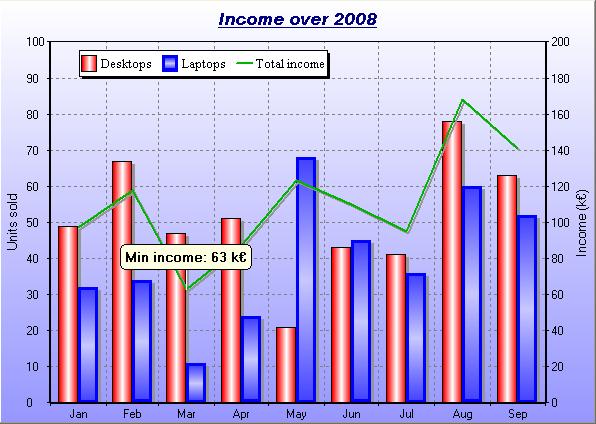
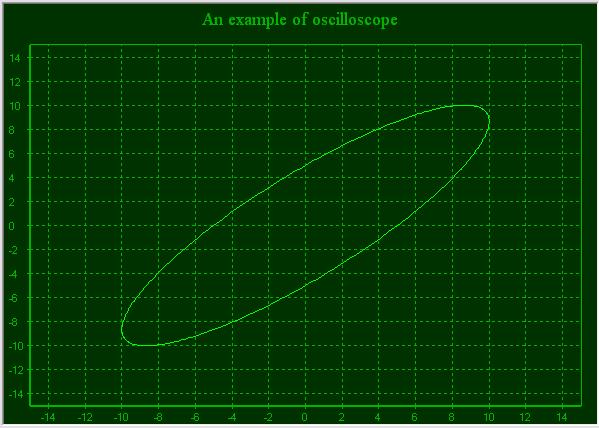
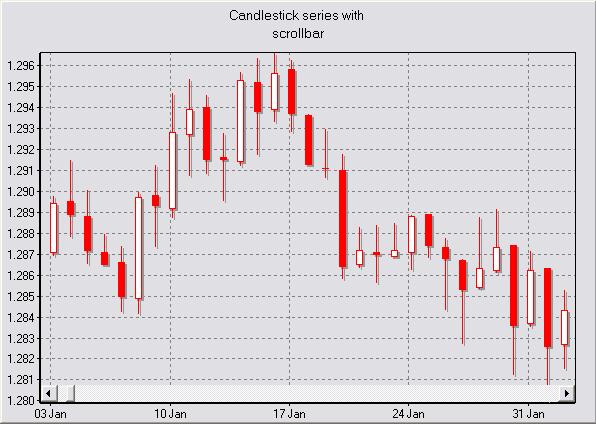
# **[High-speed Charting Control--MFC绘制图表（折线图、饼图、柱形图）控件](https://www.cnblogs.com/yutingliuyl/p/7384485.html)**

原文地址：<https://www.codeproject.com/articles/14075/high-speed-charting-control>  
本文翻译在CodeProject上的介绍（主要还是谷歌翻译。看不太明确的地方，请对照原文。敬请原谅），方便自己和后面人的学习（花费了两天时间。希望是值得的）。

推荐一个前辈写的东西：[TeeChart替代品，MFC下好用的快速画图控件-(Hight-Speed Charting)](http://blog.csdn.net/czyt1988/article/details/8740500)，自己也转载了这篇文章，在转载的文章中依据自己的实验改动了一些东西。改动了什么如今也不记得了。文章地址：[TeeChart替代品，MFC下好用的快速画图控件-(Hight-Speed Charting) .](http://blog.csdn.net/xuanyuanlei1020/article/details/50468061)  
以下是我能找到的资料(点击超链接下载)前三个是原文提供的(要想找到最新的，到[原文](https://www.codeproject.com/articles/14075/high-speed-charting-control)中找)，后三个是自己在其它的博客中找到的。  
[ChartCtrl\_demo.zip](http://download.csdn.net/detail/xuanyuanlei1020/9704421)  
[ChartCtrl\_doxygen.zip](http://download.csdn.net/detail/xuanyuanlei1020/9704423)  
[ChartCtrl\_source.zip](http://download.csdn.net/detail/xuanyuanlei1020/9704429)  
[ChartCtrl.pdf](http://download.csdn.net/detail/xuanyuanlei1020/9704416)  
[MFC动态绘制曲线图-HightSpeedChart实现.rar](http://download.csdn.net/detail/xuanyuanlei1020/9705144)  
[TeeChart和HightSpeedChart动态画图.rar](http://download.csdn.net/detail/czyt1988/6992947)

## **综述**

这个控件是用来展示2D数据，如以下的图片。  
  
  


## **介绍**

对于我之前的一个项目，我须要在图表控件上显示连续的数据流。 我决定开发自己的控件。由于我找不到不论什么能够提供所需灵活性的自由软件控件。 当中一个基本的限制是。控件必须绘制大量的数据。并能够迅速显示它（在Pocket PC上）。

控件能够通过仅绘制新的数据点而不是完整的数据序列来做到这一点而且图表还能够显示静态数据。

这样的控件是我长时间工作的结果，而且费尽周折地为了提供足够的灵活性来供须要它的人使用。

对于使用者反馈我表示由衷的感谢：一个邮件，留言板中的一一句话或仅仅是对本文评级。

当我不知道是否还有人使用它时，我就没有必要维护这个控件了。

## **免责声明**

这个控件是我花费非常长时间的开发的结果，因此我对代码的使用放置一些小条件：

该代码能够以编译的形式用于不论什么非商业和商业目的。

代码能够被又一次开发。仅仅要它提供作者名字和完整的免责声明。 更改源码须要得到作者的同意。

此代码不提供不论什么安全保证。 我不会对使用此代码造成的损失负责。 使用它须要自己承担风险。

This code may be used for any non-commercialand commercial purposes in a compiled form.

The code may be redistributed as long as it remains

unmodified and providing that the author nameand the disclaimer remain intact. The sources

can be modified with the author consent only.

This code is provided without any guarantees.

I cannot be held responsible for the damage orthe loss of time it causes. Use it at your own risks.

