**ST\_Curve开发文档**

[1 主要特点： 10](#_Toc29207)

[1.1 属性： 12](#_Toc15546)

[1.2 函数： 12](#_Toc10138)

[1.2.1 坐标轴相关： 12](#_Toc4560)

[1.2.2 坐标相关： 12](#_Toc28959)

[1.2.3 单位相关： 12](#_Toc24077)

[1.2.4 图例相关： 13](#_Toc2207)

[1.2.5 添加曲线点： 13](#_Toc16131)

[1.2.6 删除曲线点： 13](#_Toc6908)

[1.2.7 翻页相关： 13](#_Toc25942)

[1.2.8 缩放相关： 13](#_Toc6245)

[1.2.9 模式（移动模式、显示模式、网格模式）相关： 13](#_Toc531)

[1.2.10 背景图相关： 13](#_Toc20714)

[1.2.11 导出图片或导出导入文件： 13](#_Toc17265)

[1.2.12 标题脚注： 13](#_Toc1701)

[1.2.13 枚举（并修改）某条曲线： 13](#_Toc1247)

[1.2.14 枚举所有曲线： 13](#_Toc10780)

[1.2.15 枚举图例： 14](#_Toc13396)

[1.2.16 联动相关： 14](#_Toc23245)

[1.2.17 Z-Order及选中相关： 14](#_Toc16179)

[1.2.18 范围相关： 14](#_Toc21761)

[1.2.19 移动曲线： 14](#_Toc11077)

[1.2.20 辅助函数： 14](#_Toc32605)

[1.2.21 曲线长度控制相关： 14](#_Toc25754)

[1.2.22 页数量变化消息： 14](#_Toc7622)

[1.2.23 三维相关函数： 14](#_Toc20299)

[1.2.24 基点相关： 14](#_Toc7313)

[1.2.25 填充方向相关： 14](#_Toc25715)

[1.2.26 缓存控制相关： 15](#_Toc15900)

[1.2.27 曲线操作函数： 15](#_Toc7138)

[1.2.28 限制坐标相关： 15](#_Toc1600)

[1.2.29 平滑曲线相关： 15](#_Toc12359)

[1.2.30 插件相关： 15](#_Toc23099)

[1.2.31 起始点、结束点、选中点标识相关： 15](#_Toc31218)

[1.2.32 快捷键相关： 15](#_Toc16644)

[1.2.33 横坐标位置相关： 15](#_Toc15450)

[1.2.34 注解相关： 15](#_Toc20263)

[1.2.35 事件开关相关： 15](#_Toc25858)

[1.2.36 定点缩放相关： 15](#_Toc25243)

[1.2.37 自动刷新相关： 15](#_Toc28288)

[1.2.38 ToolTip相关： 16](#_Toc13874)

[1.2.39 无限曲线相关： 16](#_Toc28841)

[1.2.40 鼠标滚轮相关： 16](#_Toc20877)

[1.2.41 其它函数： 16](#_Toc28466)

[2 属性及函数： 16](#_Toc16834)

[2.1 属性（9个） 16](#_Toc16966)

[2.2 函数（232个） 18](#_Toc10255)

[2.2.1 BOOL SetVInterval(short VInterval); 18](#_Toc4321)

[2.2.2 BOOL SetHInterval(short HInterval); 18](#_Toc22356)

[2.2.3 short GetScaleInterval(); 18](#_Toc21233)

[2.2.4 SetGraduationSize(long size); 18](#_Toc15912)

[2.2.5 long GetGraduationSize(); 18](#_Toc16778)

[2.2.6 BOOL SetLegendSpace(short LegendSpace); 18](#_Toc2753)

[2.2.7 short GetLegendSpace(); 18](#_Toc3506)

[2.2.8 BOOL SetBeginValue(float fBeginValue); 18](#_Toc7954)

[2.2.9 float GetBeginValue(); 18](#_Toc14987)

[2.2.10 BOOL SetBeginTime(LPCTSTR pBeginTime); 18](#_Toc24256)

[2.2.11 BOOL SetBeginTime2(DATE fBeginTime); 18](#_Toc7664)

[2.2.12 CString GetBeginTime(); 18](#_Toc28131)

[2.2.13 DATE GetBeginTime2(); 18](#_Toc25269)

[2.2.14 BOOL SetTimeSpan(double TimeStep); 19](#_Toc28333)

[2.2.15 double GetTimeSpan(); 19](#_Toc556)

[2.2.16 BOOL SetValueStep(float ValueStep); 19](#_Toc32109)

[2.2.17 float GetValueStep(); 19](#_Toc5794)

[2.2.18 BOOL SetVPrecision(short Precision); 19](#_Toc25993)

[2.2.19 short GetVPrecision(); 19](#_Toc14063)

[2.2.20 BOOL SetUnit(LPCTSTR pUnit); 19](#_Toc8034)

[2.2.21 CString GetUnit(); 19](#_Toc19273)

[2.2.22 BOOL SetHUnit(LPCTSTR pHUnit); 19](#_Toc26177)

[2.2.23 CString GetHUnit(); 19](#_Toc3584)

[2.2.24 short AddLegend(long Id, LPCTSTR pSign, OLE\_COLOR PenColor, short PenStyle,short LineWidth, OLE\_COLOR BrushColor, 20](#_Toc28348)

[short BrushStyle, short CurveMode, short NodeMode, short Mask, 20](#_Toc16079)

[BOOL bUpdate); 20](#_Toc18739)

[2.2.25 short AddLegendHelper(long Id, LPCTSTR pSign,OLE\_COLOR PenColor, short PenStyle, short LineWidth, boolean bUpdate); 21](#_Toc1499)

[2.2.26 BOOL IsLegend(LPCTSTR pSign); 21](#_Toc6491)

[2.2.27 BOOL GetLegend(LPCTSTR pSign, OLE\_COLOR\* pPenColor, short\* pPenStyle, short\* pLineWidth, OLE\_COLOR\* pBrushColor, 21](#_Toc11169)

[short\* pBrushStyle, short\* pCurveMode, short\* pNodeMode); 21](#_Toc21230)

[2.2.28 OOL DelLegend(long Id, BOOL bAll, BOOL bUpdate); 22](#_Toc13663)

[2.2.29 BOOL DelLegend2(LPCTSTR pSign, BOOL bUpdate); 22](#_Toc27888)

[2.2.30 BOOL SetXYFormat(LPCTSTR pSign, short Format); 22](#_Toc12735)

[2.2.31 short GetXYFormat(LPCTSTR pSign); 22](#_Toc18489)

[2.2.32 short GetXYFormat2(short nIndex); 22](#_Toc8135)

[2.2.33 CString QueryLegend(long Id); 22](#_Toc11035)

[2.2.34 BOOL MoveCurveToLegend(long Id, LPCTSTR pSign); 22](#_Toc27227)

[2.2.35 BOOL ChangeLegendName(LPCTSTR pFrom, LPCTSTR pTo); 22](#_Toc856)

[2.2.36 BOOL SetHLegend(LPCTSTR pHLegend); 23](#_Toc1243)

[2.2.37 CString GetHLegend(); 23](#_Toc26832)

[2.2.38 short AddMainData(long Id, LPCTSTR pTime, float Value, short State,short VisibleState, BOOL bAddTrail); 23](#_Toc22946)

[2.2.39 short AddMainData2(long Id, DATE Time, float Value, short State, 23](#_Toc28298)

[short VisibleState, BOOL bAddTrail); 23](#_Toc359)

[2.2.40 long AddMemMainData(OLE\_HANDLE pMemMainData, long MemSize, BOOL bAddTrail); 23](#_Toc31999)

[2.2.41 void DelRange(long Id, DATE BTime, DATE ETime,short Mask, 24](#_Toc13018)

[BOOL bAll, BOOL bUpdate); 24](#_Toc2006)

[2.2.42 void DelRange2(long Id, long nIndex, long nCount, BOOL bAll, BOOL bUpdate); 24](#_Toc5101)

[2.2.43 BOOL FirstPage(BOOL bLast, BOOL bUpdate); 25](#_Toc17012)

[2.2.44 short GotoPage(short RelativePage, BOOL bUpdate); 25](#_Toc8877)

[2.2.45 BOOL SetZoom(short Zoom); 25](#_Toc20999)

[2.2.46 short GetZoom(); 25](#_Toc22935)

[2.2.47 BOOL SetHZoom(short Zoom); 25](#_Toc12540)

[2.2.48 short GetHZoom(); 25](#_Toc5060)

[2.2.49 BOOL SetMaxLength(long MaxLength, long CutLength); 26](#_Toc22262)

[2.2.50 long GetMaxLength(); 26](#_Toc5999)

[2.2.51 long GetCutLength(); 26](#_Toc4553)

[2.2.52 long GetCurveLength(long Id); 26](#_Toc8600)

[2.2.53 BOOL SetShowMode(short ShowMode); 26](#_Toc7943)

[2.2.54 short GetShowMode(); 26](#_Toc25554)

[2.2.55 BOOL SetMoveMode(short MoveMode); 26](#_Toc21271)

[2.2.56 short GetMoveMode(); 26](#_Toc11653)

[2.2.57 void SetFont(OLE\_HANDLE hFont); 27](#_Toc16886)

[2.2.58 OLE\_HANDLE GetFont(); 27](#_Toc1058)

[2.2.59 BOOL AddImageHandle(LPCTSTR pFileName, BOOL bShared); 27](#_Toc16144)

[2.2.60 void AddBitmapHandle(OLE\_HANDLE hBitmap, BOOL bShared); 27](#_Toc18977)

[2.2.61 BOOL AddBitmapHandle2(OLE\_HANDLE hInstance, LPCTSTR pszResourceName,BOOL bShared); 27](#_Toc29908)

[2.2.62 BOOL AddBitmapHandle3(OLE\_HANDLE hInstance, long nIDResource, BOOL bShared); 27](#_Toc21180)

[2.2.63 BOOL RemoveBitmapHandle(OLE\_HANDLE hBitmap, BOOL bDel); 27](#_Toc6637)

[2.2.64 BOOL RemoveBitmapHandle2(short nIndex, BOOL bDel); 27](#_Toc28875)

[2.2.65 long GetBitmapCount(); 28](#_Toc14668)

[2.2.66 OLE\_HANDLE GetBitmap(short nIndex); 28](#_Toc16682)

[2.2.67 short GetBitmapState(short nIndex); 28](#_Toc27487)

[2.2.68 short GetBitmapState2(OLE\_HANDLE hBitmap); 28](#_Toc13703)

[2.2.69 BOOL SetBkBitmap(short nIndex); 28](#_Toc4651)

[2.2.70 short GetBkBitmap(); 28](#_Toc19097)

[2.2.71 BOOL SetCanvasBkBitmap(short nIndex); 28](#_Toc30509)

[2.2.72 short GetCanvasBkBitmap(); 28](#_Toc27913)

[2.2.73 BOOL SetBkMode(short BkMode); 28](#_Toc24864)

[2.2.74 short GetBkMode(); 28](#_Toc27277)

[2.2.75 BOOL SetCanvasBkMode(short CanvasBkMode); 28](#_Toc10526)

[2.2.76 short GetCanvasBkMode(); 28](#_Toc30991)

[2.2.77 BOOL ExportImage(LPCTSTR pFileName); 28](#_Toc20927)

[2.2.78 long ExportImageFromPage(LPCTSTR pFileName, long Id, long nStartPage, long nCount, BOOL bAll, short Style); 28](#_Toc27603)

[2.2.79 long ExportImageFromTime(LPCTSTR pFileName, long Id, DATE BTime, 28](#_Toc30745)

[DATE ETime, short Mask, BOOL bAll, short Style); 28](#_Toc12020)

[2.2.80 long ExportMetaFile(LPCTSTR pFileName, long Id, long nBegin, 28](#_Toc30156)

[long nCount, BOOL bAll, short Style); 28](#_Toc9144)

[2.2.81 void BatchExportImage(LPCTSTR pFileName, long nSecond); 28](#_Toc1163)

[2.2.82 long ImportFile(LPCTSTR pFileName, short Style, BOOL bAddTrail); 28](#_Toc3202)

[2.2.83 BOOL GetOneTimeRange(long Id,DATE\* pMinTime, DATE\* pMaxTime); 30](#_Toc30391)

[2.2.84 BOOL GetOneValueRange(long Id, float\* pMinValue, float\* pMaxValue); 30](#_Toc28024)

[2.2.85 BOOL GetOneFirstPos(long Id,DATE\* pTime,float\* pValue,BOOL bLast); 30](#_Toc30217)

[2.2.86 BOOL GetTimeRange(DATE\* pMinTime, DATE\* pMaxTime); 30](#_Toc12041)

[2.2.87 BOOL GetValueRange(float\* pMinValue, float\* pMaxValue); 30](#_Toc99)

[2.2.88 void GetViableTimeRange(DATE\* pMinTime, DATE\* pMaxTime); 30](#_Toc31225)

[2.2.89 void TrimCoor(); 30](#_Toc23607)

[2.2.90 void EnableAutoTrimCoor(BOOL bEnable); 30](#_Toc16557)

[2.2.91 long TrimCurve(long Id, short State, long nBegin, long nCount, 31](#_Toc18731)

[short nStep, BOOL bAll); 31](#_Toc18361)

[2.2.92 long TrimCurve2(long Id, short State, DATE BTime, DATE ETime, 31](#_Toc5864)

[short Mask, short nStep, BOOL bAll); 31](#_Toc20484)

[2.2.93 short PrintCurve(long Id, DATE BTime, DATE ETime, short Mask, 31](#_Toc23335)

[short LeftMargin,short TopMargin,short RightMargin,short BottomMargin, 31](#_Toc22277)

