文

Second(日付式)

例

```
Sub Main()
    Dim d As Date
    Dim secnum As Integer, minnum As Integer, hournum As Integer
    d = Time
    hournum = Hour (d)
    minnum = Minute (d)
    secnum = Second (d)
        Print "現在の「時」は"; hournum
        Print "現在の「分」は "; minnum
        Print "現在の「秒」は"; secnum
End Sub
```

Sgn

数値の符号を含む整数値を返します。

構文

Sgn(数値式)

数値式の指数には任意の有効な数値式を指定できます。数値式の値が 0 より小さい場合には、Sgn は-1 を返します。数値式の値が 0 より大きい場合には、Sgn は 1 を返します。数値式の値が 0 の場合は、Sgn は 0 を返します。

例

```
Sub Main()
    Dim dbl As Double
    Dim mysign As Integer
    dbl = 3.1415
```

```
    mysign = Sgn (dbl)
    If mysign < 0 Then
        Print "db の値は負です。"
    Else
        If mysign > 0
            Print "dbl の値は正です。"
        Else
            Print "dbl の値はゼロです。"
        End If
    End If
End Sub
```

Shell

実行可能プログラムをサーバー上で実行します。

サーバー上で動くプログラムは、それらのプログラムに渡すコマンドラインの引数を持つことができますが、動作中のプログラムに対する追加の入力はできません。OS のシェルコマンドもこの関数から実行できます。Shell はコマンドインタプリターのエラーコードを返します。0 はコマンドインタプリターがリクエストの処理を始めることができたことを表します。0 以外はエラーを表します。

構文

Shell(文字列式)

引数には、サーバー上で実行するプログラムの完全なパスと実行可能な名前を指定しなければなりません。

例

```
Sub Main()
    Dim rc As Integer
    rc = Shell ("Copy C:¥FCPO¥LogFiles¥*.* C:¥Backup¥LogFiles")
    If rc = 0 Then
        Print "コピーは正常に開始されました。."
    Else
        Print "コピーを開始できませんでした。"
    End If
End Sub
```

Sin

サインの値を計算します。

Sin の戻り値はラジアンで表されます。Sin の戻り値のデータ型は倍精度浮動小数点型です。

構文

Sin(数値式)

Space

引数によって指定された個数のスペースの含む文字列を返します。

構文

Space(数値式)

引数には任意の有効な数値式を指定できます。

例

```
Sub Main()
    Dim MyStringVariable As String
    MyStringVariable = Space(15)                ' 'MyStringVariable 変数に 15 のスペース文字を返します。
        Print MyStringVariable
```

Spawn

FirstClass サーバー上でプロセスまたはアプリケーションを実行します。

この関数には Shell()関数にくらべていくつか利点があります。可能であれば、Shell()関数の代わりに使用してください。

構文

Spawn(アプリケーション名 [fcWait | fcNoWait])

アプリケーション名には、実行するアプリケーション名の文字列とコマンドラインの引数を入れてください。

2 番目の引数は定数で、つぎのどちらかを指定します。fcWait（同期）は、呼び出されている処理が完了するまでアプリケーションを待機させます(FirstClass サーバーを止めることができます）。fcNoWait（非同期）は、アプリケーションが処理を呼び出し、すぐに実行を再開します（FirstClass サーバーを止めません）。

```
Spawn ("C:¥Acrobat3¥Reader¥AcroRd32.exe", fcNoWait)          'サーバー上のプログラムを起動し、ただちにアプリケーションの処理を再開します。'
```

Sqr

任意の有効な数値式の平方根を（浮動小数点で）返します。

構文

Sqr(数値式)

例

```
Sub Main()
    Print "2 の平方根は", Sqr(2)
End Sub
```

StartupString

アプリケーションのスタートアップの文字列（コマンドラインの引数）を返します。

この関数は、同じアプリケーションの異なるインストールを実行時に区別させたり、FirstClass RAD プログラムに初期値を与える場合に使用します。

構文

StartupString

例

```
Sub Main()
    Print "このアプリケーションのコマンドラインの引数は";
    StartupString
End Sub
```

Str

（任意の有効な数値式に対して）数値を表す文字列を返します。

構文

Str(数値式)

例

```
    Dim cost As String
    Dim output As String
    cost = Str(5000)
    output = "コストの合計は " & cost
        Print output
End Sub
```

StrComp

２つの文字列式を比較します。

構文

StrComp(文字列 1, 文字列 2,[, 比較のタイプ])

オプションの比較のタイプ引数は大文字・小文字の区別を行うかどうか決めるために使用します。比較のタイプが 0 の場合、大文字と小文字は区別されます。比較のタイプが 1 の場合、大文字と小文字は区別されません。比較のタイプ属性が省略された場合、デフォルトの動作では大文字と小文字を区別します。StrComp は比較した評価にもとづいて、以下の整数値を返します。

- -1　文字列 1 < 文字列 2
- 0　 文字列 1 = 文字列 2
- 1　 文字列 1 > 文字列 2
- 不定　NULL ストリング

例

```
Sub Main()
    Dim s As String
    s = "aaa"
    Select Case StrComp(s, "bbb")
    Case -1
        Print "元の文字列は比較対象の文字列より小さいです。"

    End Select
End Sub
```

StrFill

ある文字を指定した長さまで繰り返す文字列を作成します。

構文

StrFill(数値式, 文字式)

数値式の引数は、戻り値の文字列で繰り返される文字数を指定します。文字式の引数は文字列で表すことも、ANSI 文字コードを使うこともできます。文字として文字列に渡された場合は、先頭文字のみが使用されます。

例

```
Sub Main()
    Dim MyStringVariable As String
    MyStringVariable = StrFill(5, "@")          ' @@@@@が返される
    Print MyStringVariable
    ...
End Sub
```

StrSplit

このコマンドは、二重に使われる区切り文字がスキップされる点を除いて、StrTokenと同様です。

StrToken

区切り文字で文字列を区切り、指定した順番の文字列を返します。これによりFirstClass RAD が区切り文字をはやく簡単に解析できるようになります。

構文

StrToken(元の文字列, 区切り文字, トークン番号)

StrToken には 3 つの引数が必要です。

- 元の文字列 - 区切られたサブ文字列を含む文字列
- 区切り文字 - トークンを区別するときに使われるすべての区切り文字を指定する文字列
- トークン番号 - （先頭文字列を１として）返されるサブ文字列の順序番号

例

この例では、管理者が最後にログインした時間を知り、バッチ管理の応答をその構成要素ごとに分割します。このコードはスペース文字と二重引用符文字 (Chr(34))を区切り文字の引数として使用しています。バッチ管理による引用符のない応答では、Chr(34)を使う必要はなく、単純にスペースもしくは必要に応じて他の区切り文字のみを使うことができます。

```
Sub Main()
    BatchAdmin ("Get User Admin 1230 +d")
    Print "元の文字列"; fcBatchAdminReply
    Print "文字列 1:" ; StrToken (fcBatchAdminReply, " " & Chr(34), 1)
    Print "文字列 2:" ; StrToken(fcBatchAdminReply, " " & Chr(34), 2)
    Print "文字列 3:" ; StrToken(fcBatchAdminReply, " " & Chr(34), 3)
    Print "文字列 4:" ; StrToken(fcBatchAdminReply, " " & Chr(34), 4)
End Sub
```

このコードを実行すると、以下のように出力されます（日付と時刻はお使いのシステムによって異なります）。

```
source string: 1230 14 "2000/01/04 13:30:39"
substring 1: 1230
substring 2: 14
substring 3: 2000/01/04
substring 4: 13:30:39
```

Tab

タブ文字を含む文字列を返します。

構文

Tab

例

```
Sub Main()
    Dim s As String
    s = "りんご" & Tab & "オレンジ" & Tab & "グレープ"
    Print s                 'タブで区切られた文字列リストを作成し、出力ウィンドウに表
示
End Sub
```

Tan

数値のタンジェント（正接）を計算します。

Tan の戻り値はラジアンで表されます。戻り値のデータ型は倍精度浮動小数点型です。

構文

Tan(数値式)

TCase

文字列の引数をタイトルに変換します。

構文

TCase(文字列式)

例

```
Sub Main()
    Dim s As String
    s = TCase("jon q. public")
    Print s                 '"jon q. public"と表示
End Sub
```

Time

サーバーのシステム時計から現在の時刻を返します。

構文

Time

例

コードの例は Second 内部関数をご覧ください。

Timer

午前 12 時から経過した秒数を返します。これは、アプリケーションのベンチマークを行うときに役立ちます。

構文

Timer

例

```
Sub Main()
    Dim timestart As Integer, timefinish As Integer
    timestart = Timer
    MsgBox("どのくらい早くクリックできるかな?")
    timefinish = Timer
        Print "あなたが OK をクリックするまでにかかった時間は、";
timefinish - timestart; "秒です。"
End Sub
```