鉴于开发这个控件所付出的努力。以下的要求并只是分： 假设你在在商业应用程序中使用这个控件。那么请给我发邮件让我知道。

## **主要特点**

控件的主要特点是：

* 快速画图（轴固定时），同意快速绘制数据
* 无限数量的数据序列（内存是限制）
* 每一个数据序列的数据量不受限制
* 支持线图。点图。平面图。柱状图，K线图 和甘特图系列
* 最多四个轴（左。下，右和上轴）
* 标准轴。对数轴或日期/时间轴
* 自己主动伸缩的坐标轴, 翻转的坐标轴（相互独立）
* 轴标签
* 点标签
* 平滑的曲线
* 网格
* 图例和标题
* 交互性（在控件中发生特定事件时的通知）
* 支持手动缩放和鼠标平移
* 支持鼠标指针
* 支持轴上的滚动栏
* 高度可定制（颜色，标题。标签，边缘，字体等）
* 支持UNICODE
* 支持打印和保存到图像文件

## **文档结构**

本文通过一系列简短的教程来涵盖控件的大部分功能。

阅读本文后。您将能够快速地在自己的应用程序中使用本控件。

我决定从文章中删除全部的类和函数的文档，由于它不是非常友好而且我非常难维护。 此外，随着代码的增长，要记录的类和函数的列表变得过于广泛以至于不能将全部内容放在文章中。 作为替代，我提供了一个doxygen文档，您能够从本文中（文章的开头）下载：仅仅需下载“Doxygen文档”zip文件，解压全部文件。双击“Index.html”文件，进行查看。

## **入门学习**

此图表控件同意您在屏幕上绘制一系列数据。

此控件能够加入几个不同类型数据序列而且最多能够使用四个轴。

加入到图表的数据序列与一个水平轴（底部或顶部）和一个垂直轴（右側或左側）相关联。

这两个轴控制数据序列在图表上的显示方式。  
为了能够在应用程序中使用次图表控件，您首先须要在自己的工程里加入源码zip中包括的文件。

**注意：**控件在内部使用动态转型。因此必须启用RTTI（RunTime Type Information 执行时自己主动类型识别的机制），否则可能会发生崩溃。 默认情况下，VC6没有启用RTTI。因此要启用它打开项目设置 - >“C / C ++”选项卡 - >“C ++语言”类别。并确保“Enable Run-Time Type Information (RTTI) “选项已选中。

在应用程序中使用图表控件有两种方法：手动插入，或通过资源编辑器插入。

### **手动插入**

1.#include "ChartCtrl"加入在对话框（Dialog）类的头文件里  
2.在对话框类中加入变量CChartCtrl:

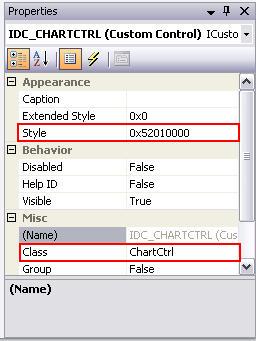
//{{AFX\_DATA(CChartDemoDlg)

//}}AFX\_DATA

CChartCtrl m\_ChartCtrl;

3.在对话框类的OnInitDialog方法中加入这个控件的Create方法。

### **使用资源管理器**

1.向对话框资源加入自己定义控件，打开控件的属性，并为Class属性指定ChartCtrl。 为了避免滚动栏上的闪烁。必须设置WS\_CLIPCHILDREN样式（0x02000000L），如图所看到的。  


2.#include "ChartCtrl.h"加入在对话框（Dialog）类的头文件里  
3.在对话框类中加入变量CChartCtrl:

//{{AFX\_DATA(CChartDemoDlg)

//}}AFX\_DATA

CChartCtrl m\_ChartCtrl;

4.在DoDataExchange函数中加入DDX\_Control（不要忘了更改ID号和控件名字）:

void CChartDemoDlg::DoDataExchange(CDataExchange\* pDX){

CDialog::DoDataExchange(pDX);

//{{AFX\_DATA\_MAP(CChartDemoDlg)

// Add this line with the appropriate ID and variable name

DDX\_Control(pDX, IDC\_CHARTCTRL, m\_ChartCtrl);

//}}AFX\_DATA\_MAP

}

## **操作数据序列**

几种类型的数据序列能够加入到控制：点序列。线序列。曲面序列，柱状图序列，K线图序列或甘特图序列。

点的数据格式可能因序列而异（比如。K线图和甘特图系列使用不同的点格式）。

| **Series type** | **Description** | **Create function** | **Point type** |
| --- | --- | --- | --- |
| Point series | Each data point is represented by a single point on the screen. The appearance of the point can be customized. | CreatePointsSerie | SChartXYPoint |
| Line series | The data points are connected through a line. The appearance of this line can be customized and it can also be smoothed. | CreateLineSerie | SChartXYPoint |
| Surface series | The data points are connected through a line and the area under this line is filled with a specific brush. The series can also be displayed vertically. | CreateSurfaceSerie | SChartXYPoint |
| Bar series | Each data point is plotted as a vertical bar of a certain width. Multiple bar series can be stacked next to each other without overlapping. The bars can also be plotted horizontally. | CreateBarSerie | SChartXYPoint |
| Candlestick series | Each data point is made of five attributes: the low value, the high value, the open value, the close value and the X value (time). Each point is drawn as a candlestick. This series is used for plotting financial data. | CreateCandlestickSerie | SChartCandlestickPoint |
| Gantt series | Each data point is made of three attributes: the start and end time and a Y value. Each point is drawn as a horizontal bar starting at the start time and finishing at the end time. The bar is positioned along the Y axis at its Y value. | CreateGanttSerie | SChartGanttPoint |

一旦你选择了一种系列，你能够通过调用上表中列出的CChartCtrl类的辅助函数之中的一个将其加入到图表中。 这些函数接受两个可选參数：两个布尔值来确定描写叙述该系列是连接到副水平轴（顶轴）或者是连接大副垂直轴（右轴）。 假设未指定參数，则数据系列将附加到主水平轴（底部轴）和主垂直轴（左轴）。

**警告:**在将不论什么系列加入到图表之前，您须要创建该系列所连接的两个轴。

假设不这样做，将导致控件失效（assert）。 有关详细信息，请參见“操纵轴”一节。

一旦将系列加入到图表后，我们就能够使用数据填充该图表。 有两种方法：将数据放到一个单元中一起加入，或者逐点加入。

后者用于有动态数据时：每次调用函数时都会更新图表。 尽管这个调用是快速的（在某些特定条件下），可是最好尽可能地将数据放到一个单元中。 以下是一个简单代码演示样例，它在图表中创建两个系列。并用数据填充它们：一个系列在初始化时全然填充，还有一个系列在调用OnDataReceived函数（仅存在于此演示样例的目的）时填充。

m\_pLineSeries，m\_pPointsSeries和m\_ChartCtrl是CMyClass类的成员变量。

void CMyClass::Init(){

.... // SNIP: Creation of the axes in the chart. This MUST be done before.

m\_pLineSeries = m\_ChartCtrl.CreateLineSerie();

m\_pPointsSeries = m\_ChartCtrl.CreatePointsSerie();

double YValues[10];

for (inti=0;i<10;i++)

XValues[i] = YValues[i] = i;

m\_pLineSerie->SetPoints(XValues,YValues,10);