[LPCTSTR pTitle, LPCTSTR pFootNote, short Flag, BOOL bAll); 31](#_Toc2438)

[2.2.94 long GetScaleNums(); 32](#_Toc17889)

[2.2.95 long ReportPageInfo(); 32](#_Toc683)

[2.2.96 BOOL ShowLegend(LPCTSTR pSign, BOOL bShow); 32](#_Toc28269)

[2.2.97 BOOL ShowCurve(long Id, BOOL bShow); 32](#_Toc7605)

[2.2.98 BOOL SelectCurve(long Id, BOOL bSelect); 32](#_Toc28010)

[2.2.99 short DragCurve(short xStep, short yStep, BOOL bUpdate); 32](#_Toc1553)

[2.2.100 BOOL VCenterCurve(long Id, BOOL bUpdate); 33](#_Toc884)

[2.2.101 BOOL GetSelectedCurve(long\* pId); 33](#_Toc22835)

[2.2.102 BOOL GotoCurve(long Id); 33](#_Toc17009)

[2.2.103 BOOL IsSelected(long Id); 33](#_Toc2745)

[2.2.104 BOOL IsLegendVisible(LPCTSTR pSign); 33](#_Toc7052)

[2.2.105 BOOL IsCurveVisible(long Id); 33](#_Toc2818)

[2.2.106 BOOL IsCurveInCanvas(long Id); 33](#_Toc31233)

[2.2.107 BOOL IsCurveClosed(long Id); 33](#_Toc19277)

[2.2.108 DATE GetTimeData(short nCurveIndex, long nIndex); 33](#_Toc14657)

[2.2.109 CString GetTimeData2(short nCurveIndex, long nIndex); 33](#_Toc21450)

[2.2.110 float GetValueData(short nCurveIndex, long nIndex); 33](#_Toc31030)

[2.2.111 short GetState(short nCurveIndex, long nIndex); 33](#_Toc842)

[2.2.112 BOOL GetPosData(short nCurveIndex, long nIndex, long\* px, long\* py); 33](#_Toc10676)

[2.2.113 BOOL InsertMainData(short nCurveIndex, long nIndex, 33](#_Toc3449)

[LPCTSTR pTime, float Value, short State, short Position, short Mask); 33](#_Toc11858)

[2.2.114 BOOL InsertMainData2(short nCurveIndex, long nIndex, 33](#_Toc13244)

[DATE Time, float Value, short State, short Position, short Mask); 33](#_Toc17995)

[2.2.115 BOOL DelPoint(short nCurveIndex, long nIndex); 33](#_Toc22060)

[2.2.116 BOOL CanContinueEnum(long Id, short nCurveIndex, long nIndex); 33](#_Toc14)

[2.2.117 long GetCurveCount(); 35](#_Toc9545)

[2.2.118 long GetCurve(long nIndex); 35](#_Toc32423)

[2.2.119 void SetCurveTitle(LPCTSTR pCurveTitle); 35](#_Toc16978)

[2.2.120 CString GetCurveTitle(); 35](#_Toc18222)

[2.2.121 long GetLegendCount(); 35](#_Toc19312)

[2.2.122 BOOL GetLegend2(long nIndex, OLE\_COLOR\* pPenColor, short\* pPenStyle,short\* pLineWidth, OLE\_COLOR\* pBrushColor,short\* pBrushStyle, , short\* pCurveMode，short\* pNodeMode); 35](#_Toc16086)

[2.2.123 long GetLegendIdCount(long nIndex); 35](#_Toc31246)

[2.2.124 long GetLegendId(long nLegendIndex, long nIdIndex); 35](#_Toc1624)

[2.2.125 BOOL SetBuddy(long hBuddy, short State); 35](#_Toc27923)

[2.2.126 short GetBuddyCount(); 35](#_Toc1482)

[2.2.127 long GetBuddy(short nIndex); 35](#_Toc17886)

[2.2.128 void EnableZoom(BOOL bEnable); 36](#_Toc21542)

[2.2.129 void EnableHZoom(BOOL bEnable); 36](#_Toc13875)

[2.2.130 BOOL SetHPrecision(short Precision); 36](#_Toc16394)

[2.2.131 short GetHPrecision(); 36](#_Toc20091)

[2.2.132 short GetCurveIndex(long Id); 36](#_Toc11130)

[2.2.133 BOOL SetCurveIndex(long Id, short nIndex); 36](#_Toc5884)

[2.2.134 BOOL SetGridMode(short GridMode); 36](#_Toc15016)

[2.2.135 short GetGridMode(); 36](#_Toc5120)

[2.2.136 void SetFootNote(LPCTSTR pFootNote); 36](#_Toc5650)

[2.2.137 CString GetFootNote(); 36](#_Toc5121)

[2.2.138 void EnableAdjustZOrder(BOOL bEnable); 36](#_Toc17011)

[2.2.139 void EnableHelpTip(BOOL bEnable); 37](#_Toc24653)

[2.2.140 BOOL SetFillDirection(long Id, short FillDirection, BOOL bUpdate); 37](#_Toc25948)

[2.2.141 short GetFillDirection(long Id); 37](#_Toc27592)

[2.2.142 void SetVisibleCoorRange(DATE MinTime, DATE MaxTime, float MinValue,float MaxValue, short Mask); 38](#_Toc17261)

[2.2.143 void GetVisibleCoorRange(DATE\* pMinTime, DATE\* pMaxTime,float\* pMinValue, float\* pMaxValue); 38](#_Toc14510)

[2.2.144 void SetBenchmark(DATE Time, float Value); 38](#_Toc28043)

[2.2.145 void GetBenchmark(DATE\* pTime, float\* pValue); 38](#_Toc13705)

[2.2.146 short GetPower(long Id); 38](#_Toc31872)

[2.2.147 BOOL ChangeId(long Id, long NewId); 38](#_Toc4792)

[2.2.148 BOOL CloneCurve(long Id, long NewId); 38](#_Toc30442)

[2.2.149 BOOL UniteCurve(long DesId, long nInsertPos, long Id, long nBegin,long nCount); 38](#_Toc23859)

[2.2.150 BOOL UniteCurve2(long DesId, long nInsertPos, long Id,DATE BTime, DATE ETime, short Mask); 38](#_Toc7020)

[2.2.151 BOOL UniteCurve3(long DesId, DATE fInsertPos, long Id, long nBegin,long nCount); 38](#_Toc20596)

[2.2.152 BOOL UniteCurve4(long DesId, DATE fInsertPos, long Id,DATE BTime, DATE ETime, short Mask); 38](#_Toc22896)

[2.2.153 BOOL OffSetCurve(long Id, DATE Time, float Value, short Operator); 39](#_Toc8368)

[2.2.154 long ArithmeticOperate(long DesId, long Id, short Operator); 39](#_Toc21764)

[2.2.155 void ClearTempBuff(); 40](#_Toc18368)

[2.2.156 BOOL PreMallocMem(long Id, long size); 40](#_Toc30357)

[2.2.157 long GetMemSize(long Id); 40](#_Toc29290)

[2.2.158 void GetMemInfo(long FAR\* pTempBuffSize, long FAR\* pAllBuffSize, 41](#_Toc19866)

[float FAR\* pUseRate, long FAR\* pId); 41](#_Toc22769)

[2.2.159 BOOL IsCurve(long Id); 41](#_Toc1381)

[2.2.160 void SetSorptionRange(short Range); 41](#_Toc25870)

[2.2.161 short GetSorptionRange(); 41](#_Toc26040)

[2.2.162 BOOL GetActualPoint(long x, long y, DATE\* pTime, DATE\* pValue); 41](#_Toc22717)

[2.2.163 long GetPointFromScreenPoint(long Id, long x, long y, short MaxRange); 42](#_Toc25811)

[2.2.164 BOOL GetPixelPoint(DATE Time, float Value, long\* px, long\* py); 42](#_Toc1168)

[2.2.165 void EnableFullScreen(BOOL bEnable); 42](#_Toc16145)

[2.2.166 DATE GetEndTime(); 42](#_Toc20825)

[2.2.167 CString GetEndTime2(); 42](#_Toc26703)

[2.2.168 float GetEndValue(); 42](#_Toc16495)

[2.2.169 void SetZLength(short ZLength); 43](#_Toc3510)

[2.2.170 short GetZLength(); 43](#_Toc24286)

[2.2.171 void SetLeftBkColor(OLE\_COLOR Color); 43](#_Toc21147)

[2.2.172 OLE\_COLOR GetLeftBkColor(); 43](#_Toc19258)

[2.2.173 void SetBottomBkColor(OLE\_COLOR Color); 43](#_Toc2445)

[2.2.174 OLE\_COLOR GetBottomBkColor(); 43](#_Toc10993)

[2.2.175 BOOL SetZOffset(long Id, short nOffset, BOOL bUpdate); 43](#_Toc29402)

[2.2.176 long GetZOffset(long Id); 43](#_Toc4947)

[2.2.177 void EnableFocusState(BOOL bEnable); 44](#_Toc19363)

[2.2.178 BOOL SetReviseToolTip(short Type); 44](#_Toc6122)

[2.2.179 short GetReviseToolTip(); 44](#_Toc14579)

[2.2.180 void LimitOnePage(BOOL bLimit); 44](#_Toc26098)

[2.2.181 BOOL FixCoor(DATE MinTime, DATE MaxTime, 44](#_Toc2454)

[float MinValue, float MaxValue, short Mask); 44](#_Toc19955)

[2.2.182 short GetFixCoor(DATE\* pMinTime, DATE\* pMaxTime, 44](#_Toc9582)

[float\* pMinValue, float\* pMaxValue); 44](#_Toc23031)

[2.2.183 BOOL RefreshLimitedOrFixedCoor(); 45](#_Toc20834)

[2.2.184 BOOL SetLimitOnePageMode(short Mode); 46](#_Toc6720)

[2.2.185 short GetLimitOnePageMode(); 46](#_Toc10168)

[2.2.186 void EnablePreview(BOOL bEnable); 46](#_Toc9924)

[2.2.187 void SetWaterMark(LPCTSTR pWaterMark); 46](#_Toc21558)

[2.2.188 long GetSysState(); 46](#_Toc10574)

[2.2.189 void SetTension(float Tension); 46](#_Toc14888)

[2.2.190 float GetTension(); 46](#_Toc7652)

[2.2.191 long LoadPlugIn(LPCTSTR pFileName, short Type, long Mask); 47](#_Toc7107)

[2.2.192 long LoadLuaScript(LPCTSTR pFileName, short Type, long Mask); 47](#_Toc8995)

[2.2.193 CString GetLuaVer(); 47](#_Toc21885)

[2.2.194 BOOL AppendLegendEx(LPCTSTR pSign, 48](#_Toc17069)

[2.2.195 OLE\_COLOR BeginNodeColor, OLE\_COLOR EndNodeColor, 48](#_Toc5461)

[OLE\_COLOR SelectedNodeColor, short NodeModeEx); 48](#_Toc6625)

[2.2.196 BOOL GetLegendEx(LPCTSTR pSign, 48](#_Toc29983)

[OLE\_COLOR\* pBeginNodeColor, OLE\_COLOR\* pEndNodeColor, 48](#_Toc19827)

[OLE\_COLOR\* pSelectedNodeColor, short\* pNodeModeEx); 48](#_Toc22595)

[2.2.197 BOOL GetLegendEx2(short nIndex, OLE\_COLOR\* pBeginNodeColor, 48](#_Toc18075)

[OLE\_COLOR\* pEndNodeColor, OLE\_COLOR\* pSelectedNodeColor, 48](#_Toc12455)

[short\* pNodeModeEx); 48](#_Toc28411)

[2.2.198 long GetSelectedNodeIndex(long Id); 48](#_Toc11146)

[2.2.199 BOOL SetSelectedNodeIndex(long Id, long NewNodeIndex); 48](#_Toc2151)

[2.2.200 void SetShortcutKeyMask(long ShortcutKey); 48](#_Toc3744)

[2.2.201 long GetShortcutKeyMask(); 48](#_Toc26674)

[2.2.202 OLE\_HANDLE GetFrceHDC() 49](#_Toc3765)

[2.2.203 BOOL SetBottomSpace(short Space); 49](#_Toc22965)

[2.2.204 short GetBottomSpace(); 49](#_Toc29839)

[2.2.205 Long GetEventMask(); 49](#_Toc18975)

[2.2.206 void SetEventMask(long Event); 49](#_Toc18944)

[2.2.207 short AddComment(DATE Time, float Value, short Position, short nBkBitmap, short Width, short Height, OLE\_COLOR TransColor, LPCTSTR pComment, OLE\_COLOR TextColor, short XOffSet, short YOffSet, BOOL bUpdate); 50](#_Toc12255)

[2.2.208 BOOL DelComment(long nIndex, BOOL bAll, BOOL bUpdate); 50](#_Toc22578)

[2.2.209 long GetCommentNum(); 50](#_Toc18876)

[2.2.210 BOOL GetComment(long nIndex, DATE\* pTime, float\* pValue, short\* pPosition, short\* pBkBitmap,Short\* pWidth, short\* pHeight, OLE\_COLOR\* pTransColor, BSTR\* pComment, OLE\_COLOR\* pTextColor, short\* pXOffSet, short\* pYOffSet); 50](#_Toc30977)

[2.2.211 short SetComment(long nIndex, DATE Time, float Value, short Position, short nBkBitmap, short Width, short Height, OLE\_COLOR TransColor, LPCTSTR pComment, OLE\_COLOR TextColor, short XOffSet, short YOffSet, short Mask, BOOL bUpdate); 50](#_Toc5638)

[2.2.212 BOOL SwapCommentIndex(long nIndex, long nOldIndex, BOOL bUpdate); 51](#_Toc5680)