TimeSerial

引数として指定された時、分、秒を、日付型のデータで返します。

無効な時刻が指定された場合、システムはエラーを表示します。

構文

TimeSerial(時, 分, 秒)

この関数の引数には、つぎの整数型の値の範囲内であればすべて受け付けます。

- 時 - 0 から 23 まで
- 分 - 0 から 59 まで
- 秒 - 0 から 59 まで

例

```
Sub Main()
    Dim meethour As Integer
    Dim meetmin As Integer
    Dim meetsec As Integer
    Dim mymeeting As Date
    meethour = 14: meetmin = 15: meetsec = 0
    mymeeting = TimeSerial(meethour, meetmin, meetsec)
```

```
    Print "ミーティングの時間は:", mymeeting
End Sub
```

TimeValue

文字列の引数で指定された値を日付型のデータで返します。

構文

TimeValue(文字列式)

この関数の引数には、有効な時刻を表す任意の文字列式を指定できます。無効な時刻の文字列が指定された場合は、システムがエラーを表示します。

例

```
Sub Main()
    Dim mytimestr As String
    Dim mymeeting As Date
    mytimestr = "2:15:00 PM"
    mymeeting = TimeValue(mytimestr)
    Print "ミーティングの時間は  ",mymeeting
End Sub
```

Today

サーバーのシステム時計から現在の日付を返します。

構文

Today

例

コード例は Day 内部関数をご覧ください。

Trim

文字の先頭および末尾にあるすべてのスペースとタブ文字を削除した文字列を返します。

構文

Trim(文字列式)

引数の文字列式には、任意の有効な文字列を指定できます。

例

```
Sub Main()
    Dim s As String
```

```
    s="  テスト用文字列  "
    Print "(" & Trim(s) & ")"
    '(テスト文字列)と出力
End Sub
```

UBound

配列の使用できる最大インデックスを表す整数を返します。

変数が配列でない場合は、エラーが発生します。

構文

UBound(配列変数)

例

コード例は IsArray 内部関数をご覧ください。

UCase

文字列の引数をすべて大文字に変換して返します。

構文

UCase(文字列式)

例

```
Sub Main()
    Dim s As String
    s = UCase(hELlo wOrld")
        Print s              '"HELLO WORLD と表示"
        Print UCase ("this is now uppercase")
' "THIS IS NOW UPPERCASE と"表示
End Sub
```

Val

文字列の引数に含まれる数値を返します。

構文

Val(文字列式)

文字列式の引数には、数字を表す任意の文字列を指定できます。文字列式に小数が含まれる場合、Val は浮動小数点の値を返します。その他の場合は整数値を返します。

例

```
Sub Main()
'このプログラムは 50 という文字を数値に変換し、その数字を計算で使用します。
    Dim s As String
    Dim num As Integer
    s = "50"
    num = Val(s)
        Print "数値 's' に５をかけると ",num *5
End Sub
```

Weekday

任意の有効な日付式に対応した曜日を指定する数値を返します。

構文

Weekday(日付式)

例

```
Sub Main()
    Dim d As Date
    Dim daynum As Integer
    Dim mydays(1 to 7) As String
    mydays(1) = "日曜日": mydays(2) = "月曜日":...
    d = today
    daynum = Weekday(d)
        Print "今日の曜日は"; mydays(daynum)
End Sub
```

Year

任意の有効な日付式から年を返します。

構文

Year (日付式)

例

```
Sub Main()
    Dim d As Date
    Dim yearnum As Integer
    d = Today
    yearnum = Year(d)
        Print "今年は西暦"; yearnum; "年です。."
End Sub
```

● FirstClass 内部関数

BatchAdmin

管理者レベルのバッチ管理を実行します。

戻り値には、要求したデータ、もしくはエラーが起こった場合の情報が含まれます。リターンコードが生成され、FCBatchAdminCode 内部関数を使って取り出すことができます。また、応答データの文字列が生成され、FCBatchAdminReply 内部関数で取り出すことができます。

構文

BatchAdmin (バッチコマンド)

BatchAdmin にはバッチコマンドの引数が必要です。BatchAdmin は引数を管理者レベル権限でバッチ管理に渡します。

例

```
BatchAdmin ("getadmin")
if FCBatchAdminCode <> 0 Then
    Print "バッチスクリプトにエラーがあります!!!"
    Print "エラーメッセージは:"; FCBatchAdminReply
Else
    Print "管理者アカウントは:"; FCBatchAdminReply          '管理者のユーザー
ID を所得して、デバッグウィンドウに出力します。
End If
```

FCAppBuildNumber

現在のアプリケーションの.fcx ビルド番号を（整数値で）返します。

ビルド番号は、プロジェクト属性フォーム上で入力した際に、.fcx にコンパイルされたアプリケーションに組み込まれます。このフォームは、FirstClass RAD プロジェクトマネージャーの［プロジェクト］タブで［オプション］をクリックすると利用できます。

構文

FCAppBuildNumber[(アプリケーション名)]

この関数は、オプションの引数であるアプリケーション名が与えられた場合、指定されたアプリケーションがサーバーにロードされていれば、そのビルド番号を返します。オプションの引数が与えられない場合は、 FCAppBuildNumber は呼び出されているアプリケーションのビルド番号を返します。

FCAppDescription

アプリケーションの記述情報を（文字列で）返します。

アプリケーションの記述情報の文字列は、FirstClass RAD 属性フォームに入力した際に、.fcx にコンパイルされたアプリケーションに組み込まれます。

構文

FCAppDescription[(アプリケーション名)]

この関数は、オプションの引数であるアプリケーション名が与えられた場合、指定されたアプリケーションがサーバーにロードされていれば、その記述情報を返します。オプションの引数が与えられていない場合は、FCAppDescription は呼び出されているアプリケーションの記述情報を返します。

FCAppDeveloper

アプリケーションの開発者情報を（文字列で）返します。

アプリケーションの開発者情報は、FirstClass RAD 属性フォームで入力した際に、.fcx にコンパイルされたアプリケーションに組み込まれます。

構文

FCAppDeveloper[(アプリケーション名)]

この関数は、オプションの引数であるアプリケーション名が与えられた場合、指定されたアプリケーションがサーバーにロードされていれば、その開発者情報の文字列を返します。オプションの引数が与えられていない場合は、FCAppDeveloper は呼び出されているアプリケーションの開発者情報の文字列を返します。

FCAppIcon

FirstClass RAD がコンパイルした、アプリケーションのアイコンのリソース番号を数値として返します。

構文

FCAppIcon[(アプリケーション名)]

オプションのアプリケーション名の引数が省略された場合、呼び出されているアプリケーションから値が返されます。

FCAppLoadState

アプリケーションの現在のロード状態を返します。アプリケーションは、オプションのアプリケーション名の引数で指定します。

FCAppLoadState は、引数が指定されない場合、呼び出されているアプリケーションのロード状態を返します。

この関数では、つぎの定数から１つが返します。

- FCAppLoaded - アプリケーションは、FirstClass RAD によりロードされています。
- FCAppUnloaded -- アプリケーションは、FirstClass RAD によりロードされていません。
- FCAppUnloading - FirstClass RAD はアプリケーションのアンロード中です。
- FCAppReloading - FirstClass RAD はアプリケーションのリロード中です。

構文

FCApploadState[(アプリケーション名)]

FCAppLaunchMethod

呼び出されているアプリケーションが起動された方法を返します。

この関数では、つぎの定数から１つを返します。

- FCServerLaunched - サーバーアプリケーションとして起動しました。
- FCSessionLaunched - クライアントのログインで起動しました。
- FCDebugLaunched - FirstClass RAD のデバッグモードで起動しました。
- FCUserLaunched - デスクトップのステーショナリから起動しました。
- FCAppLaunched - 他の FirstClass RAD アプリケーションから起動しました。

構文

FCAppLaunchMethod

FCAppName

呼び出されているアプリケーションの名前を（文字列で）返します。

構文

FCAppName

FCAppServerVersion

現在動いている FirstClass RAD のバージョン番号を返します。

バージョン番号は、元々の番号を 1000 倍した整数で返されます。例えば、RAD バージョン 1.210 の場合、1210 が返され、バージョン 2.0 では 2000 が返されます。以下同様です。

この内部関数は、アプリケーションと FirstClass RAD の間で下位互換の問題を判断するときに有効です。

構文

FCAppServerVersion

例

FCAppServerVersion = 1210

FCAppSessionUsage

FirstClass RAD 上で使用しているアプリケーションのセッション数を返します。

構文

FCAppSessionUsage[(アプリケーション名)]

