NEWTONE

インターラクティブチャートコンポーネント

TeeChart Pro 8J ActiveX

プログラミングガイド

目次

動作環境

以下の環境でご利用いただけます。

1) 対応プラットフォーム
Windows 98/Me/2000/XP(32bit)/Vista(32bit)/Server 2003(32bit)/Server 2008(32bit)

2) 対応開発コンテナ
Visual Basic 6.0、Visual C++ 6.0、IIS 4.0/5.0/6.0、Internet Explorer 6.0/5.0/4.0
Visual Studio .NET(C#.NET, VB.NET,VC++.NET)

お問い合わせについて

お問い合わせされる場合は、以下の項目をお知らせいただきますようお願いいたします。

ユーザ情報

1）ユーザ ID（ユーザ登録後に弊社より送付（送信）される書類（FAX やメール）に記載されている番号 UID-XXXXXX）
2）お名前（法人登録の場合は、会社名も併せてお知らせください。）
3）ユーザ登録している電話番号、FAX 番号もしくはメールアドレス
4）有償サポート契約を結ばれている方は契約番号

製品に関する情報

1）製品名
2）バージョン
3）製品のシリアル No.
4）OCX ファイルもしくは Cab ファイルのタイムスタンプ

開発環境に関する情報

1）OS
2）開発コンテナ
3）利用形態（ActiveX or Cab）

ご質問の内容

どのようにしたらその症状が発生するのか、あるいはどのような事柄が聞きたいのか、できるだけ簡潔にご説明ください。口頭や文章で説明するのが難しい場合は、再現性のあるプログラムをメールにて添付していただいた方がよりスムーズに解決する場合もございますので、その場合はプログラムを LZH もしくは ZIP 形式で圧縮していただき、お送りください。
また、製品とは関係のないプログラミング技法や WindowsAPI などに関するご質問にはお答えいたしかねますので、予めご了承ください。

問い合わせ先

「お使いになる前に」（パッケージ版は同梱の折込案内、ダウンロード版は別の PDF ファイル）のサポートについてをご覧ください。

注意

1 開発者ライセンスはユーザ登録された方がサポート対象となります。

インストール

●インストール方法

CD-ROM 内の「TeeChart8ActiveX.exe」を実行して（タスクバーのメニューから「ファイル名を指定して実行」など）、指示される説明に従ってインストールを進めてください。

●インストール後のフォルダの構成

TeeChart Pro 8J ActiveX のインストール後のフォルダ構成は以下の通りです。

■インストール時指定したフォルダ（デフォルトは¥Program Files¥Steema Software¥TeeChart Pro 8J ActiveXControl）

```
|          OCX ファイルなど
├────■Cab files
|          認証付 CAB ファイルなど
├────■Docs
|          | ヘルプファイルや PDF ファイルなど
|          ├────■DocImages
|          |
|          └────■Runtime Editor help source
|                     ユーザーズヘルプ関連ファイル
├────■Examples
|          サンプルプログラム
├────■Sample Data
|          サンプルプログラム用サンプルデータ
└────■Utilities
           アップデートプログラムなど
```

●インストール後の手順

TeeChart Pro 8J ActiveX のインストール後、開発を行うには以下の手順が必要です。

開発環境への組み込み

(1)Visual Basic 6.0 の場合

1.「プロジェクト(P)」メニューから「コンポーネント(O)」を選択します。
2.「コントロール」から[TeeChart Pro ActiveX Control v8]を選択して「OK」ボタンを押します。
3.ツールボックスに組み込んだコントロールのアイコンが表示されます。

(2)Visual C++ 6.0 の場合

1.「プロジェクト(P)」メニューから「プロジェクトへ追加(A)」、「コンポーネントおよびコントロール(P)」を選択します。
2.「コンポーネントおよびコントロールギャラリ」から「Registered ActiveX Controls」を選択します。
3.「TeeChart *** v8」を選択し、「挿入(S)」ボタンを押します。（***はコントロール名を指します。）

(3)Visual Basic.NET,Visual C#.NET,Visual C++.NET の場合

・バージョン 2002

1.「ツール(T)」メニューから「ツールボックスのカスタマイズ(X)」を選択します。
2.「COM コンポーネント」タブから[TeeChart *** v8]を選択して「OK」ボタンを押します。（***はコントロール名を指します。）

・バージョン 2003

1.「ツール(T)」メニューから「ツールボックス アイテムの追加と削除(X)」を選択します。
2.「COM コンポーネント」タブから[TeeChart *** v8]を選択して「OK」ボタンを押します。（***はコントロール名を指します。）

・バージョン 2005/2008

1.「ツール(T)」メニューから「ツールボックス アイテム選択(X)」」を選択します。
2.「COM コンポーネント」タブから[TeeChart *** v8]を選択して「OK」ボタンを押します。（***はコントロール名を指します。）

※本コントロールをインストールすると、レジストリへは自動的に登録されますが、その後手動で行いたい場合は以下の解説をお読みください。

●レジストリの登録及び解除について

手動で OCX をレジストリに登録したり解除したりする場合は下記のように行います。

（登録）「Regsvr32 “path¥TeeChart8.ocx”」

（解除）「Regsvr32 /U “path¥TeeChart8.ocx”」

※<<path>は OCX が存在するパスを指します。

●弊社 HP からアップデートモジュールをダウンロードして OCX ファイルを更新する場合

1.ダウンロードした OCX ファイルを既存の OCX ファイルに上書きコピーします。

2.手動で OCX ファイルをレジストリへ再登録します。（上記を参照）

TeeChart ユーザーズガイド

TeeChart ユーザーズガイドは、背景やよく使用される深さや高度なチャートデザイン技術をご利用できます。

1. 1. はじめに

TeeChart Pro 8J ActiveX は、すばらしい新機能を多数含んでいます。

チャートやデータ系列型の両方が強化され拡張されています。多くの系列型は拡張され、3D 泡系列、横ヒストグラム、極バー、滑らかな極、などのような新しい系列型を含んでいます。モード、メディアン、相関、分散、境界などの新しい関数型も拡張され、軸には拡張ラベル、奥軸(上)、等距離の軸、ベベル軸、日付書式軸、最初と最後のラベルなどの新しい機能が付加されました。

なお、異なるチャートの至る所に異なる系列を混合および適合する柔軟性も備えています。チャートの至る所でデータ系列のグループに新しい関数定義を適用できます。

数種類の新しいエクスポートフォーマットのサポートが今回のバージョンで付加されました。全てのエクスポートは、ファイルあるいは受け入れ側(例:Web ページ)に直接ストリームとして作成できます。
チャート描画の強化のために、グリッドバンド、3D チャートの置換、系列アニメーションなどの新しいツールインタフェースが付加されました。

2．1．はじめる前に

TeeChart Pro 8J をはじめてご使用になる方は、このトピックを読めばすぐに TeeChart Pro 8J の内容をご理解いただけます。何もない状態からチャートを作成する場合、TeeChart を使用すると非常に迅速に作成することができます。

2．2．TeeChartオンラインヘルプの使用

TeeChart コンポーネントの選択
TeeChart コンポーネントを使用してプロジェクトを設計する場合、ツールボックスからあるいはコンポーネントを置いたフォームでコンポーネントを選択し〔F1〕を押してください。そのコンポーネントに関連した状況感知型のヘルプ情報が表示されます。
TeeChart プロパティ、メソッド、イベントの選択
プロパティウィンドウでプロパティあるいはイベントを強調表示し、〔F1〕を押すとその項目に関するヘルプ情報が表示されます。また、ヘルプ情報の必要な単語をコードの中に書いてその単語の上にカーソルを置きクリックしてその単語を選択してから、〔F1〕を押してもヘルプは表示されます。
ヘルプの目次のページ
TeeChart ヘルプの目次ページに、TeeChart 専用のユーザーズガイドがあります。この見出しをたどって、目的のリファレンスを参照することができます。

2．3．TeeChartのコンポーネント

TeeChart には次の重要なコンポーネントがあります。

TChart

この **TChart** は、全てのチャートを作成するための基本的なツールです。
TeeChart Pro 8J は同時に次のコンポーネントも提供しています。

実行時にチャートのコントロールバーを提供します。

TeeCommander

実行時にチャート系列の ListBox を提供します。

TeeListbox

実行時にユーザがチャートエディタを使用することを可能にします。

TeeEditor

実行時にユーザが TeeChart 印刷プレビューを表示することを可能にします。

TeePreviewer

ページ分けされたチャートをページングするためのナビゲーターボタンバーです。

ChartPageNavigator

系列あるいはチャートコンポーネントへ XML データを渡します。

SeriesXMLSource

グリッド内でナビゲートするために使用します。

ChartGridNavigator

TChart に描画されたデータのグリッドデータ表現を提供します。

ChartGrid

テキストソースからデータの操作と関連したプロパティやメソッドにアクセスします。

SeriesTextSource

自動的にデータベースのデータから系列で系列を含むチャートを作成します。

CrossTabSource

チャートの WYSIWYG の印刷可能な出力を表示するための印刷プレビューパネルです。

TeePreviewPanel

標準のチャートエディタを格納するパネルです。

ChartEditorPanel

このコンポーネントは、実行時に TeeChart 機能の拡張を実現させます。

2. 4. 新しいチャートの作成

このセクションは、新しいチャートを作成するための手順を説明します。

新しいフォームを作成

新しいフォームを作成し、TeeChart のコンポーネントをそこに置いてください。TeeChart のコンポーネントの隅をドラッグしてチャートの大きさを決めてください。後で必要に応じて、チャートの大きさを調整できます。

新しいチャートを編集

新しいチャートの上にマウスポインタを置いて、右のマウスボタンをクリックしてください。チャートの編集オプションを含んでいるメニューが表示されます。

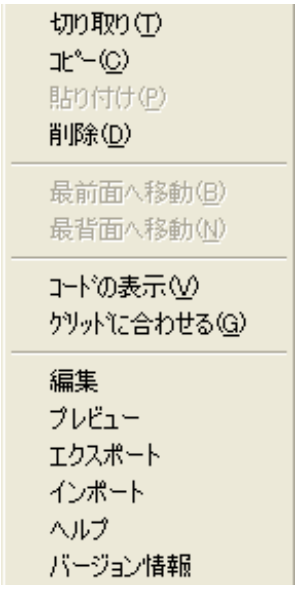
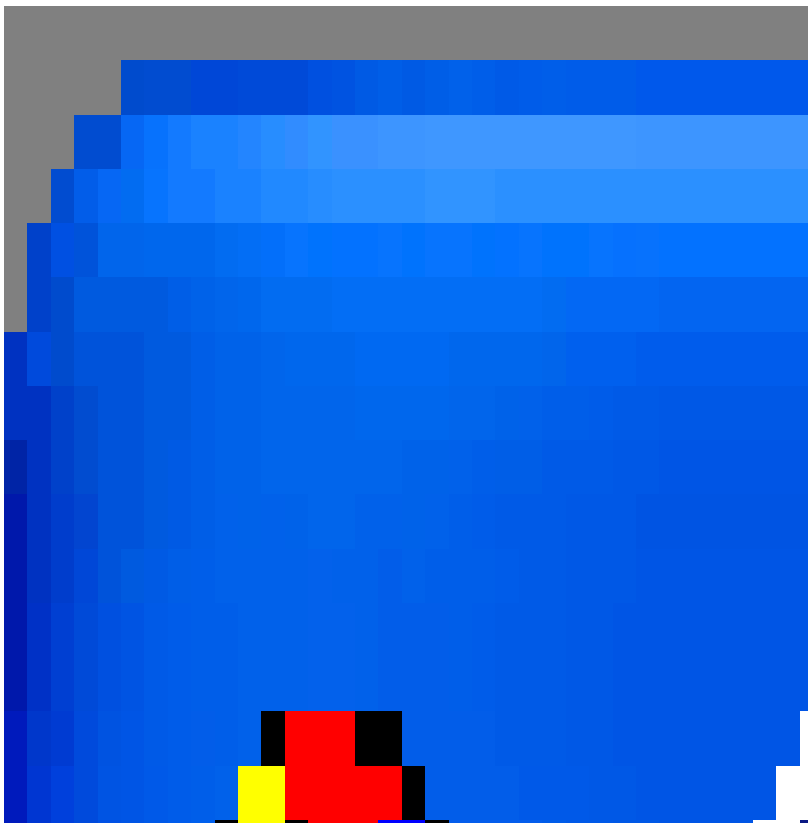


図 1. チャートの上でマウスの右ボタンをクリックするとチャートの編集オプションメニューが表示されます。

チャートの編集および系列データをチャートに設定する場合、**編集**を選択してください。
次のチャートエディタに進んでください。

2. 5. チャートエディタ

図 2.
チャートエディタ
画面



チャートエディタのチャートページ(第 1 ページ)には、チャートの定義に関する情報が入っています。一般的なチャートパラメータから特殊なチャートパラメータまでを定義する様々なセクションがあります。一部のパラメータはチャートに特定のデータ系列を定義するまで適用されません。たとえば、タイトルなどパラメータを変更してみてください。チャート上でリアルタイムで更新されるのがわかるはずです。
チャートを表示するには、データ系列を作成する必要があります。

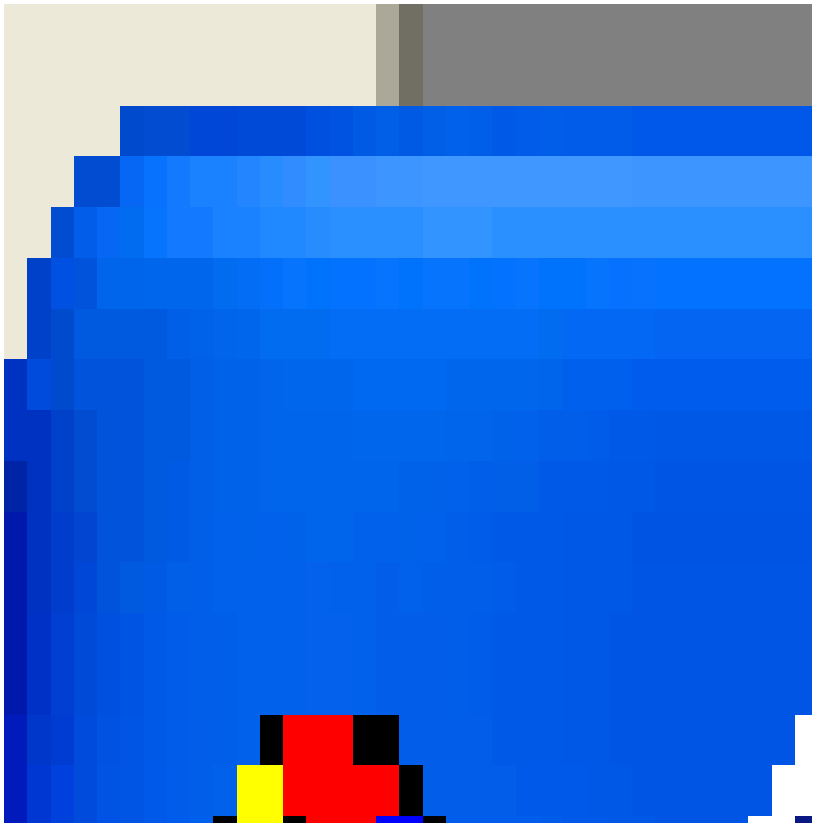
データ系列の追加

チャートページの[系列]タブセクションで[追加]ボタンを押してください。TeeChart が系列型のギャラリを表示します。どれか 1 つを選んでチャートに追加してください。データの表示の仕方を変更する場合は、この系列型を後で変更することができます。

図 3.
チャートギャラリ

マウスもしくは矢印キーで系列型を選択し、[OK]を押します。系列型をダブルクリックしても、同じ結果になります。
系列がチャートに追加されます。チャートエディタでは、新しい系列用の新しい機能タブが表示されます。
リボン系列を選択すると、エディタが以下のように表示されます。

図 4.
チャートに追加された新しい系列



いくつかのランダムな値が追加されているので、系列の外観を設計時にすぐにチャートで見ることができ、どんな変更でも簡単に理

解することができます。
プロジェクトをコンパイルします。プロジェクトは空のチャートを表示するようコンパイルします。ランダム値は、実行時には動作しませんので、次のステップでチャートエディタに戻って、データソースを追加するか、あるいは独自のコードを記述してデータ値を追加してください。

系列の編集

[系列]タブを選択すると、系列を編集することができます。次に、データを系列に追加します。以下の 2 つのトピックを参照してください。

TChart でのデータ系列の設定
TChart での ODBC/ADO データ系列の設定

2．6．TChartでのデータ系列の設定

フォーム上の TChart に系列を追加しました。これで系列を登録する準備は整いました。ポイント値をプログラムで追加します。データベース対応のチャートの作成方法については、後で説明します。

円系列の例

追加した系列が円系列の場合、以下の方法で系列を操作することができます。以下のコードを動作させるためには、系列の名前をデフォルトの Series1 のままにしておきます。

フォーム上にコマンドボタンを置き、Click イベントに進みます。

以下のコードを Command1_Click イベントにコピーします。

例 Visual Basic

```
With TChart1.Series(0)
    .Add 40, "Pencil", vbRed
    .Add 60, "Paper", vbBlue
    .Add 30, "Ribbon", vbGreen
End With
```

TeeChart のヘルプから、設計時に Add メソッドやその他の利用可能なメソッドとプロパティに関する説明を参照することができます。コードの中で Add という単語のところにカーソルを持っていき、〔F1〕を押すと、そのメソッドに関する説明が表示されます。

コードに戻り、プロジェクトを実行してください。

コマンドボタンを押すと、円グラフが表示されます。コードが実行されたのです!

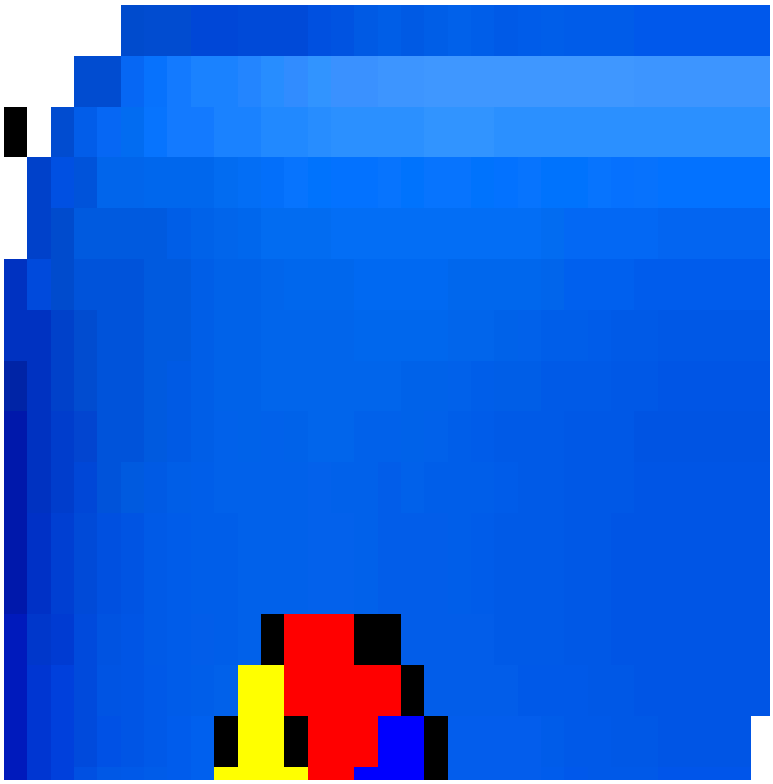
系列の編集

プログラムを閉じて、開発環境の IDE に戻ります。

設計モードでは、すべてのチャートと系列プロパティには、チャートを右クリックして[チャートの編集]オプションを選択するか、または編集するチャートをダブルクリックしてアクセスすることができます。ここで、チャートエディタを使用して、新しいチャートと系列を編集します。図.5 は最初のエディタ画面です。

系列のプロパティを編集するには、リスト内の系列名をダブルクリックするか、あるいは系列を強調表示にして[変更]を選択するか、または直接[系列]タブを選択してください。いずれの方法でも系列のエディタに移動できます。

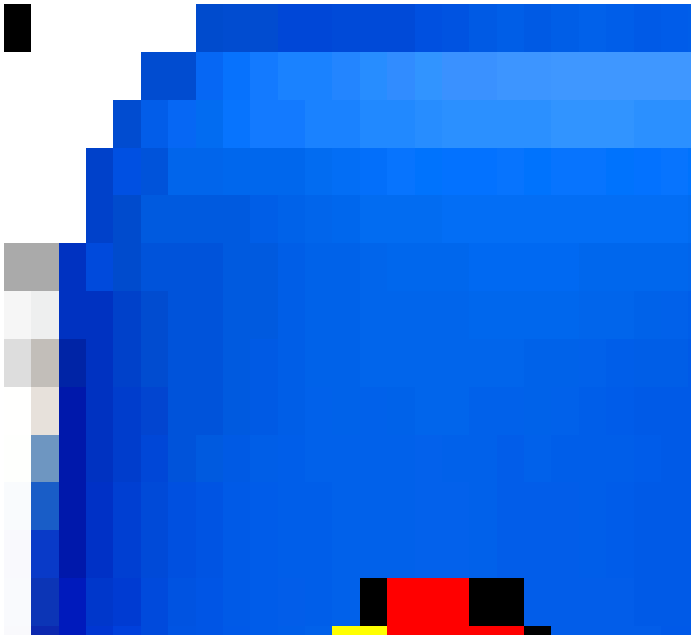
図 5.
新しい円系列を表示しているチャートエディタ画面



円系列のプロパティをいくつか変更してみてください。変更の結果がチャートに自動的に反映されているのがご理解頂けます。変更を無効にするキャンセルはありませんが、ほとんどのパラメータは簡単に切り替えることができます。設計時には、TChart コンポ

ーネントは、まだコードを実行していないため、ランダムに生成したデータのセットを表示しているということを思い出してください。プロジェクトを開始して、ユーザデータで新しいパラメータがどのように表示されるかを確認してください。

図 6.
円系列の［系列］ページ



エディタでは多くのことを、目で確認しながら行うことができます。プロパティウィンドウでパラメータをいくつか変更したり、独自のコードを使用してプロパティを変更してみてください。さらに高度なプロジェクトについては、コードを入力して試してみてください。
TChart にデータを追加する別の参考例について、ヘルプの**コードで値を挿入してチャートを作成**を参照してください。
では、ADO（ODBC による）データ系列に関するデータソースの設定について、もっと詳しく見ていきましょう。

TChart での ODBC/ADO データ系列の設定

２．７．TChartでのODBC/ADOデータ系列の設定

データベースの接続を使用する前に Microsoft の ADO コンポーネント Ver.1 以上をインストールする必要があります。

TeeChart の ODBC デモデータベース

サンプルで ODBC データソースとしてデモデータベーステーブルが DBF 形式で含まれています。**ODBC DSN（TeeChart Pro Database）**は、TeeChart がインストールされた場合に、自動的にシステムに追加されます。インターネットインフォメーションサーバ(IIS)や **Active Server Pages** でテストをしたい場合、TeeChart Pro System db を呼び出すテキストのために **TeeChart Pro System db** という System DSN を作成します。その他は、データソースの **SystemDSN** を作成してください。VB や VC++の TeeChart のサンプルは、ADO レコードセットとしてデータソースにアクセスします。

データベース対応チャート作成手順例
- チャートエディタで円系列を選択してください。
- チャートエディタの**系列ページ**を開いてください。
- 系列リストボックスから系列を選択し、データソースタブを開きます。
- リストボックスからデータソースタイプを選択します。

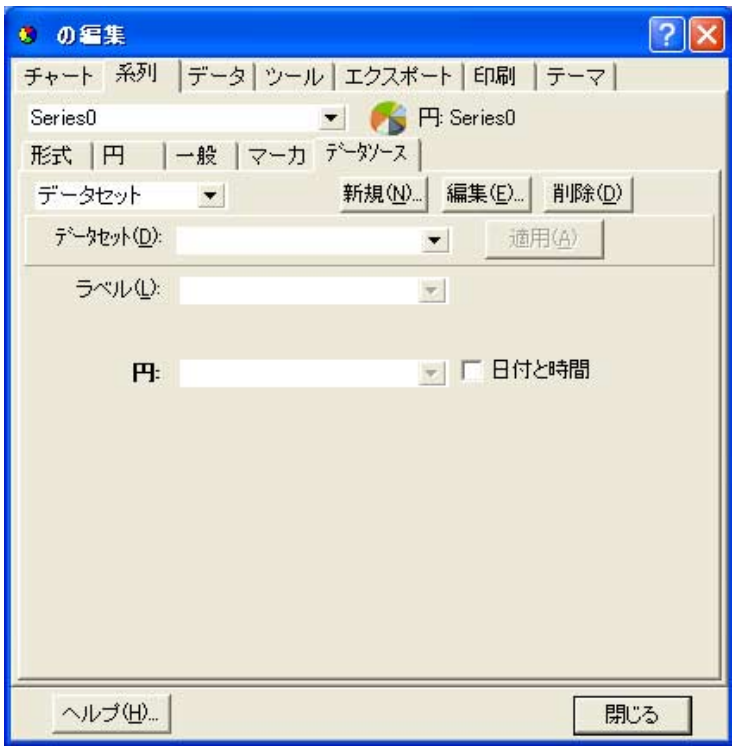
利用可能なデータソースは 8 種類あります。データソーススタイルは、次のうちの一つに該当するコンポーネントプロパティです。
1.「手動」プログラムでポイントを追加する。
2.「乱数」ランダムな値でチャート系列を描画する。
3.「関数」は 1 つ以上の系列の組み合わせと関数のアルゴリズム（最小、最大、平均など）で動作します。
4. CrossTab
5.サマリー
6.単一レコード
7.「データベース」は有効な ADO データソースで動作します。
8.XML ファイル

データセットの追加

上記の中から 7 番目のオプションである「データセット」を選択します。TChart には DSN のサンプルとして TeeChart Pro に含まれている dbf 形式の ADO (ODBC) DSN サンプル（**TeeChart Pro Database**）からのデータが含まれます。

1.フォーム上に TChart コントロールを配置します。
2.チャートエディタを表示するには、TChart コントロールを右マウスボタンでクリックし、「編集」を選択します。
3.チャートコントロールに系列を追加します（たとえば棒系列）。
4.［系列］タブの［データソース］タブをクリックします。
5.コンボボックスをクリックし、「データセット」を選択します。画面イメージは（**図.7**）のようになります。

図 7.
チャートエディタのデータソースタブ



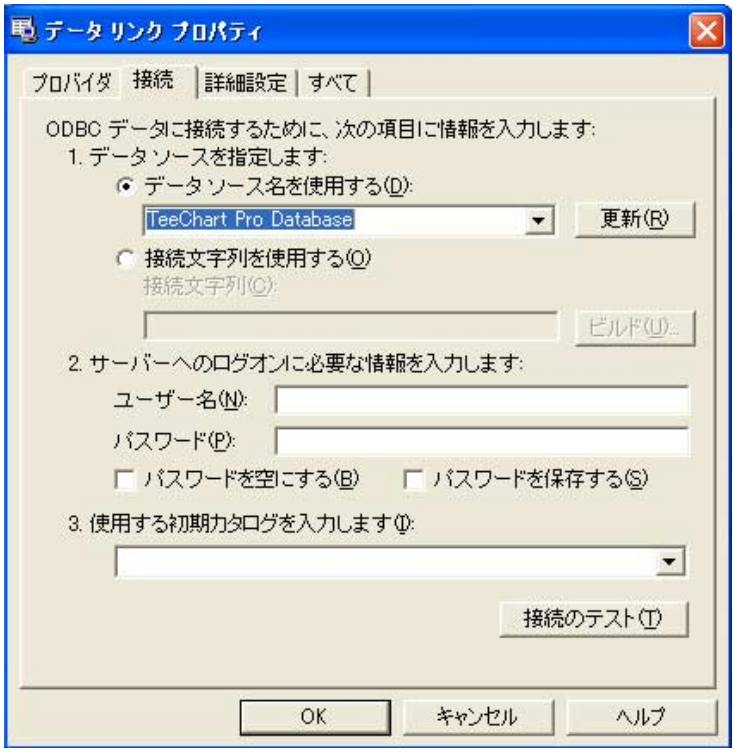
6.ADO 接続ウィンドウを開くために**新規**ボタンを選択する必要があります。