[2.2.213 BOOL ShowComment(long nIndex, BOOL bShow, BOOL bUpdate); 51](#_Toc18677)

[2.2.214 BOOL IsCommentVisiable(long nIndex); 51](#_Toc11274)

[2.2.215 BOOL SetCommentPosition(short Position); 51](#_Toc4363)

[2.2.216 short GetCommentPosition(); 51](#_Toc30200)

[2.2.217 BOOL SetFixedZoomMode(short ZoomMode); 51](#_Toc3923)

[2.2.218 short GetFixedZoomMode(); 51](#_Toc7955)

[2.2.219 BOOL FixedZoom(short ZoomMode, short x, short y, 51](#_Toc32715)

[BOOL bHoldMode); 51](#_Toc870)

[2.2.220 BOOL SetAutoRefresh(short TimeInterval, short NumInterval); 52](#_Toc17982)

[long GetAutoRefresh(); 52](#_Toc6128)

[2.2.221 void EnableSelectCurve(BOOL bEnable); 52](#_Toc2713)

[2.2.222 void SetToolTipDelay(short Delay); 52](#_Toc6160)

[2.2.223 short GetToolTipDelay(); 52](#_Toc10565)

[2.2.224 BOOL AddInfiniteCurve(long Id, DATE Time, float Value, short State, BOOL bUpdate); 53](#_Toc19205)

[2.2.225 BOOL DelInfiniteCurve(long Id, BOOL bAll, BOOL bUpdate); 53](#_Toc7959)

[2.2.226 void SetMouseWheelMode(short Mode); 53](#_Toc2719)

[2.2.227 short GetMouseWheelMode(); 53](#_Toc5569)

[2.2.228 BOOL SetMouseWheelSpeed(short Speed); 54](#_Toc15451)

[2.2.229 short GetMouseWheelSpeed(); 54](#_Toc7419)

[2.2.230 void Refresh(); 54](#_Toc14464)

[2.3 事件（15个） 55](#_Toc15797)

[2.3.1 MouseDown(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y); 55](#_Toc14068)

[2.3.2 MouseMove(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y); 55](#_Toc16402)

[2.3.3 MouseUp(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y); 55](#_Toc22493)

[2.3.4 void PageChange(long wParam, long lParam); 55](#_Toc505)

[2.3.5 void BeginTimeChange(DATE NewTime); 55](#_Toc20059)

[2.3.6 void BeginValueChange(float NewValue); 55](#_Toc27525)

[2.3.7 void TimeSpanChange(double NewTimeSpan); 55](#_Toc9676)

[2.3.8 void ValueStepChange(float NewValueStep); 55](#_Toc14232)

[2.3.9 void ZoomChange(short NewZoom); 55](#_Toc7806)

[2.3.10 void SelectedCurveChange(long NewId); 55](#_Toc14939)

[2.3.11 void LegendVisableChange(long nIndex, short State); 55](#_Toc17606)

[2.3.12 void SorptionChange(long Id, long nIndex, short State); 55](#_Toc23670)

[2.3.13 void CurveStateChange(long Id, short State); 55](#_Toc15808)

[2.3.14 void ZoomModeChange(short NewMode); 56](#_Toc31039)

[2.3.15 void HZoomChange(short NewZoom); 56](#_Toc24410)

[2.3.16 BatchExportImageChange(long FileNameIndex); 56](#_Toc32566)

[2.4 导出方法（3个） 56](#_Toc5451)

[2.4.1 extern "C" BOOL \_\_stdcall ExportImage(HBITMAP hBitmap, const unsigned short\* pFileName); 56](#_Toc28035)

[2.4.2 extern "C" LPBITMAPINFO \_\_stdcall GetDIBFromDDB(HDC hDC, HBITMAP hBitmap); 56](#_Toc25378)

[3 插件 57](#_Toc24154)

[4 关于内存使用量 57](#_Toc5027)

[5 致初学者或是对编程不太精通者 57](#_Toc3430)

[6 小窍门及FAQ 59](#_Toc26615)

[7 版权 62](#_Toc684)

[8 更新记录（从2010年开始记录） 63](#_Toc28033)

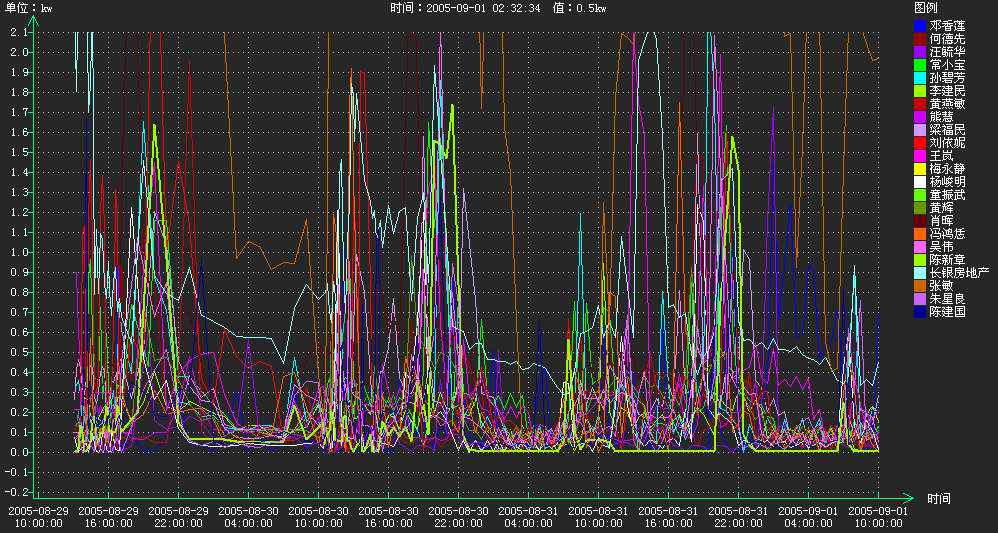
本文档描述了ST\_Curve控件的开发使用说明，包括了每一个属性、每一个函数的意义及其使用方法。

ST\_Curve是一个功能非常强大的曲线绘制控件，其主要功能有：

可同时绘制多条曲线，可绘制实时曲线（此时将智能移动曲线已保证新添加的数据可见，功能跟CListCtrl的EnsureVisible函数差不多）；

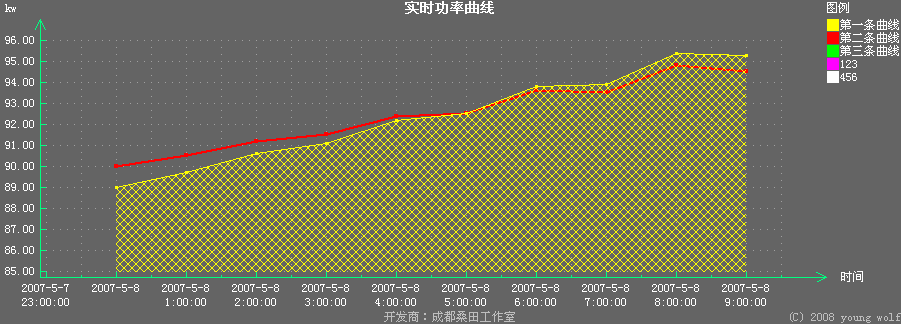
# 主要特点：

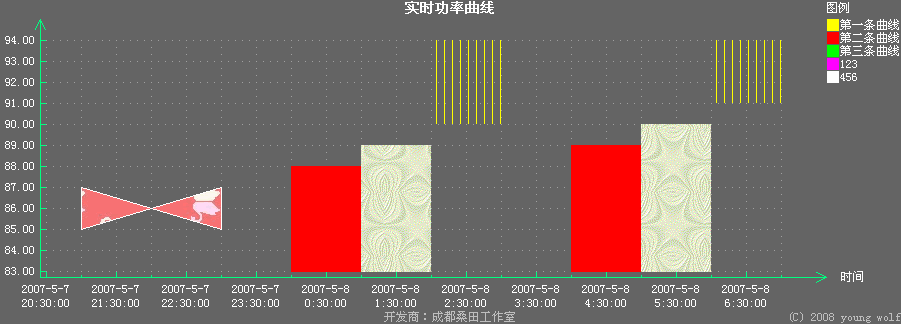
1. 强大的自定义界面，主要的可自定义属性有：背景色、坐标轴色、文字色、网格色、背景图（其中又包括拉伸、平铺、居中三种显示模式）、是否显示网格、单位、各曲线的颜色、横纵坐标值的显示精度、横纵坐标的开始值及每刻度步长值、刻度间隔数、原点位置、跟踪显示鼠标坐标、横坐标自定义显示（值或者时间）等；
2. 功能性：
   1. 任意上下左右移动曲线；
   2. 任意（这里的任意，都不是绝对的，因为计算机只考虑有限的数，是数就会有溢出，所以这里的任意不要钻牛角尖）倍数放大缩小曲线；
   3. 以任意点为中心缩放（鼠标取点），任意隐藏某些曲线；
   4. 按任意曲线居中，打印任意曲线或者全部；
   5. 导出任意曲线或者全部为图片或者图元文件，自动批量导出图片；
   6. 导出指定页面的曲线，填充曲线等；
   7. 支持三维显示，可显示全局位置预览窗口，从而快速定位；
3. 易操作性：
   1. 支持鼠标快速操作（滚轮上下移动曲线、同时按住Ctrl键则左右移动曲线、按住左键拖动鼠标则移动曲线、按住Shift键同时转动滚轮则从原点开始缩放曲线、按住alt键同时转动滚轮则从原点开始水平缩放曲线）；
   2. 支持键盘快速操作（上下左右方向键、Home/End/PageUp/PageDown键分别为上下左右移动曲线、首页、末页、上一页、下一页、F5键为垂直居中当前选中的曲线；
   3. 在图例上点左键可以使曲线被选中，并以其为基准垂直移动所有曲线（如果需要的话）；选中的曲线将变宽，且提到所有曲线的前面，以示醒目；
   4. 按数字键可按序号选择曲线（如果有的话）。在图例上点右键可以隐藏/显示曲线。按+/-再点击鼠标，则以点击处为原点缩放曲线）。
4. 效率高，移动、缩放曲线时无闪烁现象。



如上众多的曲线，滚动曲线的时候，毫无滞后感！注：图片与当前版本的ST\_Curve有小的出入，因为本控件经常在优化。图上其实是ST\_Curve早期的样子，现在效果会更好。

下面再举两个使用本控件的例子：

 曲线下面的区域可填充（填充类型参看CreateHatchBrush、CreateSolidBrush和CreatePatternBrush函数），支持三种填充模式。