この関数は、オプションの引数であるアプリケーション名が与えられた場合、指定したアプリケーションが現在使用しているセッション数を返します。

FCAppVersion

アプリケーションのバージョン名を（文字列で）返します。

アプリケーションのバージョンを表す文字列は、FirstClass RAD プロジェクトの属性フォームに入力した際に、fcx にコンパイルされたアプリケーションに組み込まれます。

構文

FCAppVersion[(アプリケーション名)]

この関数は、オプションの引数であるアプリケーション名が与えられた場合、指定されたアプリケーションがサーバーにロードされていれば、そのバージョン番号の文字列を返します。オプションの引数が与えれない場合は、FCAppVersion は呼び出されているアプリケーションのバージョンを表す文字列を返します。

FCBatchAdmin

バッチ管理のコマンドを FirstClass サーバーに送信し、結果のコードを返します。

この関数は、呼び出した結果が成功か失敗かを数値で返します。結果がゼロでない場合は、エラーが起こったことを示し、ゼロは成功したことを示します。バッチ管理の要求によって返された値（明示的なエラーメッセージを含む）は、FCBatchAdminReply 関数によって得ることができます。

構文

FCBatchAdmin(バッチ文字列)

バッチ文字列は、FCBatchAdmin が処理のために FirstClass バッチ管理へ送る単一の文字列の引数です。

FCBatchAdminCode

最後に送信したバッチ管理コマンドのリターンコードを返します。

FCBatchAdminCode がゼロでない値を返した場合は、最後に実行したバッチ管理コマンドにエラーがあることを示します。

構文

FCBatchAdminCode

FCBatchAdminReply

最後に行った FCBatchAdmin のファンクションコールによって得られた、バッチ管理の応答データを含む文字列を返します。

構文

FCBatchAdminReply

例

```
Sub Main()
    If FCBatchAdmin ("GET ADMIN") = 0 Then
        Print "デフォルトの管理者アカウントは";FCBatchAdminReply
    End If
End Sub
```

FCClientPlatform

現在 FirstClass クライアントを使っているプラットフォームをあらわす定数値を返します。

FCClientPlatform は、つぎの値から 1 つを返します。

- fcCLUI - コマンドラインユーザーインターフェース
- fcDOS - DOS クライアント
- fcFinger - Finger
- fcFTP - FTP
- fcFTP - FTP
- fcGateway - ゲートウェイ
- fcHTTP - HTTP
- fcIMAP - IMAP
- fcInternet - インターネットサービス
- fcJava - Java クライアント
- fcLDAP - LDAP クライアント
- fcMac OS - Mac OS クライアント
- fcMAPI - MAPI
- fcNNTP - NNTP
- fcOfflineServer - オフラインサーバー
- fcPersonalServer -FirstClass パーソナルサーバー

- fcPOP - POP
- fcSMTP - SMTP
- fcUNIX - UNIX クライアント
- fcUnknown -不明なクライアントプラットフォーム
- fcVoice - ボイス
- fcWindows - Windows クライアント

構文

FCClientPlatform

FCClientVersion

FirstClass クライアントのバージョン番号を表す整数値を返します。

構文

FCClientVersion

FCEventShiftState

イベントが起こったときに、Shift キーと Ctrl キーの現在の状態を返します。

FCEventShiftState は、Shift の状態を基準にしてつぎの整数値を返します。

- Shif キー、Ctrl キーとも押されていない - 0
- Shift キーが押されている状態 - 1
- Ctrl キーが押されている状態 - 2
- Shift キーと Ctr キーが押されている状態 - 3

構文

FCEventShiftState

FCGetPrivGroups

FCPrivGroup 配列にサーバーの権限グループリストを取り出します。

FCGetPrivGroups は、現在の FirstClass サーバーにある権限グループの総数を表す整数値を返します。FCGetPrivGroups を呼び出した後、アプリケーションは、FCPrivGroup 配列にアクセスしてサーバー権限グループの名前を取り出すことができます。

構文

FCGetPrivGroups

例

```
Sub Main()
    Dim i As Integer
```

```
    Dim NumGroups As Intege
    NumGroups = FCGetPrivGroups
    MsgBox(FCPrivGroup(1))
End Sub
```

FCPOFolder

プラットフォームに関わらず、FirstClass サーバーのメインのポストオフィスフォルダへのパスを返します。

構文

FCPOFolder

例

```
print FCPOFolder
```

FCPrivGroup

権限グループの名前を表す文字列値を返します。

構文

FCPrivGroup(グループ番号)

グループ番号の引数では、グループリストの権限グループで、0 で始まるか、または現在の列を基準にした数である順序数を指定します。FCGetPrivGroup 内部関数を呼び出して、権限グループのリスト化を開始しなければなりません。

例

```
Sub Main()
    Dim i As Integer
    Dim NumGroups As Integer
    NumGroup = FCGetPrivGroup
    For i = 0 to NumGroups -1
        Print FCPrivGroup(i)
    End If
End Sub
```

FCSpawnReturn

Spawn または Shell が開始した処理からリターンコードを取得します。

構文

FCSpawnReturn

FCSeverSerialNumber

FirstClass サーバーのシリアル番号を表す整数値を返します。

この関数は、アプリションのライセンス許諾でよく使用されます。

構文

FCServerSerialNumber

FCUserFirstName

現在アプリケーションを使用しているユーザーの名前（姓名の名）を含む文字列を返します。

構文

FCUserFirstName

例

```
Sub Main()
    Print "ご自身のアカウント情報"
    Print "....................................."
    Print "登録されている「名」は"; FCUserFirstName
    ...
End Sub
```

FCUserID

現在アプリケーションを使用している FirstClass ユーザーのユーザーID を含む文字列を返します。

構文

FCUserID

例

```
Sub Main()
    Print "ユーザーID は"; FCUserID
    ...
End Sub
```

FCUserLastName

現在アプリケーションを使用しているユーザーの名前（姓名の姓）を含む文字列を返します。

構文

FCUserLastName

例

```
Sub Main()
    Print "「姓」は:"; FCUserLastName
    ...
End Sub
```

FCUserMI

現在アプリーションを使用している FirstClass ユーザーのイニシャルを含む文字列を返します。

構文

FCUserMI

例

```
Sub Main()
    Print "「イニシャル」は:"; FCUserMI
    ...
End Sub
```

FCUserName

現在アプリケーションを使用している FirstClass ユーザーのフルネームを含む文字列を返します。

構文

FCUserName

例

```
Sub Main()
    Print "フルネームは:"; FCUserName
    ...
End Sub
```

FCUserPrivGroup

ユーザーが指定された権限グループに属しているか判断します。

ユーザーが権限グループに属していないか、あるいはその指定した権限グループが存在しない場合には、関数は FALSE を返します。ユーザーが指定した権限グループに属している場合は、関数は TRUE を返します。

構文

FCUserPrivGroup(権限グループ名)

例

```
Sub Main()
    If FCUserPrivGroup("Administrators") Then
        Print "あなたは管理者権限グループのメンバーです。"
    Else
        Print "あなたは管理者権限グループのメンバーではありません。"
        Print "アクセスは拒否されました。"
        End                 'プログラムを終了します。
    End If
End Sub
```

● ステートメント属性

AsyncEnable

ODBC の非同期のステートメントを含む整数値を取り出すか、または与えます。

構文

ステートメント名.AsyncEnable [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_ASYNC_ENABLE_OFF - オフ
- SQL_ASYNC_ENABLE_ON - オン
- SQL_ASYNC_ENABLE_DEFAULT - オフ

この属性のデフォルト値は SQL_ASYNC_ENABLE_OFF です。これは、ステートメントが同期処理されることを表します。

BindType

ODBC の列をバインドするステートメントを含む整数値を読み出すか、または与えます。

構文

ステートメント名.BindType [= 定数]

定数として利用可能な値は次のとおりです。

- SyntaxSQL_BIND_BY_COLUMN - 列のバインド
- SQL_BIND_TYPE_DEFAULT - 行方向バインド

この属性のデフォルト値は SQL_BIND_TYPE_DEFAULT です。

Column

列の属性とメソッドを読み出すか、または割り当てます。

構文

ステートメント名.Column(列名 | 列)[.属性] [= 値]