}

void CMyClass::OnDataReceived(double X, double Y){

m\_pPointsSeries->AddPoint(X, Y);

}

全部系列类继承自同一抽象基类：CChartSerie。

该类处理全部系列通用的功能，但对详细的数据点没有不论什么处理功能。

点的概念在子类CChartSerieBase中引入。它是一个模板类。模板參数是要操作为点的数据类型。这非常重要，由于序列可能必须处理不同的数据类型：比如点序列操作具有X和Y值的点，可是K线图系列操纵具有5个值（打开，关闭，高，低和时间值）的点。其它系列继承自CChartSerieBase并提供他们操作的数据类型。 CChartSerieBase类已经处理了大多数数据管理，并通过纯虚函数将渲染托付给子类。

每一个系列在创建时也会分配一个Id。

此标识可通过CChartSerie :: GetSerieId（）检索，并可用于从图表中删除该系列。

该系列的一个重要特征是控制点的顺序：该系列中的全部点将依据它们的值又一次排序。

默认情况下，点是基于它们的X值排序的，但您能够通过对它们的Y值排序或不正确它们进行排序来改变这样的行为（在这样的情况下，系列保持将点加入到系列中的顺序 ）。 对点进行排序会对性能产生影响：假设点是有序的。则控件能够从完整系列中检索第一个和最后一个可见点，而且仅绘制两个点之间的点。 还有一方面。你将不能绘制像椭圆形的曲线。

您能够通过调用CChartSerieBase :: SetSeriesOrdering来更改点的顺序。

控件中的不同系列的功能一般是不言自明的。 然而，柱状图系列须要一些解释。

## **柱状图系列**

这个系列有点特别，假设当中几个在同一个控件上绘制在一起。他们将互相影响。 目的是能够绘制多个条形图系列。而不会重叠：它们是彼此相邻绘制的。 为此，您须要指定每一个所属的组（一个简单的整数标识符）。 同一组的系列彼此相邻地绘制（或者对于水平条在彼此的顶部）：參见两个图形的演示样例。

设置组ID是通过SetGroupId函数完毕的。

| **Bar series with the same group Id** | **Bar series with different group Id** |
| --- | --- |
| IMG_260 | IMG_261 |

您还能够通过调用SetInterSpace静态函数来控制全部柱形图之间剩余的空间的宽度。

这将为全部系列设置以像素为单位的空间（因此，假设显示多于两个系列，则在不论什么位置使用同样的空间）。

注意，您能够通过调用SetBarWidth单独设置柱状图系列的宽度。

## **在点上加入标签**

一旦使用数据填充您的系列，您还能够在系列的特定点上加入标签：这个标签始终附加到特定点。 如今，仅仅提供一种类型的标签，气泡标签：包括文本的圆角矩形并用线连接到特定点上。 当然，假设须要，您也能够提供自己的自己定义标签（參见“扩展功能”一节）。  
有两种方式创建文本标签：静态创建标签时，或动态注冊一个对象。当标签请求时。它将提供文本。

第一种方法是最简单的，但也不太灵活。 以下是一个代码片段，显示怎样做（假设m\_pSeries已经创建并填充足够的数据）：

void CMyClass::Init(){

// SNIP...

m\_pSeries->CreateBalloonLabel(5,\_T("This is a simple label"));

}

此调用将创建一个带有“This is a simple label”文本的标签，并将其附加到带索引为5的点。该函数返回一个指向新创建的标签的指针。以便您能够改动其某些属性或存储以供以后使用。

另外一种方法有点复杂。但提供了很多其它的灵活性：比如，您能够以更方便的方式在标签中显示点属性（比如X值，Y值。…）。 为此，您必须创建一个继承自CChartLabelProvider <PointType>的类，并在创建标签时提供此类的实例。 此类是模板类，模板參数是标签附加到的系列的点类型。

这个类是一个简单的接口，你必须覆盖TChartString GetText（CChartSerieBase <PointType> \* pSerie，unsigned uPtIndex）方法。 此函数应返回必须在标签中显示的文本。

它接收指向标签所附加的系列和点索引的指针。

这里有一个这样的标签提供程序类的样例：

class CCustomLabelProvider : public CChartLabelProvider<SChartXYPoint>

{public:

TChartString GetText(CChartSerieBase<SChartXYPoint>\* pSeries, unsigned uPtIndex)

{

TChartStringStream ssText;

SChartXYPoint Point = pSeries->GetPoint(uPtIndex);

ssText << \_T("X value=") << Point.X;

return ssText.str();

}

};

此代码段显示怎样将其与标签一起使用。 注意m\_pSeries应该是一个操作SChartXYPoint点（点，线，面或者柱系列）的系列。

假设不是这样，你的代码将给出一个编译错误。

void CMyClass::Init(){

// SNIP...

m\_pLabelProvider = new CCustomLabelProvider();

m\_pSeries->CreateBalloonLabel(5, m\_pLabelProvider);

}

控件不获取指针的全部权，因此。当你不再须要时，你有责任删除它。 在上面的样例中，它一般会在CMyClass析构函数中被删除。 在上面的演示样例中。您能够为全部要加入的标签地方反复使用同样的标签类。 这也带来还有一个长处：假设你想在执行时改变标签的格式，你仅仅须要在CustomLabelProvider中加入代码。

不须要遍历全部现有标签并更改其文本。 当然，在这样的情况下，须要刷新控件，由于必须又一次绘制标签。

还要注意TChartStringStream类的使用方法。TChartStringStream类是由控件提供的别名（类似于TChartString）。

当UNICODE被定义时。它解析为std :: wstringstream，当没有定义UNICODE时，解析为std :: stringstream。

## **对轴的操作**

轴是图表的一个重要特征。由于它们控制不同系列在控制中的显示方式。 控件中最多可使用四个轴：底部，顶部，左側和右側。 控件的每一个系列必须和一个水平轴和一个垂直轴相连接。

在图表中加入系列时指定这些轴。

底部和左側轴是主轴，顶部和右側轴是辅助轴（您将在控件的某些功能中遇到此问题）。 如今有三种类型的轴供选择：标准轴，对数轴和日期/时间轴。 您能够在不同位置选用不同类型的轴。

一旦您选择了在不同位置使用哪些轴，您须要先创建它们，然后才干向控件加入不论什么数据。 为此，通过指定轴附加在哪个位置，简单地调用CreateStandardAxis。CreateLogarithmicAxis或CreateDateTimeAxis。 假设已经在该位置创建了轴，则控件将销毁它而且用新的轴替换它。 这里有一个简单的代码片段。显示怎样在底部创建日期/时间。在左側创建一个标准轴：

void CMyClass::Init(){

CChartStandardAxis\* pBottomAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::BottomAxis);