当成柱状图来绘制，可向四个方向填充，可同时向多个方向填充。

总之，本控件可轻易实现下面这个很有名的收费绘图组件的样子，如果实现不了，请联系我：

<http://www.iocomp.com/shop/shopdisplayproducts.asp?id=31&cat=Plot+Pack>

**所有属性及函数大致分类如下：**

## 属性：

1）颜色相关：

* ForeColor
* BackColor
* AxisColor
* GridColor
* TitleColor
* FootNoteColor

2）页数消息：

* PageChangeMSG
* MSGRecWnd

3）寄存器：

Register1

## 函数：

### 坐标轴相关：

SetVInterval SetHInterval GetScaleInterval GetScaleNums

SetGraduationSize GetGraduationSize

### 坐标相关：

SetBeginValue GetBeginValue GetEndValue

SetBeginTime SetBeginTime2 GetBeginTime GetBeginTime2 GetEndTime GetEndTime2

SetValueStep GetValueStep SetTimeSpan GetTimeSpan

SetVPrecision GetVPrecision SetHPrecision GetHPrecision

TrimCoor EnableAutoTrimCoor

SetVisibleCoorRange GetVisibleCoorRange

### 单位相关：

SetUnit GetUnit SetHUnit GetHUnit

### 图例相关：

AddLegend AddLegendHelper GetLegend QueryLegend IsLegend

DelLegend DelLegend2 MoveCurveToLegend ChangeLegendName

SetLegendSpace GetLegendSpace

ShowLegend ShowCurve SetXYFormat GetXYFormat GetXYFormat2

SetHLegend GetHLegend

### 添加曲线点：

AddMainData AddMainData2 AddMemMainData CloneCurve

### 删除曲线点：

DelRange DelRange2

### 翻页相关：

FirstPage GotoPage

### 缩放相关：

SetZoom GetZoom EnableZoom SetHZoom GetHZoom EnableHZoom

### 模式（移动模式、显示模式、网格模式）相关：

SetMoveMode GetMoveMode SetShowMode GetShowMode SetGridMode GetGridMode

### 背景图相关：

AddImageHandle AddBitmapHandle AddBitmapHandle2 AddBitmapHandle3

SetBkBitmap GetBkBitmap SetBkMode GetBkMode SetCanvasBkMode GetCanvasBkMode

RemoveBitmapHandle RemoveBitmapHandle2 GetBitmapCount GetBitmap

GetBitmapState GetBitmapState2

SetCanvasBkBitmap GetCanvasBkBitmap

### 导出图片或导出导入文件：

ExportImage ExportImageFromPage ExportImageFromTime BatchExportImage

ImportFile ExportMetaFile

### 标题脚注：

SetCurveTitle GetCurveTitle SetFootNote GetFootNote

### 枚举（并修改）某条曲线：

GetTimeData GetTimeData2 GetValueData GetState GetPosData

InsertMainData InsertMainData2 CanContinueEnum DelPoint

### 枚举所有曲线：

GetCurveCount GetCurve

### 枚举图例：

GetLegendCount GetLegend2 GetLegendIdCount GetLegendId

### 联动相关：

SetBuddy GetBuddyCount GetBuddy

### Z-Order及选中相关：

EnableAdjustZOrder

SetCurveIndex GetCurveIndex

SelectCurve GetSelectedCurve EnableSelectCurve

### 范围相关：

GetOneTimeRange GetOneValueRange

GetOneFirstPos

GetTimeRange GetValueRange

GetViableTimeRange

### 移动曲线：

DragCurve VCenterCurve GotoCurve

### 辅助函数：

IsSelected IsLegendVisible IsCurveVisible IsCurveInCanvas IsCurve IsCurveClosed

GetPower EnableHelpTip SetSorptionRange GetSorptionRange

GetActualPoint GetPointFromScreenPoint GetPixelPoint

### 曲线长度控制相关：

GetCurveLength SetMaxLength GetMaxLength GetCutLength

### 页数量变化消息：

ReportPageInfo

### 三维相关函数：

SetZLength GetZLength SetZOffset GetZOffset

SetLeftBkColor GetLeftBkColor SetBottomBkColor GetBottomBkColor

### 基点相关：

SetBenchmark GetBenchmark

### 填充方向相关：

SetFillDirection GetFillDirection

### 缓存控制相关：

ClearTempBuff PreMallocMem GetMemSize GetMemInfo

### 曲线操作函数：

TrimCurve TrimCurve2 OffSetCurve

ChangeId ArithmeticOperate

UniteCurve UniteCurve2 UniteCurve3 UniteCurve4

### 限制坐标相关：

LimitOnePage FixCoor GetFixCoor RefreshLimitedOrFixedCoor

SetLimitOnePageMode GetLimitOnePageMode

### 平滑曲线相关：

SetTension GetTension

### 插件相关：

LoadPlugIn LoadLuaScript GetLuaVer

### 起始点、结束点、选中点标识相关：

AppendLegendEx GetLegendEx GetLegendEx2

GetSelectedNodeIndex SetSelectedNodeIndex

### 快捷键相关：

SetShortcutKeyMask GetShortcutKeyMask

### 横坐标位置相关：

SetBottomSpace SetBottomSpace

### 注解相关：

AddComment DelComment GetCommentNum GetComment SetComment

SetCommentPosition GetCommentPosition

### 事件开关相关：

SetEventMask GetEventMask

### 定点缩放相关：

SetFixedZoomMode GetFixedZoomMode FixedZoom

### 自动刷新相关：

SetAutoRefresh GetAutoRefresh

### ToolTip相关：

SetReviseToolTip GetReviseToolTip SetToolTipDelay GetToolTipDelay

### 无限曲线相关：

AddInfiniteCurve DelInfiniteCurve

### 鼠标滚轮相关：

SetMouseWheelMode GetMouseWheelMode SetMouseWheelSpeed GetMouseWheelSpeed

### 其它函数：

SetFont GetFont PrintCurve Refresh EnableFullScreen

EnableFocusState EnablePreview SetWaterMark GetSysState GetFrceHDC

# 属性及函数：

## 属性（9个）

OLE\_COLOR ForeColor：文字色。

OLE\_COLOR BackColor：背景色。

OLE\_COLOR AxisColor：坐标轴色。

OLE\_COLOR GridColor：网格颜色。

OLE\_COLOR TitleColor：标题色，默认等于默认的ForeColor。

OLE\_COLOR FootNoteColor：脚注色，默认等于默认的ForeColor的3/4。

以下属性在设计时无效，设置也不会起作用，只能在运行时设置：

long PageChangeMSG：

当曲线页数发生变化时发送此消息给指定窗口，不为0即认为有效，页数量发生变化的情况很多，比如移动曲线，缩放曲线，隐藏、显示曲线等，消息格式如下：

wParam为当前页面前面的页数量；

lParam为当前页面后面的页数量，总页数为（ULONG)wParam + 1 + (ULONG)lParam，如果wParam和lParam均为-1，则说明一页也没有。注意，wParam和lParam在64位下，本来是64位的，但为了兼容页数量变化事件（PageChange），只使用低32位。

OLE\_HANDLE MSGRecWnd：接收PageChangeMSG消息的窗口句柄，不为0即认为有效。

OLE\_HANDLE Register1：寄存器1（目前只有一个），主要用于与64位版本控件的64位数据类型的交互，比如句柄。具体使用方法如下：当需要向控件传递一个64位数据时，把这个64位数据的高32位写入寄存器1，用低32位数据调用相应的接口；当控件传出一个64位数据时，先把高32位写入寄存器1，再返回低32位数据。

如果你的工程同时有32位和64位配置，则你肯定会需要下面我定义的宏，非常有用：

#ifdef \_WIN64

#define Format64bitHandle(C, HANDLETYPE, LOW32BIT) ((HANDLETYPE) (((ULONGLONG) C.GetRegister1() << 32) + (ULONG) LOW32BIT))

#define SplitHandle(C, H) (C.SetRegister1(GetH32bit(H)), (OLE\_HANDLE) H)

#define GetH32bit(H) ((OLE\_HANDLE) ((ULONGLONG) H >> 32))

#else

#define Format64bitHandle(C, HANDLETYPE, LOW32BIT) ((HANDLETYPE) LOW32BIT)

#define SplitHandle(C, H) ((OLE\_HANDLE) H)

#define GetH32bit(H) 0

#endif

Format64bitHandle用于从寄存器和接口返回值，组合成一个64位数据，C是控件包装类的一个实例，HANDLETYPE是64位数据的类型（64位系统下，指针，HANDLE等都是64位的），LOW32BIT是接口返回的32位数据，比如GetFont接口，使用如下：

HFONT h = Format64bitHandle(m\_ST\_Curve, HFONT, m\_ST\_Curve.GetFont());

这行语句在64位和32位工程中，都将工作正常。如果你只有32位工程，则直接：

HFONT h = (HFONT) m\_ST\_Curve.GetFont();

也是可以的，但推荐用我的宏，因为不排除将来你可能会在工程中添加一个64位配置。

SplitHandle用于向控件传送一个64位数据，比如SetFont接口，使用如下：

HFONT h = ...;//要传送的64位数据

m\_ST\_Curve.SetFont(SplitHandle(m\_ST\_Curve, h));

GetH32bit是一个辅助类，用于获取一个64位数据的高32位。

总之，为了以后的扩展，推荐使用我写的宏，哪怕你现在只有32位或者64位版本，做到两个版本都兼容是最好的，未雨绸缪嘛。

软件包根目录下的64目录下的demo工程，有使用寄存器1的例子。

需要使用寄存器1的接口及属性有（只是64位版本需要）：

MSGRecWnd SetFont GetFont AddBitmapHandle AddBitmapHandle2 AddBitmapHandle3

AddMemMainData RemoveBitmapHandle GetBitmap GetBitmapState2

SetBuddy GetBuddy GetFrceHDC

特别强调，不管是在32位还是64位版本下，ReportPageInfo接口都需要使用寄存器1，具体请参看ReportPageInfo文档，这个接口的行为在2.2.0.1版本发生了改变，这里向大家道歉。

**特别注意：**

对于OLE\_COLOR类型属性，如果想让控件自己弹出颜色选择框让用户选择，则设置颜色的最高位（从低位起第32位）为1，后面的3个字节保持当前的颜色，这样控件就会弹出颜色选择框，如果用户取消颜色选择，控件会恢复以前的颜色（后3字节），以ForeColor为例，调用方法为：

m\_ST\_Curve.SetForeColor(0x80000000 | m\_ST\_Curve.GetForeColor());

至于为什么要这么别扭，因为属性不同于方法，属性是先接受，再发送通知消息，也就是说，当我（控件）知道属性发生改变时，老的属性已经被覆盖了，此时如果弹出颜色框让用户选择，当用户取消选择时，将再也无法恢复原来的颜色了。所以我采用了上面的变通的方法，让二次开发者将老的颜色属性送给我。至于方法，就没有这个问题了，因为我可以在方法内部控制是否覆盖老的设置。

注：除了颜色，还有字体设置、打印、导出图片等也会有两种方法，一种是弹出标准对话框让用户选择，一种是不弹，不弹对话框这种方式有些时候非常有用，考虑无人值守的服务器。

## 函数（232个）

### BOOL SetVInterval(short VInterval);

### BOOL SetHInterval(short HInterval);

### short GetScaleInterval();

垂直、水平轴上，每个刻度值之间的刻度数，就像直尺一样，它最小刻度是1毫米，而刻度值（有显示的）最小为1厘米，此时刻度间隔为9（除去两次标有刻度值的那两个刻度）。这些方法主要用于让界面更简洁，同时还要照顾视觉上的要求，取值范围0到100。

对于GetScaleInterval函数，返回高8位为横坐标刻度间隔，低8位为纵坐标刻度间隔。

### SetGraduationSize(long size);

### long GetGraduationSize();

设置纵横坐标一个刻度在屏幕上的宽度（像素），调用一下看效果就知道意思了，这是了为某些特殊要求而增加的接口，比如你横坐标显示一月的每一天，有时候是31天，有时候是28天，你想刚好充满屏幕，此时就需要用到这两个接口了，同时它也有缩放功能；所以到目前为止，缩放曲线有如下几种：一是调用SetZoom函数，它会在纵横坐标上做等比例缩放；二是调用SetHZoom函数，它只会在横坐标上做缩放（同时还会与SetZoom共同作用）；三是调用SetTimeSpan和SetValueStep，这是最灵活的，想怎么缩放都可以；四是调用本接口，在TimeSpan和ValueStep一定的情况下，增加刻度宽度就等于是放大，相当于减小了TimeSpan和ValueStep。

### BOOL SetLegendSpace(short LegendSpace);

### short GetLegendSpace();

设置图例宽度，单位为像素，如果LegendSpace小于等于0，则计算将所有图例显示完全的最小宽度（假设为w），再假设当前宽度为c，然后分两种情况：LegendSpace小于0时，只有在w>c时应用w宽度；LegendSpace等于0时，直接应用w宽度。如果LegendSpace大于0，则不计算最小宽度，直接应用LegendSpace宽度。

### BOOL SetBeginValue(float fBeginValue);

### float GetBeginValue();

设置纵坐标开始值，注意，这个值不是任意值均可，所以本函数返回布尔型代表成功与否，设置这个值，要保证不会让曲线完全移出画布。这里说明一点，本控件中的曲线虽然是可任意移动、缩放的，但有一个前提是，移动缩放后画布上不能完全没有曲线。因为将所有曲线移出画布其实是没有什么实际意义的，都移出画布后，也就成了白板一块。

### BOOL SetBeginTime(LPCTSTR pBeginTime);

### BOOL SetBeginTime2(DATE fBeginTime);

### CString GetBeginTime();

### DATE GetBeginTime2();

设置横坐标的开始时间，pBeginTime为用字符串表达的时间，格式为当前操作系统默认的格式，fBeginTime为用浮点数表达的时间。如何用浮点数表达时间请参看MFC的COleDateTime类，以下不再对这个问题做再次说明。其实COleDateTime就是用DATE型数据来保存时间的，它与DATE可隐式转换，比如：

COleDateTime OldTime = COleDateTime::GetCurrentTime();

M\_Curve.SetBeginTime2(OldTime);

注意，本控件可以在横坐标上显示值，即和纵坐标显示一样的效果，请参看SetShowMode函数，当横坐标显示为值的时候，pBeginTime会转换为COleDateTime，再转换为DATE，最后直接显示DATE型数据。

### BOOL SetTimeSpan(double TimeStep);

### double GetTimeSpan();

设置横坐标刻度的步长，不能小于0.1秒，不能大于198年（大约198年）。

如何用浮点数表达时间间隔请参看MFC的COleDateTimeSpan类，以下不再对这个问题做再次说明。其实COleDateTimeSpan就是用double型数据来保存时间的，它与double可隐式转换，比如：

COleDateTimeSpan TimeSpan;

TimeSpan.SetDateTimeSpan(0, 0, 30, 0); //半小时

//SetDateTimeSpan最小只能设置为1秒，如果要指定小于1秒呢？答案是直接赋浮 //点数，1.0为一天，由此算出，给COleDateTimeSpan赋0.1秒的代码为：

//COleDateTimeSpan TimeSpan = 1.0 / 24 / 60 / 60 / 10;

//我都是使用这种方法，不管是否大于1秒，这样方便。

M\_Curve.SetTimeSpan(TimeSpan);

注意，如果横坐标显示为值，则把TimeSpan当成单纯的double型来看待。

### BOOL SetValueStep(float ValueStep);

### float GetValueStep();

设置纵坐标刻度步长，不能小于.000001f，不能大于1.0e30f。

### BOOL SetVPrecision(short Precision);

### short GetVPrecision();

设置纵坐标显示精度，非负，精度指的是小数点后面的位数，需要小于等于6。

### BOOL SetUnit(LPCTSTR pUnit);

### CString GetUnit();

设纵坐标单位，成功返回真，否则返回假，不能超过16个字符。

### BOOL SetHUnit(LPCTSTR pHUnit);

### CString GetHUnit();

设置横坐标单位（或者描述），当横坐标显示为值的时候有用，如果显示为时间，则该值无效，恒显示为“时间”。

### short AddLegend(long Id, LPCTSTR pSign, OLE\_COLOR PenColor, short PenStyle,short LineWidth, OLE\_COLOR BrushColor,

### short BrushStyle, short CurveMode, short NodeMode, short Mask,

### BOOL bUpdate);

添加图例，如果bUpdate为真，则马上刷新让操作生效，以后类似的参数不再作说明。但有一点需要说明，有些时候，就算bUpdate为假，也可能出现刷新的情况，因为本控件遵循一个原则：要么没有曲线可显示，要么画布中总要绘制一点点曲线，换句话说，无法将曲线全部移出画布。至于为什么要这样，因为曲线完全移动画布后，使用者可能再也不能将曲线移动回来了（因为他不知道往哪个方向移动才能将曲线移动到画布上，如果方向移动反了，反而会越走越远，当然，本控件提供了F5来让画线回到画布上，可是不是所有用户都能记得住的）。现在回到刚才的问题，为什么bUpdate为假时也有可能刷新曲线呢，考虑按时间截除曲线某些点，如果截除后，所有曲线落在了画布的外面，那么本控件将尝试自动移动曲线到画布，这个操作可能会调用到某一函数（比如SetBeginTime），而这个函数没有bUpdate参数（由于历史原因吧，因为对接口的更改对COM来说是致命的，为了保持接口不变，这个历史问题就遗留了下来），那么就只能马上刷新曲线了，这个问题我之所有没有想办法去解决，是因为这种情况不多见，而且，对曲线做了修改，迟早都是要刷新的，在这种情况下，仅仅是多刷新了一次而已。下面回到正题：