この属性は、名前が順序数のどちらかによって、カーソルの列を参照します。

Concurrency

ODBC の並列処理のステートメントを含む整数値を読み出すか、または割り当てます。

構文

ステートメント名.Concurrency [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_CONCUR_READ_ONLY - 更新が試みられ、カーソルは読み取り専用になります。
- SQL_CONCUR_LOCK - カーソルは、行が確実に更新できるような最も低いレベルのロックを使用します。
- SQL_CONCUR_ROWVER - カーソルは、オプティミスティック並行処理制御を使用して、行のバージョンを比較します。
- SQL_CONCUR_VALUES - カーソルは、オプティミスティック並行処理制御を使用して、値を比較します。

SQL_CONCURRENCY のデフォルト値は SQL_CONCUR_READ_ONLY です。

CursorType

ODBC のカーソルタイプのステートメントを読み出すか、または割り当てます。

構文

ステートメント名.CursorType [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_CURSOR_KEYSET_DRIVE
- SQL_CURSOR_FORWARD_ONLY
- SQL_CURSOR_DYNAMIC
- SQL_CURSOR_STATIC
- SQL_CURSOR_TYPE_DEFAULT
- SQL_CURSOR_FORWARD_ONLY - カーソルは前方スクロールタイプのみです。
- SQL_CURSOR_STATIC - 結果セットのデータは静的です。
- SQL_CURSOR_KEYSET_DRIVEN - ドライバは、SQL_KEYSET_SIZE ステートメントオプションで指定された行番号のキーを使用します。
- SQL_CURSOR_DYNAMIC - ドライバは行セットの行に対するキーを使用します。

デフォルト値は SQL_CURSOR_FORWARD_ONLY です。この属性は、オープンカーソルには設定できません。

DisplayErrors

ODBC のエラーが起きた場合に、デフォルトのエラーハンドラシステムを使用するか、それとも使えないようにするかを指定します。

DisplayErrors 属性を TRUE（デフォルトです）に設定すると、ODBC はモーダルメッセージボックス（一度開いたら、閉じるまで他の操作ができないメッセージボックス）

で、エラーの影響を受けているアプリケーションに ODBC エラーが起こっていることを伝えます。

DisplayErrors 属性を FALSE に設定すると、エラーメッセージは表示されず、通常のアプリケーション処理が続行されて、エラーは無視されます。その後、DBError 属性と DBErrorCode 属性によって特定のエラー情報を取り出すことができ、データベースのエラーハンドラをカスタマイズすることができます。

構文

ステートメント名.DisplayErrors [= TRUE/FALSE]

例

```
Sub Main ()
    Dim stmt as dbStatement
    stmt.OpenStatement(Parks, dbDynamic, dbRowver)
    stmt.DisplayErrors = FALSE
    stmt.ExecuteSQL("Select * from parkinfo")
    ...
End sub
```

DisplayWarnings

ステートメントで ODBC の警告が起こった場合、その警告メッセージを表示させるかどうかを指定します。

構文

ステートメント名.DisplayWarnings [= TRUE|FALSE]

デフォルト値は FALSE で、警告は表示されません。

KeysetSize

ODBC のキーセット駆動型カーソルで、行番号を指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.KeysetSize [= サイズ]

サイズ変数には、指定したいサイズか、デフォルトの SQL_KEYSET_SIZE_DEFAULT を設定することができます。デフォルト設定では、キーセットは完全にキーセット駆動型になります。

MaxLength

列から返すことのできるデータの最大長を指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.MaxLength [= 長さ]

長さ変数の値には、指定したい長さか、SQL_MAX_LENGTH_DEFAULT の定数を設定することができます。後者では、列がすべてのデータを返します。

MaxRows

ステートメントで返される ODBC の最大行数を指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.MaxRows [= 行番号]

行番号変数には、指定したい番号か、SQL_MAX_ROWS_DEFAULT の定数を設定することができます。後者はクエリーからすべての行が返されます。

NoScan

ODBC ドライバが ODBC のエスケープ句で SQL 文字列をスキャンするかどうか指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.NoScan [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_NOSCAN_OFF - オフ
- SQL_NOSCAN_ON - オン
- SQL_NOSCAN_DEFAULT - オン

この属性のデフォルト値は SQL_NOSCAN_ON です。これは、エスケープ句で SQL 文字列をスキャンしません。

NumCols

ステートメントに返される列の数を取り出します。

この属性は、ランタイム時のみ読み取り可能です。

構文

ステートメント名.NumCols

QueryTimeout

ODBC クエリーがタイムアウトする秒数を指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.QueryTimeout [= タイムアウト秒数]

タイムアウト秒数の変数には、指定したい秒数か、または SQL_QUERY_TIMEOUT_DEFAULT の定数を設定することができます。後者では、クエリーは完了まで続行されます。

RetrieveData

ODBC ドライバが現在の行のデータを自動的に取り出すかどうかを指定する整数値を取り出すか、または割り当てます。

構文

statementname.Retrieve [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_RD_OFF - オフ
- SQL_RD_ON - オン
- SQL_RD_DEFAULT - オン

この属性のデフォルト値は SQL_RD_ON です。これは、各列のデータを自動的に取り出します。

RowsetSize

カーソルの行セットのサイズを指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.RowsetSize [= 行数]

行数の変数には、指定したい行セットの行数か、または SQL_ROWSET_SIZE_DEFAULT を設定します。後者では、1 レコードの行セットサイズに指定されます。

SimulateCursor

シミュレートされた定位置の Update および Delete ステートメントが、 単一行だけに影響を及ぼすようにするかどうかを指定する整数値を取り出すか、または割り当てます。

構文

ステートメント名.StatementCursor [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_SC_NON_UNIQUE - 単一行だけに影響を及ぼすようにはしません。アプリケーションは、このように動作しなければなりません。
- SQL_SC_TRY_UNIQUE - 必ず単一行だけに影響を及ぼすよう試みます。ステートメントが 1 行以上の行に影響を及ぼす場合でも、ドライバは常にこのようなステートメントを実行します。
- SQL_SC_UNIQUE - ドライバは、必ず単一行だけに影響を及ぼすようにします。

UseBookmarks

ステートメントがブックマークを利用できるかどうかを指定する整数値を取り出すか、または割り当てます。

注意

カーソルでブックマークを使うには、アプリケーションでこのオプションを指定してからカーソルをオープンしなければなりません。

構文

ステートメント名.UseBookmarks [= 定数]

定数として利用可能な値は次のとおりです。

- SQL_UB_OFF - オフ
- SQL_UB_ON - オン
- SQL_UB_DEFAULT - オフ

この属性のデフォルト値は SQL_UB_OFF です。

● ステートメント メソッド

BOF

ファイルの先頭（BOF）のフラグの現在値を返します。

プログラムが結果セットの先頭からレコードポインタを動かそうとする際に、BOF フラグが TRUE に設定され、レコードポインタが結果セットの先頭レコードの位置に置かれます。

注意

セットの最前列から移動するわけではありません。行セットの途中にある場合、そこからの判定となります。

レコードポインタが結果セット内の新規レコードへ移動できると、BOF は FALSE を返します。レコードセットが空であれば、BOF と EOF は常に TRUE を返します。

構文

ステートメント名.BOF

準拠レベル

レベル 2

例

次の例は、最終レコードから先頭レコードへ向かって、データベースからすべての接続先名を表示する方法です。

```
Sub DisplayContactNames(Contact as DBStatement)
    Contact.MoveLast
    Do While Not Contact.BOF
        Print "連絡先名:";          Contact.Column("名前")
    Contact.MovePrev
    Loop
End Sub
```

CloseStatement

データベース接続のステートメントをクローズします。

更新されなかった行バッファに対して行われた変更は、破棄されます。

構文

ステートメント名.CloseStatement

準拠レベル

コアレベル

DBError

最後に発生したデータベースステートメントのエラーを含む文字列を返します。

このメソッドは、エラー処理ルーチンで発生したエラーのタイプを確認するのに有効です。

構文

statementname.DBError

準拠レベル

コアレベル

DBErrorCode

最後に実行した操作の ODBC エラーコードを表す数値を格納します。

ODBC が返す共通の戻り値を表す定数値は次のとおりです。

- db_success - 操作は正常終了しました。
- db_success_with_info - 操作は正常終了し、詳細な情報を利用可能です。
- db_no_data _found - データが要求されたときの操作からは、何のデータも返されませんでした。
- db_error - ODBC エラーが発生しました。

構文

ステートメント名.DBErrorCode

Delete

現在のレコードを削除し、レコードポインタを次のレコードに移動します。Delete メソッドは、フォームのフィールド上で利用できます。

位置指定 DELETE 形式でレコードを削除する場合には警告が表示されません。お使いのアプリケーションで、削除したいレコードだけを削除するよう気をつけてください。削除の確認はアプリケーションが行います。