図 8.
TeeChart ADO 接続ウィンドウ

ここから、データリンク(UDL)ファイルの選択、あるいは独自の接続文字列の構築が行えます。この例では、[プロバイダ]タブからデフォルトの「Microsoft OLE DB Provider for ODBC Drivers」を選択し、[接続]タブから「TeeChart Pro Database」を選択します:

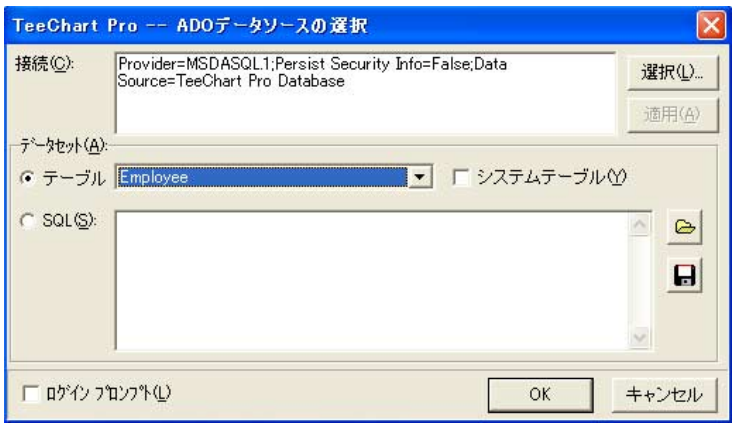
図 9.
OLE DB プロバイダ

図.10
接続ページ



7.上記の画面上の「OK」を選択した後に、下記のイメージで描画されている「Employee」テーブルを選択してください。ここでは、そのテーブルを選択する代わりに、SQL クエリの作成、あるいはそれを含むテキストファイルの取り込みもできます。

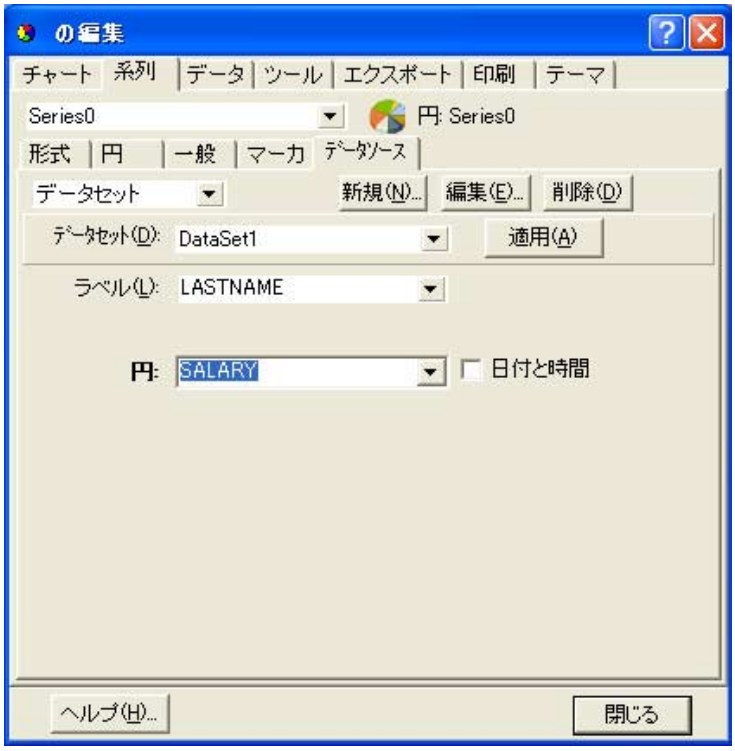
図.11
テーブル/クエリの選択



8.「OK」をクリックして ADO データソースダイアログを閉じ、チャートエディタに戻ってください。
ここで、テーブルあるいは SQL クエリからどのフィールドを取得して、系列に追加するかを設定しなければなりません。これは、各系列の項目に適用するフィールドをチャートエディタを使用して選択/設定できます。

9.下記のイメージを参照してください。チャートエディタの[データソース]タブには、例として選択されたフィールドが表示されています。

図.12
フィールドの選択



各系列は、「ラベル」フィールド(String 型あるいは Char 型)と、Integer 型または Float 型または Datetime 型の値フィールド(縦棒系列は棒フィールド)を持っています。

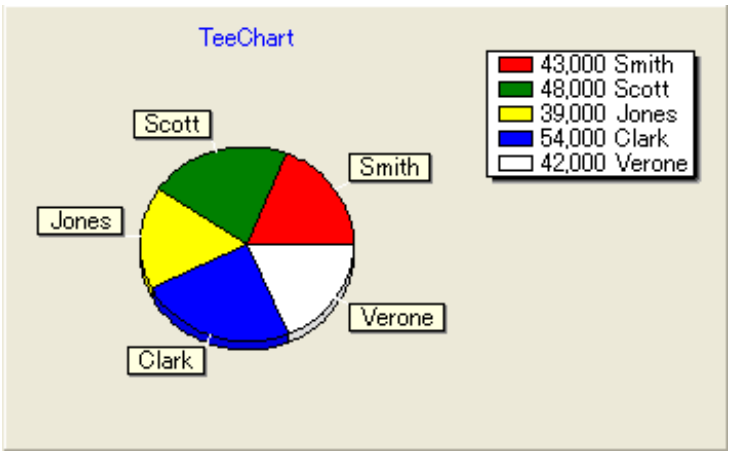
このサンプルでは、系列ラベルには「Employee」テーブルから「LASTNAME」フィールドを選択し、ポイント値には「SALARY」を選択しました。
上記のイメージの「データセット」コンボボックスを参照してください。このコンボボックスは、使用可能なテーブルあるいは作成されたクエリを表示します。このダイアログからいつでも、異なるデータセットの選択、あるいは「編集」ボタンをクリックしてデータセットの編集が行えます。
これで、適切なフィールドを割り当てたので、結果を見てみましょう。

10.「閉じる」ボタンあるいは「適用」ボタンをクリックしてチャートエディタを閉じ、開発環境に戻ってください。

これで、TChart コントロールは、選択された ADO データベースのテーブルあるいはクエリを開き、レコードの取得を開始し、選択された系列に値を追加します。

この例では、デザイン時に以下のようなチャートになります:

図.13
出力



プロジェクトを実行すると、同じチャートがご覧になれます。

2．8．チャートデータの変更

チャートデータは、各系列ポイントの Point Valueindex を使用してアクセスできます。

ポイントの削除:
Series の Delete メソッドを使います。

```
’これは、最初の系列から 5 番目のポイントを削除します。
TChart1.Series(0).Delete(5)
```

ポイントの修正:
Series の Value プロパティを使用します。

```
TChart1.Series(0).YValues.Value(3) = 27.1
’泡系列の場合
TChart1.Series(0).asBubble.RadiusValues.Value(8) = 8.1
’円系列の場合
TChart1.Series(0).asPie.PieValues.Value(3) = 111
```

特定なデータ関連のメソッドについては、ヘルプの各系列型を参照してください。

DateTime ポイント座標:
TeeChart 系列は DateTime 値を利用できます。

```
‘ 最初に、X と(あるいは)Y 値リストで、DateTime プロパティを True に設定してください。
TChart1.Series(0).XValues.DateTime = True
‘ 次に、値には Date あるいは Time あるいは DateTime のいずれかを設定してください。
TChart1.Series(0).AddXY DateValue("2000/08/25"), 25.4, "Barcelona", vbGreen
```

3．1．TeeChartのデモ

TeeChart ユーザーズガイドの「はじめる前に」のセクションには、TeeChart の最も基本的な手順についての説明があります。もう 1 つ参考にして頂きたいのは、豊富なサンプルプログラムです。

TeeChart のサンプルプログラムのフォルダのある場所

TeeChart をインストールしたフォルダの下の Examples フォルダを開いてください。このフォルダには様々な開発環境のための多くのサンプルが入っています。各サンプルにはそれぞれ特有の TeeChart の機能や系列型が用意されています。多くのフォームがイベントの操作や対話型のチャートを示すためのソースコードを含んでいます。

Visual Basic 5 & 6 フォルダで Scrolling という名前のフォルダがあります。Visual Basic で、ScrollProject1.vbp を開いてください。

Form Load イベントをクリックすると、FillDemoPoints を呼び出します。以下のコードはサンプルコードにあります。

```
Private Sub Form_Load()
    FillDemoPoints
End Sub

Private Sub FillDemoPoints()
Dim t As Integer

    With TChart1.Series(0)
        ' ランダムデータでリボン系列を描画します。
        .Clear     '<-- これは、系列(Series(0))から全てのポイントを削除します。
        ' 12:00 から 12:59 の 60 分間の時間を追加してみましょう。
        For t = 0 To 59
            .AddXY TimeValue("12:" & t & ":00"), Rnd(100), "", vbRed
        Next t
        ' 13:00 から 13:59 の 60 分間の時間も更に追加してみましょう。
        For t = 0 To 59
            .AddXY TimeValue("13:" & t & ":00"), Rnd(100), "", vbRed
        Next t
    End With
End Sub
```

このコードは、Series(0)をクリアし、0 から 100 までの 2 時間のランダム値をプロットします。

これは Form_Load イベントに置かれているため、フォームが描画される度にこのコードが実行されます。

Command1 コンポーネントをクリックして、これに関連する Click コードを参照してください。

```
Private Sub Command1_Click()
Dim h, m, s As Integer
  m = Minute(TChart1.Series(0).XValues.Last)
  h = Hour(TChart1.Series(0).XValues.Last)
  ' 新しいランダムポイントを系列に（更に 1 分間）追加します。
  s = 0
  m = m + 1
  If m = 60 Then
    m = 0
    h = h + 1
  End If
  TChart1.Series(0).AddXY (TimeValue(h & ":" & m & ":" & s)), Rnd(100), "", vbGreen
```

次のソースコードは、サンプルを目的としたものです。

新しいポイントが最後に追加される場合、水平軸のスクロールの簡単な方法は、Command1_Click イベントで以下のコードを入力します。

このイベントは新しいポイントが系列に追加されるたびに呼び出されます。

```
Private Sub Command1_Click()
    ' データの最後の時間を描画するために軸をスクロールします。
    ScrollHorizAxis
End Sub
```

この例では、最後のポイントの 55 分前、および最後のポイントの 5 分後を描画するように、下の水平チャート軸をスケーリングします。

```
Private Sub ScrollHorizAxis()
    With TChart1.Axis.Bottom      '<-- 水平スクロールの定義
        .Automatic = False        '<-- 自動スケーリングは不要

        ' この例では、最後のポイント時間に 5 分を加えた時間に 1 時間のデータを示すように
        ' 軸の最小値および最大値を設定します。
        .Maximum = TChart1.Series(0).XValues.Maximum + _
                                        TimeValue("0:5:0")
        .Minimum = .Maximum - TimeValue("1:0:0")
    End With
End Sub
```

このコードは、1 分経過する毎に最後に追加したポイントの時間を増やします。
増やされた Time 値は、系列に追加されます。このようにして、新しいポイントが生成されます。
Axis（軸）プロパティをコードで設定することにより、いろいろなことが実行できます。

各チャートコンポーネントには 5 つの軸、つまり左、上、右、下、深さがあります。各軸はデフォルトで True に設定されている論理「Automatic」プロパティを持っています。これは、TeeChart が各軸に対して Minimum（最小）および Maximum（最大）の値を常に計算することになります。

Minimum および Maximum プロパティは、Double 型になっています。これにより、浮動小数値あるいは、「Date」、「Time」、「Now」あるいはその他の有効な日付／時間値のような DateTime 表示ができます。

次に、このプロジェクトをコンパイルして実行し、各サンプルフォームを見てみましょう。

これらは、TeeChart の機能について学びながら、プロジェクトに適用できるようなアイデアを提供してくれるでしょう。

TEECHART8.HLP ファイルでは、サンプルソースコードの例をいくつか示しながら、各コンポーネントについて詳細に説明しています。

異なるサンプルフォームを使用して、新しい最小限のプロジェクトを作成し、実際のデータで TeeChart を実験し確認されることをお勧めします。

4．1．TChartコンポーネント

TChart コンポーネントは、全てのチャートの基本となっています。開発環境のツールボックスで TChart コンポーネントを選択し、フォーム上にドロップするだけで、アプリケーションにチャートを挿入することができます。
クラス定義については、TChart クラスを参照してください。
チャートエディタを使用して、チャートの表示特性の定義や、新しいデータ系列を追加することができます。チャートのプロパティとメソッドを使用して、実行時に全ての特性を設定、変更することもできます。
その場合は、TChart コンポーネントのデータ系列の設定にはコードが必要です。まず最初に、**TChart でのデータ系列の設定**を参照してください。

4．2．TChart系列

TChart の系列（Series）クラスは、すべての系列型の最上位クラスとなっています。使用している系列型のプロパティを参照する際には、その系列に含まれるプロパティに加えて、TChart の系列（Series）クラスのプロパティを確認し、すべての系列に共通なプロパティも参照してください。

リボンと折れ線
横リボン
縦棒
横棒
面
散布図
円
矢印
泡
ガント
シェープ
系列の組み合わせ

ベジェ
ボックスプロット
カレンダー
キャンドル
カラーグリッド
等高線
ドーナツ
エラー
エラーバー
ファネル
High-Low
ヒストグラム
マップ
3D 散布
極
ピラミッド
レーダー
スミス
サーフェス
サーフェス(三角)
ボリューム
ウォーターフォール

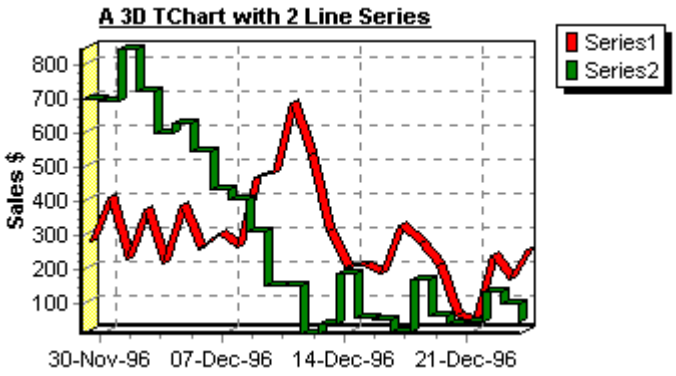
4. 2. 1. リボンと折れ線

リボンおよび折れ線の 2 つの線系列が用意されています(Y置換リボンのための横リボンも参照してください)。折れ線は高速に動作します。新しいポイントを系列に追加する際のスピードを上げるために費用がかかったために、リボン系列にある一部のプロパティが折れ線にはないと言う点でリボンとは異なります。これらの違いに関する説明については、折れ線についてのトピックを参照してください。

リボン

プロパティとメソッドの一覧については、ヘルプの**リボン系列**を参照してください。

図. 1
Stairs プロパティを True に設定した系列を示す 3D のリボン系列。Stairs は反転することができます。



折れ線

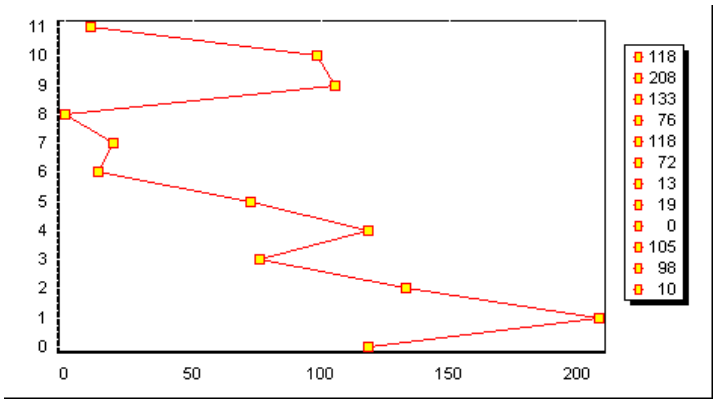
プロパティとメソッドの一覧については、ヘルプの**折れ線系列**を参照してください。

この折れ線系列は、2D でしか描画することができませんが、描画速度は非常に高速です(但し、ハードウェアによって異なります)。この系列型は本来大量に処理が要求されるアプリケーションや金融アプリケーションを処理するためのものですが、非常に多くのポイントを持ついかなるデータセットにも適しています。
スピードを高めるテクニックについて VB 5 & 6 の **FastLine AddRealTime** サンプルを参照してください。

4. 2. 2. 横リボン

プロパティとメソッドの一覧については、**横リボン系列**を参照してください。
リボン系列は、X 軸を使用して描画しますが、横リボン系列では Y 軸を使用してポイントを描画します(つまり、リボン系列を縦に描画します)。

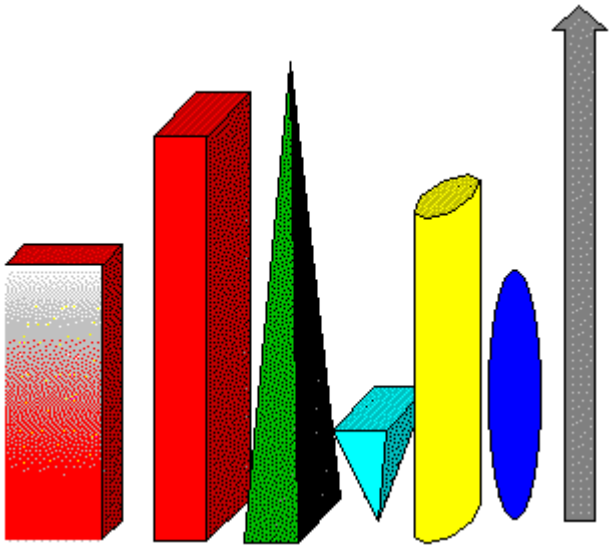
図. 2
2D の横リボン系列



4. 2. 3. 縦棒

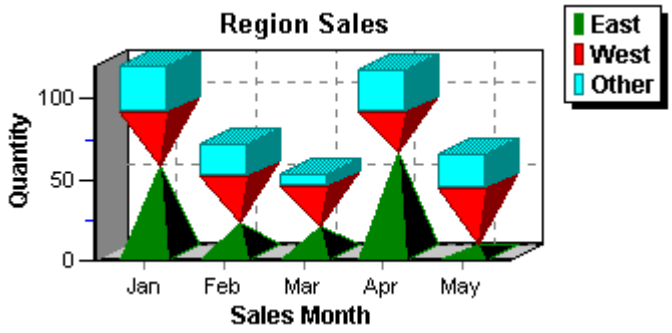
プロパティとメソッドの一覧については、ヘルプの**縦棒系列**を参照してください。
2D および 3D の縦棒系列は、長方形以外の縦棒で表すこともできます。

図. 3
チャート系列用にバースタイルを選択するか、あるいは必要に合わせて「混合と適合」を行ってください。



縦棒系列の環境設定の例
縦棒系列のスタイルを混合すると、アプリケーションによっては便利なことがあります。以下は積み重ね棒チャートの例です。

図. 4
バータイプを混合した例



このサンプルチャートを作成し、データと共にこれを登録する手順は以下のとおりです。
1.チャートコンポーネントをフォーム上にドラッグし、目的とするサイズまでドラッグします。
2.チャート上でマウスの右ボタンをクリックし、メニューで「編集」を選択します。
3 つの**縦棒系列**を追加し、最初のエディタ画面で[**タイトル**]を使用して、これらの系列に別々のタイトルを付けます(この例では、タイトルはそれぞれ「East」、「West」、「Other」になっています)。
3 つの系列のいずれか 1 つを[**系列**]タブの[**積み重ね**]ページで[**複合**]を[**積み重ね**]に設定します(この変更は 3 つの系列すべてに適用されます)。
エディタの[**チャート**]タブの[**タイトル**]入力タブを使用して、タイトルを追加します。
[軸]タブに移動します。[左軸]を選択し、タイトルを追加します。軸が自動でサイズを定義できるようにしましたが、**副目盛**は、下の軸には表示していません。
各系列は、そのバー用に定義した異なる形式を持っています。各縦棒系列の環境設定のページで、個々にこの形式を定義します。
最初のタブである[**形式**]には、[**スタイル**]のオプションがあります。
以下のようにスタイルを指定することができます。

```
With TChart1
  .Series(0).asBar.BarStyle = bsPyramid   'East
  .Series(1).asBar.BarStyle = bsInvPyramid 'West
  .Series(2).asBar.BarStyle = bsRectangle 'Other
End With
```

TeeChart オンラインヘルプで BarStyle を検索し、オプションの一覧を表示してください。
このチャートの場合、正しい結果を得るには**系列順序**を制御する必要があります。これは、設計時にチャートエディタで、[チャート]ページの系列リストで系列を選択し、順序を変更する矢印を使用することで行えます。実行時には、同様のことが以下のコードで行えます。
描画の順序は、常に系列が ParentChart に割り当てられている順序に基づきます。
順序は、実行時に ExchangeSeries メソッドによって、変更することができます。

```
TChart1.ExchangeSeries 0, 1
' Series(0)と Series(1)の Z Order を交換します。
' 1 が 0 になり、0 が 1 になります。
```

このサンプルに、ランダム値を追加しました。フォーム作成時にコードを実行することも、あるいはフォームにボタンを置いてコードを実行することもできます。

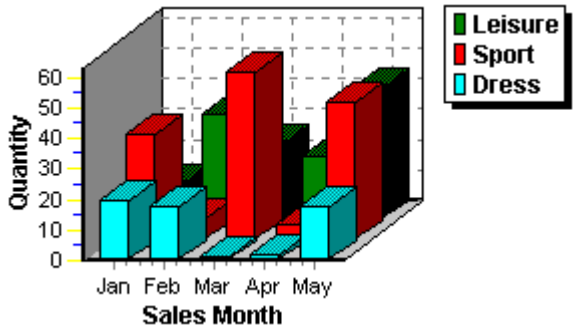
```
Private Sub Command1_Click()
Dim t As Integer
For t = 1 To 12
   TChart1.Series(0).Add Rnd(70), Format("1," & t & ",2008", "mmm"), vbRed
   TChart1.Series(1).Add Rnd(70), Format("1," & t & ",2008", "mmm"), vbYellow
   TChart1.Series(2).Add Rnd(30), Format("1," & t & ",2008", "mmm"), vbGreen
Next t
End Sub
```

12 ヶ月分のデータを登録しましたが、5 ヶ月分しか表示されていません。エディタの[チャート]ページにある[ページ]タブに、1 ページのポイント数の定義があります。

縦棒系列の表示

縦棒系列のうちの 1 つの[系列]ページの[積み重ね]タブで設定されている[**複合**]パラメータでチャートの各縦棒系列のバー表示の揃え方を設定します。各縦棒系列でパラメータを変更する必要はありません。
以下の図は、バー系列を表示するそれぞれ異なる書式を示しています。各チャートは、同じ情報を異なる表示方法で示しています。
百分率系列の表示では、実際の値は表示されませんが、系列の各要素の合計 100%に対する相対値が表示されます。

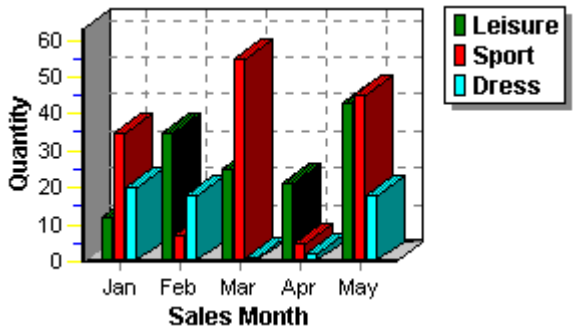
図. 5. a
同じ情報を表示する 4 つの方法を示す 3D の縦棒系列



複合
プロパティ

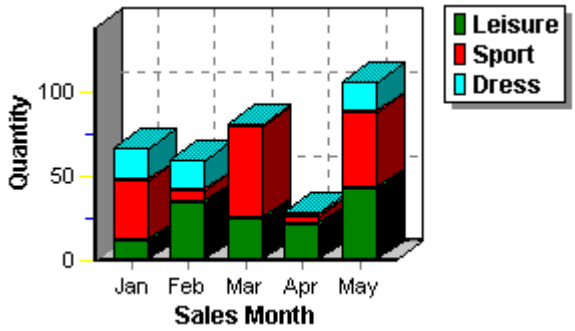
なし

図. 5. b



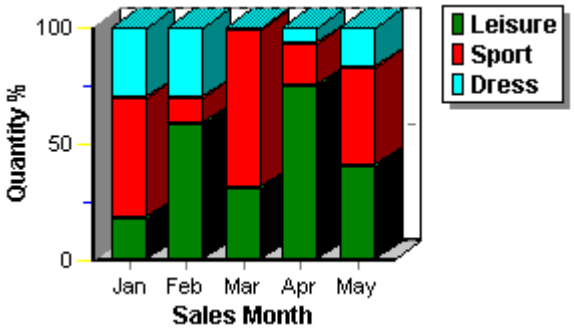
併置

図. 5. c



積み重ね

図. 5. d

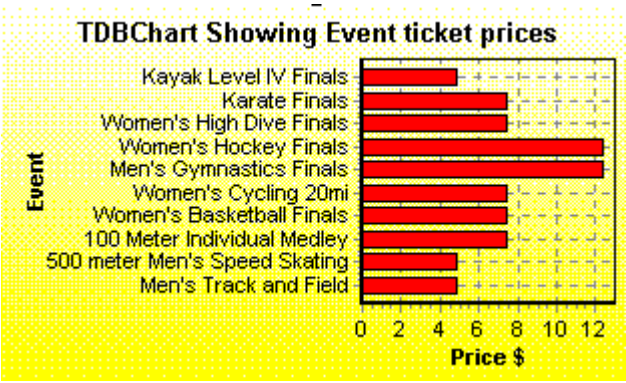


百分率

4. 2. 4. 横棒

プロパティとメソッドの一覧については、ヘルプの**横棒系列**を参照してください。
横棒系列は、縦棒系列と同じプロパティを共有します。見やすいということ以外に横棒系列を使用する 1 つの理由は、水平方向だと最も読みやすく長い軸ラベルを適切に表示できるということがあります。