Mask代表Id、PenColor、PenStyle、LineWidth、BrushColor、BrushStyle、

CurveMode和NodeMode的有效性，按位算，按前面罗列的顺序。

当要添加的图例（pSign）已存在时：

根据Mask来决定哪些值用来更新图例（Id比较特殊，它是添加，而不是更新）；如果Id有效，并且已经处于其它图例，则先从相应的图例中删除再添加到本图例（当然，在添加到本图例时，会有重复性判断的）。

当要添加的图例（pSign）不存在时：

以pSign为图例名称新建图例，并将Id添加到其中，此时所有参数都必须有效，也就是说，Mask必须等于0xFF。如果Id已经存在于其它图例，则处理方式同上。

PenStyle为画笔类型，参考CreatePen函数（取值0到255，其实没有这么多的样式，留着以后扩展，所以控件没有判断参数的值是否在CreatePen函数可识别的范围之内，不在范围之内是不会出错的）。

LineWidth为画笔宽度，从0到255，可以等于0，具体参看CreatePen函数。如果画笔的宽度大于1，则画笔类型只能是PS\_SOLID，设置其它类型将不起作用，GDI就是这样的。

BrushColor填充曲线的画刷颜色，有点像柱状图。

BrushStyle取值如下，只取低字节：

255－不填充；

127－solid brush样式，参看CreateSolidBrush（win API），颜色为BrushColor；

0-126－hatch brush样式（没有这么多的样式，留着以后扩展，所以控件没有判断参数的值是否在CreateHatchBrush函数可识别的范围之内，不在范围之内不会出错，只是无效果），具体支持的样式请参看CreateHatchBrush（win API），颜色为BrushColor；

128-254－pattern brush样式，参看CreatePatternBrush（win API），(BrushStyle - 128)即为创建刷子需要的位图的序号（位图由AddBitmap等函数添加，具体看相关文档）。

CurveMode取值如下：

0－两点之间用直线相连（默认）；1－先垂直后水平的方波；2－先水平后垂直的方波；3－平滑曲线，平滑张力参看S(G)etTension函数。

注：在绘制平滑曲线时，采用了GDI+，所以hatch brush的种类也支持的更多，可以达到52种（GDI只支持6种），具体参看GdiplusEnums.H里面的HatchStyle枚举。

如果不是平滑曲线，本控件将只使用GDI的刷子样式（为了速度，这样可以避免使用GDI+）。

另外，GDI+不支持透明的hatch brush，所以，如果为控件添加了背景的话，用GDI+的hatch brush填充，效果不如GDI的hatch brush填充。目前在使用GDI+的hatch brush填充时，背景色是控件的背景色，而在使用背景位图的情况下，背景色是不显示的，所以GDI+的hatch brush将把背景位图抹掉（变成了背景色）！

GDI+的平滑曲线绘制速度比GDI的折线曲线绘制速度要慢上百倍，不过不要被这个数字吓倒，如果GDI需要的时间为0，那么GDI+就算是100倍，仍然还是0，是不是？开个玩笑！本控件在速度方面的问题大家可以放心！

NodeMode取值如下：

0：不显示节点；1按曲线颜色显示节点；2按曲线颜色的反色显示节点。

注：当节点处于显示状态时，将比曲线要粗一点，以便观看。如果画笔为空，则本参数将无意义，节点都不会显示。

返回值也是一个Mask，里面哪个位置上为1，则说明相应位置上的属性设置失败。

### short AddLegendHelper(long Id, LPCTSTR pSign,OLE\_COLOR PenColor, short PenStyle, short LineWidth, boolean bUpdate);

添加图例辅助函数，大家一定为AddLegend函数众多的参数而头痛，大多数时候，并不需要填充曲线（除非画柱状图），却非要传BrushColor和BrushStyle等参数，而在Mask里面还要标识为无效，的确非常的麻烦，我作为作者，自己使用起来也很麻烦，于是我添加了这个辅助函数，主要用于减少参数的个数（默认一些参数）。当调用这个函数的时候：

如果图例不存在，则按BrushColor等于0（等于多少并不重要，因为后面BrushStyle等于255，代表不填充）、BrushStyle等于255、 CurveMode等于0、NodeMode等于1去调用AddLegend函数，当然Mask等于0xFF，因为图例不存在，是第一次添加。

如果图例已经存在，则按Mask等于0xF去调用AddLegend函数，也就是说，把Id、PenColor、PenStyle和LineWidth当成有效，把BrushColor、BrushStyle、CurveMode和NodeMode当成无效去调用AddLegend函数，换句话说，AddLegendHelper既无法添加也无法更改BrushColor、BrushStyle、CurveMode和NodeMode参数。

### BOOL IsLegend(LPCTSTR pSign);

判断图例是否存在，非常有讽刺意味的事又发生了（IsCurve函数时发生过一次了，详情请参看IsCurve接口说明）。

### BOOL GetLegend(LPCTSTR pSign, OLE\_COLOR\* pPenColor, short\* pPenStyle, short\* pLineWidth, OLE\_COLOR\* pBrushColor,

### short\* pBrushStyle, short\* pCurveMode, short\* pNodeMode);

获取指定图例的基本属性，如果不需要某些属性，则传入空指针即可。

### OOL DelLegend(long Id, BOOL bAll, BOOL bUpdate);

### BOOL DelLegend2(LPCTSTR pSign, BOOL bUpdate);

删除图例，如果bAll为真，则删除所有图例，Id被忽略，第二个函数不能删除所有图例，但它可以完全的删除一个图例，第一个函数则只有在图例中的所有Id均被删除后才会完全删除相应的图例（意思是，当某个图例不包括任何一条曲线时，一个图例可以包括任意多的曲线）。

注意：容器在退出的时候，不要以bAll为真来调用DelLegend函数（目的是想删除所有图例），明智的做法是根本不调用它。因为删除图例时，控件内部会有大量的状态需要去更新，控件在退出的时候，会用一种高效的方法去删除图例（不更新状态）。当然，如果容器决定在运行过程中，完全删除图例，以便重新添加图例，此时是应该调用本函数的。

### BOOL SetXYFormat(LPCTSTR pSign, short Format);

### short GetXYFormat(LPCTSTR pSign);

### short GetXYFormat2(short nIndex);

在坐标点显示XY坐标，Format按位算，从低位起：1-是否显示X值，2-是否显示Y值，3-是否隐藏单位，4-是否显示单行。对于GetXYFormat，如果返回-1，则说明图例未找到。

坐标显示的颜色由图例的NodeMode决定。

nIndex用于按序号顺序获取XY坐标的显示格式。

注意，这三个接口所提供的属性，其实是图例的属性，这就是为什么需要一个pSign参数的原因。但并没有添加到AddLegend(Helper)和GetLegend接口，原因有二：一是这两个接口的参数太多了；二是不想改动老接口，这样保证老的工程在替换新控件后，不需要重新编译仍然能运行。

### CString QueryLegend(long Id);

返回Id曲线所在的图例的名字。

### BOOL MoveCurveToLegend(long Id, LPCTSTR pSign);

把Id曲线移动到pSign图例里面，如果已经在pSign里面，则直接返回，如果已经在其它图例里面，则先删除再添加到pSign图例，如果pSign图例不存在，则失败。其实AddLegend函数也可实现此功能，比如：

m\_ST\_Curve.AddLegend(11, \_T("第三条曲线"), 0, 0, 0, 0, 0, 0, 0, 1, TRUE);

以上的代码会把11曲线添加到"第三条曲线"里面，大家看到了，参数太多太复杂，所以提供了本接口，以方便调用。

把一条曲线从一个图例移动到另外一个图例，其实际意义并不大，这样做的目的，我想无非是想快速切换一条曲线的绘制方式，但这个功能有更好的处理接口，那就是ChangeId接口，比如你有两个图例，其绘制方式不同，你给第一个图例添加曲线1，给第二个图例添加曲线2，当曲线1想要快速切换绘制方式的时候，直接用ChangeId接口把曲线1改成曲线2，就自然切换了绘制方式了。

### BOOL ChangeLegendName(LPCTSTR pFrom, LPCTSTR pTo);

修改图例的名字，前者必须存在，后者必须不存在，否则失败。修改图例名字后，可能会由于比原来的长或者比原来的短，而出现图例绘制出了屏，或者没填满，二次开发者可以通过SetLegendSpace去修正，控件不会自动调用（当然，你确定不会改变显示宽度就别调用了，这我想大家都知道）。

### BOOL SetHLegend(LPCTSTR pHLegend);

### CString GetHLegend();

设置获取水平图例，水平图例只能显示文字，由于显示一些说明性的信息，支持多行显示，显示位置在横坐标Label的下面。注意，水平图例的显示需要占用画布底部空间（具体请参看SetBottomSpace接口），如果底部空间不足（底部空间包括横坐标Label所占用的空间，显示完横坐标Label之后，剩下的空间才能用于显示水平图例），可能会造成水平图例显示不完全。

### short AddMainData(long Id, LPCTSTR pTime, float Value, short State,short VisibleState, BOOL bAddTrail);

### short AddMainData2(long Id, DATE Time, float Value, short State,

### short VisibleState, BOOL bAddTrail);

### long AddMemMainData(OLE\_HANDLE pMemMainData, long MemSize, BOOL bAddTrail);

添加一个数据到曲线中，Time为时间，即横坐标。特别注意这里的Id是一个唯一性标识，ST\_Curve用来区别多条不同的曲线，相当于曲线的ID。Id与AddLegend中的Id相对应。bAddTrail用来确定是否将新值添加到曲线的尾部，如果为真，则直接添加到尾部，这样速度很快，否则将寻找新值的插入点，如果您输入的数据是按时间预先排好序的（比如查询数据库时使用order by排序），则强烈建议将该值置为真，这样速度快(如果时间不能保证已按升序排好了序，而又要想绘制一条一次曲线的话，一定要将bAddTrail置为假，否则显示不正常)。

VisibleState从低位起：

1－是否马上绘制添加的点（绘制实时曲线）

2－保持纵坐标不变（在第1位为1的情况下有效）

3－保持横坐标不变（在第1位为1的情况下有效）

4－在纵坐标上做最少的移动（在第1位为1的情况下有效）

5－在横坐标上做最少的移动（在第1位为1的情况下有效）

6－当上一个点在画布中不可见时，不自动移动曲线（在第1位为1的情况下有效）

四五两位的意义在于让控件在需要移动的时候尽量少移动，达到类似Windows任务管理器性能栏的CPU使用记录的表现效果。当第六位为1时，如果上一个点不可见，绘制实时曲线失效。

若正在绘制实时曲线（VisibleState低位为1），则不允许添加隐藏点（State等于2），如果强行绘制隐藏点，控件将自动更改为普通点（State等于0）。

State为该点状态，低字节取值如下：

0－普通状态，意义仅仅是非其它状态；

1－断点，该点与前一点之间不用线相连；

2－隐藏点，该点前一点与后一点直接相连，跳过这一点。注意，曲线的首尾点设置这个状态无效。

高字节按位算，取值如下：

1－不显示节点，哪怕图例指示需要显示。

下面要说的是新添加的函数AddMemMainData，应网友的要求，增加了这个接口用于一次添加多个点，而且这些点还可以分别属于不同的曲线，pMemMainData应该是一个地址，这个地址里面保存的就是要一次性添加的点，地址长度为Size，按字节算。为了效率，控件没有判断这Size个字节的可读性，请调用者自己保证，否则程序会崩溃。内存里面的数据组成为：

Id（4）+ 时间（8）+ 值（4）+ State（2）

这里的State与AddMainData函数的State是相同的。

大家不难看出，批量添加点的步骤其实很简单，首先确定要添加多少个点，然后点数乘以18得到要分配的内存，然后将每个点写入这片内存中，最后调用AddMemMainData函数即可。初学者请看我的demo，里面就是用这种方法添加曲线的。

现在说说pTime这个参数，当横坐标显示为时间时，它就是时间的字符串表达方式，比如：2011-3-17 1:0:0；当横坐标显示为值的时候，它就是一个double数据类型的字符串表达方式，比如：1.0911。在2.1.0.1及其之前的版本中，这个pTime无论在什么情况下，都是时间的字符串表达方式，在2.1.0.2中，做了上述修改（同样，InsertMainData也做了同样的修改）。

另外，对于AddMainData和AddMainData2函数，其实它只会返回0，1和2三种结果，返回0代表失败；返回1代表成功，但没有新添加曲线，也就是说曲线已经存在，本次调用只是给它增加了一个点；返回2代表成功，并且添加了新的曲线，本次调用是添加曲线的第一个点。这样做有什么用呢？这要从控件的接口设计谈起，我下面要说的，请大家不要觉得我像唐僧，我只是想让大家完全理解为什么要这样，而不是记住是这样，这是我的原则：