CChartLogarithmicAxis\* pLeftAxis =

m\_ChartCtrl.CreateLogarithmicAxis(CChartCtrl::LeftAxis);

}

一旦创建了这些轴，就能够对它们设置一些属性。 大多数属性在全部轴类型之间共享（比如自己主动模式。最小值和最大值。轴标签，…）。 轴能够设置为三种“自己主动”模式：全自己主动，屏幕自己主动和手动模式。

* 全自己主动模式基于附加到该轴的全部系列计算轴最小值和最大值（全部系列的全部点的最小值用作轴的最小值。并使用全部系列的全部点的最大值 作为轴的最大值）。
* 屏幕自己主动模式基于与该轴相关的全部系列的全部可见点计算轴最小值和最大值。 比如，假设图表仅显示连接到手动底部轴和屏幕自己主动左側轴的一个系列。则左側轴将自适应于当前可见的点，而且不考虑这些点有可能超过底轴的范围（在全自己主动模式下，底轴外部的点将被考虑）。

**警告：**假设系列的两个轴都处于屏幕自己主动模式，则结果没有定义。

* 在手动模式下，轴最小和最大值由用户设置。不由控件计算。

在使用自己主动轴模式下，假设将数据动态加入到控件。假设新的数据点位于轴的范围之外，那么控件将自己主动刷新。

这里是一个代码片段（继续前一个代码段），显示一个全自己主动轴（底部轴）和一个手动轴（左轴，它是一个对数轴）：

void CMyClass::Init(){

// SNIP ...

pBottomAxis->SetAutomaticMode(CChartAxis::FullAutomatic);

// The call to SetAutomaticMode(CChartAxis::NotAutomatic) is not

// really needed because this is the default.

pLeftAxis->SetAutomaticMode(CChartAxis::NotAutomatic);

pLeftAxis->SetMinMax(0.01,1000);

}

## **处于离散模式下的轴**

轴有一个模式是离散模式(默认禁用)。此模式指定轴不显示连续值。而仅仅显示离散值，这些值是轴上刻度指定的值。而轴将不显示其它的值。尝试绘制不同于显示的节拍值的值是不可能的。让我们举一个样例：假设你有一个底部标准轴，间隔为1.0（所以，显示的蜱是1，2。3等等）。尝试绘制X值为0.5的点将在同样位置显示该点，就好像它的值为1.0。其实。你能够觉得两个刻度之间的区域是一个常量值。这就是为什么刻度标签显示在两个刻度的中间。而不是刻度本身。

这里有一个小代码片段，显示离散轴对系列显示方式的影响。代码片段下的两个图像显示启用离散模式（第一个图像）或禁用（第二个图像）的结果。

void CMyClass::Init(){

CChartStandardAxis\* pBottomAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::BottomAxis);

pBottomAxis->SetMinMax(0, 10);

CChartStandardAxis\* pLeftAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::LeftAxis);

pLeftAxis->SetMinMax(0, 10);

pBottomAxis->SetTickIncrement(false, 1.0);

pBottomAxis->SetDiscrete(true);

CChartLineSerie\* pSeries = m\_ChartCtrl.CreateLineSerie();

double XVal[20];

double YVal[20];

for (int i=0; i<20; i++)

{

XVal[i] = YVal[i] = i/2.0;

}

pSeries->SetPoints(XVal,YVal,20);

}

| **Discrete mode enabled** | **Discrete mode disabled** |
| --- | --- |
| IMG_262 | IMG_263 |

## **使用日期/时间轴**

使用日期/时间轴有点特别，以下是怎样利用这个功能的解释。

要了解日期/时间轴的重要一点是它们在COleDateTime对象内部工作。原因非常简单：COleDateTime中有DATE类型的类。DATE类型是一个双精度型。由于图表中的点表示为双精度值，因此它非常适合：使用标准点（非日期/时间）和日期/时间点之间没有差异。这使得后者的使用不太复杂。全部点仍然存储为双精度型，不管是否是日期/时间。

创建日期/时间轴后，能够在控件中填充数据。

为此目的，没有改变：你必须从CChartSerie类调用void AddPoint（double X，double Y）或void SetPoints（double \* X。double \* Y，int Count）。 CChartCtrl类提供了两个静态函数。让你从COleDateTime转换为双精度。反之亦然：

double DateToValue(const COleDateTime& Date)

COleDateTime ValueToDate(double Value)

假设您有还有一种格式的日期（比如time\_t或SYSTEMTIME）。这不是一个问题，由于COleDateTime对象能够从不同的时间格式构造（检查COleDateTime类的MSDN文档。以了解从哪种格式能够构造它）。

填充数据后，能够配置轴以显示所需的内容。 与日期/时间轴相关的几个功能可用：

void SetDateTimeIncrement(TimeInterval Interval, int Multiplier)void SetDateTimeFormat(bool bAutomatic, const TChartString& strFormat)void SetReferenceTick(COleDateTime referenceTick)

第一个同意您指定轴上显示的两个节拍之间的间隔。两个节拍之间的间隔将遵守正确的时间，这意味着假设指定1个月的节拍增量(Interval=CChartAxis::tiMonth and Multiplier=1)，则两个节拍之间的间隔将是不规则的（28,30或31天）。第二个函数同意您指定刻度标签的格式。控件依据刻度间隔自己主动格式化刻度标签，但您能够通过调用此函数覆盖它。检查MSDN上的COleDateTime :: Format函数的文档以获取很多其它信息。最后。SetReferenceTick（COleDateTime referenceTick）函数同意您为轴指定一个參考标记。參考标记是用作绘制标记的參考的日期：在该日期总是存在标记。

当您在SetDateTimeIncrement函数中指定的multiplier 不是1时，这非常实用。

比如，假设您指定了3个月的单位增量。而且您希望在2月（因此。5月。8月。…）有一个单位，那么您能够调用此函数将2月1日设置为參考单位。默认设置为2000年1月1日。

以下是一个简单的代码片段。它创建一个日期/时间轴。并显示不同函数的使用方法：

void CMyClass::Init(){

// Sets the axis min value to January 1st 2006 and the axis

// max value to December 31st 2007.