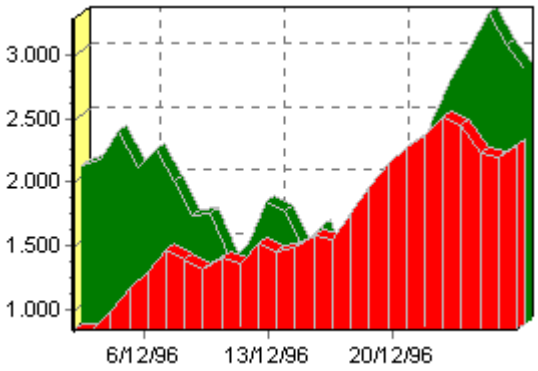
図. 6
2D の横棒



4. 2. 5. 面

プロパティとメソッドの一覧については、ヘルプの**面系列**を参照してください。
面系列には、リボン系列と似た特性（塗りつぶし）があります。

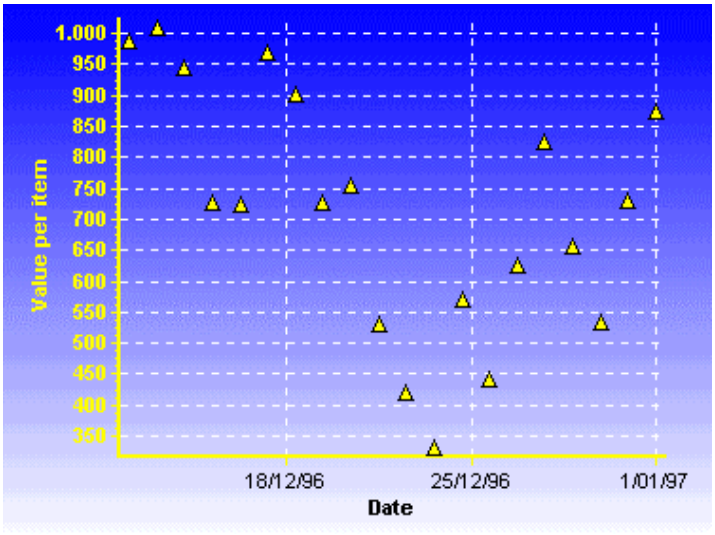
図. 7
3D 面系列



4. 2. 6. 散布図

プロパティとメソッドの一覧については、ヘルプの**散布図系列**を参照してください。
散布図系列は、その定義において線のないリボン系列に似ています。

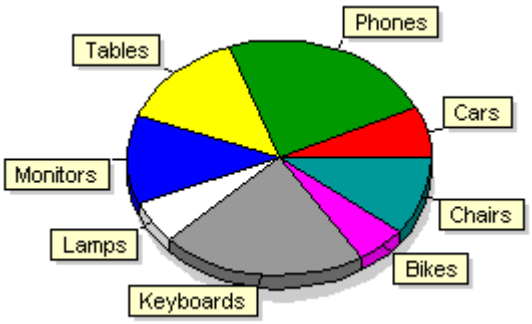
図. 8
2D の散布図



4. 2. 7. 円

プロパティとメソッドの一覧については、ヘルプの**円系列**を参照してください。
円系列は、軸をまったく必要としないという点でその他とは異なる系列です。チャートの中で円系列と軸を必要とする別の系列とを混在させることも可能です。

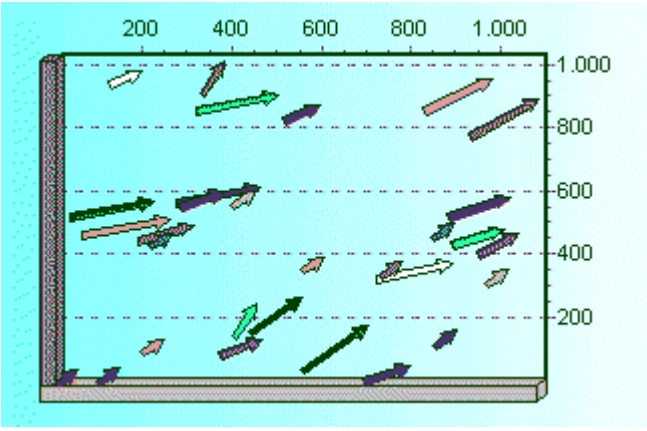
図. 9
3D の円



4. 2. 8. 矢印

プロパティとメソッドの一覧については、ヘルプの**矢印系列**を参照してください。

図. 10
3D の矢印系列

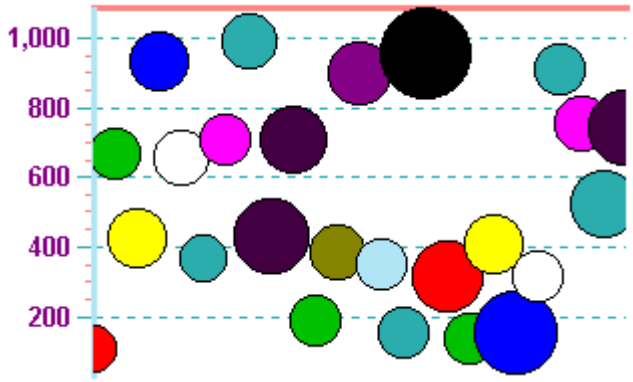


矢印系列は、多くのイベントの開始点および終了点をそれぞれ表示する際に便利です。

4. 2. 9. 泡

プロパティとメソッドの一覧については、ヘルプの**泡系列**を参照してください。
泡系列には、3 つの設定可能なパラメータがあります。これらのパラメータは、系列のデータの値を定義します。
XValues、YValues、RadiusValues
泡系列は、特に重要な部分を強調するのに便利です。たとえば、大量に売れている製品の収益を比較しても、他のあまり売れていない製品の収益に関する情報は得られません。大きな泡が重要であることが一目でわかります。

図. 11
2D の泡チャート

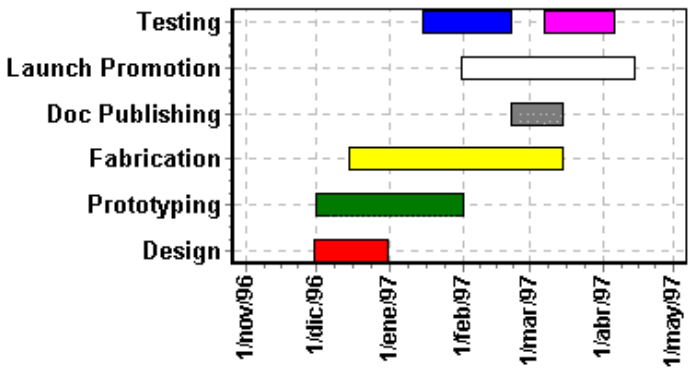


形状
泡系列は、三角形など様々な形状に設定することができます。

4. 2. 10. ガント

プロパティとメソッドの一覧については、ヘルプの**ガント系列**を参照してください。
計画などの一連の動作の進捗状況を確認するために、ガントチャートを使用できます。
ガント系列は、開始値および終了値を持つバーを描きます。これらの値は、DateTime 形式になっている場合もあります。バーの垂直位置に対する Y 軸の値を定義し、「NextBar」を定義して、バーとバーの間の接続線を描くことができます。

図. 12
ガント系列を持つ 2D のチャート



コードでガントバーを追加する方法
AddGantt メソッド、もしくは AddGanttColor メソッドを使用します。
例:

```
With TChart1.Series(0).asGantt
    .AddGantt DateValue("1,1,2008"), DateValue("31,1,2008"), 2, "Hello"
    .AddGantt DateValue("2008,1,15"), DateValue("2008,2,15"), 1, "Nice"
    .AddGantt DateValue("2008, 2, 1"), DateValue("2008, 2, 28"), 0, "World"
    .AddGantt DateValue("2008, 3, 1"), DateValue("2008, 3, 31"), 2, ""
    .AddGantt DateValue("2008, 4, 1"), DateValue("2008, 4, 30"), 0, ""
```

```
        .AddGantt DateValue("2008, 3, 15"), DateValue("2008, 4, 15"), 1, ""
        ' 接続線の変更
        .ConnectingPen.Width = 3
        .ConnectingPen.Color = vbBlue
        ' バーの高さの増加
        .Pointer.VerticalSize = 16
    End With
```

あるいは...

```
    With TChart1.Series(0).asGantt
        .AddGantt DateValue("1,1,2008"), DateValue("31,1,2008"), 2, "Hello", vbGreen
    End With
```

上記のコード内で「2」は、このバーの希望する垂直方向の位置を示します。
希望する垂直方向の位置を選択してください。

ガントバーを接続するには次の手順に従います。
1)「AddGantt」メソッド、もしくは「AddGanttColor」メソッドの Long 型の戻り値を格納します。

```
    Dim tmp1, tmp2 As Integer

    With TChart1.Series(0).asGantt
        tmp1 = .AddGantt.DateValue("1,1,2008"), DateValue("31,1,2008"), 2, "Development"
        tmp2 = .AddGantt.DateValue("2008,1,15"), DateValue("2008,2,15"), 1, "Testing"
    End With
```

2)次に NextTask プロパティを使用します。

```
    TChart1.Series(0).asGantt.NextTask.Value(tmp1) = tmp2
```

これにより、「Development」ガントバーから「Testing」ガントバーへ線を描きます。「ConnectingLinePen」プロパティは、線を描くペンです。

TeeChart はデフォルトで、ポイントが追加された順番でオーダーが決まります。NextTask を使用してガントポイントに接続するために、AddGantt の戻り値をインデックスとして使用しますが、このポイント（たとえば、インデックスを 5 とした場合）の前にガントポイントを追加した場合、このインデックスはプラス 1 のインデックス番号（インデックスが 6）になります。これは、ポイントを描画したいインデックス順に追加すれば問題はありません。しかし、ガントポイントを追加する前に、Order を loNone に設定すると問題がなくなります。このように AddGantt により戻されるインデックスは、後で追加されたポイントに関係なく不変のままになります。

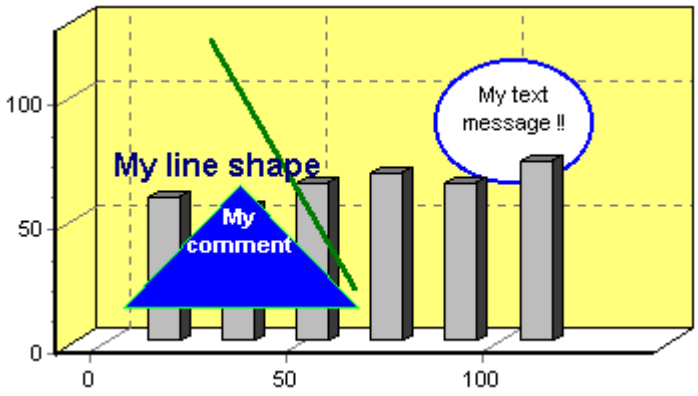
例.

```
TChart1.Series(0).asGantt.StartValues.Order = loNone
```

4. 2. 11. シェープ

プロパティとメソッドの一覧については、ヘルプの**シェープ系列**を参照してください。
シェープ系列は、実行時に例外的な情報を得た場合などに、チャートに追加情報を追加したり、定義する際に便利です。チャートに追加したどの図形にでもテキストを追加することができ、その図形を別の系列に関連付けることもできます。

図. 13
シェープ系列

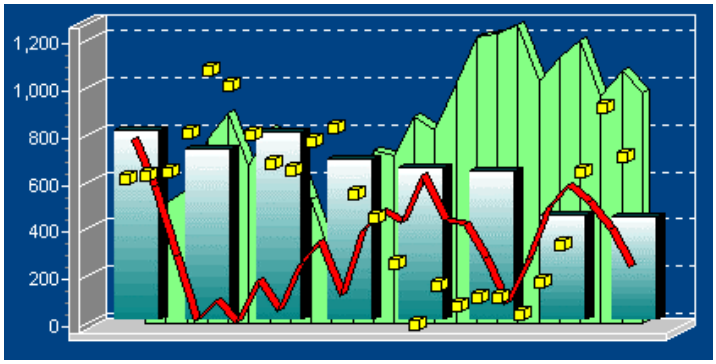


各シェープは対応する2つの座標値、つまりそのシェープを囲む目に見えない長方形の左上および右下の座標値を持っています。テキストをボックスに追加することもできます。これらの座標値およびメッセージは、コードで実行時に更新すればチャートのどこにでも動的に配置できます。

4. 2. 12. 系列の組み合わせ

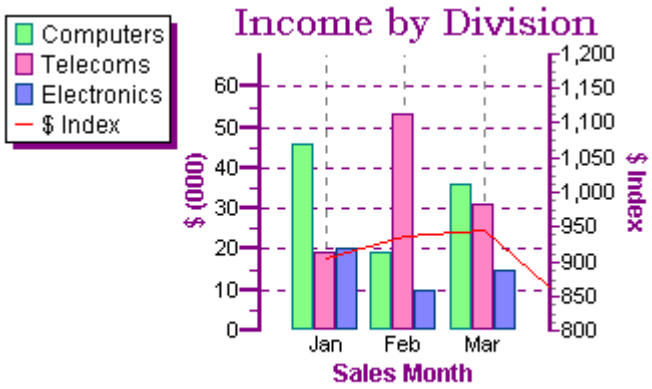
同時にチャートへ追加できる系列の数には実質上制限はありません。
ほとんどの系列型は、異なる系列型と混在させることができます。系列間の軸の定義が矛盾する場合など、ある特定の組み合わせについては実用的でないことがあります。そのような場合、その系列は使用できず、間違ってそのような系列を使用しないよう、系列ギャラリで使用できない系列は淡色表示になります。
系列型の組み合わせに関する詳細な内容については、関数に関するトピックを参照してください。関数は算術的な関係を作成し表示するために別の系列と一緒に動作します。

図. 14
系列の組み合わせ



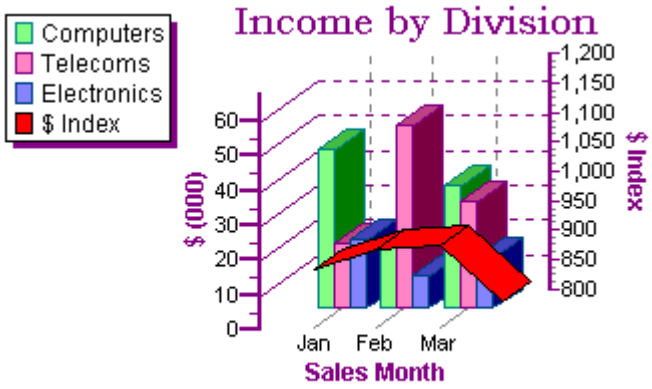
系列の組み合わせの例
1つのチャートで系列型を組み合わせると、情報の価値が増加します。次の例は部門ごとの収益を示しており、同じチャートに$インデックスを描画して、収益への外部の影響を測定できるようになっています。

図. 15
2D の系列型の組み合わせ



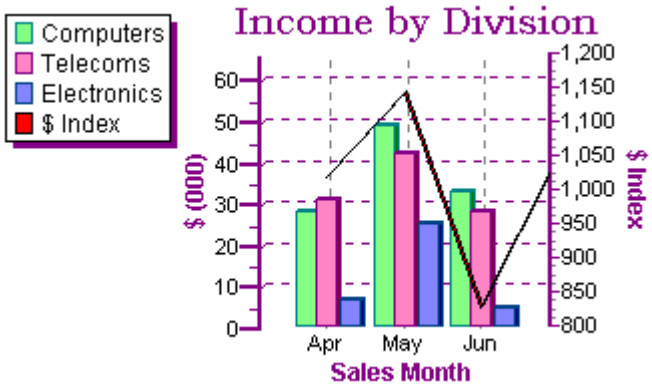
3D チャートで系列を組み合わせることもできます。上記の例を以下に 3D で表示しています。チャートは非常に見やすく仕上がっていますが、3D の度合が高いため、部門ごとの収益の月々の$インデックスが見にくくなっています。

図.16
3D の系列型の組み合わせ(60%の 3D)



3D の効果(3D のパーセント率)を減らすことにより、この影響を最低限に押さえることができます。

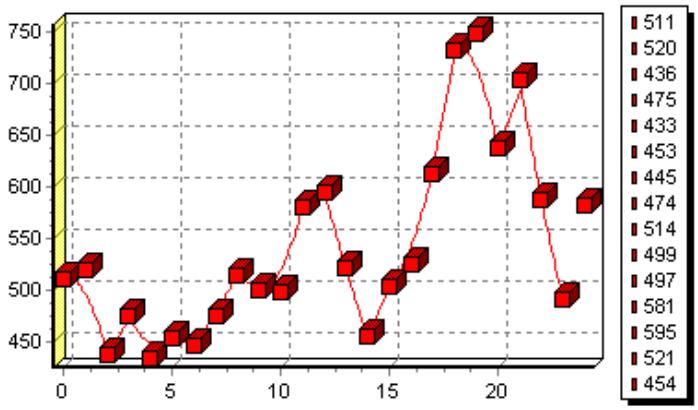
図. 17
3D の系列型の組み合わせ(6%の 3D)



4. 2. 13. ベジェ

プロパティおよびメソッドの一覧については、ヘルプの**ベジェ系列**を参照してください。
ベジェ線は、系列の 3 ポイント毎に線を描きます。ベジェ系列ポイントの計算方法はいくつかあります。この系列は、Windows（GDI BezierTo 関数)メソッドを使用しています。

図. 18
ベジェ系列



4. 2. 14. ボックスプロット系列

プロパティおよびメソッドの一覧については、ヘルプの**縦ボックスプロット系列**や**横ボックスプロット系列**を参照してください。
ボックスプロット系列は、統計学の系列型です。これは、データセットの位置と変化情報を識別するために使用できます。

図. 19
縦ボックスプロット系列

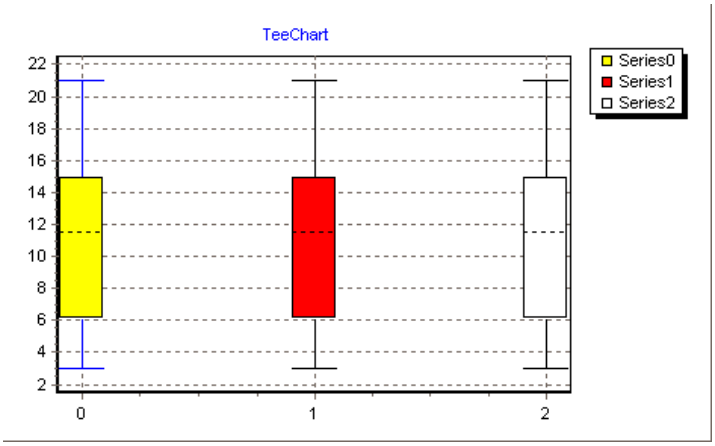
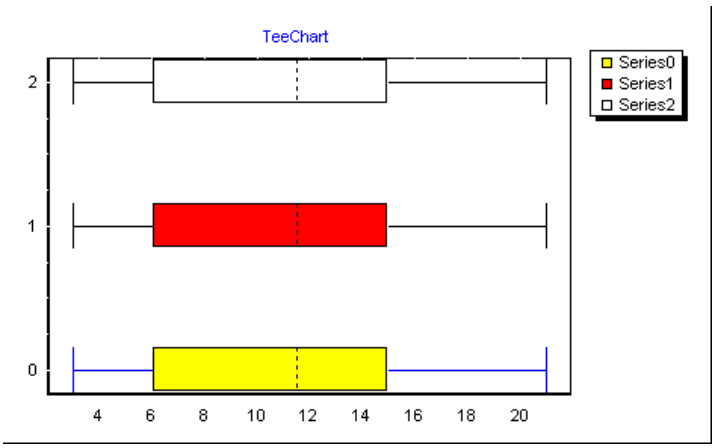


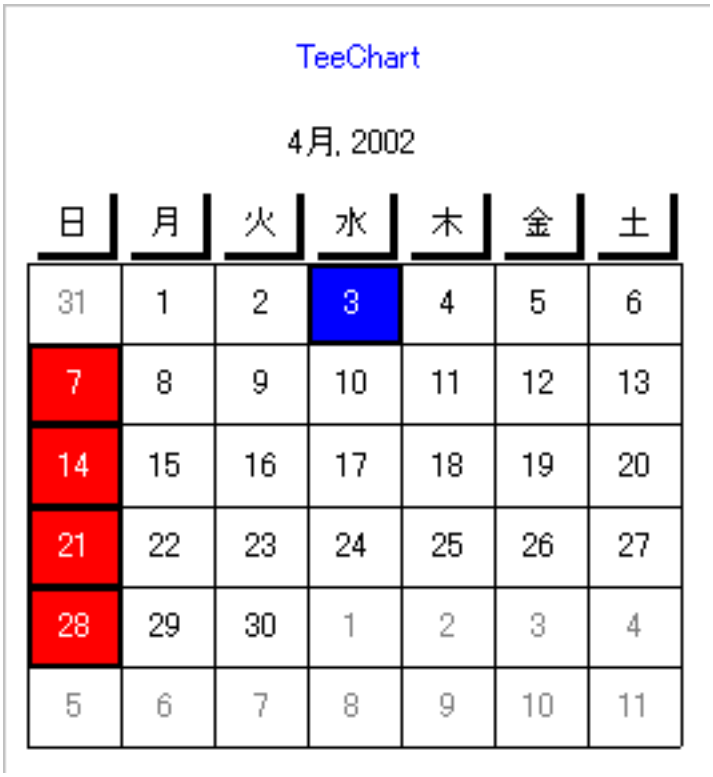
図. 20
横ボックスプロット系列



4. 2. 15. カレンダー系列

プロパティおよびメソッドの一覧については、ヘルプの**カレンダー系列**を参照してください。
日付関連を表示するために、カレンダー系列を使用してください。

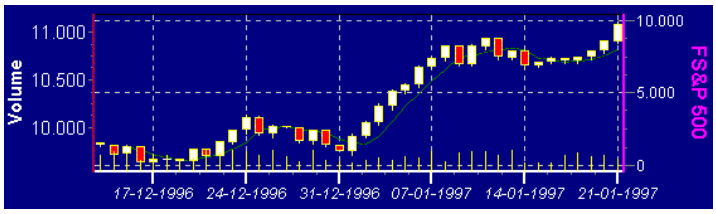
図. 21
カレンダー系列



4. 2. 16. キャンドル

プロパティおよびメソッドの一覧については、ヘルプの**キャンドル系列**を参照してください。
HighValues、LowValues、OpenValues、CloseValues、DateValues のプロパティがあります。キャンドルは、金融市場情報にとても便利です。

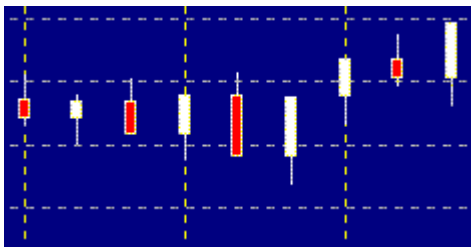
図. 22
キャンドル系列



例
ズームしたキャンドルの図を見る場合、金融情報の流れを見ることができます。

白いバーは、市場の上昇を表し、中心線の下の部分が始値で上の部分が終値です。赤いバーは、市場での下降を表し、中心線の下の部分が終値で上の部分が始値です。

図. 23
キャンドルのズーム



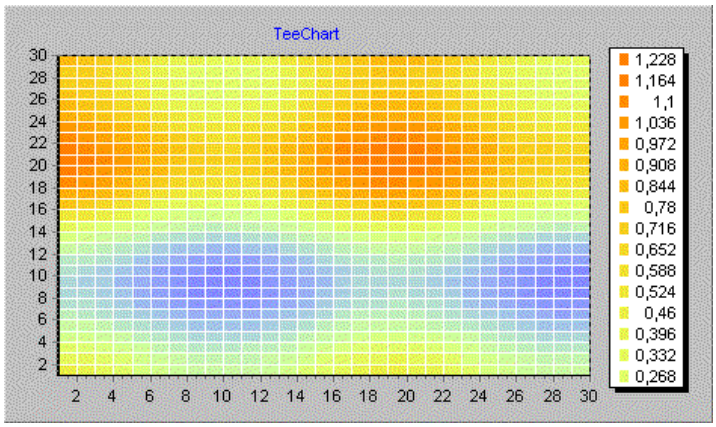
次は、AddCandle メソッドの使用方法を表しています。このコードは、等しい間隔で画面にランダムデータで系列を表示します。ランダムデータの代わりに、独自のデータソースあるいはチャートエディタにより直接データセットに接続することもできます。

```
Dim tmpopen, tmp As Integer
With TChart1.Series(0)
  .Clear
  tmpopen = 1000 + Rnd(100)
  For t = 1 To 30
    tmp = Int(100 * Rnd - 50)
    .asCandle.AddCandle Date + t, tmpopen, tmpopen + 20, tmpopen - 20, tmpopen + tmp
    tmpopen = tmpopen + tmp
  Next t
End With
```

4. 2. 17. カラーグリッド

プロパティおよびメソッドの一覧については、ヘルプの**カラーグリッド系列**を参照してください。
カラーグリッド系列は、X、Y グリッドに基づいて色で値を描画します。

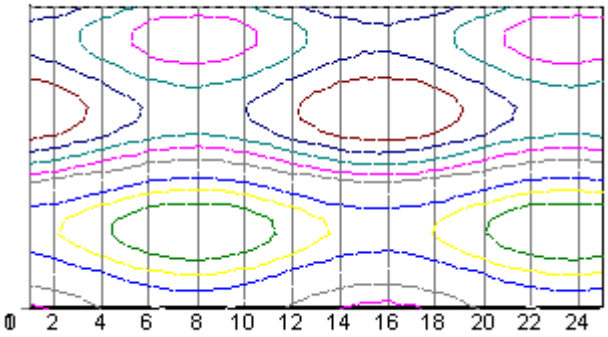
図. 24
カラーグリッド系列



4. 2. 18. 等高線

プロパティおよびメソッドの一覧については、ヘルプの**等高線系列**を参照してください。

図. 25
等高線系列



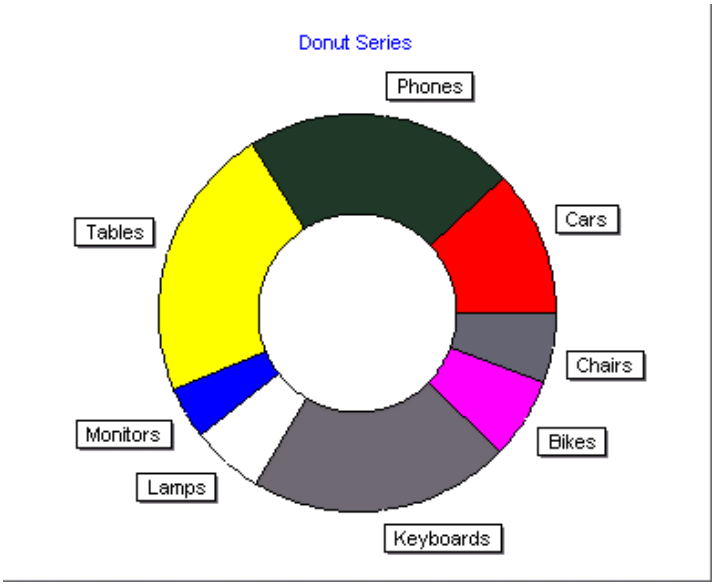
各レベルは、次のように設定すると異なる色を使用して描画されます。
TChart1.Series(0).ColorEachPoint = True
レベルは、指定した Y 軸の位置に描画することができます。
TChart1.Series(0).asContour.YPosition = 123
各レベルは、このプロパティを設定することにより、値に応じた Y 軸に描画されます。
TChart1.Series(0).asContour.YPositionLevel = True

等高線系列は、デフォルトで各等高線レベルに一致する項目を凡例に描画します。Value および Color のレベルは自動的に設定されますが、等高線系列の OnGetLevel イベントを使用することで無効にすることもできます。