大家注意到，控件并没有明确的接口用于添加一条曲线，删除一条曲线（AddMainData和AddMainData2只是添加点，DelRange和DelRange2也只是删除点），本着方便开发者的原则，添加曲线和删除曲线都由控件自己做了，在添加点的时候，如果曲线不存在，就会添加一条曲线，删除点的时候，如果某条曲线的所有点都删除完了，则删除整条曲线，这些对二次开发者都是透明的，凡事有利就有弊，太智能就会缺少灵活性，具体到我上面说的问题，因为曲线有它自身的属性，最典型的属性就是填充方向和幂次（幂次因为是只读的，所以这里不讨论），现在我以填充方向为例：

比如对某条曲线执行SetFillDirection（设置一个非默认的填充方向，默认填充方向向下）后，由于种种原因，后来又调用DelRange2删除了一些点，如果DelRange2的执行结果是所有点都被删除了，则相应的曲线将被删除，如果后面又要添加点，则曲线将被重新添加，此时填充方向恢复成默认的向下填充，原来SetFillDirection产生的结果就丢失了，现在二次开发者可以对AddMainData和AddMainData2函数的返回值进行验证，如果是2，则重复调用SetFillDirection函数即可。出现这样的问题的根本原因是二次开发者不知道什么时候曲线被删除（控件没有提供这样的接口），什么时候被创建，现在对AddMainData和AddMainData2函数的返回值做了如上的修改，基本上解决了这个问题，虽然可能不是最好的解决方案。

### void DelRange(long Id, DATE BTime, DATE ETime,short Mask,

### BOOL bAll, BOOL bUpdate);

### void DelRange2(long Id, long nIndex, long nCount, BOOL bAll, BOOL bUpdate);

DelRange删除指定时间段内的曲线，Mask代表BTime和ETime的有效性，从低位起，第1位表示BTime的有效性，第二位表示ETime的有效性，这样做的目的是很显然的，就是如果要从曲线头开始删除，或者要删除到曲线尾的时候，不需要去获取曲线头或者尾的具体时间，可以随便给BTime或者ETime赋个值，然后在Mask的相应位置置0，表示相应的时间无效，此时对于BTime将取所有曲线的最小时间，对于ETime将取所有曲线的最大时间。

ExportImageFromTime和PrintCurve函数也有一个Mask参数，它们的意义与这里的Mask完全一样，以后不再注释。如果bAll为真，则操作所有曲线，此时Id无效。

DelRange2删除曲线从nInde开始的nCount个点，nCount必需大于0，或者等于-1，此时将从nIndex开始删除到曲线尾。活用这两个函数（必要时加上GetCurveLength函数，例如删除某条曲线的右边10个点，则程序可这样写：DelRange2(100, GetCurveLength(100) - 10, -1, FALSE, TRUE)），可以达到按时间或点数量对某条曲线或所有曲线进行截左、截中、截右操作，使用时多发挥想像。

注意，在调用DelRange2函数的时候有个技巧，比如要删除第2、4、6点（从0开始），那么应该是从后面开始：

M\_Curve.DelRange2(Id, 6, 1, FALSE, FALSE);

M\_Curve.DelRange2(Id, 4, 1, FALSE, FALSE);

M\_Curve.DelRange2(Id, 2, 1, FALSE, TRUE);

如果非要从前面开始删除，则要对Index进行处理：

M\_Curve.DelRange2(Id, 2, 1, FALSE, FALSE);

M\_Curve.DelRange2(Id, 4 - 1, 1, FALSE, FALSE);

M\_Curve.DelRange2(Id, 6 - 2, 1, FALSE, FALSE);

这个“问题”在所有存在下标或者迭代器概念的地方都有，比如MFC的CListCtrl，STL的vector等，并不是控件的BUG。

注意，显然是DelRange2函数的效率高，所以如果需要删除整条曲线，一定要使用DelRange2函数。

另外容器在退出的时候，不要以bAll为真来调用这两个函数（目的是想删除所有曲线），明智的做法是根本不调用它们。因为删除曲线时，控件内部会有大量的状态需要去更新，控件在退出的时候，会用一种高效的方法去删除曲线（不更新状态）。当然，如果容器决定在运行过程中，完全删除曲线，以便重新添加曲线，此时是应该调用本函数的。

### BOOL FirstPage(BOOL bLast, BOOL bUpdate);

### short GotoPage(short RelativePage, BOOL bUpdate);

翻页，如果bLast为真，则跳到末页，否则跳到首页。当RelativePage为正时向后翻，为负时向前翻。GotoPage返回值的绝对值代表具体翻了多少页，注意不一定等于RelativePage，因为如果翻页后，当前页没有任何可显示的曲线，则会继续翻页，直到屏幕上有显示为止。

### BOOL SetZoom(short Zoom);

### short GetZoom();

设置缩放率，正为放大，负为缩小，0代表不缩放。注意1个单位仅代表1/4，也就是当Zoom等于4时，才放大一倍，等于-4时才缩小1倍。

### BOOL SetHZoom(short Zoom);

### short GetHZoom();

设置水平缩放率，上面的SetZoom我称之为普通缩放率，普通缩放率对纵横坐标均有效；而对于水平缩放率，则会只应用于横坐标，至于为什么提供此功能，可以联想一下K线图。那么，这个功能和二次开发者调用SetTimeSpan有何区别呢？答案是一样的，但调用本接口会更方便，而且二次开发者不用记住老的横坐标间隔，想恢复到非缩放状态时，只需要用0调用SetHZoom即可（跟普通缩放率一样）。

提供水平缩放的另一个原因是，我在2.1.0.0版本里面新增加了一个快捷操作——按住alt键滚动鼠标，此时会执行水平缩放（联想一下按住shift键滚动鼠标），既然提供了水平缩放这个功能，那GetHZoom接口就必须得提供了，最后我一不做，二不休，把SetHZoom接口也提供了（反正按住alt键滚动鼠标这个操作也需要封装一下）。

水平缩放的参数与普通缩放完全一样，区别仅仅是前者只应用于横坐标。

如果普通缩放率和水平缩放率都不为0，那么真正的缩放率是多少呢？对于纵坐标，当然是等于普通缩放率（因为水平缩放率不作用于纵坐标），对于横坐标，则为两种缩放率相加，比如普通缩放率为1（放大1/4），水平缩放率为-1（缩小1/4），则最后真正的缩放率为1 - 1 = 0，即不缩放。

### BOOL SetMaxLength(long MaxLength, long CutLength);

### long GetMaxLength();

### long GetCutLength();

### long GetCurveLength(long Id);

设置每条曲线的最大点数，MaxLength取值为-1表示不限制最大点（此时CutLength无效），否则取值大于0（此时CutLength必需小于MaxLength且大于0），CutLength为当曲线中的点超过MaxLength时，截掉曲线前面点的个数。

GetCurveLength函数则用来获取指定曲线中的点数量，包括隐藏点和断点，如果找不到曲线，则返回-1。

### BOOL SetShowMode(short ShowMode);

### short GetShowMode();

设置显示模式，也就是坐标系，取值如下（低7位）：

0－Y轴向上，X轴向右，原点在左下角（默认）；

1－Y轴向上，X轴向左，原点在右下角；

2－Y轴向下，X轴向右，原点在左上角；

3－Y轴向下，X轴向左，原点在右上角。

上面使用了低两位，第三位如果为1，则不显示年月日，第四位如果为1，则不显示时分秒，如果第8位为1，则横坐标将按值的方式显示（此时第3、4位只要设置任意一位，即表示不显示横坐标）。

注意，在导出图元文件时，横坐标要么导出为浮点数格式，要么导出为完整的日期时间格式，不受这里的第三、四位的影响，否则将无法导入。

### BOOL SetMoveMode(short MoveMode);

### short GetMoveMode();

设置获取曲线的移动模式，从低位起：

1－是否允许在水平上移动曲线

2－是否允许在垂直上移动曲线

3－是否为快速移动模式，即在按住鼠标移动过程中移动曲线，否则称为慢速移动模式，即在鼠标左键弹起时移动曲线

第8位如果为1，则在鼠标移动（非拖动曲线）时，显示为手型，否则显示为十字架（默认）；此时坐标提示将失效，吸附效应将失效，按住ctrl或者shift键控制鼠标只能在水平或者垂直上移动功能也将失效。

如果通过本接口去禁止曲线的移动行为，只会影响到通过鼠标滚动和鼠标拖动这两种操作，其它的操作（比如方向键）仍然可能移动曲线，需要二次开发者做其它方面的限制（如果你想完全禁止曲线被移动的话）。

当设置为禁止曲线被移动时，这里指的是禁止操作员对曲线的移动，二次开发仍然可以通过接口移动曲线（比如DragCurve等），以后类似的不再说明。

### void SetFont(OLE\_HANDLE hFont);

### OLE\_HANDLE GetFont();

设置字体，hFont为字体句柄（HFONT），不为0即为有效，如果为0，则本控件弹出字体选择框让用户选择，建议采用这种方式设置字体，当然也可以传入一个事先生成好了的字体，后面这种用法在一些特殊情况下有用，比如后台服务器，此时一般没有操作员在旁边，而且每次让用户选择字体也麻烦。GetFont返回字体句柄。

### BOOL AddImageHandle(LPCTSTR pFileName, BOOL bShared);

### void AddBitmapHandle(OLE\_HANDLE hBitmap, BOOL bShared);

### BOOL AddBitmapHandle2(OLE\_HANDLE hInstance, LPCTSTR pszResourceName,BOOL bShared);

### BOOL AddBitmapHandle3(OLE\_HANDLE hInstance, long nIDResource, BOOL bShared);

添加位图到控件，控件内部有能力保存一系列的位图句柄，以便以后使用（目前可以使用的地方有背景位图和曲线下面区域的填充位图，以后可能还会有更多的地方使用）。可以想像CListCtrl的ImageList，当建好一个ImageList后，使用的时候只需要指定位图序号即可。AddImageHandle函数从文件里面加载图片，支持bmp、png、jpg、gif格式。AddBitmapHandle函数只能接受位图句柄（HBITMAP），非0即认为是一个合法的HBITMAP,如果bShared为真，则位图共享，这里的共享的概念是，外面可以将控件内部的位图拿来使用（参看GetBitbmp函数）。

注意，AddImageHandle、AddBitmapHandle2和AddBitmapHandle3函数调用会使控件自己创建位图，这样的位图，如果为共享模式，外界也可以拿来使用，但一定不能调用DeleteObject函数释放资源。

### BOOL RemoveBitmapHandle(OLE\_HANDLE hBitmap, BOOL bDel);

### BOOL RemoveBitmapHandle2(short nIndex, BOOL bDel);

移除位图，如果bDel为真，则为位图句柄调用DeleteObject函数（不管位图是否共享），如果位图为控件自己创建，则不管bDel是否为真，都将调用DeleteObject函数。

控件在退出的时候，将自动释放下面两种情况下的位图：

一，控件自己创建的位图；

二，外界按非共享模式传入的位图（以bShared为假调用AddBitmapHandle函数）。

所以，对于外界传入的位图句柄：

一，如果是按共享模式添加到控件的，则二次开发者要么显示调用上面两个函数释放资源（bDel必须要为真），要么显示调用DeleteObject释放资源。

二，如果是按非共享模式添加到控件的，则一定不要调用DeleteObject函数去释放资源，除非是在按bDel为假调用上面两个函数之后，再去调用DeleteObject。

对于外界传入的非共享位图，二次开发者最明智的做法是不去管资源的释放问题。

### long GetBitmapCount();

### OLE\_HANDLE GetBitmap(short nIndex);

### short GetBitmapState(short nIndex);

### short GetBitmapState2(OLE\_HANDLE hBitmap);

枚举位图，包括句柄和状态，GetBitmapState和GetBitmapState2用short返回状态，目前的状态有两种（最低位为1时为共享，第二位为1时为控件自己内部创建的位图），以后可能会有更多的状态，如果返回-1，则说明序号非法或者没有找到hBitmap句柄。注意在用GetBitmap函数获取位图句柄时，如果位图不是共享的，则将返回NULL，如果nIndex指向的序号非法，也返回NULL。

### BOOL SetBkBitmap(short nIndex);

### short GetBkBitmap();

设置整个控件的背景位图，nIndex即为位图句柄序号，-1表示没有设置背景位图。

### BOOL SetCanvasBkBitmap(short nIndex);

### short GetCanvasBkBitmap();

设置画布（就是绘制网格的区域）背景位图，nIndex即为位图句柄序号，-1表示没有设置画布背景位图。

### BOOL SetBkMode(short BkMode);

### short GetBkMode();

### BOOL SetCanvasBkMode(short CanvasBkMode);

### short GetCanvasBkMode();

设置背景位置显示模式，前者为控件背景，后者为画布背景，取值为（低7位）：0－平铺；1－居中；2－拉伸；对于控件背景，高8位如果为1，则绘制区域将裁剪掉画布区域（当无画布背景或者画布背景模式为剧中时，才有意义，否则裁不裁剪最终效果是一样的，不裁剪效率还会高一些），对于画布背景，高8位无意义，必须为0。

### BOOL ExportImage(LPCTSTR pFileName);

### long ExportImageFromPage(LPCTSTR pFileName, long Id, long nStartPage, long nCount, BOOL bAll, short Style);

### long ExportImageFromTime(LPCTSTR pFileName, long Id, DATE BTime,

### DATE ETime, short Mask, BOOL bAll, short Style);

### long ExportMetaFile(LPCTSTR pFileName, long Id, long nBegin,

### long nCount, BOOL bAll, short Style);

### void BatchExportImage(LPCTSTR pFileName, long nSecond);

### long ImportFile(LPCTSTR pFileName, short Style, BOOL bAddTrail);

导出当前页图片到文件中（不管有没有数据），对于ExportImage函数，如果pFileName为空串，则弹出文件浏览对话框，让用户选择，否则直接导出到pFileName文件中，格式由文件名后辍决定，成功返回真。