COleDateTime minValue(2006,1,1,0,0,0);

COleDateTime maxValue(2007,12,31,0,0,0);

pBottomAxis->SetMinMax(CChartCtrl::DateToValue(minValue),

CChartCtrl::DateToValue(maxValue));

// Sets the tick increment to 4 months (disable automatic tick increment)

pBottomAxis->SetTickIncrement(false, CChartDateTimeAxis::tiMonth, 4);

// Sets the tick label format for instance "Jan 2006"

pBottomAxis->SetTickLabelFormat(false, \_T("%b %Y"));

}

## **自己定义外观**

控件的外观方面能够依据不同的应用场景做出更改。比方控件的不同部分（图例，标题，背景，…）都能够改动。

全部与这些对象的交互是通过CChartCtrl类来实现：一些将依据须要创建（比如axes或series），一些在创建控件时创建（legend，titles，…）。

一般来说，你永远不会自己创建这些对象，而是将该任务委派给CChartCtrl类。

唯一的例外是当您要使用自己定义轴或自己定义系列（请參阅“扩展功能”部分）。 比如。以下是一个代码段。设置渐变背景，并将图例放在控件的底部：

void CMyClass::Init(){

// SNIP

// Disable the refresh of the control

m\_ChartCtrl.EnableRefresh(false);

// Set the gradient for the background

m\_ChartCtrl.SetBackGradient(RGB(255,255,255),RGB(125,125,255),gtVertical);

// Dock the legend at the bottom

m\_ChartCtrl.GetLegend()->DockLegend(CChartLegend::dsDockBottom);

// Specifies that the legend entries are horizontally stacked

m\_ChartCtrl.GetLegend()->SetHorizontalMode(true);

// Re-enable the refresh of the control

m\_ChartCtrl.EnableRefresh(true);

}

**重要：**从版本号1.4的控件。每次调用控件上的一个属性将导致控件的全然刷新（即使像改变一些文本的字体或对象的颜色）。

为了避免在没有必要时刷新控件（比如。当您同一时候更改多个属性时）。应首先禁用刷新，更改属性，然后又一次启用刷新，如上面的代码段所看到的 。

自从1.5版的控件開始支持UNICODE。 全部出现的std :: string对象已被TChartString对象替换，这仅仅是一个typedef，假设未启用UNICODE，则解析为std :: string。并在启用UNICODE时解析为std :: wstring。

## **响应鼠标事件**

有时，应用程序须要响应用户鼠标操作。

比如。假设用户点击点，则程序能够显示关于被点击的点的信息。这一节将解释怎样做到。

尽管原理是有点不同，可是不管你想听在图表上的一般鼠标事件本身（点击轴。图例，…）或你是否对特定系列的鼠标事件感兴趣。 这两种情况都非常easy实现。

### **响应图表上的鼠标事件**

你必须实现CChartMouseListener接口。覆盖你感兴趣的方法，并通过调用CChartCtrl :: RegisterMouseListener（CChartMouseListener \* pMouseListener）将该类的实例注冊到图表控件。

依据鼠标事件发生在控件的哪个部分：标题，图例，轴或画图区，调用该接口上的不同函数。 对于全部这些函数。总是传递两个參数：MouseEvent。它是列出鼠标事件类型（鼠标移动，左键单击，…）的枚举，以及一个CPoint对象，它包括的发生事件的点的屏幕坐标。 对于某些函数。须要时传递一些其它參数。 比如。当单击一个轴时，指向该轴的指针被传递给该函数。

以下是CChartMouseListener的实现。它对轴的点击作出反应。并显示一个消息框：

class CCustomMouseListener : public CChartMouseListener

{public:

void OnMouseEventAxis(MouseEvent mouseEvent, CPoint point,

CChartAxis\* pAxisClicked)

{

if (mouseEvent == CChartMouseListener::LButtonDoubleClick)

{

MessageBox(\_T("Axis clicked"), \_T("Info"), MB\_OK);

}

}

};

然后你必须创建一个这个类的实例并注冊它：

m\_pMouseListener = new CCustomMouseListener();

m\_ChartCtrl.RegisterMouseListener(m\_pMouseListener);

这里也须要自己删除指针。

### **响应系列上的鼠标事件**

响应系列上的事件与响应一般事件非常类似，仅仅是监听器是CChartSeriesMouseListener的一个实例，它是一个模板类，模板參数是系列的点类型。 这是须要的，以避免当您要检索点的特定值时不必要的转型。 还有一个差别是。您必须在系列本身上注冊监听器。而不是在图表控件上注冊。

以下是CChartSeriesMouseListener的实现，它对系列的点击做出反应，假设点击发生在点上。它将显示一个带有点的Y值的消息框：

class CCustomMouseListener : public CChartSeriesMouseListener<SChartXYPoint>

{public:

void OnMouseEventSeries(MouseEvent mouseEvent, CPoint point,

CChartSerieBase<SChartXYPoint>\* pSerie, unsigned uPointIndex)

{

if (mouseEvent == CChartMouseListener::LButtonDoubleClick &&

uPointIndex != INVALID\_POINT)

{

TChartStringStream ssText;

SChartXYPoint Point = pSeries->GetPoint(uPointIndex);

ssText << \_T("Y value=") << Point.Y;

TChartString strText = ssText.str();

MessageBox(NULL,strText.c\_str(), \_T("Info"), MB\_OK);

}

}

};

**注意:**当用户不点击一个点时，OnMouseEventSeries函数也能够被调用。 比如当用户在两个点之间但仍然在该系列上点击时情况。 在这样的情况下，为uPointIndex參数传递INVALID\_POINT。

然后，您必须创建此类的实例并将其注冊到系列中：

m\_pMouseListener = new CCustomMouseListener();

m\_pSeries.RegisterMouseListener(m\_pMouseListener);

**注意：**仅仅有当系列操作SChartXYPoint类型的点（点。线，面或者柱状图系列）时，这才会起作用。 假设不是这样，您的代码将生成编译错误。

出于性能原因，禁止检測系列上的鼠标移动事件。 要启用它。请參阅doxygen文档中的CChartSerie :: EnableMouseNotifications函数。

## **使用光标**

您还能够向控件加入光标。 支持两种类型的光标：“十字线”光标和“拉线”光标。

第一个是在鼠标移动的画图区域上显示的简单十字，第二个是与特定轴关联的水平或垂直线，您能够通过单击它并使用鼠标移动来拖动。

对于每一个光标，您能够注冊一个侦听器。以便在移动光标时通知它。 这里是一段代码。用于创建与底部和左側轴相关联的“十字准线”光标以及与底部轴相关联的“拉线”光标：

// Creates a cross-hair cursor associated with the two primary axes.

CChartCrossHairCursor\* pCrossHair =

m\_ChartCtrl.CreateCrossHairCursor();// Creates a dragline cursor associated with the bottom axis.

CChartDragLineCursor\* pDragLine =

m\_ChartCtrl.CreateDragLineCursor(CChartCtrl::BottomAxis);// Hides the mouse when it is over the plotting area.

m\_ChartCtrl.ShowMouseCursor(false);

注意到对CChartCtrl :: ShowMouseCursor的调用结束。

默认情况下，鼠标总是可见的，可是当您使用十字光标时。当它在画图区域时隐藏有时是须要的。

假设希望在光标位置更改时收到通知，则必须实现CChartCursorListener接口，创建其实例并使用光标注冊它：

class CCustomCursorListener : public CChartCursorListener

{public:

void OnCursorMoved(CChartCursor \*pCursor, double xValue, double yValue)

{

TChartStringStream ssText;

ssText << \_T("Cursor moved: xPos=") << xValue << \_T(", yPos=") << yValue;

// Do something with the string...