4. 2. 19. ドーナツ

プロパティおよびメソッドの一覧については、ヘルプの**ドーナツ系列**を参照してください。
ドーナツ系列は、中心にリサイズ可能な穴がある以外は、円系列にとても似ています。

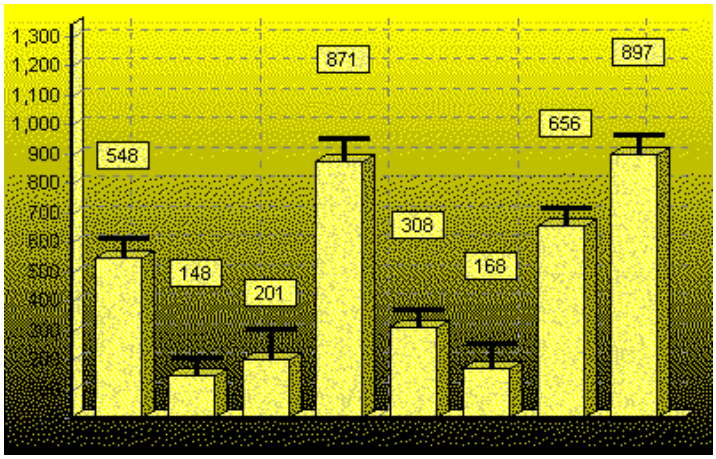
図. 26
ドーナツ系列



4. 2. 20. エラーバー

プロパティおよびメソッドの一覧については、ヘルプの**エラーバー系列**を参照してください。

図. 27
エラーバー系列



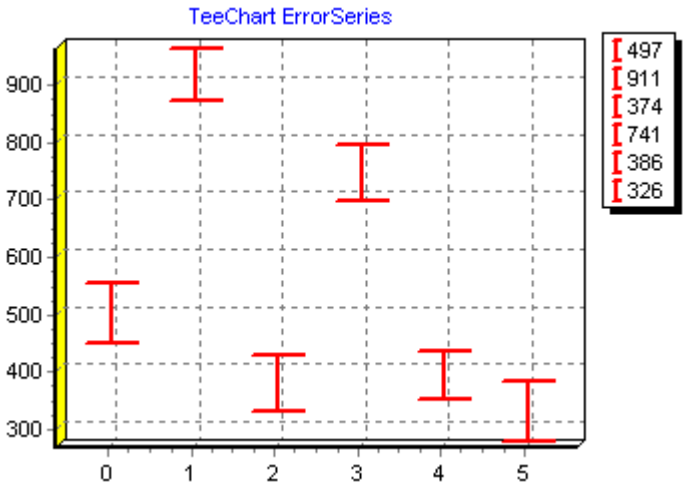
エラーバー系列のバー上部の T の高さは、エラーの大きさを描画します。

エラーバー系列は、実体値、評価値、成功、失敗レベルなどを持つどのようなデータにでも適用できます。

4. 2. 21. エラー

プロパティおよびメソッドの一覧については、ヘルプの**エラー系列**を参照してください。エラー系列はエラーバー系列から作成され、エラーバー系列にとても似ています。

図. 28
エラー系列



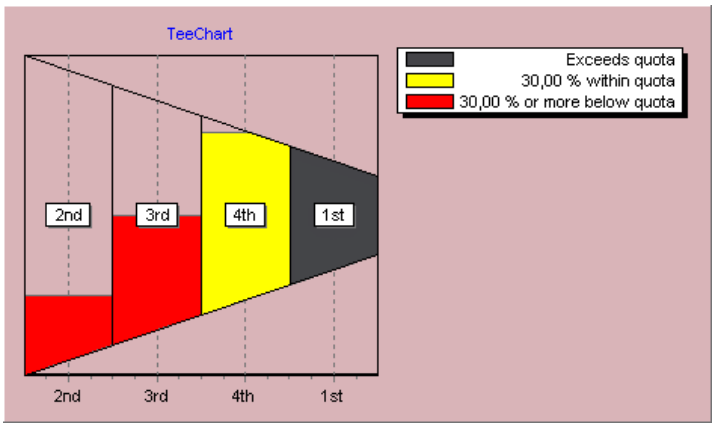
T はエラーポイントの異なる大きさで描画します。T の大きさはエラーの大きさを示しています。

エラー系列は、実体値、評価値、成功、失敗レベルなどを持つどのようなデータにでも適用できます。

4. 2. 22. ファネル

プロパティおよびメソッドの一覧については、ヘルプの**ファネル系列**を参照してください。
これは、パイプラインチャートとして知られています。

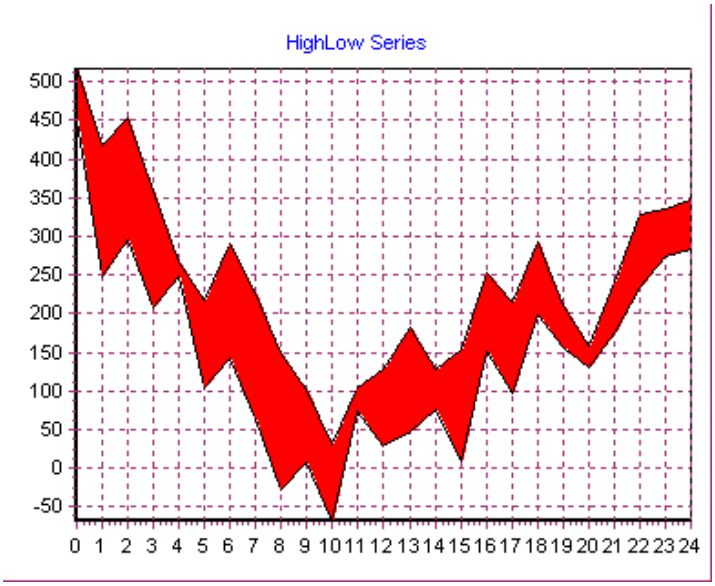
図. 29
ファネル系列



4. 2. 23. High-Low

プロパティおよびメソッドの一覧については、**High-Low 系列**を参照してください。

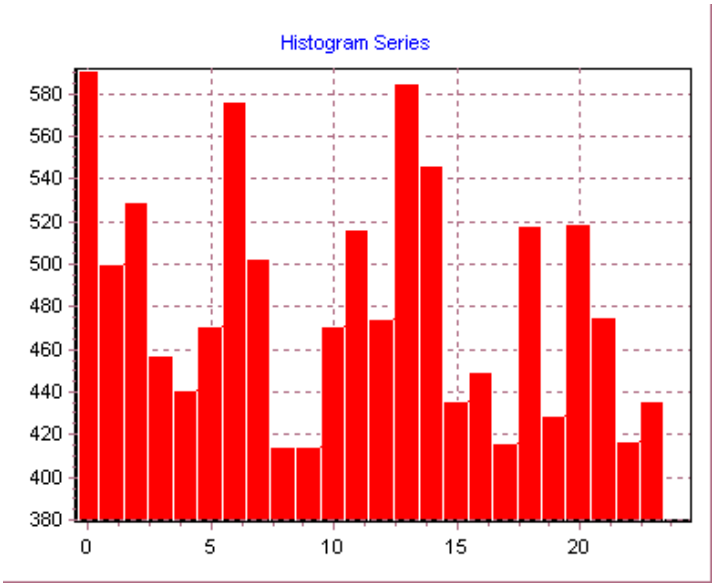
図. 30
High-Low 系列



4. 2. 24. ヒストグラム

プロパティおよびメソッドの一覧については、ヘルプの**ヒストグラム系列**を参照してください。

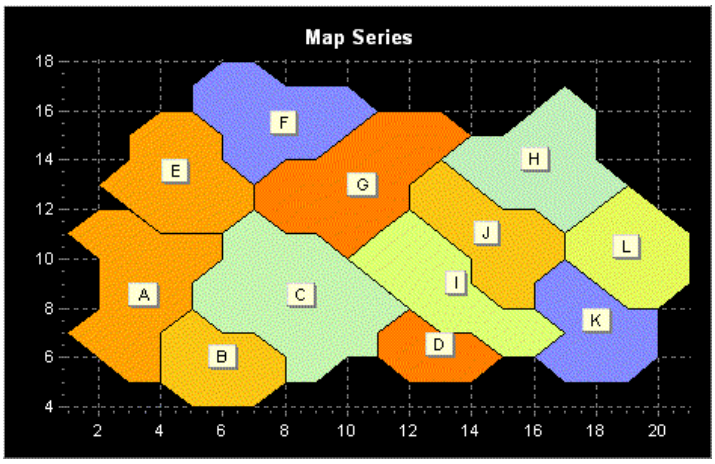
図. 31
ヒストグラム系列



4. 2. 25. マップ

プロパティおよびメソッドの一覧については、ヘルプの**マップ系列**を参照してください。
マップ系列は、X、Y 座標でシェープを表示します。シェープは、無制限のサイド(境界辺)のポリゴンを使用し、マップ系列の各ポリゴンはポリゴンサイドの異なった数を持つことができます。マップ系列は 3D で描画できますが、2D での描画の方が見やすくなります。この理由は、Z 値に付加的な情報(例えばポリゴンの色)を保持できるからです。

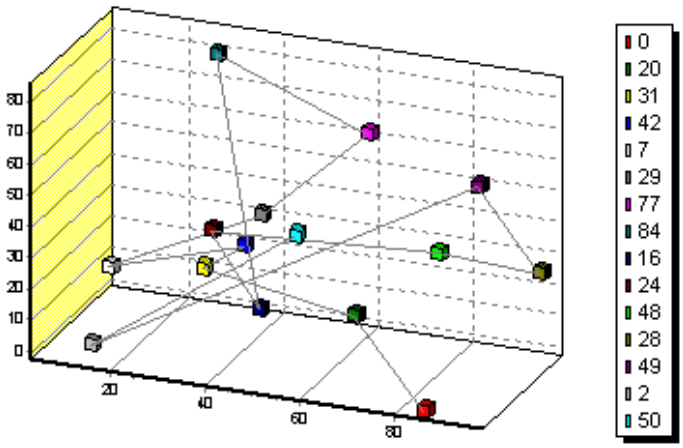
図. 32
マップ系列



4. 2. 26. 3D散布

プロパティおよびメソッドの一覧については、ヘルプの **3D 散布系列**を参照してください。

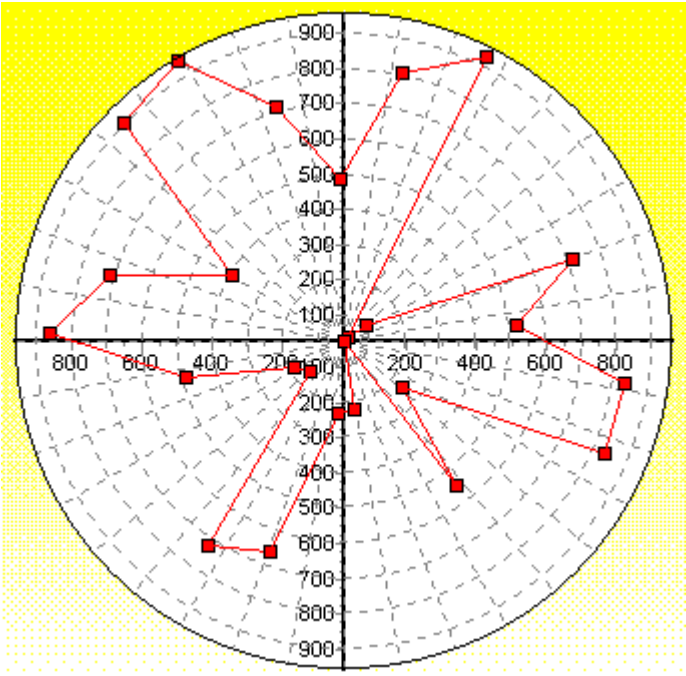
図. 33
3D 散布系列



4. 2. 27. 極

プロパティおよびメソッドの一覧については、ヘルプの**極系列**を参照してください。

図. 34
2D 極系列

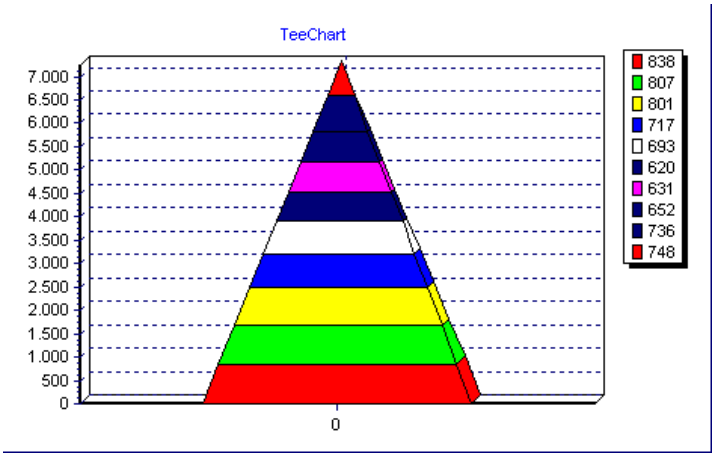


極系列は、XValues を 0° から角度の回転で描画します。YValues は原点からの距離で描画されます。

4. 2. 28. ピラミッド

プロパティおよびメソッドの一覧については、ヘルプの**ピラミッド系列**を参照してください。

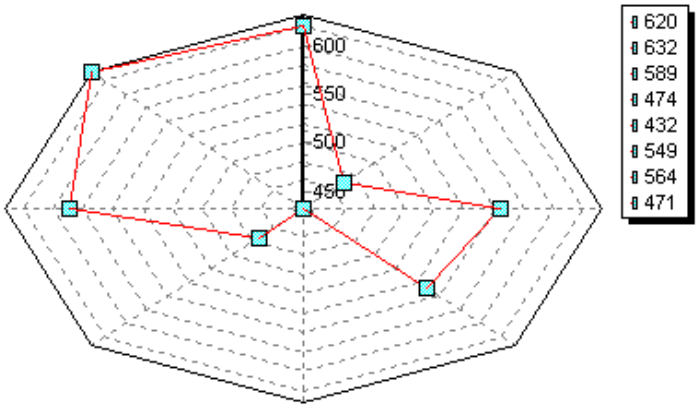
図. 35
ピラミッド系列



4. 2. 29. レーダー

プロパティおよびメソッドの一覧については、ヘルプの**レーダー系列**を参照してください。

図. 36
レーダー系列

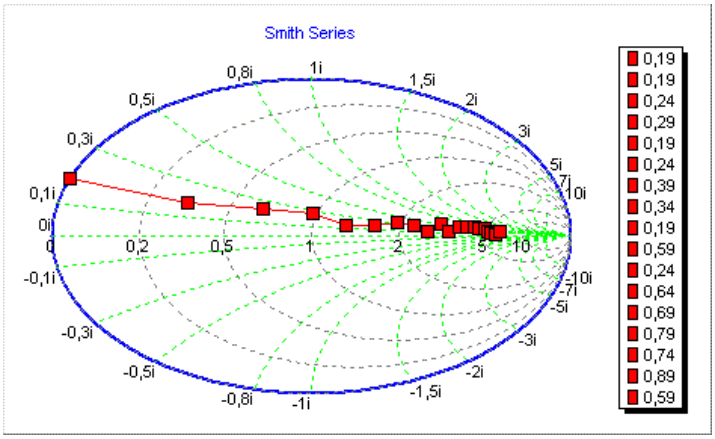


極系列のすべてのプロパティは、レーダーにも適用されます。
極系列と同じ LeftAxis 軸および BottomAxis 軸を使用することにより、グリッド線とラベルを制御できます。
最初の系列は、CirclePen プロパティを制御します。(上の図の青色において示されます)
極もレーダーも、Brush プロパティを使用してポイントで領域を塗りつぶすことができます。

4. 2. 30. スミス

プロパティおよびメソッドの一覧については、ヘルプの**スミス系列**を参照してください。

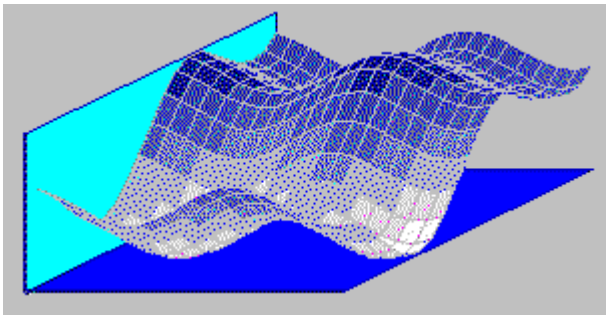
図. 37
スミス系列



4. 2. 31. サーフェス

プロパティおよびメソッドの一覧については、ヘルプの**サーフェス系列**を参照してください。

図. 38
サーフェス系列

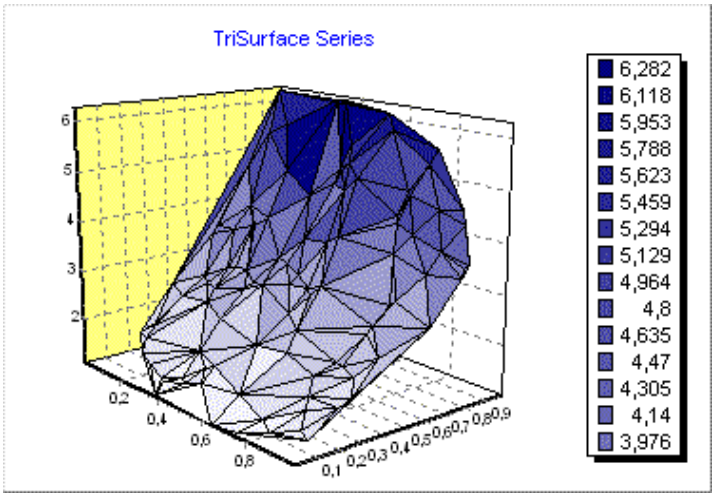


サーフェス系列は、3D の座標を使用します。TeeChart のサーフェス系列は Null 値をサポートします。Null 値の部分は、サーフェスに穴が開いているように見えます。

4. 2. 32. サーフェス(三角)

プロパティおよびメソッドの一覧については、ヘルプの**サーフェス(三角)系列**を参照してください。

図. 39
サーフェス(三角)系列

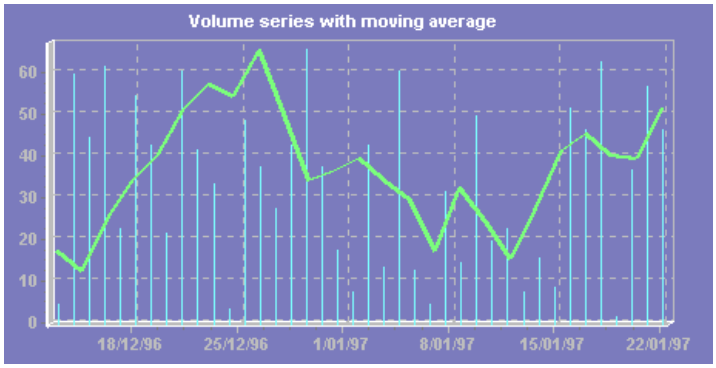


サーフェス(三角)系列は、サーフェス系列に似ておりますが、サーフェス系列とは異なる三角測量を使用してサーフェスを構築します。

4. 2. 33. ボリューム

プロパティおよびメソッドの一覧については、ヘルプの**ボリューム系列**を参照してください。

図. 40
ボリューム系列

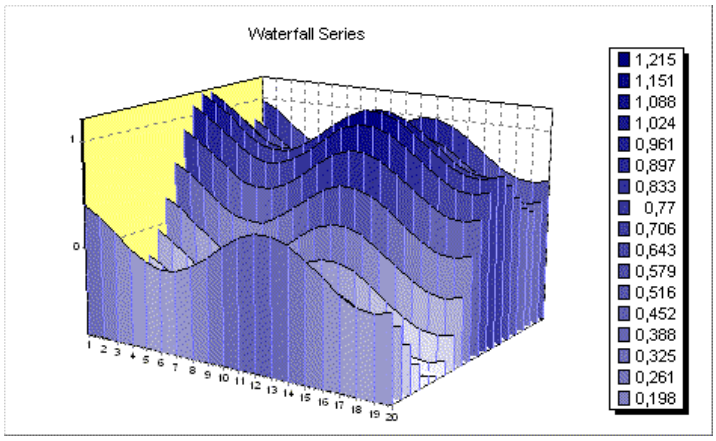


ボリューム系列は、金融市場のデータに使用すると便利です。バー系列と非常に似ていますが、バーの代わりに細い線で描画されます。線は積み重ねて描画できません。

4. 2. 34. ウォーターフォール

プロパティおよびメソッドの一覧については、ヘルプの**ウォーターフォール系列**を参照してください。

図. 41
ウォーターフォール系列



ウォーターフォール系列は、サーフェス系列に似ています。ウォーターフォールは、サーフェスのZ面を示すために垂直方向のスライスで描画します。

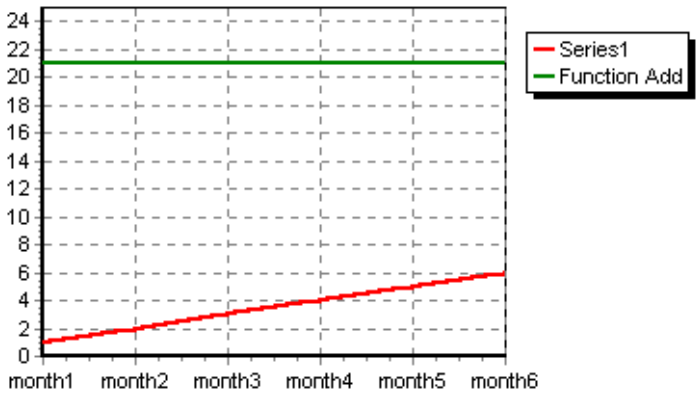
4. 3. 関数

和

プロパティとメソッドの一覧については、ヘルプの**和関数**を参照してください。

和関数は、1 つまたは複数の系列からデータを追加します。リボン系列の「Series1」とリボン系列の「和関数」を作成し、和関数系列を「Series1」の和として定義し、これ以上何もしないとします。この場合、「Series1」が表示されたチャートが示され、「和系列」は「Series1」のすべての値の合計を示すフラットな線として表示されます。図では、合計 1 + 2 + 3 + 4 + 5 + 6 = 21 が示されています。

図. 1
インプット系列が 1 つの 2D の和関数

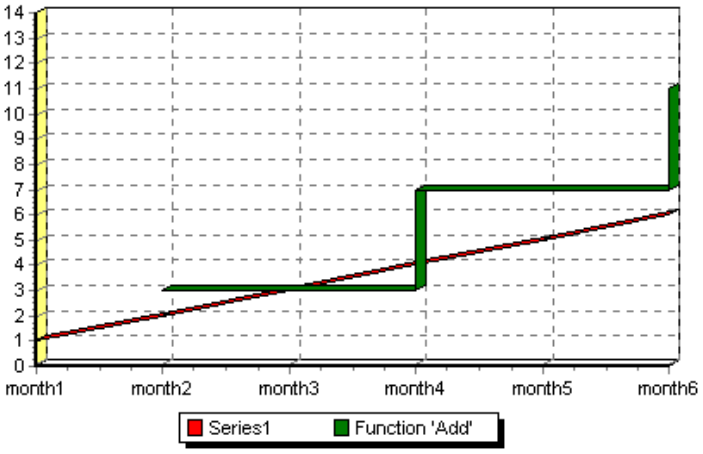


2 ポイントのグループ化(1+2)、(3+4)、(5+6)で「Series1」の和を表示するように和関数系列を変更することができます。ここでは、Period プロパティ(チャートエディタの[データソース]タブの適用ボタン)を使用します。コードは以下のようになります。

```
TChart1.Series(2).FunctionType.Period= 2
' Series(2)は関数系列です。
' チャートエディタによって関数を定義すれば、関数名は自動的に割り当てられます。その場合、その関数名はプロパティウィンドウ内に表示されます。
```

Period を 2 に定義すると、和関数が 2 ポイントごとに追加を行うように設定されます。

図. 2
Period = 2 の和関数

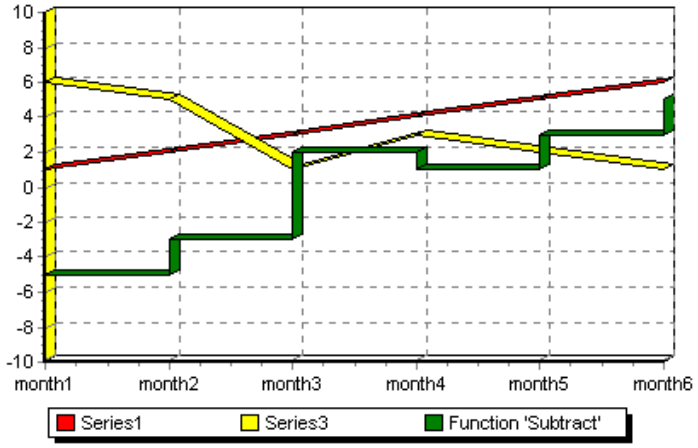


差分

プロパティとメソッドの一覧については、ヘルプの**差分関数**を参照してください。

差分は、2 つの入力系列を必要とします。関数に 1 つ以上の系列があると、デフォルトで Period は 1 軸ポイントに設定されます。リストの 2 番目の系列がリストの 1 番目の系列から減算されます。

図. 3
差分関数

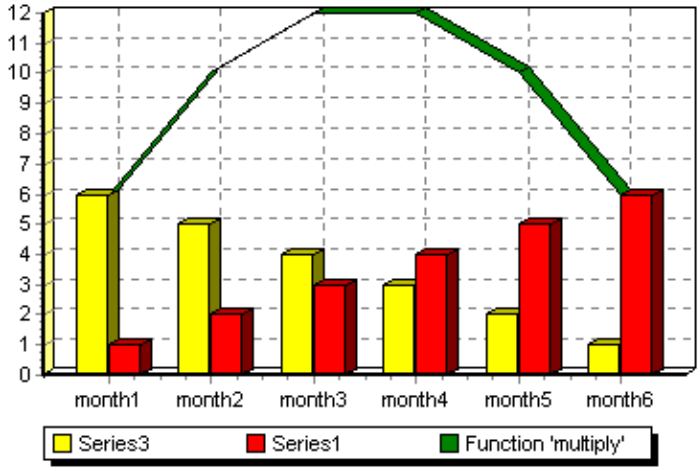


積

プロパティとメソッドの一覧については、ヘルプの**積関数**を参照してください。

積関数のデフォルトの Period は 1 です。積関数に系列をいくつでも追加できます。

図. 4
積関数



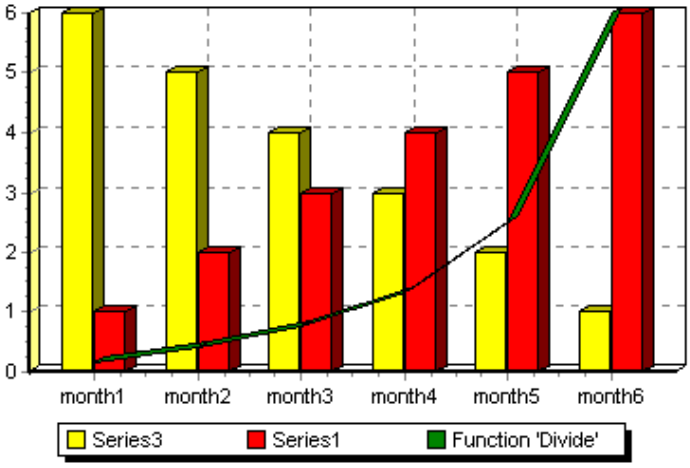
商

プロパティとメソッドの一覧については、ヘルプの**商関数**を参照してください。

商関数は、少なくとも 2 つの入力系列を必要とし、商関数のデフォルトの Period は 1 です。リストの 2 番目の系列が、分母になります。

3 つ以上の系列を追加する場合、1 番目を 2 番目で割り、次に 2 番目を 3 番目で割るというようになります。

図. 5
商関数

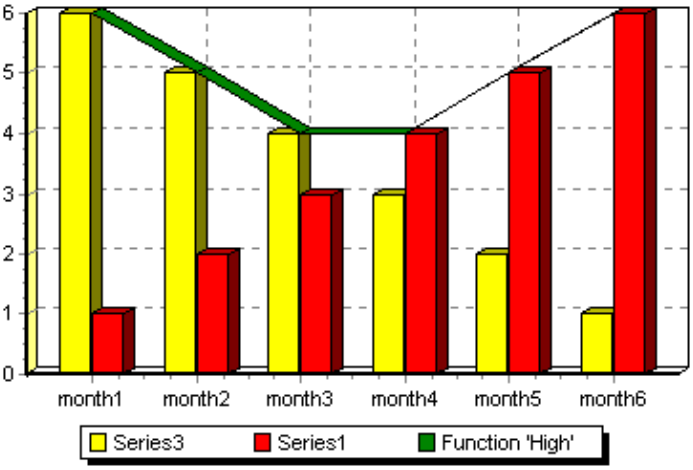


最大

プロパティとメソッドの一覧については、ヘルプの**最大関数**を参照してください。

最大関数では複数の入力系列が使用でき、常にこれらの系列の中で各 Period ポイントで最高のポイントを表示します(1 つの系列のデフォルトの Period は 0、2 つあるいはそれ以上の系列のデフォルトの Period は 1 になります)。