第2－4个函数用于导出一部分数据到图片中。如果bAll为假，则只导出Id这一条曲线，否则导出所有曲线，此时Id无效。注意nStartPage从1开始（其它涉及到序号的，比如DelRange2等，都是从0开始的），nCount为-1时代表导出到末页；Mask参数意义参看DelRange接口。返回导出图片的张数。

第5个函数用于定时导出当前页的图片，这在绘制实时曲线时有用，如果没有绘制实时曲线，则定时导出无意义，因为每次导出都是完全一样的。nSecond是导出频率，单位是秒，如果nSecond小于等于0，则定时导出结束。

除第一个函数外，所有导出函数生成的文件名都依赖于pFileName（所以它不能为空，而且还必须满足一定的格式要求），通过\*号作为替换符，比如pFileName为c:\\*\*\*\*.jpg，则导出的文件名依次是：c:\0001.jpg、c:\0002.jpg……，如果要导出的文件已存在，将自动累加数字，直到找到一个可用的文件名。\*号替换符只要连续并且大于等于1个即可，位置任意，比如：c:\123\*\*\*456.png等，只要保证出现在文件名字里面即可，替换符越少，则可导出的图片数量就越少，比如两个星，则最多导出99张图片。如果没有后缀名（包括无法识别的后缀名），则导出为bmp，否则导出相应格式的图片。

上面说的这些，都是在Style等于1的情况下，Style还可取2到6的值，表示导出曲线到文本文件或者二制文件之中，Style取如下：

1－导出图片文件（bmp png jp(e)g gif）；

2－导出曲线到文本文件（ansi）；

3－导出曲线到文本文件（unicode）；

4－导出曲线到文本文件（unicode big endian）；

5－导出曲线到文本文件（utf8）；

6－导出曲线到二制文件；

对于按2至6的导出，返回的是导出数据的个数。

对于导出函数，如果文件名非空，则由Style决定文件的导出格式，而不是按后辍名，如果文件名为空（Style不等于1时，文件名是可以为空的），则弹出文件选择框供用户选择，此时Style仅仅相当于一个建议的格式，即在文件保存对话框的类型过滤下拉框里面，会默认的选中Style建议的格式，但用户是可以修改的。

对于ImportFile函数，它用来从文件中导入曲线数据，Style只能取2或者6，2表示文本文件，6表示二进制文件。如果文件名非空，则由Style决定文件的格式而不是按后辍名，此时如果Style为2，则根据文件内容判断具体是哪一种文本格式，为6的话直接按二进制导入文件。如果文件名为空，将弹出文件选择框供用户选择（此时Style无效，控件将根据用户在打开文件对话框里的选择，决定是文本文件还是二进制文件，如果是文本文件，则根据文件内容决定是ansi、unicode、unicode big endian还是utf8）。如果返回-1，则说明参数出错，或者用户取消了文件选择，否则返回高2字节为总的数据个数，低2字节为成功添加到控件的数据个数，均当成无符号数看待（若发生溢出，也不影响导入）。

在按文本格式导出曲线的时候，为了兼容，我没有添加任何自定义的格式信息（这样windows和其上的其它软件，都可以打开图无文件并正确读取到内容而不至于显示乱码），于是就出现一个问题：时间的格式是普通的格式呢，还是浮点数格式（控件在导出的时候，会根据当前横坐标的显示格式来导出，但有一点不同，只要是按时间格式显示的，都导出为完全时间，而不管当前显示的是只有日期还是只有时间）？目前的做法是，对第一行数据进行智能判断，以确定是普通时间格式还是浮点数格式，得出结论后，后面所有行直接使用，这也是为了效率，所以一个图元文件里面，不应该即有普通格式的时间，又有浮点数格式的时间。

速度上，导出二进制最快，按浮点数格式导出时间次之，按标准时间格式导出最慢。

注意，bAddTrail参数无意义（从7/5/2014日的更新中开始），相当于它衡等于真，保留它只是为了接口的一致性。

关于ExportMetaFile函数，它只能导出图元文件，功能与ExportImageFromPage函数和ExportImageFromTime函数，在Style大于1的情况下是完全一样的（Style的意义也一样，所以它的Style参数只能取值2到6），不同的地方是它可以精确到点数量。这个功能有一个用处，比如：

在曲线很长的情况下，考虑到内存的问题，一般都会定时的截掉曲线的一部分点，以免内存占用无限的增长。若让控件自动删除点（参看SetMaxLength函数），当然是很省事的，但带来一个问题，就是删除了的点再也找不回来，如果用户要求将点保存在图元文件里面再删除点呢，那么SetMaxLength函数就无能为力了。此时应该由二次开发者来定时的删除某些点（同时去掉控件自动限制长度的功能），这样用ExportMetaFile函数配合DelRange2函数，就可以达到要求了，因为这两个函数都是可精确到点数量的。

注意，2-5这些导出方式，可能会有精度的损失，如果按时间导出，则只能精确到秒，如果按double导出，则采用%f格式，他们都可能存在不同程度的精度损失；当然，纵坐标也一样，它永远都是采用%f格式导出的。

### BOOL GetOneTimeRange(long Id,DATE\* pMinTime, DATE\* pMaxTime);

### BOOL GetOneValueRange(long Id, float\* pMinValue, float\* pMaxValue);

### BOOL GetOneFirstPos(long Id,DATE\* pTime,float\* pValue,BOOL bLast);

获取指定曲线的时间范围，值范围，第一个点及最后一个点坐标，返回假表示找不到曲线。如果只想得到最小量或最大量，则把不需要的量的指针指定为空即可。

### BOOL GetTimeRange(DATE\* pMinTime, DATE\* pMaxTime);

### BOOL GetValueRange(float\* pMinValue, float\* pMaxValue);

获取所有曲线的时间及值范围，用法和上面的函数一样。

### void GetViableTimeRange(DATE\* pMinTime, DATE\* pMaxTime);

获取本控件支持的时间范围，其实也就是COleDateTime所支持的时间范围。

### void TrimCoor();

### void EnableAutoTrimCoor(BOOL bEnable);

修整坐标，在定点缩放的时候，为了保证缩放点不被移动，经常需要重新设置横纵坐标的开始值，这样坐标的开始值会变得零乱（比如纵坐标的开始值可能为1.00001），该函数将修整他们，以便于更好的观看。注：ST\_Curve不会自动调用本函数，除非按TRUE调用EnableAutoTrimCoor函数，这样，每次在定点缩放之后，控件都将自动调用TrimCoor函数（在调用EnableAutoTrimCoor函数的时候，如果bEnable为真，控件将马上调用一次）。

修整坐标的方法是去掉小数（如果横坐标显示为时间，等于是精确到天），这样一来，有可能出现曲线完全被移出画布的情况，此时控件将做必要的平移操作，最终的结果是：修整后的坐标原点，在经过0次或多次平移后，可让原点坐标为整数。

如果对上面的修整坐标不满意，控件还增加了通过插件或者Lua脚本修整坐标的功能，具体参看LoadPlugIn和LoadLuaScript接口。

### long TrimCurve(long Id, short State, long nBegin, long nCount,

### short nStep, BOOL bAll);

### long TrimCurve2(long Id, short State, DATE BTime, DATE ETime,

### short Mask, short nStep, BOOL bAll);

修剪曲线，即更改指定点的状态，State和AddMainData2或AddMainData函数的State参数一样。本函数将Id曲线的从第nBegin起的nCount个点之间的每隔nStep个点的状态设置为State状态，如果nCount等于-1，则一直操作到曲线结尾，否则它应该大于0，如果bAll为真，则作用于所有曲线，此时Id无效，成功返回真，如果未找到指定曲线，返回假。这个功能的作用我举个例子：当曲线被缩小到很小的时候，可能整条曲线上都画满了曲线点，此时可以用这个函数让控件每两个点只显示一个，或者每三个点只显示一个，达到精简曲线的目的，同样，当曲线被放大后，用这个函数又可以将隐藏的点再重新显示出来。至于TrimCurve2，作用与TrimCurve一样，只是用两个时间来确定要操作的范围，Mask参数的低位起第一位代表BTime的有效性，如果无效就从曲线头开始，第二位代表ETime的有效性，如果无效一直操作到曲线尾。返回值代表成功更改状态的点数量，如果为0，可能是空范围或者曲线不存在。

这个函数还用来巧妙的将曲线变柱状图，将柱状图变曲线，请看demo的“曲线与柱状图互换”。

### short PrintCurve(long Id, DATE BTime, DATE ETime, short Mask,

### short LeftMargin,short TopMargin,short RightMargin,short BottomMargin,

### LPCTSTR pTitle, LPCTSTR pFootNote, short Flag, BOOL bAll);

打印曲线，Id可指定某一条曲线，BTime和ETime确定打印范围，Mask决定BTime和ETime参数的有效性。如果用户选择打印当前页，则BTime参数无效。四个Margin参数为边距（屏幕象素），所有边距推荐设置为50。pTitle为标题，pFootNote为脚注（由于控件本身也有标题和脚注，而且在打印的时候也会显示出来，所以本接口里面的标题与脚注，大家可以理解为页眉页脚，这是显示在控件窗口外面的，具体可以采用demo的打印功能来查看结果，对于没有打印机的朋友，这里推荐一个虚拟打印机软件pdfFactory，用它不但可观看打印效果，还能将结果转换成PDF文档）。参数Flag从低位起：

第一位：是否打印页序号；

第二位：是否后台打印，此时打印过程不受用户干涉，控件将获取默认打印机做默认打印，但打印方向为横向；

第三位：是否强行垂直居中，如果不强行垂直居中，则在需要时垂直居中（居中的依据是当前选中的曲线，如果无选中曲线，则按画布中第一条可见曲线来居中）；

第四位：是否保持本页纵坐标，如果设置该位，可能会出现某一页画布中完全没有曲线的状态，控件将跳过这样的页不打印。如果三、四位同时为1，则第四位有效；

1. 六位：意义与第三、四位一样，但只在打印第一页时有效，换句话说，第三、四位只在打印非第一页时有效；

至于为什么要设置三、四位和五、六位，考虑这种情况，在控件窗口很小时，如果保持原点不变，打印到纸上的效果可能是曲线在纸的最下边，上边留下很大一片空白，此时可以通过设置五、六让控件在打印第一页时，在垂直上居中，然后打印后面页的时候，为了让每一页的坐标原点相同，可以通过三、四位让控件保持纵坐标不变。

第七位：如果为1，则按位图方式打印前景（在绘制平滑曲线时，如果打印机不支持，可以采用这种方式，这种方式优点是解决了平滑曲线的打印问题，缺点是画面粗糙）；

如果bAll为真，则Id无效，将打印所有处于显示状态的曲线。

返回值，0-成功；-1-成功，但无曲线可打印；1-打印失败（调用GDI打印函数失败，比如StartDoc等函数）；2-参数无效；3-用户取消打印；4-打印区域不存在（纸张太小）；5-不存在默认打印机；6-打印机不支持按位图打印。

另外说明一点，TitleColor和FootNoteColor这两个属性，虽然指的是控件里面的标题与脚注的颜色，但在打印的时候，这两个属性同样会被应用于打印时的标题与脚注（参看PrintCurve接口说明文档第一段），由于打印纸是白色的，如果设置了一个与白色十分相似的颜色，那么打印到纸上将看不见（可能大家会说，控件里面的标题与脚注为什么可见呢，如果不可见，也不会设置这样的颜色！答案是，控件有背景，比如背景是黑色的，设置标题颜色为白色，这是很合适的，而在打印的时候，情况变了，PrintCurve接口的pTitle内容将变得不可见），怎么办呢？控件会自动判断两种颜色的相似程度，并在合适的时候，取其反色来打印。同样，对于其它不可设置的颜色，控件都会做这样的处理，这些颜色包括：

水印、帮助文字的颜色，正常情况下，它们始终等于前景色的二分之一，当与背景色很相似的时候，将取反；

版权文字的颜色，正常情况下，它始终等于前景色的四分之三，当与背景色很相似的时候，将取反。

标题及脚注颜色不会做上面的处理，因为它们是有接口可以更改的，如果因为与背景色太相似而不可见，控件认为是有意而为之，所以不处理。

### long GetScaleNums();

获取坐标刻度个数，注意不包括第一个，只要是在坐标上绘制出来的刻度，不管它是长的还是短的，都算一个。返回值高16位为横坐标刻度个数，低16位为纵坐标刻度个数。

### long ReportPageInfo();

请求报告页数信息（控件只在页数信息改变的时候才会主动发送消息），处理方法与PageChangeMSG消息完全一样，本函数返回lParam，同时将wParam写入Register1。注意，这是唯一一个在32位版本下仍然会写Register1的接口。

如果设置了PageChangeMSG和MSGRecWnd，则本函数还会发送PageChangeMSG消息，所以如果有PageChangeMSG消息的响应函数的话，可以直接仍掉本函数的返回值而等待消息。同样，如果允许控件触发PageChange事件，则调用本函数也会触发PageChange事件，该事件的参数就是PageChangeMSG消息的wParam和lParam的低32位。

### BOOL ShowLegend(LPCTSTR pSign, BOOL bShow);

### BOOL ShowCurve(long Id, BOOL bShow);

显示、隐藏指定的图例，或者曲线。后者要求曲线已经存在，否则会失败；

曲线本身没有显示还是隐藏这个属性，让曲线显示，其实还是让图例显示。

### BOOL SelectCurve(long Id, BOOL bSelect);

选中曲线或者取消选中曲线。不管是选中还是取消，如果曲线处于隐藏状态，则本函数会将其更改为显示状态。

### short DragCurve(short xStep, short yStep, BOOL bUpdate);

拖动曲线，模拟用鼠标拖动曲线的效果，xStep大于0时曲线右移，yStep大于0时曲线上移，返回值从低位起，如果第一位为1，则在水平轴上移动过曲线，如果第二位为1，则在垂直轴上移动过曲线。换句话说，有可能无法移动曲线，无法移动曲线的情况在前面已经说过了，就是不能移动到画布上没有任何曲线这种状态。当然，水平轴和垂直轴上是否可移动是独立的，即可能水平轴上无法移动曲线，而垂直轴上却可以移动。