準拠レベル

レベル 2

例

次の例では、指定したレコードを検索し、その情報を出力し、位置指定 DELETE 形式を使ってデータベースから削除します。

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")                ' "MyDSN"データソースをオープン
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM MyTable WHERE CustNum = 12345")
    Print "顧客の削除"
    Print "名前"; stmt.Column("名前")
    Print "電話番号:"; stmt.Column("電話番号")
    Print "最後に連絡をとった日時";
stmt.Column("CustLastContact")
    stmt.Delete             '現在のレコードを削除
    stmt.CloseStatement
    cnct.CloseConnection            '接続をクローズ
End Sub
```

Empty

現在のレコードセットにレコードがあるかどうかを返します。

レコードセットにレコードがある場合、Empty は FALSE を返します。ない場合は、TRUE を返します。

構文

ステートメント名.Empty

準拠レベル

コアレベル

例

コードの例は、ExecuteSQL ステートメントメソッドをご覧ください。

EOF

ファイルの末尾（EOF）のフラグの現在値を返します。

プログラムが結果セットの末尾からレコードポインタを動かそうとする際に、EOF フラグが TRUE に設定され、レコードポインタが結果セットの最終レコードの位置に置かれます。

注意

セットの最後尾から移動するわけではありません。行セットの途中にある場合、そこからの判定となります。

レコードポインタが結果セット内の新規コードへ移動できると、EOF は FALSE を返します。レコードセットが空であれば、BOF と EOF は常に TRUE を返します。

構文

ステートメント名.EOF

準拠レベル

コアレベル

例

次の例は、先頭レコードから最終レコードへ向かって、データベースからすべての接続先を表示する方法です。

```
Sub DisplayContactNames(Contact as DBStatement)
    Contact.MoveFirst
Do While Not Contact.EOF
        Print "連絡先"; Contact.Column("名前")
    Contact.MoveNext
    Loop
End Sub
```

ExecuteSQL

SQL ステートメントを実行し、妥当な場合、カーソルを返します。

構文

ステートメント名.ExecuteSQL(クエリー)

準拠レベル

コアレベル

例

次の例では、データソースへ接続してステートメントを開き、クエリーを実行して特定の顧客レコードが存在しているか確認します。該当するレコードが存在していればその顧客情報を表示します。存在していなければ、注意を表示します。

```
Sub Main()
    Dim cnct As DBConnection
Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")              '"MyDSN"データソースを開く。
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM MyTable WHERE CustNum = 12345")
    If (stmt.Empty) Then              'レコードが１つも返されない場合
        Print "データベースに該当する顧客は存在しません!"
```

```
    Else
        Print "顧客情報："
        Print "名前"; stmt.Column("名前")
        Print "電話番号"; stmt.Column("電話番号")
        Print "最後に連絡した日時";stmt.Column("最後に連絡した日時")
    End If
    stmt.CloseStatement
    cnct.CloseConnection                '接続を閉じる
End Sub
```

Filter

SQL の WHERE 節を利用して、レコードセットの再クエリーを実行します。

Filter メソッドは、レコードセットに含まれるレコードの範囲を速く、効率的に絞り込むことができる方法です。Filter は、SQL の WHERE 節がない場合には、SQL ステートメントで生成されるどのようなステートメントのカーソル上でも実行することができます。

Filter は、バインドされたテーブルのアプリケーション内でレコードを検索するのに適したメソッドです。

構文

ステートメント名.Filter(WHERE 節)

Filter コマンドは、WHERE 節引数で指定された WHERE 節を元の SQL ステートメントに追加して、データベースの再クエリーを実行します。クエリーが終わると、レコードポインタがレコードセット内の先頭レコードに移動します。

WHERE 節を""にして Filter メソッドを呼び出すと、現在の Filter メソッドの WHERE 節が削除され、Filter 実行前の完全なレコードセットが返されます。

準拠レベル

コアレベル

例

次の例では、フォームから読み込まれた文字列を使って、顧客名でデータ環境内の顧客テーブルに Filter を実行します。

```
Sub Click()
    Dim filterstr As String
    filterstr = "MyCustName Like "          ' & txtSearchText.Text & "%'"
    Customers.Filter(filterstr)
End Sub
```

FindFirst, FindLast, FindNext, FindPrev

レコードセット内のレコードを検索します。

構文

ステートメント名.FindFirst(条件),
ステートメント名.FindLast(条件),
ステートメント名.FindNext(条件),
ステートメント名.FindPrev(条件)

このメソッドでは、レコードポインタは、条件が TRUE と評価されるか、またはレコードセットの末尾に到達するまでレコードセットを移動します。レコードポインタが移動する度に、条件が再評価されます。

レコードポインタの開始位置と移動方向は、使用するメソッドにより異なります。

- FindFirst - 先頭レコードで開始され、レコードセットを前に移動します。
- FindLast - 最終レコードで開始され、レコードセットを後ろに移動します。
- FindNext - 現在のレコードで開始され、レコードセットを前に移動します。
- FindPrev - 先頭レコードで開始され、レコードセットを後ろに移動します。

条件が TRUE と評価されずにレコードセットの末尾へ到達した場合には、NoMatch フラグが TRUE に設定されます。

条件の引数には、任意の有効な条件式を指定できます。しかし、最も適切な指定方法は、カーソル内の列を指定することです。そうしなければ、条件が TRUE と評価される可能性はほとんどありません。

注意

非同期のシステム間でのデッドロックを回避するために、Find では、移動と条件の評価の度にサーバーへの再接続が必要になります。再接続では大量のオーバーヘッドが生成されますので、たいていの場合、速く効率的にレコードを検索するには、Find メソッドではなく Filter メソッドを使用する方がいいでしょう。

準拠レベル

レベル 2

例

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")              '"MyDSN"データソースを開く
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL ("SELECT * FROM MyTable")
    'レコードセットの先頭レコードから開始して、
```

```
    '顧客番号 = 12345 という条件が TRUE と評価される最初のレコードを検索
    stmt.FindFirst(stmt.Column("顧客番号") = 12345)
    If (stmt.NoMatch) Then              'レコードが見つからなかった場合
        Print "該当する顧客はデータベースに存在しません!"
    Else
        Print "顧客情報:"
        Print "名前: ";stmt.Column("名前")
        Print "電話番号: ";stmt.Column("電話番号")
        Print "最後に連絡した日時: ";stmt.Column("最後に連絡した日時")
    End If
    stmt.CloseStatement
    cnct.CloseConnection                '接続を閉じる
End Sub
```

GetTables

現在のデータソースに関するテーブル情報を取り出し、結果セットとして返します。

構文

ステートメント名.GetTables

準拠レベル

レベル 1

Insert

現在の行バッファをクリアにし、レコードセットの末尾に挿入位置を確保して書き込む準備をします。Update メソッドが実行されるまで、そのレコードは実際にはデータベースに書き込まれません。

構文

ステートメント名.Insert

準拠レベル

レベル 2

例

次の例では、Insert を利用して、レコードセットにレコードを挿入します。ステートメントがいったん挿入モードになると、行バッファに値が指定されてから更新され、データベースに新規レコードが書き込まれます。

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
```

```
    cnct.OpenConnection("MyDSN")              ' "MyDSN"データソースを開く
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM My Table")
    stmt.Insert               '新規レコードの挿入
    stmt.Column("顧客名") = "春田"
    stmt.Column("住所") = "東京都千代田区内神田 2-14-3"
    stmt.Column("最後に連絡した日時") = Today
    stmt.Update               'データベースに新規レコードを書き込み
    stmt.CloseStatement
    cnct.CloseConnection              '接続を閉じる
End Sub
```

Lock

現在のレコードをロックし、他のユーザーが更新できないようにします。

Lock メソッドにより、トランザクションに対応していない接続環境で、アプリケーションが並行処理制御を行うことを可能にします。一般に、並行処理レベルとトランザクションに対応しているデータソースは、Lock メソッドに対応していません。

擬似的にトランザクションを行うには、アプリケーションは Lock メソッドを使って、トランザクション処理を行うそれぞれの行をロックします。つぎに、Update メソッドまたは Delete メソッドを使って、それぞれの行を更新または削除します。トランザクションの全処理が完了したら、アプリケーションは Unlock メソッドを使って、それぞれの行のロックを解除します。

行は、アプリケーションが Unlock メソッドを呼び出すか、あるいはステートメントが閉じられるまで、Lock メソッドによってロックされたままになります。

構文

ステートメント名.Lock

準拠レベル

準拠レベル 2 のドライバ

例

次の例では、現在のレコードをロックし、更新してからロックを解除して、他のユーザーが修正できるようにします。

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")              '"MyDSN"データソースを開く
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM My Table WHERE CustNum = 12345")
```

```
    stmt.Lock
    stmt.Column("名前") = "春田"
    stmt.Column("住所") = "東京都千代田区内神田 2-14-3"
    stmt.Column("最後に連絡した日時") = Today
     stmt.Update              'データベースに新規レコードを書き込み
    stmt.Unlock
    stmt.CloseStatement
    cnct.CloseConnection              '接続を閉じる
End Sub
```