図. 6
最大関数

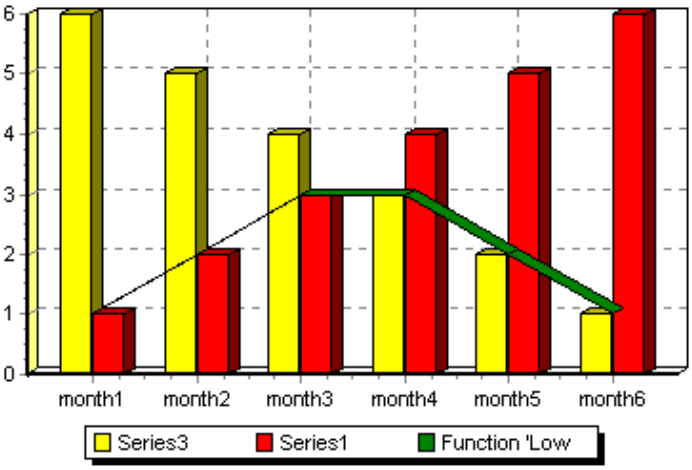


最小

プロパティとメソッドの一覧については、ヘルプの**最小関数**を参照してください。

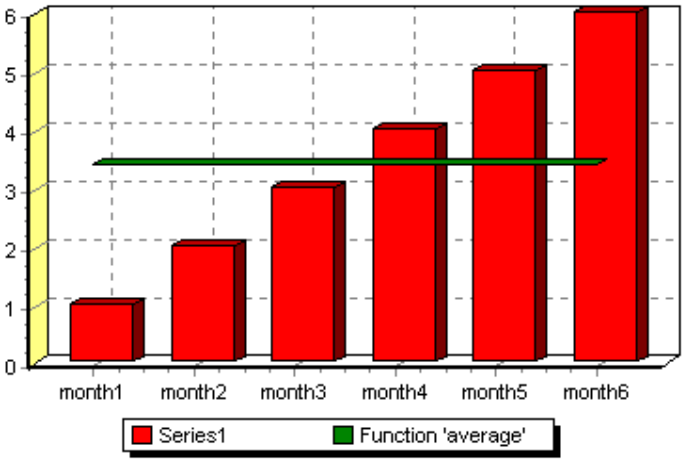
最小関数では、複数の入力系列が使用できます。これは、常にこれらの系列の中で各 Period ポイントで最低のポイントを表示します(1 つの系列のデフォルトの Period は 0、2 つあるいはそれ以上の系列のデフォルトの Period は 1 になります)。

図. 7
最小関数



平均値
プロパティとメソッドの一覧については、ヘルプの**平均値関数**を参照してください。

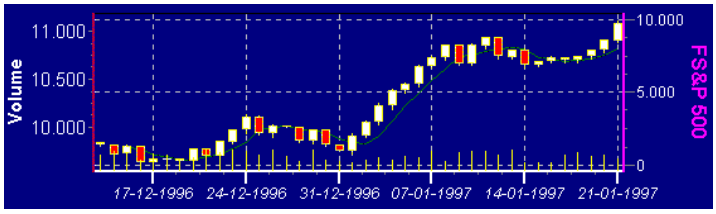
図. 8
平均値関数



系列が 1 つの平均値関数のデフォルトの Period は 0（すべて）で、これはチャート全体にわたるその系列の平均を出します。複数の系列がある場合は、Period は 1 軸ポイントになります。
平均曲線の周期を変更するには Period を変更してください。

移動平均値
移動平均値関数は、現在のデータの平均値を計算します。平均値を計算する範囲を定義できます。

図. 9
キャンドル系列のデータに適応した移動平均値



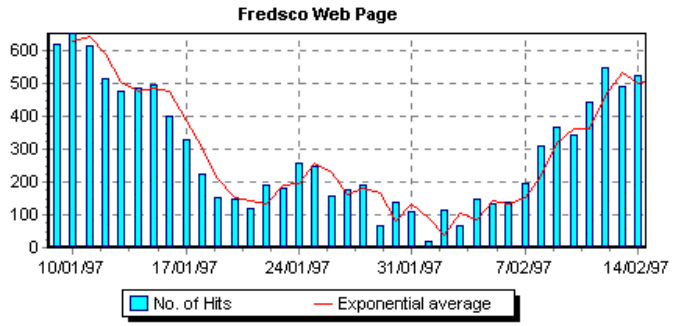
R.S.I 関数（Relative Strength Index）は、金融のアプリケーションでよく使用されます。

図. 10
キャンドル系列の最後の 20 データポイントを計算した 2D R.S.I

指数平均値
指数平均値関数は、移動平均値に似ていますが、最新のデータをより反映するために重み付け要素を加えます。

下記の**図. 11** は、0.2 の重み付けをした指数平均を表します。

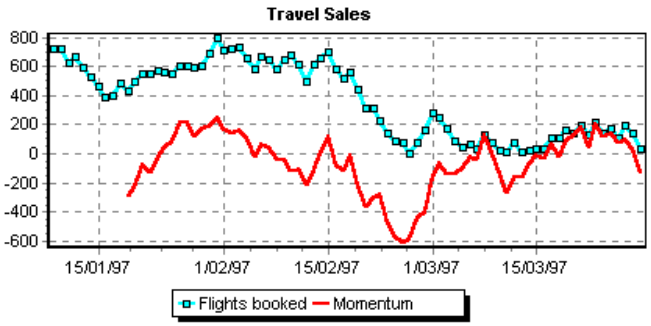
図. 11
2D 指数平均値



運動量

運動量関数は、範囲を使用して定義されます。グラフの線は、範囲の最後の値から最初の値を引きます。

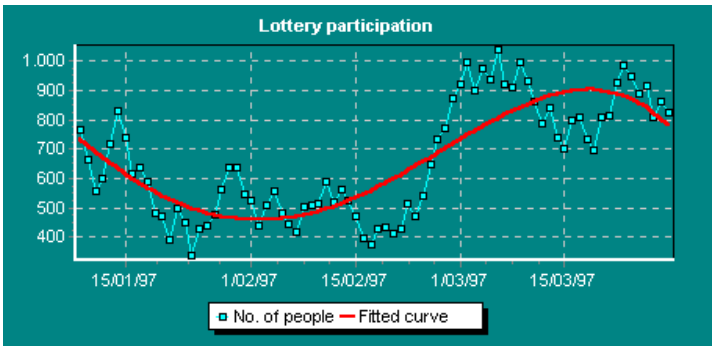
図. 12
範囲が 10 の 2D 運動量



系列組み合わせ

系列組み合わせ関数は、滑らかな曲線を描くため、基となるデータ系列を最小ニ乗法で計算します。

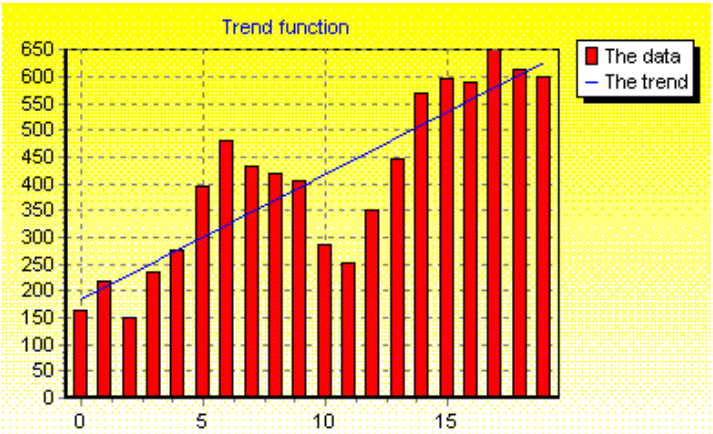
図. 13
2D 系列組み合わせ



トレンド

トレンド関数は、系列組み合わせに似ていますが、直線を描くため、基となるデータ系列を最小ニ乗法で計算します。範囲は、トレンドに適応します。

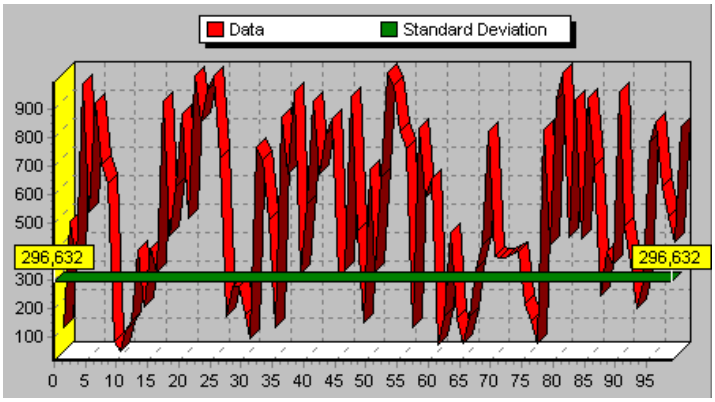
図. 14
2D のトレンド



標準偏差

標準偏差関数は、データの平均値から標準偏差を描画します。範囲は、設定できます。

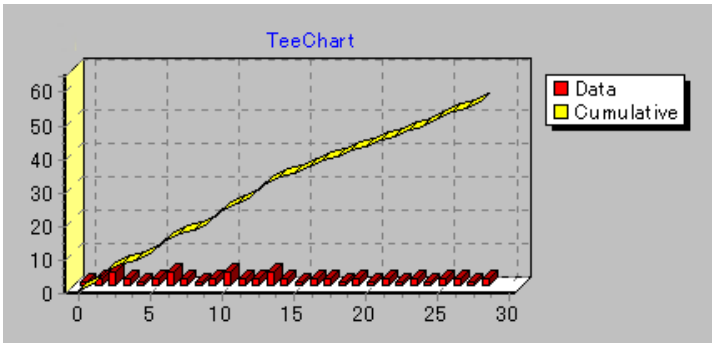
図. 15
3D 標準偏差



累積

累積関数は、入力データの累積しているラインを描画します。範囲は、設定できます。

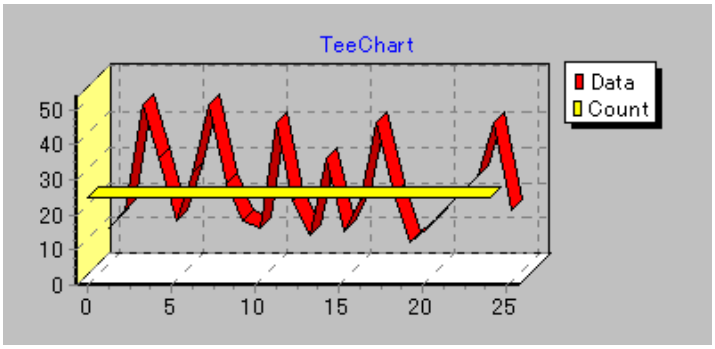
図. 16
累積



計算

計算関数は、入力ポイント数を返します。範囲は、設定できます。

図. 17
計算



5．1．チャートおよび系列の操作方法

TeeChart の設計上のパラダイムを理解するには、チャートの内容、すなわちデータ系列と、チャート自体、たとえば軸のフォーマット、凡例およびタイトルなどを概念上区別する必要があります。チャートから独立して系列コンポーネントを参照する、チャートの境界を越えた系列を定義し、操作することもできます。その場合でも、チャートのコピーアンドペーストが可能で、定義された内容、軸、凡例、系列がすべてコピーされます。

チャートクラス
最も重要なチャートクラスです。これを使用することにより全てのサブクラスにアクセスできます。

チャートのサブクラス
チャートクラスにはサブクラスがあります。たとえば、チャートの軸のプロパティにアクセスしたり変更する場合、次のサブクラスのいずれかのプロパティを変更することになります。

TChart.Axis.Bottom
TChart.Axis.Top
TChart.Axis.Left
TChart.Axis.Right
TChart.Axis.Depth

軸プロパティの詳細については、ヘルプの **Axis** クラスを参照してください。
また、凡例は TChart サブクラスのもう 1 つの例です。

系列の背景としてのチャート
チャートクラスは、データ系列用の背景を提供します。チャート全体の外観は、コードまたはチャートエディタのチャートプロパティで制御できます。すべてのチャートプロパティの一覧については、オンラインヘルプを参照してください。

背景として、キャンバスのプロパティやメソッドを使用することにより、値から座標を計算できます。また、その逆もできます。キャンバスは、画面にチャートを描画する前後にアクセスできます。

系列とチャートの関連付け
チャートを開いて、エディタで系列を追加するか、あるいは AddSeries メソッドを使用すると、新しい系列がチャートに自動的に接続されます。

チャートエディタを使用する場合
チャートがフォームに配置された後、右クリックすると[編集]メニューを呼び出すことができます（チャートをダブルクリックして呼び出す方法もあります）。チャートの編集メニューの最初の画面で、系列の型や系列のタイトルを追加、削除、複製、変更するためのボタンオプションのリストが表示されます。このメニューでデータ系列を追加すると、自動的に新しいデータ系列がそのチャートと関連付けられます。

コードから使用する場合
チャートに系列を追加するには、TChart.AddSeries メソッドを使用してください。
例:
この例は、現在、存在する系列を削除し、新しい系列を作成しデータに接続します。サンプルの「Add Series」で、このコードの内容が参照できます。

```
With TChart1
    '既にチャートに系列が存在する場合
    If .SeriesCount > 0 Then
        '系列を削除します。
        .RemoveAllSeries
    End If
    '新しい系列を追加し、ランダムデータを系列に接続します。
    '例:  .AddSeries(scHorizBar)は、.AddSeries(2)としても構いません。
```

```
    .AddSeries (Combo1.ListIndex) 'コンボリストは、系列型に関連付けられています
    'ランダム値で新しい系列を描画します
    .Series(0).FillSampleValues 10
    .Series(0).Title = Combo1.List(Combo1.ListIndex) & " Series"
End With
```

頻繁に使用するパラメータ

チャートに関連するパラメータは、いずれも実際に必要になったときに重宝します。頻繁に使用されるいくつかのパラメータをここで紹介しておきます。

ラベルのインクリメント、最小間隔および角度

軸上の軸ラベルすべてを画面に表示するのは困難な場合があります。たとえば、軸上で頻繁に変更される長い日付ラベルなどがそうです。

ラベルのインクリメント

TeeChart はチャート軸のラベルの単位を制御することを**ラベルのインクリメント**と言います。チャートエディタの**軸–スケール–期待する増加量**に進み、軸を選択してこのプロパティにアクセスできます。これは、時間インクリメントの標準オプションのリストを表示します(例:1 秒、1 分… 1 日、など)。日付ラベルの軸の場合に、「1 日」を選択すると一日単位で軸上に日付を表示します。

コードにより、適切な軸のためにサブクラスのプロパティにアクセスできます。

例:

```
With TChart1
  .Axis.Bottom.Increment = TChart1.GetDateTimeStep(dtOneHour)
  .Axis.Right.Increment = 1000
End With
```

最小間隔

軸のラベルが重なっている場合、最小間隔を大きくすると、ラベルとの表示間隔が広がります。ラベルの特徴とチャートのリサイズにより、一部のラベルは非表示になる場合があります。

チャートエディタの**軸–ラベル–最小間隔**で設定できます。

最小間隔を 0 に設定すると、ラベルの重なりをチェックしないので、すべての軸ラベルが表示されます。

角度

軸のラベルの(回転)角度を制御します。このオプションは、チャートエディタの**軸–ラベル–角度**で変更できます。

JPEG、BMP、メタファイルのエクスポートファイル形式

TeeChart Pro 8J ActiveX コントロールは、チャートを保存するために JPEG、Bitmap(BMP)、Metafile 形式を提供します。メタファイル形式は、WMF(Windows Metafile Format)や EMF(Extended Metafile Format)をサポートします。

Bitmap 形式は、TeeChart の内部で使用され、メタファイルより速く描画されます。しかし、他のアプリケーションにチャートを「エクスポート」したり、コンテナアプリケーション(MSWord など)に埋め込んだりする場合、メタファイルの方がより良くリサイズされ描画されます。

ズーム

TeeChart は、すべてのチャート上にデフォルトでズームの機能を提供します。実行時にズームをする場合は、マウスの左ボタンを右下の方向にドラッグしてください。選択した範囲が長方形になります。マウスの左ボタンを離すとズームされます。ズームは何度でも繰り返すことで、どこまでも拡大できます。ズームを元に戻すためには、マウスの左ボタンを左上の方向にドラッグしてください。

メモ:

データ系列のポイントに関連するコードを有効にするためマウスクリックを使用できます。OnClick イベントは、ズームより優先されます。この場合、チャートの StopMouse メソッドを使用して、OnClick イベントとズームの切り替えを設定できます。

チャートエディタ – チャートページ – 一般タブでズームの機能を無効または有効に設定できます。「ズームを有効」の設定は、コードでも設定できます。

```
TChart1.Zoom.Enable = False
```

「False」は、ズームを無効にします。

ズームは、コードでも設定できます。この場合、ズームさせたい領域の画面ピクセル座標が必要となります。

例 1:
「ピクセル」座標で領域をズームします:

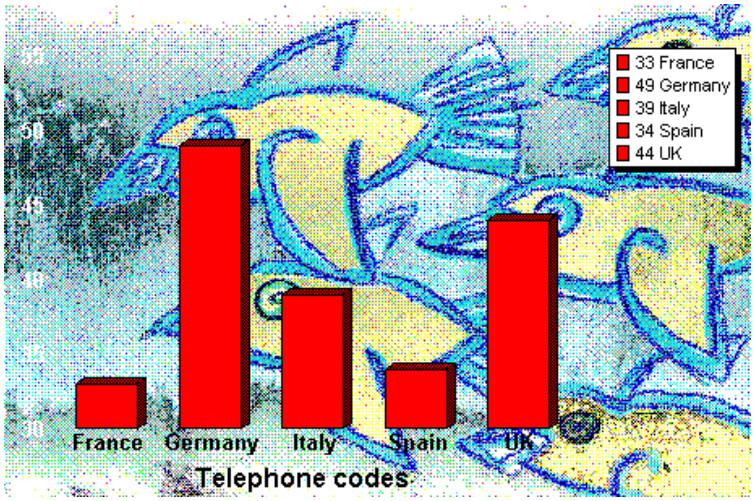
```
Dim Left as integer, Top as integer, Right as integer, Bottom as integer
Left = 123
Top = 67
Right = 175
Bottom = 100
TChart1.Zoom.ZoomRect Left, Top, Right, Bottom
```

例 2:
ポイント座標で領域をズームします:
最初に、ピクセル座標の値からポイントに調整する必要があります。そのためには、Axis クラス、あるいは Series クラスが使用できます。

```
Left   = TChart1.Series(0).CalcXPosValue(22.5)
Top    = TChart1.Series(0).CalcYPosValue(5000)
Right  = TChart1.Series(0).CalcXPosValue(57.6)
Bottom = TChart1.Series(0).CalcYPosValue(15000)
TChart1.Zoom.ZoomRect Left, Top, Right, Bottom
```

背景
チャートの背景にビットマップを貼りつけることができます。

図. 1
チャート背景としてのビットマップ



チャートエディタの[チャート]ページの[パネル]タブに、チャート背景を定義するパラメータがあります。

以降では、TeeChart を扱うときに考慮しなければならない設計上の事項といくつかの約束事をとりあげます。アプリケーションの設計に役に立つポイントを紹介します。

5．2．クリックイベント

OnClickSeries
OnClick
系列の OnClick および OnDblClick

チャートの OnClickSeries

TChart の OnClickSeries イベントにより、チャートの任意のアクティブな系列にアクセスすることができます。OnClickSeries イベントに次のコードを追加して、コードでどのようなことができるかをお試しください。

```
Private Sub TChart1_OnClickSeries(ByVal Series As TeeChart.ISeries, ByVal ValueIndex As Long, ByVal Button As
TeeChart.EMouseButton, ByVal Shift As TeeChart.EShiftState, ByVal X As Long, ByVal Y As Long)
  MsgBox (" Clicked Series: " + TChart1.Series(0).Name + " at point: " + Str(ValueIndex))
End Sub
```

Valueindex は、チャート内の系列データポイント（ポイント、バーなど）のインデックスを参照します。これを使用して、X および Y 値にアクセスすることができます。次に例を示します。

```
MsgBox ("Clicked Series: " + TChart1.Series(0).Name + " at point: " +_
Str(TChart1.Series(0).XValues.Value(ValueIndex)) + "," +_
Str(TChart1.Series(0).YValues.Value(ValueIndex)))
```

チャートの OnClick

チャートの OnClick イベントを使用して、同じ情報を得ることができます。

```
Private Sub TChart1_OnClick()
Dim t As Integer
  For t = 0 To TChart1.SeriesCount - 1
    If TChart1.Series(t).GetMousePoint <> -1 Then
      MsgBox (" Clicked Series: " + TChart1.Series(t).Name + " at point: " + Str(TChart1.Series(t).GetMousePoint))
    End If
  Next t
End Sub
```

系列の OnClick および OnDblClick

系列の OnClick イベントは、クリックイベントを系列レベルで見つけます。これにより、複数の系列チャートに対して、特定の系列のデータへのアクセスを制限することができます。

```
Private Sub TChart1_OnClickSeries(ByVal SeriesIndex As Long, ByVal ValueIndex As Long, ByVal Button As
TeeChart.EMouseButton, ByVal Shift As TeeChart.EShiftState, ByVal X As Long, ByVal Y As Long)
  MsgBox("hello: " & TChart1.Series(SeriesIndex).Name & " at point" & Str(ValueIndex))
End Sub
```

5．3．チャート上のカスタム描画

テキストボックス、あるいはチャート軸に関連するその他の図形を追加する場合、シェープ系列を選ぶのが最適です。シェープ系列が特定の要求に合わない場合には、独自の線および(あるいは)図をチャートに描画することができます。
TeeChart では、軸もしくは画面ピクセルでチャート領域にアクセスできるようになっています。TeeChart の VB サンプルの **Dragging points** および **Interactive** は、チャートの値から画面のポイントを変換したり、チャート上でカスタムな描画をすることに役立ちます。

座標値の計算

ここでは、ポイント座標値からピクセルへの変更方法、およびこの逆の変更方法について説明します。また、チャートコンポーネントの各グラフィック要素の正確な座標値の定義方法についても説明します。
ポイント値は、ユーザによるカスタムスケールで示されます。チャート軸は、ポイント値を表示する場合、ポイント値を適切な X および Y 画面ピクセル座標値に変換します。
軸クラスも系列クラスも画面座標値(ピクセル単位)から軸あるいはポイント値(ユーザ定義の単位)に変換する関数があります。
軸変換関数を使用するか、あるいは系列変換関数を使用するかの違いは、軸の方では 3D モードで最上位置の座標値に変換するだけなのに対して、系列の方は座標値をその Z オーダーの位置へ調整します。
メモ: 変換関数の使用はチャートを描いた後で、画面あるいはプライベートな非表示のキャンバスに対してのみ有効となります。

軸の値から画面の座標値を割り出すメソッド

CalcPosValue

任意の軸を使用して、指定した値の画面位置を計算することができます。

```
Dim MyPos As Long
MyPos = TChart1.Axis.Left.CalcYPosValue(100)
MsgBox MyPos
```

ここでの MyPos は、TChart1 の左軸(垂直)の「100.0」に対するピクセル座標値を持ちます。「100.0」は浮動少数値です。
この座標値を使用して、カスタム描画を行ったり、あるいはマウスクリックを計算したりすることができます。
ピクセル座標値から軸値に変換したい場合、反対の関数を使用することができます。

CalcPosPoint

```
Dim MyPos As Double
MyPos = TChart1.Axis.Left.CalcPosPoint(100)
MsgBox MyPos
```

ここでの MyValue は、TChart1 の左軸(垂直)での「100」に対する軸値の座標値を持っています。「100」は、画面ピクセル座標値です。

メモ: ピクセル座標値は、0(チャートの起点ポイント0,0)から始まります。これは、画面に対して有効ですが、メタファイルに描く場合、カスタムキャンバスに描く場合、あるいは印刷する場合には、チャートの起点ポイントはオプションで 0,0 以外の値にすることができます。

CalcSizeValue

軸には、指定された軸範囲を表示するのに必要な画面スペースを計算する別の関数があります。

```
Dim Space As Long
Space = TChart1.Axis.Left.CalcYSizeValue(1000)
```

軸は、DateTime にすることができるということに注意してください。たとえば日付範囲をピクセル単位で変換することができます。

```
Dim Space As Long
Space = TChart1.Axis.Bottom.CalcXSizeValue(DateValue("2008/12/31") - DateValue("2008/01/01"))
```

CalcYPos および CalcXPos

CalcYPos および CalcXPos を使用することができます。XPos および YPos 座標値を使用して描く場合、座標値 0,0 はチャートの長方形(ChartRect)の左上になり、ChartRect はチャートの 4 つの軸で囲まれる領域であることに注意してください。

次の例は、Y 軸の任意のポイントからチャート上に線を描きます。キャンバスプロパティを使用していることに注意してください。

```
Private Sub Command3_Click()
Dim MyHalfwayPoint, YPosition As Integer
With TChart1.Series(0).YValues
    MyHalfwayPoint = CInt(((.Maximum - .Minimum) * 0.5) + .Minimum)
    ' 上の画面ピクセル座標値を計算します。
    YPosition = TChart1.Axis.Left.CalcYPosValue(MyHalfwayPoint)
    With TChart1.Canvas
      ' ペンを変更し、線を描画する。
      .Pen.Width = 3
      .Pen.Style = psSolid
      .Pen.Color = vbBlack
      With TChart1
        .Canvas.MoveTo .Axis.Bottom.CalcXPosValue(0), YPosition
        .Canvas.LineTo .Axis.Bottom.CalcXPosValue(0) + .Aspect.Width3D, YPosition - .Aspect.Height3D
        .Canvas.LineTo .Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)+ .Aspect.Width3D, YPosition - .Aspect.Height3D
      End With
  End With
End With
End Sub
```