}

};

CCustomCursorListener\* pCursorListener = new CCustomCursorListener;

pDragLine->RegisterListener(pCursorListener);

OnCursorMoved函数接收一个X和Y值。但对于拖动光标，仅仅使用这些值中的一个：假设光标与水平轴相关联，则使用X值。否则使用Y值。

## **使用平移和缩放功能**

在版本号1.1的控件中，缩放和平移功能已被加入到控件。 使用鼠标左键控制缩放，用鼠标右键控制平移。 要缩放图表的特定部分。仅仅需左键单击图表（这将是缩放矩形的左上角）。然后拖动到右下角。

将出现一个矩形。 一旦松开鼠标按钮。四个轴将自己主动调整到您选择的区域。 默认情况下启用缩放，但您能够通过调用CChartCtrl :: SetZoomEnabled（bool bEnabled）来禁用缩放。 您还能够通过调用CChartAxis :: SetZoomLimit（double dLimit）为每一个轴指定缩放限制。 它指定缩放时轴的最小范围。

默认值为0.001。

要平移控件，右键单击控件上的某处并移动鼠标。

鼠标下的点将“尾随”鼠标的移动（实际上，轴的最小和最大值将改变）。 默认情况下启用平移，但您能够通过调用CChartCtrl :: SetPanEnabled（bool bEnabled）来禁用它。

假设您左键单击图表（比如開始缩放），但假设您移动到左上角，全部使用缩放和平移功能所做的改动将被取消（控制将处于它的状态 在使用平移和缩放操作之前）。 最后，还有一种方法通过调用CChartAxis :: SetPanZoomEnabled（bool bEnabled）禁用特定轴的平移和缩放功能。

## **利用快速功能**

线和点系列同意以快速率绘制数据。 这通常在要绘制来自外部设备（比如，传感器）的数据时完毕。 这是可能的。由于当您向此类系列加入点时。控件不会全然刷新，仅仅会绘制最后一个点（或最后一个线段）。这是非常有效的。 可是。假设希望控件能够足够快地绘制数据，则必须考虑几点。

一个重要的事情是，使用自己主动轴可能会减少非常多性能。

这是由于假设一个点绘制在轴范围之外，则轴范围将被自己主动调整，这意味着控制将被全然刷新。

因此，假设您使用自己主动底部轴线并具有“滚动”轨迹，则每一个新点都将位于轴的当前范围之外，而且将对每一个点执行控制刷新。处理的更好的方法是使用固定轴而且每秒手动地添加轴的范围（或以合理的速率）。

还有一个重要的点是，你不应该在向一个系列加入一个新点之后调用RefreshCtrl。

这当然会全然地刷新控件，可是应该避免这样做。

最后，假设您须要同一时候应用几个改动或加入几个点到控件。您应该在EnableRefresh（false）和EnableRefresh（true）之间封装这些调用（请參阅“自己定义外观”部分）。

## **扩展功能**

在某些特定情况下，您须要使用新功能扩展控件，比如新的系列类型。眼下，您能够自己定义四个组件：序列。轴，点标签和光标。

要提供新轴，新标签或新光标。您仅仅需继承基类（CChartAxis，CChartLabel或CChartCursor）并实现所需的虚拟函数。一旦完毕。您能够通过调用不同函数的自己定义版本号（CChartCtrl :: AttachCustomAxis。CChartCtrl :: AttachCustomLabel或CChartCtrl :: AttachCustomCursor）附加您的新对象。 CChartLabel类是一个模板类。这个主题有点广泛，进入了非常多细节。但最简单的方法是看看不同的现有类。

假设你想提供新的系列。这有点不同：你首先要考虑你想要在你的系列中操纵的点的类型。假设你仅仅须要使用X和Y值来操作点，那么你能够继承CChartXYSerie，它提供了非常多功能来操作这些点。

然后，您必须实现所需的虚拟函数。

看看以下的系列：CChartLineSerie，CChartPointSerie。CChartSurfaceSerie和CChartBarSerie详细演示样例。

假设你的系列操纵其它类型的点，那么你首先必须为点包括以下方法创建一个结构：double GetX()。double GetXMin()，double GetXMax()，double GetY()，double GetYMin()和double GetYMax()。

一旦完毕。您必须继承CChartSerieBase并将此点作为模板參数。

然后，您必须提供所需的虚拟功能。看看以下的系列详细样例：CChartCandlestickSerie和CChartGanttSerie。

## **Upgrading from Version 1.x to Version 2.0**

在版本号2.0中。对控件进行重构，导致API的更改。 基本的可见变化是每一个轴类型如今有其单独的类（CChartStandardAxis。CChartDateTimeAxis和CChartLogarithmicAxis）。

这也意味着默认情况下没有创建轴，而且您必须在向图表加入系列之前自己创建轴（否则代码将断言）。

这包括在“操纵轴”部分。

还有一个变化是加入系列到图表的方式：AddSerie已经在CChartCtrl类中删除。并已被帮助函数替代，以创建特定的系列类型（CreateLineSerie。CreatePointsSerie，…）。 这些函数返回确切的系列类型。因此不再须要铸造。

这在“操纵系列”一节中有详细描写叙述。

## **Upgrading from Version 2.x to Version 3.x**

版本号3.0.0的主要变化是，系列基类如今已经作为模板类，模板參数是系列操作的点类型。假设您没有通过提供新的系列类型扩展控件，这将不会在您的代码中有所不同。

假设你提供了一个新的系列类型。你的类必须继承CCharSerieBase并提供它操作的点的类型。

假设你的系列使用仅仅有X和Y值的点，你能够简单地继承CChartXYSerie。

看看现有的系列很多其它的样例。

还有一个小的改动是标签提供程序如今也是模板类（出于同样的原因）。而且监听系列中的鼠标事件如今从图表上的鼠标事件中分离出来。这两点在“在点上加入标签”部分和“鼠标事件通知”部分中有非常好的解释。

最后，CChartAxis :: SetAutomatic方法已被标记为已弃用。您应该使用CChartAxis :: SetAutomaticMode（已经引入了一个额外的自己主动模式）。

## **样例**

本节仅仅是两个代码片段。显示了怎样使用控件。 第一个片段再现了示波器演示样例的图像（參见本文顶部）。第二个演示样例再现了“2008年收入”图像。

代码是文档化的，所以它不应该太难理解。

Oscilloscope example:

// Disable the refresh of the control (avoid multiple refresh).

m\_ChartCtrl.EnableRefresh(false);// Create a bottom and left axes

CChartStandardAxis\* pBottomAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::BottomAxis);

CChartStandardAxis\* pLeftAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::LeftAxis);// Sets the min and max values of the bottom and left axis to -15 -> 15

pBottomAxis->SetMinMax(-15,15);

pLeftAxis->SetMinMax(-15,15);

// Add a new series of type line to the control and add data to it

CChartLineSerie\* pLineSeries = m\_ChartCtrl.CreateLineSerie();// Specifies that the points in the series are not ordered (needed to be able// to draw an ellipse).

pLineSeries->SetSeriesOrdering(poNoOrdering);for (int i=0;i<361;i++)

{

double X = 10 \* sin(i/360.0 \* 2 \* 3.141592);

double Y = 10 \* cos( (i-60)/360.0 \* 2 \* 3.141592);

pLineSeries->AddPoint(X,Y);

}

// Defines the different colors (back color, axes color, ...)