MakeList

選択リスト（入力可または入力不可）として利用するのに適したリストを、データベースの列から作成します。

MakeList ステートメントにより、結果を連結して、リストをセミコロンで区切ることで、ステートメントに含まれる全アイテムのリストを作成することができます。

構文

MakeList(列名, [列番号])

列名とはリストの中身を作るのに使われるデータの列名です。列番号はオプションの引数で、リスト内のアイテムに数値を追加する場合に使用します。

例

```
...
Dim stmt As DBStatement
stmt.OpenStatement (MyConnection)
stmt.ExecuteSQL ("SELECT NAME ID FROM NAMES WHERE ID < 40")
1stNames.List=stmt.MakeList ("NAME", "ID")
stmt.CloseStatement
...
```

MoveFirst, MoveLast, MoveNext, MovePrev

レコードポインタを移動して、バインドされた列を更新します。

Move メソッドは、Update メソッドによってデータベースに変更内容が書き込まれなかった場合、修正もしくは挿入されたレコード上のいかなる変更も自動的に破棄します。

構文

ステートメント名.MoveFirst, ステートメント名.MoveLast,

ステートメント名.MoveNext, ステートメント名.MovePrev

レコードポインタの開始位置と移動方向は、使用するメソッドによって異なります。

- MoveFirst - レコードポインタをレコードセットの先頭レコードに移動します。
- MoveLast - レコードポインタをレコードセットの最終レコードに移動します。
- MoveNext - レコードポインタをレコードセットの次のレコードに移動します。
- MovePrev - レコードポインタをレコードセットの前のレコードに移動します。

準拠レベル

レベル 2

例

次の例では、先頭レコードから最終レコードまで、データベースからすべての連絡先を表示します。

```
Sub DisplayContactNames(Contact as DBStatement)
    Contact.MoveFirst
    Do While Not Contact.EOF
        Print "連絡先:"; Contact.Column("名前")
        Contact.MoveNext
    Loop
End Sub
```

NoMatch

最後に実行した Find メソッドが合致するレコードを見つけたかどうかによって、TRUE もしくは FALSE を返します。

構文

ステートメント名.NoMatch

準拠レベル

レベル 2

例

コード例は、Find ステートメントメソッドをご覧ください。

OpenStatement

データベース接続を開きます。

構文

ステートメント名.OpenStatement(連結名 [,データベースカーソルタイプ] [データベース並行処理])

連結名には、有効なデータソース上ですでに接続している有効な DBConnection オブジェクト名を指定します。

オプションの引数であるデータベースカーソルタイプとデータベース並行処理は、それぞれステートメントのカーソルタイプと並行処理方法を設定することができます。

データベースカーソルタイプ引数では、次の定数から１つを設定することができます。

- dbForwardOnly - 前方スクロールタイプカーソル
- dbStatic - 静的カーソル
- dbKeyset - キーセット駆動カーソル
- dbDynamic - 動的カーソル

データベース並行処理には、次の定数から１つを設定することができます。

- dbReadOnly - カーソルは読み取り専用です。
- dbLock - カーソルは、行レベルロックの並行処理を使用します。
- dbValues - カーソルはオプティミスティック並行処理を使用して、値を比較します。
- dbRowver - カーソルはオプティミスティック並行処理を利用して、行の新旧のバージョンを比較します。

カーソルタイプと並行処理方式は Cursor Type 属性と Concurrency 属性で明示的に設定することもできますが、データベースカーソルタイプ引数とデータベース並行処理引数を使うと設定が簡単になります。

ステートメントのカーソルタイプと並行処理方式を明示的に指定しない場合、ステートメントはデフォルトで前方スクロールタイプの読み取り専用カーソルを作成します。

準拠レベル

コアレベル

例

次の例では、データソースに接続してステートメントを開き、クエリーを実行して特定の顧客レコードが存在しているか確認します。該当するレコードが存在していればその顧客情報を表示します。存在していなければ、注意を表示します。

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")              '"MyDSN"データソースを開く
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM MyTable WHERE CustNum = 12345")
    If (stmt.Empty) Then                      'レコードが１つも返されない場合
        Print "データベースに該当する顧客は存在しません!"
    Else
        Print "顧客情報:"
```

```
        Print "名前:"; stmt.Column("名前")
        Print "電話番号:"; stmt.Column("電話番号")
        Print "最後に連絡した日時";stmt.Column("最後に連絡した日時")
    End If
    stmt.CloseStatement
    cnct.CloseConnection              '接続を閉じる
End Sub
```

Refresh

現在の行バッファの値をリフレッシュします。

元となるデータベースの現在の行に変更があった場合、Refresh メソッドを呼び出すと、ローカルカーソルが更新され、該当する変更内容がバインドされたコントロール（テキストボックスやリストなど）にも適用されます。

行バッファの現在値を修正する場合、Refresh を呼び出すと、データベースから元の行の新規コピーが作成されて、変更が上書きされます。

構文

ステートメント名.Refresh

準拠レベル

レベル 2

例

```
Sub Click()
    MsgBox("現在のレコードを更新しますか?",FCYesNo, "更新")
    If MsgBoxResponse = FCYes Then
        MyStmt.Update              '現在のレコードを更新
    Else
        MyStmt.Refresh             '変更を破棄して元の値が取り出す
    End If
End Sub
```

Requery

現在のステートメントのクエリーを再実行します。これは Refresh メソッドとよく似ていますが、カーソル内のすべての行がリフレッシュされる点が異なります。Requery メソッドが呼び出されている状態では、レコードポインタは結果セットの先頭レコードに位置しています。

構文

ステートメント名.Requery

準拠レベル

コアレベル

Unfilter

Filter メソッドで指定する WHERE 節を使わないで、レコードセットのクエリーを再実行します。

Unfilter メソッドを呼び出すことは、WHERE 節を""にして Filter メソッドを呼び出すのと同じことになります。Unfilter は現在の Filter の WHERE 節を削除して、Filter 実行前の完全なレコードセットを返します。

構文

ステートメント名.Unfilter

準拠レベル

コアレベル

Unlock

現在のレコードのロックを解除し、他のユーザーが更新できるようにします。

構文

ステートメント名.Unlock

準拠レベル

準拠レベル 2 のドライバ

例

コード例は、Lock ステートメントメソッドをご覧ください。

Update

データベースの現在の行を更新します。可能な場合には、その行の変更内容が全てデータベースに書き込まれます。Update メソッドは、新規レコードの挿入を確認する場合にも使用されます。

構文

ステートメント名.Update

準拠レベル

レベル 2

例

次の例では、SQL を実行してレコードを取り出し、いくつかの列の値を変更して、その変更を位置指定した Update メソッドを使ってデータベースに書き込みます。