系列の値から画面の座標値を割り出すメソッド

系列にも座標値からポイント値への変換、およびこの反対の変換を行う似たようなメソッドがあります。主な違いは、系列メソッドを使用すると、計算の際に正確な軸クラスを理解している必要がないという点です。

これは、右または上の軸に関連する系列、あるいは各軸に関連する複数の軸がある場合に、大変便利です。

CalcPosValue

このコードは、1000 の値を持つ Series1 のポイントを画面上のどこに置くかを計算します。

```
Dim MyPos as Double
MyPos = TChart1.Series(0).CalcPosValue(100)
```

CalcXPos および CalcYPos

特定のポイントあるいは特定のポイント値に対する、X および Y の両方の座標値を計算することができます。

```
MyXPos = TChart1.Series(0).CalcXPos(TChart1.Series(0).XValues.Last)
' 最後のポイント
あるいは
MyXPos = TChart1.Series(0).CalcXPos(TChart1.Series(0).XValues.Locate(DateValue("1/12/1999")))
```

XScreenToValue および YScreenToValue

画面ピクセルをポイント値に変換するには、次のコードを使用してください(水平座標値用は XScreenToValue)。

```
Dim MyValue as Double
MyValue = TChart1.Series(0).YScreenToValue(Y)
```

Series.Clicked 関数を使用して、指定した XY 画面座標値の下のポイントを問い合わせることができます。

チャートのキャンバス

Canvas プロパティを使用してチャートの外観を制御することができます。

キャンバスへの書き込み

以下の例では、チャートの長方形の背景領域を5等分のセグメントに分割し、色配列に従ってこれらに色付けを行います。このコードを TChart の OnBeforeDrawSeries イベントに入れてください。

```
Dim MyColors(5) As ColorConstants
Dim t, partial, tmpLeft, tmpTop, tmpRight, tmpBottom As Integer
    MyColors(0) = RGB(215, 0, 0)
    MyColors(1) = RGB(255, 128, 0)
    MyColors(2) = RGB(128, 128, 255)
    MyColors(3) = RGB(0, 0, 153)
    MyColors(4) = RGB(215, 0, 0)

With TChart1
  ’描画する領域を設定します。
  tmpTop = TChart1.Axis.Left.CalcYPosValue(TChart1.Axis.Left.Maximum)
  tmpBottom = TChart1.Axis.Left.CalcYPosValue(TChart1.Axis.Left.Minimum)
  tmpLeft = TChart1.Axis.Bottom.CalcXPosValue(TChart1.Axis.Bottom.Minimum)
  tmpRight = tmpLeft
  ’チャート全体の幅を 5 で割る
  partial = (TChart1.Axis.Bottom.CalcXPosValue(TChart1.Axis.Bottom.Maximum) - tmpLeft) / 5

  ’ ブラシスタイルを変更する
  .Canvas.Brush.Style = bsSolid
  .Canvas.Pen.Style = psSolid

  ’各セクションに対して、特定の色で塗りつぶす
  For t = 0 To 4
    ’矩形の大きさを調整する
    tmpRight = tmpRight + partial
    ’ブラシの色を設定する
    .Canvas.Brush.Color = MyColors(t)
    .Canvas.Pen.Color = MyColors(t)
    ’ペイント !!!
    .Canvas.Rectangle (tmpLeft + TChart1.Aspect.Width3D), _
                      (tmpTop - TChart1.Aspect.Height3D), _
                      (tmpRight + TChart1.Aspect.Width3D), _
                      (tmpBottom - TChart1.Aspect.Height3D)
    ’矩形を調整する
    tmpLeft = tmpRight
  Next t
End With
```

内部ビットマップ

TChart コンポーネントは内部ビットマップオブジェクトを持っています。これは、非表示の「バッファ」を描く際に使用されます。描画が終了すると、この「バッファ」は、これを表示するために画面のビデオメモリにコピーされます。

TChart Canvas プロパティは、内部ビットマップキャンバスオブジェクトを返します。

メモ: メタファイルに描画する場合、あるいは印刷する場合には、TChart Canvas プロパティはメタファイルあるいはプリンタキャンバスオブジェクトを参照します。これらの場合、ビットマップは使用されません。

チャートが内部ビットマップに描画され、画面キャンバスにコピーされた後で、TChart Canvas プロパティは、その基となった「本物の」チャートキャンバスを参照します。

再描画

チャートコンポーネントを強制的に再描画させるためには、TChart1.Repaint を呼び出してください（ほとんどのケースでは、このメソッドを呼び出す必要はありません）。

いつ描くか

キャンバスに描画する順序が重要です。

チャート上でカスタム描画を行うのに最も適した場所は、

1.チャート系列上にカスタム描画アイテムを表示したい場合、TChart1.**OnAfterDraw** イベントを使用してください。

TChart1.**OnAfterDraw** イベントは、チャートが再描画される度に、画面にビットマップをコピーする直前に発生します。

2.系列の OnBeforeDrawSeries イベントにコードを入れて、チャートのグリッド上やチャートの系列の下にカスタム描画アイテムを配置することもできます。

チャート領域

軸は、チャートパネルの内側に描画されます。軸の Position プロパティは、軸の位置を返します。

3D のチャートでは、各系列は、Z 軸（奥）上のそれぞれの位置に表示されます。

チャートの Width3D および Height3D は、3D チャートのオフセット（Z 軸（深さ）方向の位置）をピクセル単位で指定します。この指定した位置にカスタムな描画を移動することができます。

描画

基本的な例を用いて、描画を開始してみましょう。

このコードは、TChart1 のちょうど中心に水平線を描きます。

```
Private Sub TChart1_OnAfterDraw()
With TChart1
    .Canvas.Pen.Color = vbYellow
    .Canvas.MoveTo .Canvas.Left, ((.Canvas.Height) / 2)
    .Canvas.LineTo .Canvas.Left + .Canvas.Width, ((.Canvas.Height) / 2)
  End With
End Sub
```

軸スペースの中に描きます。

```
Private Sub TChart1_OnAfterDraw()
  With TChart1
    .Canvas.Pen.Color = vbYellow
    .Canvas.MoveTo .Axis.Left.Position, .Axis.Top.Position + ((.Axis.Bottom.Position - .Axis.Top.Position) / 2)
    .Canvas.LineTo .Axis.Right.Position, .Axis.Top.Position + ((.Axis.Bottom.Position - .Axis.Top.Position) / 2)
  End With
End Sub
```

Series(0)の各ポイントに線を描きます。

```
Private Sub TChart1_OnAfterDraw()
Dim i, gridwidth
With TChart1
  .Canvas.ClipRectangle .Axis.Left.Position, .Axis.Top.Position, .Axis.Right.Position, .Axis.Bottom.Position
  If .Series(0).Count > 0 Then
    gridwidth = .Series(0).CalcXPos(1) - .Series(0).CalcXPos(0)
```

```
        .Canvas.Pen.Width = 2
        .Canvas.Pen.Color = .Series(0).Color
        For i = 0 To .Series(0).Count - 1
          If i > 0 Then
            .Canvas.LineTo .Series(0).CalcXPos(i) - (gridwidth / 2), .Series(0).CalcYPos(i)
          Else
            .Canvas.MoveTo .Series(0).CalcXPos(i) - (gridwidth / 2), .Series(0).CalcYPos(i)
          End If
          .Canvas.LineTo .Series(0).CalcXPos(i) + (gridwidth / 2), .Series(0).CalcYPos(i)
        Next i
      End If
    End With
    End Sub
```

メモ: テキストを描く

画面上やプリンタやメタファイルで同じフォントサイズが必要であれば、Font.Size を使用する代わりに、常に TChart1.Canvas.Font.Height を負の値に設定してください。

5．4．軸の操作

軸スケールの設定

チャートクラスには 5 つの軸、LeftAxis、RightAxis、TopAxis、BottomAxis、DepthAxis があります。各軸は、Axis クラスのインスタンスになっています。

軸は、系列ポイントのピクセル座標値を計算し、スクロールやズームがいつでも行えるように有効な範囲を割り当てます。新しい系列が挿入されると、あるいは新しいポイントが系列に追加されると、軸はデフォルトでその最小値および最大値を再計算します。

Automatic プロパティを False に設定することにより、軸スケールの自動再計算をオフにすることができます。

```
TChart1.Axis.Left.Automatic = False
```

また、軸の最小値および軸の最大値はともにオプションで、それぞれ独立して「自動」（軸のスケールの「自動」チェックボックス）をオン／オフにすることができます。

```
TChart1.Axis.Left.AutomaticMaximum = False
TChart1.Axis.Left.AutomaticMinimum = True
```

Minimum プロパティや Maximum プロパティを使用して、軸スケールを変更することができます。

```
With TChart1.Axis.Left
    .Automatic = False
    .Minimum = 0
    .Maximum = 10000
End With
```

また、Axis の SetMinMax メソッドを使用しても変更できます。

```
TChart1.Axis.Left.SetMinMax 0, 10000
```

DateTime 軸

メモ:

関連付けられた系列コンポーネントの XValues.DateTime プロパティあるいは YValues.DateTime プロパティが True の場合、軸は DateTime スケールになります。軸のための DateTime プロパティはありません。

DateTime 軸でスケールを変更することは、非 DateTime 値に対するものと同じことです。

```
With TChart1.Axis.Left
    .Automatic = False
    .Minimum = DateValue("1990/03/16")
    .Maximum = DateValue("1999/05/24")
End With
```

Logarithmic Axis（対数軸）

軸は、軸の最小値や最大値がゼロ以上の場合にのみ、Logarithmic（対数）にすることができます。これは、直線に設定する場合と対数スケールに設定する場合の唯一の違いです。

メモ:

軸ラベルは、対数インクリメントでは表示されません。カスタムな軸ラベルを生成することができます。

Inverted Axis（反転軸）

軸の最小値や最大値を入れ替えるように軸を反転させることができます。この使用によって間違った結果になってしまう可能性を最小限にするために、できるだけ Inverted = True を使用しないことをお薦めします。

軸スタイルとインクリメント

軸は目盛りの線や、グリッド、ラベルなどを付けたり付けなかったりしてさまざまな方法で表示することができます。色、フォント、ペンスタイルなどすべての書式に関するプロパティをカスタマイズすることができます。軸の Increment プロパティは、グリッド線とラベルの数、およびこれらの間隔を制御します。

デフォルトでは、この値はゼロになっており、軸はラベルインクリメントを自動的に計算します。軸の Increment プロパティの設定と、軸スケールの設定とは別個のものです。自動の軸最大値および軸最小値を設定して、さらにインクリメントを固定に設定することもできます。

Increment プロパティは、すべての軸ラベルが表示されるまで、自動的に大きくなります。この自動機能を無効にするには、軸の LabelsSeparation をゼロに設定してください。

```
TChart1.Axis.Left.Labels.Separation = 0
```

警告：

LabelsSeparation がゼロになっていると、ラベルサイズに対するチェックが一切行われませんので、ラベルが重なり合わないように注意してください。

以下のコードは、垂直方向の軸のインクリメントを 30 に設定しています。

```
TChart1.Series(0).Clear
TChart1.Series(0).Add 20, 50, 120
TChart1.Axis.Left.Increment = 30
```

デフォルトでは、最初の軸ラベルは一番近いインクリメントから開始されます。RoundFirstLabel を False に設定すると、ラベルは軸の最大値から開始されます。

```
TChart1.Axis.Left.RoundFirstLabel = False
```

DateTime インクリメント

DateTime 軸の軸インクリメントを指定する場合には、定義済みの DateTimeStep 定数配列を使用してください。

```
TChart1.Axis.Bottom.Increment=TChart1.GetDateTimeStep(dtOneMonth)
```

軸ラベルをちょうど DateTime の境界、たとえば月の最初の日などにする場合は、ExactDateTime プロパティを True に設定します。

```
TChart1.Axis.Bottom.ExactDateTime = True
```

メモ：

対数軸では、Increment プロパティを直線として使用します。

グリッド線

軸のグリッド線は、各インクリメント位置、あるいは各軸ラベルの位置に表示されます。軸の TickOnLabelsOnly プロパティは、この機能を制御します。

```
TChart1.Axis.Bottom.TickOnLabelsOnly = False
```

軸ラベル

軸ラベルにはさまざまなスタイルがあります。軸の LabelStyle プロパティは、軸ラベルのモードを制御します。

```
TChart1.Axis.Bottom.Labels.Style = talValue
```

LabelStyle の有効な値を次に示します。

talValue	ラベルは軸スケールを表示します。
talMark	ラベルは系列ポイントマーカを表示します。
talText	ラベルは系列の XLabels を表示します。
talNone	ラベルは表示されない。
talAuto	スタイルは自動的に計算されます。

LabelStyle が talText になっている場合、系列に XLabels がないと、TeeChart はこれを自動的に talValue に設定します。
talMark や talText スタイルの場合、軸ラベルはちょうど系列ポイントの位置に表示されるため、軸の Increment プロパティは使用しません。
OnGetAxisLabel イベントを使用して、ラベルテキストをカスタマイズすることができます。

```
Private Sub TChart1_OnGetAxisLabel(ByVal Axis As Long, ByVal SeriesIndex As Long, ByVal ValueIndex As Long,
LabelText As String)
    If Axis = atLeft then
        If ValueIndex < 2 then
            LabelText = ""
        End if
    End if
End Sub
```

複数軸

TeeChart は、無制限にカスタム軸をサポートします。軸は、デザイン時にはチャートエディタを使用し、実行時にはコード/エディタを使用して追加できます。

例 (Visual Basic)

```
Dim numVertaxis, numHorizaxis

With TChart1.Axis
  'Series(2)用のカスタム軸を作成します。
  numVertaxis = .AddCustom(False)
  numHorizaxis = .AddCustom(True)
  With .Custom(numVertaxis)
    .EndPosition = 45
    .PositionPercent = 7 / 15 * 100
    .AxisPen.Color = vbWhite
    .Ticks.Color = vbWhite
    .Title.Angle = 90
    .Title.Font.Color = vbWhite
    .Title.Caption = "Variance"
    .Title.Font.Color = vbWhite
    .Labels.Font.Color = vbButtonFace
  End With
  With .Custom(numHorizaxis)
    .StartPosition = 7 / 15 * 100
    .AxisPen.Color = vbWhite
    .Ticks.Color = vbWhite
    .PositionPercent = 55
    .Labels.Font.Color = vbButtonFace
  End With
  'Series(2)に結合します
  TChart1.Series(2).VerticalAxisCustom = numVertaxis
  TChart1.Series(2).HorizontalAxisCustom = numHorizaxis
End With
```

CustomDraw(軸)

CustomDraw は軸をコピーするだけであるという点で複数軸とは異なります。独自の Axis オブジェクトを作成しません。軸をコピーすることにより好きなだけ追加できます。CustomDraw は既存の軸を新しい位置に「コピー」することにより軸を追加します。CustomDrawMinMax および CustomDrawMinMaxStartEnd も参照してください。TeeChart の複数の軸機能も使用すると、よりチャートの強化ができます。

以下の例では、リボン系列をランダムデータを使って登録しています。そして、軸を 2 つ追加作成しています。
ラベル、タイトル、軸を置く場所として、それぞれPosLabels、PosTitle、PosAxisという3つの設定可能な位置があります。最後のパラメータである GridVisible は、論理パラメータで、軸グリッドを新しい軸にまで伸ばすかどうかを定義します。

```
Private Sub Command1_Click()
Dim t As Integer
  For t = 0 To 20
    TChart1.Series(0).AddXY t, ((100 * Rnd) + 1) - ((Rnd * 70) + 1), "", vbRed
  Next t
End Sub

Private Sub TChart1_OnBeforeDrawSeries()
    Dim posaxis As Integer
    With TChart1
        ’スクロールあるいはズームする場合、常にグリッド線がチャート矩形に含まれる状態にします。
        .Canvas.ClipRectangle .Axis.Left.Position, _
        (.Axis.Left.CalcYPosValue(.Axis.Left.Maximum)), _
        (.Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)), _
        .Axis.Bottom.Position
        ’常に、チャートの中間ポイントに 2 番目の垂直軸を描画します。
        posaxis = (.Axis.Left.Position) _
                + (((.Axis.Bottom.CalcXPosValue(.Axis.Bottom.Maximum)) _
                - (.Axis.Left.Position)) * 0.5)
        .Axis.Left.CustomDraw posaxis - 10, posaxis - 20, posaxis, True
        ’垂直軸上の 10 レベルの所に 2 番目の水平軸を描画します。
        posaxis = (.Axis.Left.CalcYPosValue(10))
        .Axis.Bottom.CustomDraw posaxis + 10, posaxis + 40, posaxis, True
        .Canvas.UnClipRectangle
    End With
End Sub
```

5. 5. 系列の操作

ここでは、系列の操作方法、および系列ポイントとその他の系列内部データの操作方法について説明します。

実行時に系列を作成
系列は実行時に作成することができます。
例.

```
TChart1.AddSeries  scLine
```

他のデザイン時に作成した系列と同じように、系列にポイントを追加し制御できます。
ESeriesClass scLine は、リボン系列を追加します。(例:scBar は、縦棒系列を追加します。scArea は、面系列を追加します)。
ESeriesClass のすべてのオプションについては、オンラインヘルプを参照してください。

系列配列プロパティ
チャートクラスは、系列リストにすべての系列を格納します。
系列配列(Series)プロパティを使用することにより、このリストに読み込み専用でアクセスできます。

```
TChart1.Series(0) 'これは、チャートの最初の系列を表します。
```

ExchangeSeries メソッドを使用することにより、系列のインデックスの順番で系列を交換することができます。

```
TChart1.ExchangeSeries 1, 0
```

SeriesCount プロパティ
TChart1.SeriesCount プロパティは、系列リストにある系列の数を返します。
全てのチャート系列を調べる場合に、SeriesCount を使用することができます。

```
For t = 0 To TChart1.SeriesCount - 1
    With TChart1.Series(t)
        .Color = vbBlue
    End With
Next t
```

系列の削除
系列を非表示にするには、2 つの方法があります。

A) 系列の Active プロパティを設定します。
TChart1.Series(0).Active = False

B) 系列を完全に削除します。
TChart1.RemoveSeries(0)

実行時に系列の Z オーダー(順序)を変更

3D モードでは(TChart1.View3D が True になっている場合)、全ての系列が Z オーダー位置に設定されます。これは、系列が描かれる順序が、チャートの 3D 面の中で最も遠い系列から開始することになります。
以下のメソッドを使用して、系列が描かれる順序を制御することができます。

```
TChart1.ExchangeSeries 0, 1
```

ポイントの追加

全ての系列型は、各ポイントに対して少なくとも 2 つの値を持っています。これらの値は、X や Y ポイント座標値として設計されます。
メモ: 値は、「Double」の浮動小数点変数として定義されます。
拡張系列型は、泡系列が各ポイントに対して X、Y、および半径値をもっているように、2 つ以上の値を持っています。
従って、各系列型にはポイントを追加するのに適切なメソッドがありますが、リボン、縦棒、散布図および面など最もよく使用される系列型は、ポイントを追加するのに一般的な Add メソッドを共有しています。
以下のコードは、円系列を空にし、これにいくつかのサンプル値を追加します。

```
TChart1.Series(0).Clear
TChart1.Series(0).Add 1234, "USA", vbBlue
TChart1.Series(0).Add 2001, "Europe", vbRed
```

拡張系列型でポイントを追加するにはどのメソッドを使用するかについては、ヘルプのそれぞれ特定の系列リファレンスを参照してください。

系列にデータ配列を直接追加

系列にデータ配列を直接追加すると、Add メソッドあるいは AddXY メソッドを使用してデータを追加するより速くできます。数百ポイント追加する場合、速さの違いはごくわずかですが、ポイントの数が何十万あるいは数百万以上になるとかなり違ってきます。利用可能な 2 つの AddArray メソッドがあります:
AddArray – X 値や Y 値を持つ 2D 系列
AddArrayXYZ – X 値、Y 値、Z 値を持つ 3D 系列
AddArray メソッドは、3 つの引数を設定します。
例.

```
Dim a, b
  a = Array(2, 3, 4, 5, 7, 1, 4, 5, 2)
  b = Array(4, 1, 8, 6, 2, 3, 3, 7, 1)
  TChart1.Series(0).AddArray UBound(a) + 1, a, b
```

最初の変数は配列のサイズを設定しますが、最初の X 値のみ追加したい場合は数値になります。最後の変数(ここではb)が省略された場合、Add メソッドと同様に X 値に 0、1、2、などの連続したインデックス値を自動的に割り当てます。

Null 値

状況によっては、特定のポイントに対して値が無いことがあります。そのような場合は、それらのポイントを「ゼロ」として追加するか、「ヌル」値として追加します。
「ゼロ」だと通常どおり表示されますが、ヌル値の場合は表示されません。
以下のコードでは、いくつかのポイントとヌルポイントを追加しています。

```
With TChart1.Series(0)
  .Clear
  .Add 10, "John", vbGreen
  .Add 20, "Anne", vbYellow
  .Add 15, "Thomas", vbRed
  .AddNull "Peter"
  .Add 25, "Tony", vbBlue
  .Add 20, "Mike", vbCyan
```

```
    End With
```

各系列型は、ヌル値をそれぞれ異なった方法で処理しています。縦棒、横棒、リボン、面および散布図は、ヌルポイントを表示しません。円系列は、ヌル値を「ゼロ」として使用します。

ポイントオーダーの制御

ポイントは、オプションで X 値あるいは Y 値でソートすることができます。Series.XValues や Series.YValues の Order プロパティは、ポイントの順序を制御します。

```
    TChart1.Series(0).XValues.Order = loAscending
```

使用できる値は、loNone、loAscending、loDescending です。

デフォルトでは、XValues の Order は loAscending に、YValues の Order は loNone に設定されています。これは、新しく追加されたポイントは、X 座標値に従って順序付けされるということになります。非 XY チャートでは、X 座標値は常にゼロから開始するポイント位置になります。

ポイント順は、系列クラスがポイントを描くのに使用します。

メモ: 順序は、ポイントを系列に追加する**前**に設定しなければなりません。

Order プロパティは実行時に変更でき、その変更後に Sort メソッドを呼び出します。

例:

TChart をフォーム上に置き、リボン系列を追加します。

Command1 を置き、このコードを Command1_Click に入れます。

```
    With TChart1.Series(0)
      .Clear
      .Add 10, "John", vbGreen
      .Add 20, "Anne", vbYellow
      .Add 15, "Thomas", vbRed
      .Add 25, "Tony", vbBlue
      .Add 20, "Mike", vbCyan
    End With
```

ここで、別の Command2 を置き、このコードを Command2_Click に追加します。

```
    With TChart1.Series(0)
        .YValues.Order=loAscending
        .YValues.Sort
    End With
    TChart1.Repaint
```

ここで実行し、Command1 をクリックして Series(0)を描画します。次に、Command2 をクリックして Series(0)ポイントが Series(0)の YValues の昇順で描かれても、オリジナルの X 座標値を持っていることを確認します。

メモ: X 座標値を使用していない場合、コードがもう 1 行必要です。

Command3 を置き、Command3_Click に以下のコードを入れてください。

```
    TChart1.Series(0).XValues.FillSequence
    TChart1.Repaint
```

新しいポイントは、Series1 の XValues 軸の中でゼロから開始する番号で「再番号付け」されます。これは、ポイントの順序をつけ直します。つまりここでは、ポイントは新しい順序でもともと系列に追加されていたかのように描かれます。

この「2 ステップ」でのポイントのソートにより、リボン系列を垂直方向に描くことができます。

XY ポイント

X 座標値をポイントに追加すると、系列コンポーネントがユーザ固有の水平方向のポイント位置を計算します。

メモ: バーチャートでは、X 座標値での解釈が難しい場合があります。

円系列では X 座標値は使用できません。

X 座標値は、AddXY メソッドを使用するだけで追加できます。

TChart を置き、散布図系列を追加します。

```
    With TChart1.Series(0)
      .Clear
      .AddXY 10, 10, "Barcelona", vbBlue
      .AddXY 1, 10, "San Francisco", vbRed
```

```
    End With
```

メモ: 泡系列を使用している場合には、BubbleSeries.AddBubble メソッドを使用してください。

例:

```
    TChart1.Series(0).asBubble.AddBubble 3, 4, 3, "", clTeeColor
```

ポイントを X 座標値でソートしたくない場合は、XValues.Order を loNone に設定してください。

ポイントの制限

1 つの系列で最大ポイント数は 134217727 です。1 つのチャートで最大系列数は 134217727 です。

ポイントの削除

ポイントインデックスを引数として渡して、Series.Delete メソッドを呼び出すだけです。ポイントインデックスはゼロから開始します。

```
    TChart1.Series(0).Delete (0)  ' Series1 の最初のポイントを削除します。
    TChart1.Series(0).Delete (TChart1.Series(0).Count - 1)  ' Series1 の最後のポイントを削除します。
```

存在しないポイントを削除しようとすると、「領域外のリスト」という例外が発生します。常にポイントを削除する前には、ポイントが系列にあることを必ず確認してください。

```
    If TChart1.Series(0).Count > MyIndex then
      TChart1.Series(0).Delete(MyIndex)
    End If
```

Delete を呼び出すと、関数の再計算とチャートの再描画を行います。

系列ポイント

ポイントの取り出しと変更

ポイントを追加すると、ポイントの座標値を取り出したり、あるいは変更したりすることができます。

XValues プロパティや YValues プロパティを使用することができます。

```
    Dim MyValue as Double

    MyValue = TChart1.Series(0).YValues(0).Value' 最初の Y 値を取得します。
```

これらの配列を検索し、計算を実行することができます。

```
    Dim MyTotal As Double
    Dim t As Integer

    MyTotal = 0
    For t = 0 To TChart1.Series(0).Count - 1
        MyTotal = MyTotal + TChart1.Series(0).YValues.Value(t)
    Next t

    MsgBox Str(MyTotal)
```

拡張系列型は、BubbleSeries.RadiusValues などの追加配列プロパティを持っています。XValues あるいは YValues 配列と同じ方法で、これらのプロパティにアクセスすることができます。

```
    If TChart1.Series(0).asBubble.RadiusValues.Value(Index)>100 then
```

上記プロパティを使用して、ポイント値の変更を行うことができます。

```
        TChart1.Series(0).YValues.Value(0) = TChart1.Series(0).YValues.Value(0) + 1
        TChart1.Series(0).RefreshSeries
    End If
```

ポイントの検索

XValues や YValues の Locate 関数は、リストで特定の値を検索し、見つかった場合、ゼロから始まる値インデックスを返します。

```
    Dim MyIndex as Integer

    MyIndex = TChart1.Series(0).YValues.Locate( 123 )

    If MyIndex = -1 then
```

```
    Msgbox("  123 not found in Series(0) !! ")
  Else
    Msgbox("  123 is the "+Str$( MyIndex+1 )+"th point in Series(0) !!")
  End If
```