COLORREF BackColor = RGB(0,50,0);

COLORREF GridColor = RGB(0,180,0);

COLORREF TextColor = RGB(0,180,0);

COLORREF SerieColor = RGB(0,255,0);

// Specifies a sunken border for the control

m\_ChartCtrl.SetEdgeType(EDGE\_SUNKEN);

// Sets the color of the border and the back color

m\_ChartCtrl.SetBorderColor(TextColor);

m\_ChartCtrl.SetBackColor(BackColor);

//Sets the color of the different elements of the bottom axis

m\_ChartCtrl.GetBottomAxis()->SetAxisColor(TextColor);

m\_ChartCtrl.GetBottomAxis()->SetTextColor(TextColor);

m\_ChartCtrl.GetBottomAxis()->GetGrid()->SetColor(GridColor);

// Sets the color of the different elements of the left axis

m\_ChartCtrl.GetLeftAxis()->SetAxisColor(TextColor);

m\_ChartCtrl.GetLeftAxis()->SetTextColor(TextColor);

m\_ChartCtrl.GetLeftAxis()->GetGrid()->SetColor(GridColor);

// Sets the color of the title, change the font to Times New Roman// and add a string

m\_ChartCtrl.GetTitle()->SetColor(TextColor);

m\_ChartCtrl.GetTitle()->SetFont(140,\_T("Times New Roman"));

m\_ChartCtrl.GetTitle()->AddString(\_T("An example of oscilloscope"));

// Change the color of the line series

pLineSeries->SetColor(SerieColor);

// Finally re-enable the refresh of the control. This will refresh the// control if any refresh was still 'pending'.

m\_ChartCtrl.EnableRefresh(true);

“Income over 2008” example:

srand((unsigned int)time(NULL));

// Disable the refresh

m\_ChartCtrl.EnableRefresh(false);COleDateTime Min(2008,1,1,0,0,0);COleDateTime Max(2008,10,1,0,0,0);// Create the bottom axis and configure it properly

CChartDateTimeAxis\* pBottomAxis =

m\_ChartCtrl.CreateDateTimeAxis(CChartCtrl::BottomAxis);

pBottomAxis->SetMinMax(Min,Max);

pBottomAxis->SetDiscrete(true);

pBottomAxis->SetTickIncrement(false,CChartDateTimeAxis::tiMonth,1);

pBottomAxis->SetTickLabelFormat(false,\_T("%b"));// Create the left axis and configure it properly

CChartStandardAxis\* pLeftAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::LeftAxis);

pLeftAxis->SetMinMax(0,100);

pLeftAxis->GetLabel()->SetText(\_T("Units sold"));// Create the right axis and configure it properly

CChartStandardAxis\* pRightAxis =

m\_ChartCtrl.CreateStandardAxis(CChartCtrl::RightAxis);

pRightAxis->SetVisible(true);

pRightAxis->GetLabel()->SetText(\_T("Income (kEuros)"));

pRightAxis->SetMinMax(0,200);

// Configure the legend

m\_ChartCtrl.GetLegend()->SetVisible(true);

m\_ChartCtrl.GetLegend()->SetHorizontalMode(true);

m\_ChartCtrl.GetLegend()->UndockLegend(80,50);// Add text to the title and set the font & color

m\_ChartCtrl.GetTitle()->AddString(\_T("Income over 2008"));

CChartFont titleFont;

titleFont.SetFont(\_T("Arial Black"),120,true,false,true);

m\_ChartCtrl.GetTitle()->SetFont(titleFont);

m\_ChartCtrl.GetTitle()->SetColor(RGB(0,0,128));// Sets a gradient background

m\_ChartCtrl.SetBackGradient(RGB(255,255,255),RGB(150,150,255),gtVertical);

// Create two bar series and a line series and populate them with data

CChartBarSerie\* pBarSeries1 = m\_ChartCtrl.CreateBarSerie();

CChartBarSerie\* pBarSeries2 = m\_ChartCtrl.CreateBarSerie();

CChartLineSerie\* pLineSeries = m\_ChartCtrl.CreateLineSerie(false,true);int lowIndex = -1;int lowVal = 999;for (int i=0;i<9;i++)

{

COleDateTime TimeVal(2008,i+1,1,0,0,0);

int DesktopVal = 20 + rand()%(100-30);

pBarSeries1->AddPoint(TimeVal,DesktopVal);

int LaptopVal = 10 + rand()%(80-20);

pBarSeries2->AddPoint(TimeVal,LaptopVal);

int Income = DesktopVal + LaptopVal\*1.5;

if (Income < lowVal)

{

lowVal = Income;

lowIndex = i;

}

pLineSeries->AddPoint(TimeVal,Income);

}// Configure the series properly

pBarSeries1->SetColor(RGB(255,0,0));

pBarSeries1->SetName(\_T("Desktops"));

pBarSeries2->SetColor(RGB(68,68,255));

pBarSeries2->SetGradient(RGB(200,200,255),gtVerticalDouble);

pBarSeries2->SetName(\_T("Laptops"));

pBarSeries2->SetBorderColor(RGB(0,0,255));

pBarSeries2->SetBorderWidth(3);

pLineSeries->SetColor(RGB(0,180,0));

pLineSeries->SetName(\_T("Total income"));

pLineSeries->SetWidth(2);

pLineSeries->EnableShadow(true);

// Add a label on the line series.

TChartStringStream labelStream;

labelStream << \_T("Min income: ") << lowVal;

CChartBalloonLabel<SChartXYPoint>\* pLabel =

pLineSeries->CreateBalloonLabel(lowIndex, labelStream.str() + \_T(" kEuros"));

CChartFont labelFont;

labelFont.SetFont(\_T("Microsoft Sans Serif"),100,false,true,false);

pLabel->SetFont(labelFont);

// Re enable the refresh

m\_ChartCtrl.EnableRefresh(true);

## **Feedback**

Quite a lot of work is involved in the development of this control and, as any other software project, it might still contain bugs or errors in the documentation. If you encounter such a problem, please let me know (even if you fixed it yourself) so that I can fix the issue as soon as possible. Other users of the control will thank you for that. The same if you encounter errors in the documentation or typos in the article.