```
Sub Main()
    Dim cnct As DBConnection
    Dim stmt As DBStatement
    cnct.OpenConnection("MyDSN")              '"MyDSN"データソースを開く
    stmt.OpenStatement(cnct, dbKeyset, dbRowver)
    stmt.ExecuteSQL("SELECT * FROM MyTable WHERE CustNum = 12345")
    stmt.Column("名前") = "上田"
    smt.Column("住所") = "港区六本木 1-2-3000"
    stmt.Column("最後に連絡した日時") = Today
    stmt.Update               '変更をデータベースに書き込む
    stmt.CloseStatement
    cnct.CloseConnection              '接続を閉じる
End Sub
```

4 サンプルファイルについて

● サンプルファイルの内容

FirstClass RAD 日本語版には以下の３つのサンプルファイルを収録しています。お客様ご自身でアプリケーションの作成方法を学習するにあたり、ぜひこれらサンプルファイルをお試しください。

サンプルファイルは、FirstClass RAD でプログラム開発を行うことができるユーザーで FirstClass にログインしたときの、［FirstClass RAD］フォルダ >［Sample Apps］フォルダに収録されています。

サンプル１：ディスク容量チェック

FirstClass デザイナーと FirstClass バッチ管理を利用して、ユーザーに割り当てられているディスク容量と現在のディスク使用量を表示するアプリケーションです。

ファイル名

DiskUse

サンプル２：ユーザー用欠席連絡フォーム

Microsoft 社の Access で作成したデータベースと連携するアプリケーションで、ユーザーがフォーム上で欠席願いをすることができます。

ファイル名

Absence

サンプル３：管理者用出欠管理フォーム

サンプル２の「ユーザー用欠席連絡フォーム」から送られたユーザーの欠席願いを、管理者が許可あるいは拒否するフォームです。

ファイル名

AbsenceAdmin

エラーメッセージ一覧

FirstClass RAD のエラーメッセージは、エラーメッセージ番号に基づいたカテゴリに大まかに分けられています。

コードエラーメッセージ (1～107)

1

メッセージ: **文法誤り**

コードの文法に誤りがあります。エラーが発生した行番号を確認してください。

2

メッセージ: **メモリ確保失敗**

システム上の使用可能なメモリ量を超えてメモリを要求しました。

3

メッセージ: **プログラムのエントリポイントが見つかりません，Sub Main()がありません**

プログラムコードのどこにも Sub Main()が見つかりません。

4

メッセージ: **括弧の数が合わない**

プログラムコード内に必要な括弧(開括弧または閉括弧のいずれか)が抜けています。

5

メッセージ: **等号が必要**

プログラムコード内に適切な場所に等号がありません。このエラーは代入ステートメントまたは変数の代入時に発生します。

6

メッセージ: **ラベルが重複**

.プログラムコード内に同じ名前のラベルが 2 つあります。

7

メッセージ: **未定義のラベル**

プログラムコード内で使用しているラベルが定義されていません。

8

メッセージ: **THEN が必要**

THEN コマンドが抜けています。

9

メッセージ: **TO が必要**

TO コマンドが抜けています。

10

メッセージ: **NEXT には FOR が必要**

FOR コマンドが抜けています。

11

メッセージ: **不正な識別子名**

識別子名に不正な文字が使われています。

12

メッセージ: **AS が必要**

AS のない DIM コマンドがあります。

13

メッセージ: **不正な variable / type / sub / function の再定義**

変数、type、サブルーチン、関数を 2 回または間違って設定しています。

14

メッセージ: **宣言されていない識別子**

宣言せずに識別子を使用しています。

15

メッセージ: **型の不整合エラー**

異なる型同士で代入を行っています。

16

メッセージ: **ゼロ除算エラー**

コード内でゼロで除算を行おうとしました。この計算はできません。

17

メッセージ: **END IF が必要**

必要な END IF コマンドが抜けています。

18

メッセージ: **END IF には IF が必要**

必要な IF コマンドが抜けています。

19

メッセージ: **ELSE には IF が必要**

必要な IF コマンドが抜けています。

20

メッセージ: **WEND には WHILE が必要**

必要な WHILE コマンドが抜けています。

21

メッセージ: **WEND が必要**

必要な WEND コマンドが抜けています。

22

メッセージ: **配列の引数が不正な値**

配列を定義するときに小さすぎるか大きすぎる引数を使用しています。

23

メッセージ: **識別子が必要**

必要な識別子名が抜けています。

24

メッセージ: **配列の引数が範囲を超えた**

配列の定義された境界値外の引数でアクセスしました。

25

メッセージ: **変数は配列ではない**

変数が配列として定義されていません。

26

メッセージ: **定数の再定義は禁止**

同じ名前の 2 つの定数があります。

27

メッセージ: **END SELECT が必要**

必要な END SELECT コマンドが抜けています。

28

メッセージ: **END SELECT には SELECT CASE が必要**

必要な SELECT CASE コマンドが抜けています。

29

メッセージ: **CASE が必要**

必要な CASE コマンドが抜けています。CASE は SELECT CASE のサブセットです。

30

メッセージ: **サブルーチンの引数が不正な値**

サブルーチン呼び出し時の引数の数に誤りがあります。

31

メッセージ: **配列に対する範囲が不正**

誤った範囲で配列が定義されています。たとえば、配列の範囲として負数は使用できません。また配列の下限値は上限値より小さくしなければなりません。

32

メッセージ: **キーワードは変数として使用禁止**

キーワードが変数として使用されています。キーワードはプログラム言語の一部で、変数名として使用することはできません。

33

メッセージ: **メモリ確保エラー, 配列のサイズが大きすぎる**

システムで使用可能なメモリより大きいサイズの配列を作成しようとしています。

34

メッセージ: **不明なメソッド参照**

存在しないメソッドを参照しています。

35

メッセージ: **未定義のオブジェクト**

コード内で定義されていないオブジェクトがあります。

36

メッセージ: **不正なオブジェクト参照**

.存在しないオブジェクトを参照しました。

37

メッセージ: **IF が必要**

IF…THEN ELSE 命令文で必要な IF コマンドが抜けています。

38

メッセージ: **式が必要**

コード内で式が抜けています。

39

メッセージ: **ELSEIF には IF が必要**

IF…ELSEIF 命令文で IF コマンドが抜けています。

40

メッセージ: **ELSE の後の ELSEIF**

ELSE コマンドの後に ELSEIF コマンドがあります。ELSE は最後の節になければなりません。

41

メッセージ: **ステートメントの終わりまたは改行が必要**

1 行にいくつかの命令文があります。

42

メッセージ: **データ形式が必要**

データ形式が正しく定義されていません。

43

メッセージ: **NEXT ステートメントのカウンタが不正**

ループに対して間違ったカウンタが使用されています。

44

メッセージ: **FOR には NEXT が必要**

FOR…NEXT 命令文の NEXT コマンドが抜けています。

45

メッセージ: **SELECT CASE と最初の CASE との間のステートメントは禁止**

SELECT CASE と最初の CASE コマンドの間にコードがあります。

46

メッセージ: **数式, IS または ELSE が必要**

IS または ELSE コマンドが無いのに CASE 数式があります。

47

メッセージ: **IS の後には比較演算子が必要**

IS コマンドの後に比較演算子がありません。

48

メッセージ: **CASE には SELECT CASE が必要**

SELECT CASE コマンドが抜けています。

49

メッセージ: **LOOP には DO が必要**

DO…LOOP 命令文で DO コマンドが抜けています。

50

メッセージ: **EXIT DO は DO-LOOP 間でのみ使用可能**

DO LOOP コマンドペアの外側に EXIT DO コマンドがあります。

51

メッセージ: **DO には LOOP が必要**

DO…LOOP 命令文で LOOP コマンドがありません。

52

メッセージ: **UNTIL または WHILE が必要**

UNTIL または WHILE コマンドがありません。

53

メッセージ: **EXIT FOR は FOR-NEXT 間でのみ使用可能**

FOR NEXT コマンドの外側に EXIT FOR コマンドがあります。

54

メッセージ: **等号または AS が必要**

コマンドに等号または AS がありません。

55

メッセージ: **ラベル定義のエラー**

ラベル定義が間違っています。

56

メッセージ: **重複した OPTION BASE ステートメント**

OPTION BASE コマンドが複数あります。

57

メッセージ: **BASE が必要**

OPTION コマンドで BASE オプションがありません。

58

メッセージ: **0 または 1 が必要**

BASE オプションで 0 または 1 以外を使用しています。

59

メッセージ: **整数または定数が必要**

文字列中で文字の数を指定しましたが、TYPE 命令文では実際の整数値が用意されていません。

60

メッセージ: **AS には DIM, REDIM または TYPE が必要**

コード命令文で変数を定義するときに、AS コマンドに対する DIM, REDIM または TYPE コマンドが省略されています。

61

メッセージ: **TO には直前のステートメントが必要**

行中に他のコマンドのない TO コマンドがあります。

62

メッセージ: **TYPE が必要**

END　TYPE コマンドに対する TYPE コマンドがありません。

63

メッセージ: **TYPE にはメンバが必要**

TYPE コマンドによるユーザ定義タイプで、メンバを一つも定義していません。

64

メッセージ: **TYPE には END TYPE が必要**

TYPE コマンドでタイプする END TYPE コマンドがありません。

65

メッセージ: **END TYPE には TYPE が必要**