ポイントの統計

XValues プロパティや YValues プロパティは、以下の統計値を保持しています。

Total	リスト内の全ての値の合計
TotalABS	全ての値の合計の絶対値（正の値）
MaxValue	リストの最大値
MinValue	リストの最小値

コードで RecalcMinMax メソッドを呼び出して MinValue および MaxValue を再計算することもできます。Total および TotalABS は自動的に維持されます。
これらの値は、一部の軸スケールの計算のスピードアップに使用されたり、またパーセント計算にも役立ちます。
TChart の IValueList オブジェクトは、ポイント値を操作するためのその他のメソッドとプロパティをいくつか持っています。詳しくはオンラインヘルプを参照してください。

通知

ポイントを追加、削除、あるいは変更する場合、系列は必ず内部通知イベントを発生します。これらのイベントは、他の系列ポイントに依存している系列の再計算を行うために使用されます。

```
TChart1.Series(0).RefreshSeries
```

これは、強制的に従属系列のポイントを再計算させます。

ポイントの色

すべての系列は、色の内部リストを持っています。系列の各ポイントに対して 1 つのリストがあります。
PointColor 配列プロパティを使用して、このリストにアクセスし、ポイントの色を取り出したり変更したりすることができます。

```
Dim MyColor As Long

MyColor = TChart1.Series(0).PointColor(0)

TChart1.Series(0).PointColor(1) = vbBlue
```

TeeChart は、clTeeColor という名前の汎用色定数を定義します。
clTeeColor 色のポイントは、SeriesColor 色で描かれます。
Visual Basic では、標準カラー（vbBlue、vbRed など）の定数をあらかじめ定義しています。カラーは RGB 形式も使用できます。画面の色を 16k 色以上にすると、より綺麗に表示されます。

色の定数をサポートしていない環境では、RGB カラーで定義できます。
例. MS VC++

```
m_Chart1.GetSeries(0).SetPointColor(1,RGB(0,0,255));
```

VBScript

```
TChart1.Series(0).PointColor(1) = RGB(0,0,255);
```

ポイントラベル

各ポイントは、String として宣言された XLabel と呼ばれる関連したテキストを持っています。
ポイントラベルは、軸ラベル、チャート凡例、およびポイントマーカで使用されます。
ラベルは、系列の PointLabel 配列プロパティに格納されます。
PointLabel ポイントテキストにアクセスしたり、変更したりすることができます。

```
TChart1.Series(0).PointLabel(0) = "Sales"
```

実行時に系列型を変更

すべての系列型は、それぞれ異なるコンポーネントのクラスに対応しています。
系列型を変更することは、系列コンポーネントのクラスを変更することになります。

つまり、新しいクラスの系列を新規に作成し、次に古い系列プロパティをこの新しいクラスのインスタンスに代入します。そして最後に古い系列を破棄する必要があります。
チャートエディタダイアログおよびギャラリを使用して設計時に系列型を手動で変更する場合、上記のことはすべて自動的に行われます。
次のコードを呼び出して、実行時に系列型を変更することができます。

```
TChart1.ChangeSeriesType Self, MySeries
```

警告:
変更できるのはプライベートな系列クラスです。
フォームに付随している系列クラスも変更できますが、変更後は使用することができません。

```
TChart1.ChangeSeriesType 0, scBar
' 最初の系列を縦棒系列に変更します。
```

変更前と変更後の変数の数が異なる場合、系列型は変更できません。(たとえば、縦棒系列は X と Y で、キャンドル系列は High、Open、Low、Close)

系列型の特性

TeeChart Pro ActiveX コントロールは、統一した環境で異なるタイプの系列が動作するようにするため、内部的にクラスを継承して使用します。これが、1 つのチャートで多くの異なる系列を混在して柔軟に使用できるという TeeChart の大きな特徴です。

例

共通の系列プロパティは、系列クラスを使用してアクセスできます。

```
TChart1.Series(0).Color = vbBlue
```

系列の特定のプロパティは、系列型のインタフェースを使用してアクセスが可能です。たとえば、キャンドル系列の場合は、チャートの他の系列と同じように TChart.Series(Index)で指定し、この特定のプロパティ、メソッドは下記のようにしてアクセスできます。

```
TChart1.Series(0).asCandle.AddCandle DateValue("2,12,97"),  110,102,119,114
```

これらを混在しても構いません。

```
With TChart1.Series(0)
   .Clear
   .Title = "My Candle Series"
   .asCandle.UpCloseColor     = vbGreen
   .asCandle.DownCloseColor = vbYellow
   .asCandle.AddCandle DateValue("2,12,97"),110,102,119,114
End With
```

系列の特定のプロパティやメソッドは、系列プロパティを使用してアクセスできます。
asArea、asArrow、asBar、asBubble など。詳しくはオンラインヘルプを参照してください。

5．6．TeeChartツール

TeeChart は前バージョンで技術サポートとして要求されたタスクを容易にするツールを提供します。これらには、系列ポイントのヒント、ドローライン、カーソル、カラーバンドなどの機能が含まれています。
チャートエディタのツールページを使用してデザイン時に全てのツール型を追加できます。チャートにツールが設定されている場合、ツールリストにアクセスしてツールをアクティブや非アクティブにできます。

デザイン時にツールを追加
デザイン時にツールを追加するには:
- チャートエディタを開きます。
- ツールページをクリックします。
- 「追加」ボタンを選択します。
- リストからツールを選択します。

各ツールは、エディタで定義される独自の特性を持ちます。たとえば「カラーバンド」ツールは、軸スケールで設定された位置に指定した色のバンドを描画します。各ツールに関連する詳細については、ヘルプファイルのインタフェース定義を参照してください。

実行時にツールを追加
実行時にツールを追加する方法:
例 - カラーバンドツール (Visual Basic)

```
With TChart1
  .Series(0).FillSampleValues 10
  .Tools.Add tcColorband
  With .Tools.Items(0).asColorband
    .EndValue = TChart1.Series(0).YValues.Maximum - 40
    .StartValue = TChart1.Series(0).YValues.Minimum + 40
  End With
End With
```

ToolList の操作
チャートにいくつかの異なるツールを追加することができます。各ツールは、共通プロパティ(Active、Description、ToolType)や、'Items(xx).asToolType'プロパティによりアクセスできる、各ツール独自の特性を持ちます。
例 - マーカチップツールの遅延時間を変更します。(Visual Basic)

```
TChart1.Tools.Items(0).asMarksTip.Delay = 500   'ミリ秒
```

チャート内に複数のツールが存在する場合、プロパティ変更を適用する前に ToolType を確認することができます。
例 (Visual Basic)

```
Dim i As Integer
  With TChart1.Tools
    For i = 0 To .Count - 1
      Select Case .Items(i).ToolType
        Case tcMarksTip: .Items(i).asMarksTip.Style = smsLabelPercent
        Case tcColorband: .Items(i).asColorband.Brush.Style = bsBDiagonal
      End Select
    Next i
  End With
```

詳細については、TeeChart のオンラインヘルプ ITools や IToolList インタフェース定義を参照してください。

5．7．チャートの印刷

ここでは、チャートの印刷方法、および印刷プロセスを制御する場合に使用するプロパティとメソッドについて説明します。

印刷

マージン

印刷の際、用紙のマージンを定義できます。PrintMargins プロパティは、全ページサイズに対するパーセントで用紙マージンを設定します。

```
‘ デフォルトは 15%の印刷マージンです。
With TChart1.Printer
    .MarginLeft = 15
    .MarginTop = 15
    .MarginRight = 15
    .MarginBottom = 15
    .PrintLandscape
End With
```

PrintMargins を使用して、ページ内の任意のサイズの領域でも定義することができます。

メモ:

印刷の向きを変更する場合は、マージンは再計算されます。

解像度

チャートはメタファイル（16 ビット WMF、32 ビット EMF）フォーマットで印刷されます。

メタファイルはスケーラブルなベクトルフォーマットなので、プリンタに送ると「wysiwyg」効果（チャートが画面に表示されている状態）が得られます。ただし、画面表示に対するプリンタのより大きな解像度が必要です。

TChart1.Printer.Detail プロパティは、チャートをプリンタに送る際にどのようにチャートをスケーリングするかを制御します。デフォルトでは、Printer.Detail はゼロになっています。これをゼロから-100 までの負の数値にすると、チャートはその値に比例して大きくなるので、軸ラベルに使用できるスペースが増えます。画面で見るより用紙の方がはっきりと見えるので、より小さいフォントが使用できます。

これを正の値に設定すると、フォントのサイズが大きくなります。

Print、PrintLandscape など

チャートコンポーネントを印刷するには、以下のようなメソッドがあります。

```
TChart1.Printer.PrintChart
TChart1.Printer.Print  ’PrintChart と同様
TChart1.Printer.PrintPortrait
TChart1.Printer.PrintLandscape
```

上記メソッドのすべては、同じ処理を行います。つまり、新しいページにチャートを印刷し、ページを**イジェクト**(フォームフィード)します。

```
TChart.Printer.Orientation(AOrientation:TPrinterOrientation)
```

上記コードは、印刷の向きを変更します。

```
TChart1.Printer.Orientation
```

現在の用紙の印刷方向で TChart1 を印刷します。

```
TChart1.PrintChart
```

PrintPartial

以下のメソッドを使用すると、印刷に関してさらに高度な制御が行えます。PrintPartial ジョブは改ページを行わないので、同じページ/印刷ジョブに 1 つ以上のチャートを送れることができます。TeeChart の印刷ジョブを開始/終了させるために TChart1.Printer.BeginDoc メソッドおよび TChart1.Printer.EndDoc メソッドを使用してください。

```
TChart1.Printer.BeginDoc
TChart1.Printer.PrintPartial 0, 0, 400, 400
TChart1.Printer.EndDoc
```

上記は、画面座標を使用し、印刷ページ上のチャートは「小さく」なります。チャートの印刷を定義するには、プリンタの用紙サイズに合わせて使用することをお薦めします。

例.

```
With TChart1.Printer
    .BeginDoc
    .PrintPartial 0, 0, .PageWidth, .PageHeight / 2
    .EndDoc
End With
```

BeginDoc メソッドは、印刷を開始します。EndDoc メソッドは、印刷を終了します。TeeChart のチャートと、TeeChart 以外の何かを一緒に出力したい場合は、PrintPartialHandle メソッドを使用してください。

例.

```
Private Sub Command2_Click()
Dim HWidth, HHeight, I, Msg1, Msg2 ’変数の宣言
    On Error GoTo ErrorHandler ’エラーハンドルの設定
    Msg1 = "This is printed on page before Chart"
    Msg2 = "This is printed on page after Chart"
    HWidth = Printer.TextWidth(Msg) / 2 ’半分の幅を取得
    HHeight = Printer.TextHeight(Msg) / 2 ’半分の高さを取得
    Printer.CurrentX = Printer.ScaleWidth / 3 - HWidth
    Printer.CurrentY = Printer.ScaleHeight / 3 - HHeight
    Printer.Print Msg1 & Printer.Page & "."  ’印刷
    With TChart1.Printer
      .Orientation = poPortrait
      .PrintPartialHandle Printer.hDC, .PageWidth / 3, (.PageHeight / 3) + 10 _
            , (.PageWidth / 3) * 2, (2 * (.PageHeight / 3)) - HHeight - 10
    End With
    Printer.CurrentY = 2 * Printer.ScaleHeight / 3 - HHeight
    Printer.CurrentX = Printer.ScaleWidth / 3 - HWidth
    Printer.Print Msg2 & Printer.Page & "." ’印刷
    Printer.EndDoc ’印刷の終了
    Exit Sub

ErrorHandler:
```

```
        MsgBox "There was a problem printing to your printer."
        Exit Sub
    End Sub
```

複数のチャートを 1 ページに入れる

PrintPartial メソッドおよび PrintPartialHandle メソッドのサンプルは、TeeChart サンプルプロジェクトで使用されています。代わりに、複数のチャートを PrintPreviewPanel に送り、印刷ページにチャートを再配置するオプションがあります。

例

```
        TeePreviewPanel1.AddChart TChart1
        TeePreviewPanel1.AddChart TChart2
```

最初に追加されたチャートが再配置されるため、アクティブになります。2 番目のチャートをアクティブにするには、逆順に再追加してください。チャートの位置が変更されるのではなく、非アクティブなチャートがアクティブになります（TeeChart のデモプロジェクトを参照してください）。

Windows およびプリンタの制限

メタファイルは、小さくて高速でスケーラブルであるという点で優れています。

Microsoft Knowledge Base(www.mskb.com)において記述されるようにメタファイルを使用する際に、いくつかの制限が起こります。TeeChart はそれらの制限を引き継ぎます。

クリッピング

クリッピングは、メタファイルの物理座標値で格納されます。これは、クリップしたメタファイルを移動またはスケールすると、クリッピング領域のスケール、移動が行われず、期待した結果が得られないことを意味しています。

TeeChart はポイントの一部だけを描画できないのでクリッピングはされません。ズームしたチャートに描画するといくつかのポイントはチャート軸から外れて表示されることがあります。

囲いを丸くする

メタファイルはスケーリングできるので、スケーリングの際に囲みを楕円にできます。

フォントを回転させる

回転したフォントは、非比例的にスケーリングしたメタファイル上で正確に位置を揃えることはできません。

ドットペンスタイルおよびペン幅

ソリッド（べた塗り）でないペン（ドット、ダッシュ）は、スケーリングしたメタファイルでソリッドとして描くことができます。

FAQ からの抜粋 － TeeChart 印刷時の実際的問題

ここでは、TeeChart コンポーネントを印刷する方法および印刷を改善する方法について説明します。

はじめに

TeeChart のライブラリは、「メタファイル」と呼ばれる Windows グラフィックスフォーマットを使用して、チャートをプリンタに送ります。このフォーマットは、ビットマップフォーマットに比べていくつかの長所と短所があります。

- ピクセルではなくベクトル命令でグラフィックス情報を格納するため、サイズがかなり小さい。したがって、ほとんどの場合において印刷速度が速い。
- 「大きな」イメージ（チャート）を印刷するためにプリンタに大量のメモリを搭載する必要がない。
- ピクセルの固定配列でなく命令のシーケンスであるため、解像度をそのままに、非常に正確に、メタファイルをサイズ変更（または「引き伸ばし」）することができる。
- 新型のプリンタは、ソフトウェアドライバまたは「GDI ハードウェア」を使用して、メタファイルフォーマットをネイティブにサポートして

いる。

Windows9x 系と WindowsNT 系は、拡張版のメタファイルフォーマットに対応しています。これは「拡張メタファイル」と呼ばれています。メタファイルの拡張子は *.wmf、拡張メタファイルは *.emf です。

デザインに関する問題

TeeChart でメタファイルを使用する主な目的は、wysiwyg（what you see is what you get）を提供することです。つまり、できるだけ画面で見た通りに、チャートを印刷するということです。
そうするために、TeeChart は、チャートイメージのメタファイルを作成して、このメタファイルをプリンタに送ります。この時点で、Windows の GDI モジュールとWindows のプリンタドライバは用紙にチャートを「ペイントする」仕事を引き受けます。そのときに、画面座標値をプリンタのピクセル値に変換して、メタファイルを「引き伸ばす」または「サイズ変更する」作業も実行されます。
例:
画面上のチャートは次の長方形です。
左　: 100
上　: 100
右　: 300
下　: 400
これを用紙の次の長方形に出力します。
左　: 400
上　: 1000
右　: 1200
下　: 1500
この例で、チャートは横幅と高さの両方ともスケール変更する必要があります。
画面上の幅= 300 − 100 = 200
用紙の幅　= 1200 − 400 = 800
用紙/画面比 = 800 / 200 = 4　<---
画面上の高さ= 400 − 100 = 300
用紙の高さ= 1500 − 1000 = 500
用紙/画面比 = 500 / 300 = 1.666... <---
問題は、横と縦の伸長比が同じでないことです。
4 <> 1.666...
つまり、垂直よりも水平の寸法を大きく拡大しなければならないということです。この場合、Windows は、新しい寸法比で、フォントサイズとペン幅を拡大します。

Windows（95/NT）は、フォントを再スケールします。テキストは、他のチャート部分とオーバーラップし、不正な位置に出力されます。これを修正するために、「プロポーショナルに印刷する方法」を参照してください。

Windows（95/NT）は、拡張メタファイルフォーマットを使用して最適な処理をするので、テキスト寸法は正確に計算されます。

プロポーショナルに印刷する方法

PrintProportional プロパティは、画面上の縦横比と用紙の縦横比が同様になるようにプリンタマージンを計算します。
PrintProportional プロパティは、デフォルトは True です。

もう一つの方法として、用紙の縦横比に合わせて画面上の縦横比を変更するやり方もあります。

```
With TChart1
        .Printer.Orientation = poPortrait
        .Height=Rnd((.Printer.PageHeight)/.Printer.PageWidth)*.Width
        .Printer.PrintChart
End With
```

その他の問題

解像度の引き上げ:

メタファイルフォーマットでは、「解像度」という概念は意識されません。つまり、メタファイルイメージ内に解像度情報は格納されないということです。印刷前に「解像度」を変更するには、次のプロパティを設定します。

TChart1.Printer.Detail = -100

マイナスの値（上記の-100 など）は、解像度のパーセント増分を表します。ゼロは wysiwyg です。
チャートのすべてのフォントを小さく、線を細くすれば、解像度は高くなります。解像度を高くすると、メタファイルのサイズは大きくなります。プリンタドライバの計算の不確実性も大きくなります。

非実線の印刷:
プリンタと Windows の組み合わせによっては、「点」や「破線」などの非実線が実線として印刷されることがあります。唯一の対策は、チャートのペン幅のプロパティをゼロに設定することです。

TChart1.Axis.Left.GridPen.Width = 0

解像度を上げることで（上記参照）、非実線を印刷できるケースもあります。

色:
多くのプリンタは、利用できるカラーのうち、一定サブセットのみを受け付けます。したがって、チャートのカラーをサポート外パレットカラーに設定すると、そのカラーがプリンタで使用されておらず、なにも描画されない、という可能性が生じます。この場合には、vbRed、vbBlue、vbYellow、vbGreen などの確実なソリッドカラーをご使用ください。
プリンタによっては、「カラーマッピング」設定ダイアログが用意されている場合もあります（「プリンタのプロパティ」ダイアログ）。

直接印刷:
プリンタ GDI ハンドル、あるいはキャンバスに直接チャートを描画することができます。以下のコードを使用してください。

例.Visual Basic

```
With TChart1
    .Printer.BeginDoc
    .Draw.Canvas.HandleDC, 0, 0, .Printer.PageWidth / 2, .Printer.PageHeight / 2
    .Printer.EndDoc
End With
```

直接印刷については、いくつかの問題があります。:
-- チャート背景が（白ではなく）グレー色になる。
-- フォントサイズが非常に小さい。
-- 多くの軸グリッド線。
-- 線がとても細い。
複雑ではありますが、上記の問題の回避方法の 1 つは、全てのフォントサイズやペンの幅を変更することです。

Visual Basic の Printer クラスは、上記の方法でキャンバスを供給しません。同様の方法で PictureBox に描画することができます。MS VC++の pDC で Draw を使用することができます。

プリンタドライバの設定:
かならず最新バージョンのプリンタドライバを使用してください。Windows のプリンタドライバの解像度設定を変更して、（Windows NT 系の場合は）スプーラメソッドを EMF と RAW 両方のモードをお試しください。EMF は、すべての出力がメタファイルフォーマットでプリンタに送られます。

印刷リファレンス
次のプロパティおよびメソッドに拡張された内容とサンプルについてのヘルプファイルを参照してください。
Examples フォルダには、いくつかのカスタム印刷サンプルがあります。

印刷プロパティ:
チャートの印刷には、次のプロパティが関係します。
TChart1.Printer.MarginTop、MarginBottom、MarginLeft、MarginRight

パーセント値による、用紙ページ 4 辺の余白

TChart1.Printer.Detail

画面と用紙の寸法比

印刷メソッド:

TeeChart のコントロールには、印刷用に設計されたいくつかのメソッドがあります。

印刷して排出するメソッド:

デフォルトのメソッドであり、TeeChart で多用されるメソッドです。

TChart1.Printer.PrintChart

デフォルトの印刷方向とマージンを使います。

TChart1.Printer.PrintPortrait

印刷の向きを縦方向に設定し、デフォルトのマージンを使用します。

TChart1.Printer.PrintLandscape

印刷の向きを横方向に設定し、デフォルトのマージンを使用します。

印刷後にページを排出しないメソッド:

このメソッドでは、同じ用紙ページに複数のチャートを印刷することができます。あるいは、別のアイテムとチャートコンポーネントを同じ用紙ページに印刷することができます。

このメソッドでは、Printer.BeginDoc と EndDoc を独自に呼び出す必要があります。

TChart1.Printer.PrintPartial

指定した範囲で、プリンタのデバイスコンテキストにチャートを描画します。

TChart1.Printer.PrintPartialHandle

指定したデバイスコンテキストに、指定した範囲でチャートを描画します。

関連メソッドとプロパティ:

印刷に直接は関係しませんが、高度な印刷に役立ちます。

TChart1.Draw

画面モードで、渡されたキャンバスにチャートを出力します。

「画面モード」とは、背景を灰色にし、メタファイルフォーマットを使用しません。

このメソッドは、直接キャンバスに出力します。

TChart1.Printer.JobTitle

印刷ジョブのタイトル

TChart1.Printer.PrinterCount、PrinterIndex、PrinterDescription

複数のプリンタ環境で、プリンタの指定および設定ができます。

TChart1.Printer.PrintProportional

画面上のチャートの寸法に比例する、印刷時のチャートの寸法を設定します。

詳細情報

印刷とメタファイルに関する情報を参照するには、Microsoft Windows 32 ビット版 SDK ヘルプファイルを使用します(Win32.hlp)。「ヘルプの目次」(インデックスではありません)で、「メタファイル」までスクロールダウンします。

5. 7. チャートのズームとスクロール

チャートのスクロールおよびズームは、単に軸スケールを希望する値に設定しているだけです。チャートのズームやスクロールの後、すべての系列が新しい位置にそのポイントを再ペイントします。
メモ: 円系列は、スクロールやズームは行えません。チャートマージンあるいは円カスタム半径プロパティを使用して円の大きさを制御できます。

ズーム
動画ズーム
コードによるズーム
ズームの解除
ズームイベント

スクロール
スクロールイベント
スクロールの制御
キーボードスクロール

ズーム
チャートはプログラムあるいはマウスをドラッグしてズームすることができます。チャートの Zoom.Enabled プロパティは、ズームを使用するかどうかを制御できます。

```
TChart1.Zoom.Enabled = True
```

詳細を見たいチャート領域の範囲を指定して、ズームすることができます。
メモ:
左上から右下にマウスをドラッグしてください。反対方向にドラッグすると、軸スケールをリセットします。
マウスボタンから手を放すと、TeeChart はズームしたエリアを描画するように再ペイントします。

動画ズーム
TeeChart がズーム位置を一度に計算(即時ズーム)するか、あるいはズームした状態の画面になるまで、短い「ステップ」で(徐々に)ズームを計算するかを制御できます。
これは、「動画」ズーム効果を出すことができます。これにより、ズーム領域をより明確にすることができます。このコードでは、動画ズームをアクティブにしています。

```
TChart1.Zoom.Animated = True
```

AnimatedZoomSteps プロパティを、お好きなズーム倍率に設定してください。

```
TChart1.Zoom.AnimatedSteps = 5
```

コードによるズーム
以下のメソッドのどれを使用しても、チャートのズームインあるいはズームアウトが行えます。
ZoomRect は、Rect パラメータ領域を示すよう軸スケールを調整します。指定領域は、画面ピクセル座標値で示されます。指定領域をチャート矩形より小さい範囲に指定するとズームインになり、チャート矩形より大きい範囲を指定するとズームアウトになります。

```
TChart1.Zoom.ZoomRect 100, 100, 200, 200
```

ズームの解除
Zoom.Undo メソッドは、軸スケールを自動最小値および最大値にリセットします。

```
TChart1.Zoom.Undo
```

これは、コードまたはマウスを使用して、これ以前に行ったズームインあるいはズームアウトの操作を解除して、すべての系列ポイントを表示します。
メモ:
ズームを解除した後で、軸スケールを特定の値にしたい場合、以下に説明するチャートの OnUndoZoom イベントを使用することができます。
Zoomed プロパティは、4 つのチャート軸がすべて自動になっているかどうかを返します。

ズームイベント

OnZoom イベントは、ズームをチャートに適用する場合はいつでも、呼び出されます。

```
Private Sub TChart1_OnZoom()
  Button1.Visible =True
'「ズーム解除」ボタン可視に設定してください
End Sub
```

OnUndoZoom イベントは、プログラムあるいはマウスによるズームの解除の際に呼び出されます。

スクロール

スクロールはズームに非常に似ています。軸スケールがインクリメントまたはデクリメントされ、チャート全体が系列ポイントを新しい位置に示すように再ペイントされます。

チャートの Scroll.Enable プロパティは、マウスをドラッグすることによりチャート内容をスクロールできるかどうかを制御します。使用できる値は以下のとおりです。

pmNone	スクロールなし
pmHorizontal	水平スクロールのみ
pmVertical	垂直スクロールのみ
pmBoth	水平および垂直スクロール

例:

```
TChart1.Scroll.Enable = pmNone
' スクロールなし
```

Axis Scroll メソッドを使用してプログラムでチャートのスクロールが行えます。

```
Procedure Scroll(Offset As Double, CheckLimits As Boolean)
```

例:

```
TChart1.Axis.Bottom.Scroll 1000, True
```

上記コードは、下の軸スケールを 1000 ずつインクリメントします。これは以下と同様の処理を行います。

```
With TChart1.Axis.Bottom
  .SetMinMax .Minimum+1000, .Maximum+1000
End with
```

そして Axis.Bottom.Automatic プロパティを False にします。

チャートは再ペイントされ、水平方向の下の軸は軸スケールで「1000」の量だけ左に「スクロール」されます。

「CheckLimits」パラメータは、スクロールする方向に系列ポイントがある場合のみ軸をスクロールするように指示します。

スクロールイベント

チャートの OnScroll イベントは、チャートをスクロールするたびに呼び出されます。

```
Private Sub TChart1_OnScroll()
    Label1.Caption = "This Chart has scrolled ! "
End Sub
```

スクロールの制御

OnAllowScroll イベントは、コードで予定していたスクロールを受け入れたり、拒否したりするために使用されます。

```
Private Sub TChart1_OnAllowScroll(ByVal Axis As TeeChart.EAxisType, AMin As Double, AMax As Double, AllowScroll As Boolean)
    If Axis = atBottom Then
        If AMax > 1000 Then AllowScroll = False
    End If
End Sub
```

上記のコードは、下の軸の最大値を 1000 よりも大きい値に設定しようとすると、スクロールを拒否します。同じチェックを DateTime 軸上でも行うことができます。

```
If Axis = atBottom Then
    If AMax > DateValue("31,12,1999") Then AllowScroll = False
End If
```

キーボードスクロール

矢印キーを押すと、フォームの OnKeyDown イベントを使用してスクロールできます。まずフォームの KeyPreview プロパティを True に設定する必要があります。

KeyDown イベントで、押した矢印キーに応じて、4 つのチャート軸が Axis Scroll メソッドを使用してスクロールされます。

5．8．リアルタイムチャートおよびスピード

リアルタイムチャートでパフォーマンススピードを向上させるには、3 つの重要なルールが適用されます。
1.できるだけ少ない数のポイントをプロットする。
2.できるだけ高速のハードウェアを使用する。
3.チャートにデータを追加するために、TeeChart の AddArray サポート（可能である場合）を使用する。**ISeries.AddArray** を参照してください。
これら 3 つのルールを一緒に適用すれば、チャートを何回も継続して描く際のスピードが一段と速くなります。