I’m also more or less constantly working on this control to add new features. If you have some requirement for a nice feature that could be useful for others, please let me know and I’ll add it to my wishlist. However, as I’m working on this control in my spare time, my time is rather limited.

Finally, if you liked this control, do not hesitate to drop me a word in the discussion forum or to rate the article, this is much appreciated. Thank you.

## **History**

* 08/05/2006: Release of version 1.0
* 19/08/2006: Release of version 1.1
  + Bug fix in ScreenToValue function (CChartAxis)
  + Bug fix in RemoveAllSeries function (CChartCtrl)
  + Added support for manual zoom
  + Added support for mouse panning
  + Ability to specify a tick increment on the axis
  + Added support for resizing the control
* 09/04/2007: Release of version 1.2
  + GDI leak corrected
  + Invisible series are not taken in account for auto axis and legend (thanks to jerminator-jp)
  + Ability to change the text color of the axis
  + Ability to change the color of the border of the drawing area
  + Surface series added
* 16/02/2008: Release of version 1.3
  + Added date/time axis
  + Bug fix in how the logarithmic labels are displayed (trailing 0)
  + Ability to change the color of the zoom rectangle
  + Removed compiler warnings for VC2005
  + Bug fix in the zoom
* 14/04/2008: Release of version 1.4
  + Added support for scrollbars
  + Bar series added
  + Legend can be docked on any side or floating
  + Support for legend in horizontal mode
  + Support for transparent background on the legend
  + Support for shadow for several objects
  + RemovePointsFromBegin, RemovePointsFromEnd and AddPoints in the CChartSeries class
  + Support for gradient background
  + EnableRefresh and UndoPanZoom functions added in CChartCtrl
  + Possibility to enable/disable the zoom for a specific axis and to set its limit
  + Speed improvement on the series (min and max cached, ordering of the series)
  + Series can be removed using their pointers
  + Bug fix for invisible series in the legend
  + Bug fix for logarithmic axis (1 digit was not displayed)
  + Bug fix when removing series from the control
  + Bug fix if the pen width is bigger than 1 for line series
  + Bug fix for automatic axis
  + 20/08/2008: Release of version 1.5
  + Added support for UNICODE
  + Added support for printing
  + Auto-hide scrollbars
  + Baseline selection for bar series
  + Performance patch
  + Scrollbar flickering removed (see here)
  + Bug fix: scrollbar is now updated when axis is panned
  + Bug fix: calling AddPoint was not drawing the new point
  + Bug fix: tick labels for log axis were not always correct (rounding error)
  + Bug fix: last point of ChartPointSerie was not displayed
  + Bug fix: moving the mouse outside the control doesn’t stop the zoom or pan operation (the button can be released outside the control)
* 13/04/2009: Release of version 2.0
  + The different axis types are now separated into different classes
  + Modified the way to add series to the control for improved flexibility
  + Added cursors
  + Ability to display discrete axes
  + Ability to be notified about mouse events occurring on the control
  + Added labels on points
  + Ability to display a smooth curve
  + Added ChartFont: allows for italic, bold or underlined fonts
  + Added the SetReferenceTick function for date/time axis
  + Ability to store user data for each point
  + Series now have an Id
  + Removed the CChartObject class
  + Points are now stored in a standard array instead of a std::vector for efficiency
  + Binary search implemented for finding the first and last visible points (for efficiency)
  + The line series now uses PolyLine instead of MoveTo/LineTo (efficiency)
  + Bug fix when using date/time axis with a tick interval in years
  + Bug fix: bar series were drawn from the wrong axis
* 11/06/2009: Release of version 2.0.1
  + Optimization: the pan feature has been smoothed
  + Optimization: points with the same X and Y values are not plotted anymore for the line series.
  + Bug fix: in some situations, the code was crashing when accessing points outside the valid range
  + Bug fix: when series were removed, the legend was accessing removed series (which crashed)
  + Bug fix: when a series was cleared, new points were not drawn properly
  + Bug fix: inserting a point for which the X value already existed in the series did not add the point properly
  + Bug fix with the CChartFont class
* 07/08/2009: Release of version 2.0.2
  + Bug fix: the control was crashing when a series with no points and no ordering was added
  + Bug fix: the shadow of the line was not drawn correctly
  + Bug fix: when an automatic date/time axis was used without any data, the code crashed
* 28/12/2009: Release of version 3.0.0
  + Series are now template classes with the template parameter being the point type. This allows the control to manipulate any type of points
  + Added candlestick and Gantt series
  + Added support to save the chart to an image file
  + Bar series can be stacked
  + Added a new automatic mode for axes: the screen automatic mode
  + Listening for mouse events on a series has been moved to a CChartSeriesMouseListener class
  + Bug fix: when a point X or Y value is modified, the series is reordered
  + Bug fix: setting a tick increment on a standard axis did not show the digits properly
* 17/01/2010: Release of version 3.0.1
  + Bug fix: when using labels with the points series, the border of the points was changing color. Fixed by providing a way to specify the border color.
  + Bug fix: the code was crashing when clicking on a series without having registered a mouse listener on the series.
  + Bug fix: detection of mouse events on certain series type was crashing
  + Bug fix: CChartTitle::SetVisible was not implemented
* 13/07/2010: Release of version 3.0.2
  + Bug fix: the high-speed functionality has been removed by mistake
  + Bug fix: the draw function of the line series was not drawing points
  + Bug fix: replaced Clear() by clear() in the ClearSerie function.
  + Bug fix: Added implementation of ctor/dtor for the CChartCursorListener class
  + Bug fix: memory leak when the series was cleared (labels were not deleted)

## **Thanks**

I would like to thank all the people from this community, they were a great help when I started programming. Thanks also to all the people who contributed to this control with their various help or feedback: toxcct, Chris Maunder, Kevin Hoffman, jerminator-jp, Laurie Gellatly, Eugene Pustovoyt, Andrej Ritter, Nick Holgate, Nick Schultz, Johann Obermayr, Pierre Schramm and Kevin Winter. A special thanks to Bruno Lavier for the time spent working on the control. I hope I didn’t forget anybody.

## **Links**

* [Chris Maunder’s Colour Picker Control](http://blog.csdn.net/xuanyuanlei1020/article/details/50468061) (used in the demo application)
* [Drawing smooth curves with Bezier Primitives](http://www.codeproject.com/KB/graphics/BezierSpline.aspx)

## **License**

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)