END TYPE コマンドで、対応する TYPE コマンドがありません。

66

メッセージ: **ELSEIF...THEN の後に行の終わりが必要**

行の終わりに改行がありません。

67

メッセージ: **数式の中からの変数や関数呼び出しは禁止**

定数が必要な場所に変数や関数呼び出しがあります。

68

メッセージ: **固定長文字列の範囲は 1-65536**

範囲より長い文字列長を定義しました。

69

メッセージ: **STEP 0 は使用不可**

ループ増分値として STEP 0 を使用しました。プログラムはループしません。

70

メッセージ: **GOSUB のない RETURN**

RETURN コマンドで対応する GOSUB がありません。

71

メッセージ: **ERROR が必要**

ON コマンドで対応する ON ERROR コマンドがありません。

72

メッセージ: **GOTO または RESUME が必要**

ON ERROR コマンドで GOTO または RESUME コマンドがありません。

73

メッセージ: **NEXT が必要**

FOR コマンドで対応する NEXT コマンドがありません。

74

メッセージ: **未定義のデータ型**

プログラムコードでデータ形を定義していません。

75

メッセージ: **エラーハンドラのない RESUME**

RESUME コマンドで対応する ON ERROR コマンドがありません。

76

メッセージ: **エラーのない RESUME**

RESUME コマンドでエラータイプの定義がありません。

77

メッセージ: **不正な変数名**

誤った変数名を定義しました。

78

メッセージ: **1 行でネストされた IF..NEXT はサポートしない**

同じ行に 2 つの IF…THEN コマンドがあります。

79

メッセージ: **識別子が長すぎる**

128 文字の上限より長い識別子を定義しようとしました。

80

メッセージ: **システム関数の不正使用**

関数で戻り値を使用していません。たとえば、IF…THEN 命令文や代入文として戻り値を使用しなければなりません。

81

メッセージ: **タイプサフィックスのない識別子が必要**

配列の参照で、配列要素を特定していません。

82

メッセージ: **SUB には END SUB が必要**

SUB コマンドで対応する END SUB コマンドがありません。

83

メッセージ: **FUNCTION には END FUNCTION が必要**

FUNCTIONコマンドで対応するEND FUNCTIONコマンドがありません。

84

メッセージ: **ネストされた手続きは使用不可**

同じ手続き内で 2 つまたはそれ以上の手続きを定義しました。

85

メッセージ: **手続きの外側でステートメントは使用不可**

手続きの外側に命令文があります。

86

メッセージ: **不正な END SUB / FUNCTION**

正しくコマンドを終了させていません。

87

メッセージ: **未定義のオブジェクトメソッド**

オブジェクトメソッドを正しく入力していません。

たとえば、 Form.load であるところを Form.loaf と入力しています。

88

メッセージ: **ラベルは手続きや関数の外側へ配置禁止**

手続きや関数の外側にラベルがあります。

89

メッセージ: **不正な手続き呼び出し**

サブルーチンや関数呼び出しに誤りがあります。サブルーチンや関数の定義を確認してください。

90

メッセージ: **REDIM での不正なリサイズ操作**

配列の開始数がオリジナルの変数と異なるか、配列の次数が一致していません。

91

メッセージ: **配列の次数は変更不可**

配列の次数を変更しようとしました。

92

メッセージ: **サブルーチン内での EXIT FUNCTION は不可**

サブルーチン内に EXIT FUNCTION があります。(EXIT SUB を使用してください)

93

メッセージ: **関数内での EXIT SUB は不可**

関数内で EXIT SUB を使用しました。(EXIT FUNCTION を使用してください)

94

メッセージ: **オーバーフロー**

代入値が変数の範囲を超えました。

95

メッセージ: **手続き内の不正なキーワード**

手続き内で TYPE または OPTION キーワードを使用しました。

96

メッセージ: **手続き外のコード**

手続きの外側にコードが書かれています。

97

メッセージ: **不正な属性値**

属性に対して間違った値を代入しようとしました。

98

メッセージ: **クオートされた文字列が終了していない**

リテラル文字列のクオートが閉じられていません。

99

メッセージ: **関数か変数が必要**

変数名を指定せずに間違ってメソッドを呼び出そうとしました。

100

メッセージ: **ソースデータベースのバージョンが不正**

ソースコードが、RAD 1.1 より前のバージョンで生成されています。

101

メッセージ: **関数かサブルーチンが空**

サブルーチンや関数の中にコードがありません。

102

メッセージ: **宣言内でのラベル使用は禁止**

コードの宣言セクション内にラベルがあります。

103

メッセージ: **操作を行うためのファイルがオープンされていない**

閉じられたか開かれていないファイルに対して操作を行いました。

104

メッセージ: **ファイルオープン失敗**

ファイルが存在しないか、壊れているか、使用中です。

105

メッセージ: **ファイルポインタの設定ができない**

ファイルの長さより大きい位置または負の位置にファイルポインタを設定しようとしました。

106

メッセージ: **ボックスアレイのインデックスが範囲を超えた**

ボックス要素の範囲外またはゼロ以下を参照しました。

107

メッセージ: **データベース列は NULL をサポートしない**

データベースカラムに NULL 値を入力しました。

データベースエラーメッセージ (500～509)

500

メッセージ: **データソース接続が確立していない**

プログラムで使用するデータベース接続が確立されていません。

501

メッセージ: **不正なカーソル状態**

閉じられたステートメント上でデータベース操作を行いました。

502

メッセージ: **ODBC エラー**

ODBC エラーが返されました。

503

メッセージ: **定義されていない列**

存在しない列にアクセスしようとしました。

504

メッセージ: **定義されていない属性**

存在しない属性にアクセスしようとしました。

505

メッセージ: **サポートしていないデータ型**

サポートしない型のデータをフィールドに代入しようとしました。

506

メッセージ: **接続に応答する前にステートメントは閉じていなければならない**

同じ接続でステートメントを 2 回開こうとしました。

507

メッセージ: **新しいデータソースに接続する前に接続は閉じていないければならない**

接続を閉じる前に新しいデータソースに接続しようとしました。

508

メッセージ: **実行時には属性は読み取り専用**

変更できないデータベースの属性を設定しようとしました。

509

メッセージ: **ODBC 警告**

ODBC 操作に問題があります。操作は完了しているでしょう。

FirstClass エラーメッセージ (1000～1018)

1000

メッセージ: **属性かメソッドが見つからない**

存在しない属性やメソッドを呼び出そうとしました。

1001

メッセージ: **アプリケーションが見つからない**

存在しないアプリケーションを実行しようとしました。

1002

メッセージ: **アプリケーションのエントリポイントが見つからない**

コードモジュールで Sub Main 手続きが抜けています。アプリケーションの開始位置がわかりません。

1003

メッセージ: **全ての ODBC データベース接続が使用済み**

ODBC ドライバの接続制限を越えました。接続を共有に設定してください。

1004

メッセージ: **一度にひとつのアプリケーションしか実行できない**

同じアプリケーションを複数実行させようとしました。このアプリケーションは 1 つのみ実行できるように設定されています。開発環境で、オプション > Project Attribute フォームで[クライアントセッションを無制限にする]にチェックを入れて再ビルドすることで設定を変更できます。

1005

メッセージ: **全てのアプリケーションセッションが使用中**

オプション > Project Attribute フォームで設定した最大アプリケーションセッション数を超えました。

1006

メッセージ: **フォーム ID 番号が不正**

フォーム ID 番号を 1 より小さい値に設定しようとしました。

1007

メッセージ: **不正なファイルフォーマット**

実行中の RAD バージョンで認識できない FCX ファイルをロードしようとしました。

たとえば、RAD2.0 用アプリケーションをそれより前の RAD で実行させようとした場合です。

1008

メッセージ: **ファイルから読み込みができない**

壊れた FCX ファイルをロードしようとしました。このエラーはアプリケーションのロード時に発生します。

1009

メッセージ: **ファイルへ書き込みができない**

壊れた FCX ファイルをロードしようとしました。このエラーはアプリケーションのビルド時に発生します。

1010

メッセージ: **ODBC データソースへの接続失敗**

機能しない接続を設定しています。たとえば、ユーザ ID やパスワードが不正です。

1011

メッセージ: **email メッセージの受取人が指定されていない**

Send 属性が呼び出された時に email の To 属性が設定されていません。

1012

メッセージ: **email メッセージの送信者が指定されていない**

Send 属性が呼び出された時に email の From 属性が設定されていません。

1013

メッセージ: **ライセンス制限: 最大同時アプリケーションセッション数に到達**

同時アプリケーションセッションライセンス制限を超えました。このエラーは 5.5Gold より前の FirstClass サーバおよびサーバライセンスの入っていないサーバで発生します。

1014

メッセージ: **ファイルコピーに失敗**

COPYFILE キーワードが失敗しました。たとえば、パスやディレクトリが正しくない場合です。

1015

メッセージ: **フィールドが拡張データベースリストではない**

正しくない型のフィールド要素を参照しようとしました。

1016

メッセージ: **このフィールドのための属性かメソッドが使用可能ではない**

指定したフィールド型でサポートしていない属性やメソッドを使用しようとしました。

索　引（言語のみ）

FirstClass RAD 2.6 日本語版　リファレンスマニュアル

2005 年 8 月 1 日発行　第 1 版

会社名	株式会社エフ・シー・マネジメント
住　所	〒101-0047 東京都千代田区内神田 2-14-3
電話番号	03-5209-5780（代表）
FAX 番号	03-5209-5781
URL	http://www.fcm.co.jp/