これ以外にも、以下のことをお勧めします。
- 2D を使用する。3D チャートは、2D チャートよりもペイントの速度が遅くなる。
- チャートを小さくする。チャートが大きいほど、埋めるピクセルがたくさん必要になる。
- 可能であれば、チャートの凡例とタイトルを削除する。
- デフォルトのフォントとフォントサイズを使用する。
- 多くのポイントを描画する場合は最も高速な折れ線系列を使用する。
- ソリッドなペンとブラシスタイルを使用する。
- 円形の図形や円形のバースタイルを使用しない。
- 背景ビットマップやパネルのグラデーション効果を使用しない。
- チャートの BevelInner および BevelOuter プロパティを bvNone に設定する。
- 可能であれば、TChart1.AxisVisible を False に設定し、軸を削除する。
- ビデオモードの解像度および色の表示色数をビデオカードに従って最適な値にする。
- 平均的なビデオカードでは、800x600x256 色の組み合わせの方が、1024x768x32k 色よりも高速になる。
- ビデオカードに最速のドライバを使用する。

’より速いメソッド
折れ線系列により速くポイントを追加するメソッドである AddRealTime メソッドを参照してください。AutoRepaint プロパティは、ポイントが追加されるまでチャートを再描画するかしないかを制御します。
例.

```
TChart1.AutoRepaint = False
TChart1.Series(0).Add ……’ここでポイントを追加します。
TChart1.AutoRepaint = True
TChart1.Repaint
```

5. 9. 関数

系列型の1つを「キャリア」系列として使用し、他の系列を操作してそのソースデータを作成するように、関数を定義することができます。定義された関数の動作は、他の任意の系列と変わりません。したがって、他の関数の上に、関数をビルトすることもできます。関数には、計算式、合計など、他のデータソースとの十分な関連付けのための情報が格納されるので、データソースの定義という点でのみ、他の系列と区別されます。

はじめに

関数は、系列を使用する独立したクラスです。新しい関数を追加するということは、新しい系列を追加し、関数の定義が適用されることになります。通常、関数は、1 つ以上の他の系列に対応した結果を出力します。たとえば、A 系列を定義し、次に「最大」の関数から成る B 系列を定義すると(この時、B 系列を A 系列に関連付けます)、B 系列は A 系列のデータから最大値を取得する系列として使用できます。「最大」の関数(B 系列のデータソース)は、A 系列からデータを読み込み最大値を取得して、その結果を描画します。系列型を組み合わせるルールに従えば、関数のデータを表す系列型はどんな系列でも設定できます。他のチャート系列のデータを使用して関数を定義することもできます。

メモ:

チャートエディタの TeeChart ギャラリから関数を追加する場合、デフォルトで、リボン系列が追加されます。系列型を追加した後、「変更」ボタンで他の系列型に変更することができます。

一般に、すべての関数は基本的に同じルールで処理を行います。基となる系列型(リボン、縦棒など)のプロパティは、関数系列に適用されます。つまり、縦棒系列として定義された関数では、標準的な縦棒系列のプロパティを持っています。標準系列と、関数系列の重要なプロパティの違いは、範囲の設定です。(標準系列には、範囲はありません。)範囲は、関数の反復に適用されます。たとえば、「Series1 のすべてのポイントを含む平均関数を出力する」の場合は、チャートに 1 つの直線を描画します。そして、「月単位で Series1 のすべてのポイントを含む平均関数を出力する」の場合は、1 年間のデータは月単位(12 個所の各平均値)でチャートに描画します。

重要

関数に 1 つの系列しか無い場合、デフォルトで、範囲は適用されません。必要な場合は、コードで範囲を設定しなければなりません。関数に 2 つ以上の系列が存在する場合、デフォルトで、Period = 1 になっています。

チャートエディタを使用して関数を追加

関数はチャートエディタを使用して、設計時に追加することができます。系列を追加するのと同様に、TeeChart ギャラリを使用して関数を選択することができます。ギャラリでは、すべての関数はまずリボン系列関数として表示されます。関数を追加した後で、系列型を変更することができます。

関数を追加したら、[系列]ページの[データソース]タブに移動してください。ここで、どの入力系列(あるいは系列)を関数に追加するかを選択します。リストの全メンバーを操作しない関数は選択中のリストボックスで上から数えて最初の系列を操作します。

例: 差分.

図. 1
チャートエディタで定義した差分関数

チャートエディタ画面を見ると、Series3 の入力用の系列として Series1 および Series2 が定義されています。リストの系列の順序は、どの系列が減算を行うかを定義します。
ここでは次のようになります。:Series3 = Series1 − Series2

チャートエディタを使用して関数を削除
関数用のキャリアとして追加した系列を削除することができます(系列をチャートエディタの最初のページから削除する)。あるいは、[系列]ページの[データソース]タブに、異なるデータソースを持っているものとして系列を再定義することもできます。

チャートエディタを使用して FunctionType を変更
FunctionType を変更するオプションは、関数系列の[系列]ページの[データソース]タブの中にあります。ドロップダウンコンボボックスには、すべての関数型のリストが入っています。リスト内から任意のものを選択することができます。コピー関数は、入力系列を直接コピーします(系列を重複させる)。

コードで関数を追加
関数はコンポーネントです。新しい関数を追加する場合、系列を追加した後で、新しい関数を定義し、これを系列用の FunctionType として設定します。

```
TChart1.Series(0).SetFunction(tfAdd)
```

異なる関数型を追加する方法については、オンラインヘルプを参照してください。各関数は、同じ系列メソッド SetFunction を使用します。

コードで関数を削除
関数を削除するために、系列を削除するか、あるいは系列のデータソースの定義を異なるデータソース型に変更してください。

Period
関数を使用して作業していると、Period プロパティが非常に便利であることに気付くでしょう。これは、関数の再計算を行う周期を定義するのに使用されます。

例:
値のある 6 つのデータポイント(たとえばバー系列のバー)があるとします。
3, 8, 6, 2, 9 および 12

Period 0(1 つの系列だけが関数への入力になっている場合のデフォルト)で関数系列を定義します。平均値は次のようになります。

6.667

Period を使用して、2 に設定します。関数からの出力として 3 つの平均値を取得します。

5.5, 4 および 10.5

これらの値は、Period の範囲の中心にプロットされます。つまり入力系列のバー1 とバー2 の間に 1 番目の値、バー3 とバー4 の間に 2 番目の値というようになります。

プロパティウィンドウで関数を選択して、Period を定義することも、あるいは FunctionType を使用して実行時に Period を変更することもできます。

たとえば、系列 2 が関数系列になっている場合は次のようになります。

```
TChart1.Series(2).FunctionType.Period = 2
```

5. 10. ADOデータソースの操作

データソースは、データ系列により使用されるチャート単位で定義できます。ADO データ系列とプログラムでデータを設定した系列あるいは統計関数系列を混在して使用できます。

データセットの作成
TeeChart のチャートは、ADO Ver.1 以上のレコードセットに接続します。系列のためのデータソースのフィールドは、ADO DSN のテーブル、あるいは SQL クエリーに属します。

データベースのデータセットを接続
チャートエディタの系列ページで新しい**系列**を選択する場合は、**データソース**タブを選択します。データ系列に新しいデータセットを接続する場合、コンボボックスから**データセット**を選択してください。「**新規**」ボタンを選択すると、新しいダイアログボックスは、ADO データソーステーブル・SQL クエリー・他の ADO 支援データソースの選択あるいは宣言を表示します。
チャートエディタの[系列]タブの[データソース]ページに正確な内容を返すことは、選択した系列型に依存します。ここで、系列型によって異なるパラメータを変更します。次のテーブルは、よく使用される系列型の一部の利用可能なオプションを示します。

系列型	データソースプロパティ
基本	
リボン / 横リボン	XValues, YValues, XLabel
折れ線	XValues, YValues, XLabel
縦棒	XValues, YValues, XLabel
面	XValues, YValues, XLabel
散布図	Xvalues, YValues, XLabel
円 / ドーナツ	PieValues, XLabel
矢印	StartXValues, StartYValues, XLabel, EndXValues, EndYValues
泡	Xvalues, YValues, XLabel, RadiusValues
ガント	StartValues, EndValues, AY (Y軸レベル), AXLabel (オプションでY軸にマーカとして表示されるラベル)
シェープ	X0 (上), Y0 (下), X1 (左), Y1 (右)

拡張	
ベジェ	XValues, YValues, XLabels
キャンドル	OpenValues, CloseValues, HighValues, LowValues, DateValues
等高線	XValues, YValues, ZValues, Labels
エラー	XValues, YValues, XLabel, StdErrorValues
エラーバー	XValues, YValues, XLabel, ErrorValues
ヒストグラム	XValues, YValues, XLabel
3D散布	XValues, YValues, XLabel, ZValues
極	XValues, YValues, Labels (角度と半径を持っています)
レーダー	XValues, YValues, Labels (角度と半径を持っています)
サーフェス / ウォーターフォール / サーフェス (三角)	XValues, YValues, ZValues
ボリューム	XValues, YValues (VolumeValues), XLabel

データソースのコーディング
コードで系列をチャートに追加し、その系列のフィールドを定義することにより実行時にチャートにデータを接続できます。
この例は、**TeeChart Pro ODBC DSN** がインストールされていると仮定します。
LabelsSource プロパティや YValues.ValueSource プロパティの文法は:

```
TChart1.Series(0).YValues.ValueSource = "SALARY"
TChart1.Series(0).LabelsSource = "LASTNAME"
```

系列データソースの文法は:

```
例.
TChart1.Series(0).DataSource="DSN=TeeChart Pro Database; TABLE=employee"
```

SQL クエリーの文法は:

```
TChart1.Series(0).DataSource="DSN=TeeChart Pro Database; SQL=select * from employee"
```

あるいは、外部的に作成されたレコードセット:

```
Set Conn = CreateObject("ADODB.Connection")
Set rst = CreateObject("ADODB.Recordset")
Conn.Open "DSN=TeeChart Pro Database"
rst.Open "select * from employee", Conn, 1, 1
```

```
TChart1.Series(1).DataSource = rst
```

系列をデータソースから切断するには、このように簡単に記述できます。:

```
TChart1.Series(0).DataSource = ""
```

データベースの値が変更された場合、接続を「更新」して最新の値を取得するには、次のようにしてください。 :

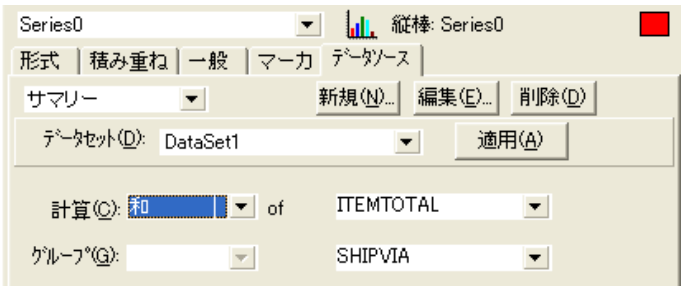
```
TChart1.Series(0).CheckDataSource
```

系列のすべてのレコードを再度取得するために「CheckDataSource」メソッドを使用してください。

サマリーグループ

TeeChart は、フィールドによりレコードセットデータを要約することができます。チャートエディタでは、サマリーとして使用可能なフィールドが表示されます。エディタでサマリーグループを設定するためには、エディタの[系列]タブの[データソース]ページでドロップダウンコンボボックスから「サマリー」を選択してください。次に、「新規」あるいは「編集」ボタンを使用して ADO データソースを選択してください。データセット(レコードセット)を選択した後は、[データソース]ページに戻ります。

[データソース]ページは、以下のフィールドを表示します:



- **計算**フィールドでは、計算タイプ(例:和、平均値、計算、など)を選択します。
- **of**フィールドでは、和あるいは計算などをさせたいフィールドを設定します。
- 右の**グループ**フィールドでは、要約したいフィールドを選択してください。

上記の設定をコードにすると:

```
.Series(0).YValues.ValueSource = "#Sum#ItemTotal"
.Series(0).LabelsSource = "ShipVia"
```

この結果は、SQL クエリーの結果と等しくなります。

"Select Sum(ItemTotal), Shipvia from Orders group by ShipVia"

TeeChart では、TeeChart 内部でグループ化を設定できるので直接テーブルにアクセスして、クエリーを構築する必要がありません。TeeChart の VB サンプルを参照してください。

単一レコード

データセットレコードを使用してデータに接続するために、このテクニックを使用してください。たとえば、データが、フィールド付きレコードとして保持される場合:

Jan,Feb,Mar,Apr,May,June,Year

6 つのバーの縦棒系列としてチャートデータができ、一月毎に 1 つのデータで、チャートタイトル、あるいはアノテーションコメントとして使用される年を取得します。

データソース型で「単一レコード」が選択された場合、以下のような TeeChart 系列データソースページになります。

利用可能なフィールドは、系列を構成するように付加できるレコードフィールドを示します。系列に必要なフィールドを選択してください。

5. 11. ASPリファレンス

概観

TeeChart Pro 8J ActiveXは、IISやASPでクライアントサイドコンポーネント、あるいはサーバサイドコンポーネントとして動作します。TeeChart は、VBScript あるいは JScript で使用できます。クライアントページオブジェクトの場合、ページイベントやユーザアクションで動作します。また、TeeChart は、ホットスポットされた静的なイメージとしてエクスポートもできます（ASP サンプルコードを参照してください）。テンポラリファイルなしで、サーバからブラウザに動的や静的のエクスポートができます。

インフォメーションソース

以下を参照してください：

TeeChart の ASP ライブコードサンプル（「インターネットサンプル」インストールオプションを選択した場合、TeeChart プログラムマネージャグループを使用して使用可能になります）。そのコードがインストールされない場合、サンプルフォルダの下の「IIS & ASP」フォルダ内に保持されます。

TeeChart のチュートリアル 「インターネットアプリケーション」

TeeChart の他のコントロール

1. TeeChart の他のコントロールの使用

TeeChart Pro 8J ActiveX には他にもいくつかのコントロールがあります。
「代表的なもの」


TeeCommander


TeeEditor


TeePreviewer


TeeListBox

2. コンポーネントの選択と貼り付け

TeeChart Pro 8J を Visual Basic のコンポーネントパネルに追加すると、ほかのコントロールも自動的に追加されます。VisualC++などではコントロールが別々にレジストリに追加されるので、使用するコントロールを選択して追加してください。

3. TeeChart に他のコンポーネントの接続

「Chart」や「ChartLink」プロパティを使用することにより、TeeChart に「TeeCommander」や他のコンポーネントを接続することができます。VisualC++や他の環境では、ChartLink プロパティを使用してください。(ChartLink プロパティによりチャートの位置を取得できます。)

Visual Basic では

```
TeeCommander1.Chart = TChart1
```

あるいは

```
TeeCommander1.ChartLink = TChart1.ChartLink
```

VisualC++や他の環境では

```
m_Commander1.SetChartLink(m_Chart1.GetChartLink());
```

4. TeeCommander

TeeCommander は実行時にチャートのナビゲーションやコントロールのパラメータを変更することができます(Chart Editor を使用します。)。ナビゲーションボタンを選択し、マウスの左ボタンを押してチャートをドラッグするとチャートが移動します。
「TeeCommander」が呼ばれると、「EditorLink」プロパティと「PreviewerLink」プロパティは、実行時にコントロールが「Chart Editor」と「Previwer」にリンクされるために使用されます。

5. TeeEditor

「TeeEditor」のコンポーネントは実行時には表示されません。このコンポーネントを使用することにより実行時にエディタを表示することができます。「TeeEditor」は「ShowEditor」メソッドで呼び出すことができます。
また「TeeCommander」に接続することにより「TeeCommander」からエディタを実行することもできます。

6. TeePreviewer

「TeePreviewer」のコンポーネントは実行時には表示されません。このコンポーネントを使用することにより実行時に印刷プレビューを表示することができます。「Previewer」は「ShowPreviewer」メソッドで呼び出すことができます。また「TeeCommander」に接続することにより「TeeCommander」から印刷プレビューを実行することができます。

7. TeeListBox

「TeeListBox」は系列のリストを実行時にチャートに表示します。ListBox を使用することにより、チャートの表示・非表示を切り替えたり、機能を制限することができます。

VisualC++での使用について

---

1. クイックスタート

1-1. TeeChart のプロジェクトへの追加

ファイルメニューの「新規作成」を選択し、プロジェクトページから「MFC AppWizard」を選択してください。ここではダイアログベースのアプリケーションを選択します。Wizard の 2 ページ目で「ActiveX コントロール」にチェックを入れてください。(デフォルトで設定されています)

プロジェクトを作成したら、プロジェクトに TeeChart Pro 8 を挿入する必要があります。メニューから「プロジェクト」－「プロジェクトへ追加」－「コンポーネント及びコントロール」－「Registered ActiveX Controls」を選択し、TeeChart Pro 8 の ActiveX コントロールを選択します。そうすると、コントロールギャラリに TeeChart Pro 8 コントロールのアイコンが表示されます。

次に、リソースウィンドウの Dialog フォルダからダイアログを選択して、TeeChart Pro 8 をダイアログボックスに貼ります。

1-2. チャートの名前

メニューの「表示」－「ClassWizard」を選択し、メンバー変数タブを選択します。IDC_TCHART1 に変数を追加してください。(たとえば、m_Chart1)

1-3. チャートにデータを追加

デザイン時に系列を追加することもできますが、下記のようにコードで系列を追加することもできます。

なお、ここではランダムなデータを系列にセットします。

```
void CMyDlg::OnButton1()
{
        //Add this include to the implementation file
        #include "Series.h"
        // Add a Bar Series
        m_Chart1.AddSeries(1);
        // Fill with random data
        m_Chart1.GetSeries(0).FillSampleValues(10);
}
```

2. TeeChart の VC++でのプログラミング

2-1. デザイン時でのチャートのプロパティの設定

デザイン時にプロパティを設定するには、チャートの上でマウスを右クリックし、「プロパティ」メニューを選択し、「TeeChart Pro Editor」タブから「チャートの編集」を選択してください。

(注意)チャートの編集は「TeeChart Pro ActiveX control v8 オブジェクト」－「編集」から実行しないでください。

この場合、デザイン時のプロパティが保存されません。

2-2. TeeChart のイベント

チャートのイベントを追加するには、チャートの上でマウスを右クリックしてイベントを選択してください。リストボックスから必要なイベントを選択してコーディングしてください。

2-3. 実行時の TeeChart のコーディング

チャートのクラスは ClassView で表示されます。また、前記で設定したチャートの変数(m_Chart1)でプロパティやメソッドにアクセスすることができます。

下記の例はチャートのパネルの表示をグラデーションに設定します。

```
//Add includes
#include "Panel.h"
#include "Gradient.h"
void CMyDlg::OnButton1()
{
        //Enable Panel Gradient
        m_Chart1.GetPanel().GetGradient().SetVisible(true);
}
```

(注意)プロパティにアクセスする関数は「Get」や「Set」で始まります。

2-4. オンラインヘルプについて

オンラインヘルプは Visual Basic 用に記述されていますが、VC++については下記の表を参考にしてください。

VB	VC++
Add method TChart1.Series(0).Add 3, "Pears", vbRed	#include "series.h" m_Chart1.GetSeries(0).Add(3,"Pears",RGB(255,0,0));
Axis Title TChart1.Axis.Left.Title.Caption="My left axis"	#include "axes.h" #include "axis.h" #include "axisTitle.h" m_Chart1.GetAxis().GetLeft().GetTitle().SetCaption("My Left Axis");

Chart Title
With Chart1.Header.Text
.Clear
.Add("ACME Monthly Sales")
.Add("Year: 2008")
End with
	#include "Titles.h"
	#include "Strings.h"
	
	COleVariant var1(CString ("ACME Monthly Sales"));
	COleVariant var2(CString ("Year: 2008"));
	CTitles hd = m_Chart.GetHeader();
	hd.GetText().Clear();
	hd.GetText().Add(*(LPCVARIANT)var1);
	hd.GetText().Add(*(LPCVARIANT)var2);
	
	//if items (header lines) already in list (+ alternative COleVariant declare syntax)
	
	m_Chart1.GetHeader().GetText().SetItem(0,
	COleVariant("ACME Monthly Sales"));
	m_Chart1.GetHeader().GetText().SetItem(1,
	COleVariant("Year: 2008"));

Canvas draw
TChart1.Canvas.MoveTo(0,0)
TChart1.Canvas.LineTo(100,100)
TChart1.Canvas.TextOut(50,50,"My output text)
	#include "Canvas.h"
	
	m_Chart1.GetCanvas().MoveTo(0,0);
	m_Chart1.GetCanvas().LineTo(100,100);
	m_Chart1.GetCanvas().TextOut(50,50,"My output text");

2-5. 定数定義

VC++では TeeChart の列挙定数が使用できませんので、サンプルフォルダにある「TeeChartDefines.h」をプロジェクトに追加して、この定義文字列を使用するようにしてください。

2-6. サンプル

(Canvas の使用例)

「OnAfterdraw」イベントにキャンバスを使用する例です。

```
void CVCExampleDlg::OnOnAfterDrawTchart1()
{
    // Draw line to 5th point and label it
    if (m_Chart1.GetSeries(0).GetCount() > 4)
    {
       CCanvas cv = m_Chart1.GetCanvas();
       CAxes ax = m_Chart1.GetAxis();

       //Setup pen and brush
       //See section Constant defines for
       // how to define bsClear
       cv.GetPen().SetColor(RGB(0,0,0));
       cv.GetBrush().SetStyle(bsClear);
       cv.TextOut(m_Chart1.GetSeries(0).CalcXPos(5), m_Chart1.GetSeries(0).CalcYPos(5), "HELLO");
       cv.MoveTo(ax.GetLeft().GetPosition(), ax.GetTop().GetPosition());
       cv.LineTo(m_Chart1.GetSeries(0).CalcXPos(5), m_Chart1.GetSeries(0).CalcYPos(5));
    }
}
```

(ODBC の使用例)

ODBC を使用するサンプルコードです。

```
#include "series.h"
#include "valuelist.h"
#include "marks.h"
void CVCExampleDlg::OnButton5()
{
    // Add a Pie series and define the datasource
    m_Chart1.AddSeries(5);
    m_Chart1.GetSeries(0).SetDataSource("DSN=TeeChart Pro Database; SQL=select * from employee");
    m_Chart1.GetSeries(0).GetYValues().SetValueSource("SALARY");
    m_Chart1.GetSeries(0).SetLabelsSource("LASTNAME");
    m_Chart1.GetSeries(0).GetMarks().SetVisible(false);
}
```

3. 他の TeeChart コントロールの使用

TeeChart Pro 8J ActiveX には他にもいくつかのコントロールがあります。

「代表的なもの」


TeeCommander


TeeEditor


TeePreviewer


TeeListBox

3-1. TeeCommander

「ChartLink」プロパティにより「TeeCommander」コントロールと「TeeChart」コントロールを接続します。

例

```
m_Commander1.SetChartLink(m_Chart1.GetChartLink);
```

3-2. TeeEditor

「ChartLink」プロパティにより「TeeEditor」コントロールと「TeeChart」コントロールを接続します。

例

```
m_TeeEditor1.SetChartLink(m_Chart1.GetChartLink);
```

3-3. TeePreviewer

例

```
m_TeePreviewer1.SetChartLink(m_Chart1.GetChartLink);
```

3-4. TeeListBox

例

```
m_TeeListBox1.SetChartLink(m_Chart1.GetChartLink);
```

作成したアプリケーションの配布

TeeChart Pro 8J を使用して作成したアプリケーションをユーザに配布する場合は、そのアプリケーションの実行時に以下のファイルが必要になりますので、忘れずに配布リストに加えてください。

A.Windows アプリケーション

TeeChart8.ocx

TeeChart8.ocx を使用し、実行時にチャートエディタを使用する場合は、TeeUserX8.hlp が必要です。

また、PNG を利用する場合は Lpng.dll も必要です。

B.Web アプリケーション（1 サーバー毎に WebServer ランタイムライセンスが必要）

下記の二通りの方法があります。

1. CAB ファイルをサーバーからクライアントに転送し、クライアント側で実行する場合

   TeeChart8.cab または TeeSmall8.cab

   cab ファイルは ocx ファイルが圧縮されたファイルであり、インターネット等で配信される時にメーカ名を明らかにする認証情報が埋め込まれています。

   TeeSmall8.cab は、TeeChart8.cab に比べて機能が制限されているため、TeeChart8.cab よりもファイルサイズが小さくなっています。

   制限事項（TeeChart8.cab と比較して利用できない機能）:

   TeeChart エディタ、JPEG のサポート、印刷プレビュー、バージョン情報スクリーン

   ODBC のサポート、OpenGL のサポート

   TeeChart8.cab を使用し、実行時にチャートエディタを使用する場合は、TeeUserX8.hlp が必要です。

   また、PNG を利用する場合は Lpng.dll も必要です。

2. サーバー側で OCX を実行する場合

   TeeChart8.ocx

   PNG を利用する場合は Lpng.dll も必要です。

その他、TeeChart Pro 8J 以外のファイルでセットアップディスク作成時に必要なファイルについては、その開発環境のマニュアルなどをご覧ください。

注意事項

TeeChart8.cab と TeeSmall8.cab はレジストリへ登録される GUID が同じため、同一 PC にそれぞれ別々の CAB ファイルがダウンロードされた場合、バージョンが同じであれば最初にレジストリへ登録された CAB ファイルが有効になります。そのため、TeeChart8.cab のフル機能を使用した場合に発生すると思われるトラブルを回避するために、できるだけ TeeSmall8.cab の機能を使用して、Web アプリケーションを作成することをお勧めいたします。（ASP 中に記述する CAB ファイルは TeeChart8.cab をお勧めいたします。）

また、TeeUserX8.hlp についてもレジストリへの登録が必要になりますので、CAB ファイル同様に注意が必要です。（こちらは後でレジストリへ登録された方が有効になります。）

詳しくは、インストールしたフォルダの階層化にある Docs¥Runtime Editor help sourceHelp フォルダの ReadMe.txt をご覧ください。

製品に付属のドキュメントを再配布することはできません。

TeeChart Pro 7 からの移行

Visual Basic 及び Visual C++で作成したプロジェクトについては、インストールしたフォルダの階層化にある Utilities フォルダの Update.exe を使用することにより、TeeChart Pro 4 から 7 を TeeChart Pro 5 から 8 のコンポーネントに置き換えることができます。

Visual Studio .NET で作成したプロジェクトについては、インストールしたフォルダの階層化にある Utilities フォルダの UpdateVSNet.exe を使用することにより、TeeChart Pro 7 から TeeChart Pro 8 のコンポーネントに置き換えることができます。

それ以外の言語についてはユーティリティが使用できないため、手動でコンポーネントを置き換える必要があります。
ただし、TeeChart Pro 7 と TeeChart Pro 8 は同一プロジェクトに共存できないため、一度 TeeChart Pro 7 のコンポーネントをフォームから削除した後で、プロジェクトに TeeChart Pro 8 を組み込んでコンポーネントを配置してください。

索引

TeeChart Pro 8J ActiveX

プログラミングガイド

発行 株式会社ニュートン

〒940-0015
新潟県長岡市寿１－６－43
TEL 0258-24-7900
FAX 0258-24-7905
http://www.newtone.co.jp/