### BOOL VCenterCurve(long Id, BOOL bUpdate);

按指定的曲线为中心垂直移动曲线，让Id处于屏幕中间。如果曲线处于隐藏状态，则本函数会把其状态改为显示状态；如果曲线在当前页中不可见或者找不到曲线，则返回假。

### BOOL GetSelectedCurve(long\* pId);

获取当前选中的曲线，选中的曲线在绘制的时候会绘制到最上层（如果允许控件自动调整Z-Order的话，参看EnableAdjustZOrder函数），便于观看，同时还会加粗。

### BOOL GotoCurve(long Id);

跳转到指定曲线的开始值，必要是移动纵坐标。如果曲线 处于隐藏状态，则本函数会先将其更改为显示状态。

### BOOL IsSelected(long Id);

判断曲线是否处于选中状态。

### BOOL IsLegendVisible(LPCTSTR pSign);

判断图例是否处于可见状态。

### BOOL IsCurveVisible(long Id);

判断曲线是否可显示（可显示不一定就是可见，这 要看它在当前页上有没有显示，这用IsCurveInCanvas来判断）。

### BOOL IsCurveInCanvas(long Id);

判断曲线是否在画布上有显示。

### BOOL IsCurveClosed(long Id);

判断曲线是否封闭（屏幕坐标相同，非实际坐标）。

### DATE GetTimeData(short nCurveIndex, long nIndex);

### CString GetTimeData2(short nCurveIndex, long nIndex);

### float GetValueData(short nCurveIndex, long nIndex);

### short GetState(short nCurveIndex, long nIndex);

### BOOL GetPosData(short nCurveIndex, long nIndex, long\* px, long\* py);

### BOOL InsertMainData(short nCurveIndex, long nIndex,

### LPCTSTR pTime, float Value, short State, short Position, short Mask);

### BOOL InsertMainData2(short nCurveIndex, long nIndex,

### DATE Time, float Value, short State, short Position, short Mask);

### BOOL DelPoint(short nCurveIndex, long nIndex);

### BOOL CanContinueEnum(long Id, short nCurveIndex, long nIndex);

以上函数组用于枚举指定曲线中的所有点，并可在枚举过程中操作曲线，具体如下：

通过GetCurveIndex接口，得到想要枚举的曲线的nCurveIndex，这是第一步。

GetTimeData、GetValueData、GetState、GetPosData获取nIndex位置的时间、值、状态及屏幕坐标，参看AddMainData函数。

InsertMainData、InsertMainData2用于插入值，参数的意义与AddMainData一样，除了hCurve和Position参数。Position取值有：

-1：插入到nIndex之前，此时Mask无意义；

0：更改nIndex点，此时Mask有意义：按位算，从低位起，1－Time有效，2－Value有效，3－State有效；

1：插入到nIndex之后，此时Maks无意义。

注意，这两个函数不会对插入点按时间排序，插入位置完全由Position决定。

DelPoint函数用于删除nIndex点。

因为InsertMainData、InsertMainData2和DelPoint函数都没有bUpdate参数，所以要让插入和删除操作生效，最后还需要调用Refresh函数。

枚举过程中如果要增加删除点，则一定要小心重复枚举和枚举不全的情况，比如当你正要枚举第2个值时候，在第一个值的前面又添加了一个点，此时你再枚举第2个值的时候，其实还是枚举的原来的第1个值，即第1个值被枚举了两次；同样，删除点的时候，则可能造成枚举不全；另有甚者，如果你正要枚举第100个值，第100个值及其以后的所有点被删除，此时枚举将越界。

在枚举越界的情况下，如果接口直接返回想要枚举的值，比如GetDataTime，将返回一个默认值；如果接口返回BOOL，枚举值通过指针获取，比如GetPosData，将返回FALSE；所以，对于前者，不好判断是否枚举成功，为了统一起见，建议大家要保证枚举过程不越界，如何保证呢，首先可以代码上保证（枚举过程中不增加不删除不交换曲线层次），其次，如果实在不放心，可以用CanContinueEnum来检测是否可继续枚举。

注意：CanContinueEnum只是用来检测越界问题，在删除点时，可能造成枚举不全，在增加点时，可能造成重复枚举，这两种情况下，CanContinueEnum都可能是无能为力的。

下面我举个例子，这个例子中，在枚举过程中还对曲线进行了删除点操作（增加删除点操作不是不允许，只是要小心，使用对了就没事）。代码摘自demo，里面有更详细的注释，大家可以去看看。

short nCurveIndex = m\_ST\_Curve.GetCurveIndex(11);

if (nCurveIndex >= 0)

{

long CurveLen = m\_ST\_Curve.GetCurveLength(11);

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

{

if (!m\_ST\_Curve.CanContinueEnum(11, nCurveIndex, i))

break; //无法再继续枚举了

float value = m\_ST\_Curve.GetValueData(nCurveIndex, i);

CString str;

str.Format(\_T("%s, %f"), ((COleDateTime) m\_ST\_Curve.

GetTimeData(nCurveIndex, i)).Format(), value);

AfxMessageBox(str);

if (value > 92.0f) //删除y值大于92.0f的点，模拟一边枚举一边删除

{

m\_ST\_Curve.DelPoint(nCurveIndex, i);

--i; //当前i的位置删除了（后面的补了上来），继续在这个位置上枚举

--CurveLen; //因为删除了一个点，所以点的总数要减一

}

}

m\_ST\_Curve.Refresh();

}

### long GetCurveCount();

### long GetCurve(long nIndex);

前者获取曲线的条数，后者用于按序号获取曲线的Id，枚举曲线会有用。

### void SetCurveTitle(LPCTSTR pCurveTitle);

### CString GetCurveTitle();

设置曲线标题。

### long GetLegendCount();

### BOOL GetLegend2(long nIndex, OLE\_COLOR\* pPenColor, short\* pPenStyle,short\* pLineWidth, OLE\_COLOR\* pBrushColor,short\* pBrushStyle, , short\* pCurveMode，short\* pNodeMode);

### long GetLegendIdCount(long nIndex);

### long GetLegendId(long nLegendIndex, long nIdIndex);

获取图例的个数，获取指定图例控制的曲线个数，最后一个函数用于枚举曲线，对于GetLegend2函数，它的功能和GetLegend函数是一样的，只不过是通过序号获取。

GetLegendId的nLegendIndex参数就是GetLegendIdCount函数的nIndex参数。

### BOOL SetBuddy(long hBuddy, short State);

### short GetBuddyCount();

### long GetBuddy(short nIndex);

设置联动关系，联动关系中，有一个控件被指定为服务器，其它可以有1个或多个客户机，服务器与这些客户机组成一组联动关系，这一组中任何一个曲线的横坐标改变时，将会使其它曲线也跟着改变横坐标，达到的目的就是，这一组控件的横坐标开始值及横坐标间隔值保持相同。这个效果有点像是在一个控件里面绘制了两种性质完全不同的曲线。这个用法我举个例子，比如要绘制电度及功率曲线，但是电度和功率这两个东西无法绘制在同一个控件里面，此时可以放置两个控件，一个绘制电度，一个绘制功率，并让他们形成联动关系，这样，任意时刻的电度和功率就一目了然了。当然，这两个控件在容器里面应该是左对齐的、长度完全相同的，并且，这两个控件应该属于同一个容器，否则联动关系没有什么意义。下面解释一下SetBuddy函数的具体用法：

hBuddy == 0时

如果控件为联动服务器，则取消联动服务器，并关闭与所有联动客户机的连接

如果控件为联动客户机，则取消与联动服务器的连接

hBuddy != 0时（此时应该是某个控件的窗口句柄）

如果State == 0，则hBuddy看成联动客户机，当前控件（调用者）自动成为联动服务器，并把hBuddy添加到自己的联动客户机队列里面。

如果State == 1，则hBuddy看成联动客户机，并将其从当前控件的联动客户机队列里删除，此时当前控件必须已经是联动服务器。

注：删除联动客户机有两种方法，一种是对联动服务器调用SetBuddy（此时hBuddy为联动客户机，State为1）；一种是对联动客户机调用SetBuddy（此时hBuddy为0，State无意义）。

删除联动服务器只能是对联动服务器调用SetBuddy（此时hBuddy为 0，State无意义）。

### void EnableZoom(BOOL bEnable);

### void EnableHZoom(BOOL bEnable);

是否允许缩放曲线，后者用于水平缩放，关于什么是水平缩放，参看SetHZoom接口。

当设置为禁止曲线被缩放时，这里指的是禁止操作员对曲线的缩放，二次开发仍然可以通过接口缩放曲线（SetZoom、SetHZoom）。禁止曲线被缩放，实际上是禁止了以下功能键：+ -键（定点缩放）、鼠标滚轮加shift键（原点缩放，EnableZoom）、鼠标滚轮加alt键（水平缩放，EnableHZoom）。所以在禁止缩放时，以前的缩放率仍然有效，二次开发者随时可以通过Set(H)Zoom(0)来恢复到不缩放状态。水平缩放默认不开启。

由于添加了水平缩放，GetSysState接口将受影响而返回多一位，但仍是向前兼容的。

在水平缩放开启状态下，alt键将被控件消化掉，所以用户是无法通过alt键去打开菜单的（当然只是控件获得焦点时才会这样，所以不用担心给操作上带来不便）；如果水平缩放未开启，则按alt将会走windows标准流程。

### BOOL SetHPrecision(short Precision);

### short GetHPrecision();

当横坐标显示为值的时候，其显示精度，参看SetVPrecision。

### short GetCurveIndex(long Id);

### BOOL SetCurveIndex(long Id, short nIndex);

更改、查询指定曲线在整个曲线链表里面的位置，从0开始。更改曲线的位置，可以达到更改曲线的层次，就像Z-Order差不多。当然，由于本控件绘制的是平面曲线，如果曲线没有相交，则他们的Z-Order并不会表现出现，如果曲线发生相交甚至重叠时，Z-Order就有用了，越大的nIndex，绘制在越上层。

### BOOL SetGridMode(short GridMode);

### short GetGridMode();

网格显示模式，从低位起，第一位：是否显示横向网格；第二位：是否显示纵向网格；第三位：是否显示为实线；第四位：是否只在主刻度（短的刻度叫次刻度，可以联想直尺的样子）上绘制风格。默认的GridMode为3，即用虚线显示所有纵横网格。

### void SetFootNote(LPCTSTR pFootNote);

### CString GetFootNote();

设置曲线脚注，比如显示公司名什么的，自己发挥想像。控件在默认情况下，是没有脚注的。

### void EnableAdjustZOrder(BOOL bEnable);

是否允许自动更改Z-Order，如果允许，控件将在某曲线被选中时，把它的Z-Order放在最大的位置上（最上层）。但该曲线在取消选中时，不会还原Z-Order，当另外某曲线又被选中时，又将对另外那条曲线做上面同样的操作，这样经过几次调整后，曲线间的Z-Order将是不确定的。如果你想让所有曲线保持他们的Z-Order不变，则可以通过本函数来实现。注意，就算设置不让控件自动更改Z-Order，你仍然可以通过SetCurveIndex函数来更改其Z-Order（但SelectCurve函数则不行）。

按FALSE来调用本接口之后，曲线仍然可被选中，但被选中时，不会被提到最前面。

### void EnableHelpTip(BOOL bEnable);

是否允许显示简单的帮助信息在背景上，只是一些简单的鼠标键盘快捷操作方法，默认情况下，控件在刚开始运行时会显示帮助，并持续10秒后自动消失。

### BOOL SetFillDirection(long Id, short FillDirection, BOOL bUpdate);

### short GetFillDirection(long Id);

填充方向，对于FillDirection，按位算从低位起，可同时向多个方向填充：

1－向下填充 2－向右填充 3－向上填充 4－向左填充

注意，要填充，还要图例的支持，如果图例指示不填充，则这个设置无效。反之，如果图例指示要填充，但填充方向又是0（没有向任何方向填充），则也不会填充，所以它俩要共同作用来完成填充，在AddMainData(2)的时候，会设置填充方向为向下，所以默认情况下，只要图例指示填充，就会填充，如果图例指示要填充，而属于这个图例的某一条曲线又不想填充，那么SetFillDirection就派上用场了！

第5、6位当成一个整体来看，如果不为0，则在填充的矩形里面显示坐标Y值（柱状图一般要求这样），显示位置是上下左右均居中，目前只支持这一种位置。如果填充的图形不是一个矩形，则控件会计算出来一个矩形来，比如在向下填充时，所有点（这一次执行填充的点）的Y坐标的平均值当成矩形的top值，开始绘制的第一个点的X当成矩形的left值，最后一个绘制点的X当成矩形的right，画布的bottom当成矩形的bottom，其它填充方向类似。现在说5、6位的意义，1－显示第一个绘制点的值，2－显示最后一个绘制点的值，3－显示所有绘制点的平均值。

注：绘制柱状图的时候，应该每两个点为一柱，如果多于两个点为一柱，则不显示值还没事，如果显示值的话，会随着曲线的拖动，值有可能变化，比如3个点为一柱，当第一个点拖出屏幕后，屏幕上显示的值将变成第二个（如果设置为显示第一个绘制点的话，即1），这不是控件的BUG，而是设计即是这样，因为采用了优化的绘制方法。对于这个问题，可以看演示“方案二”，红线显示的值不会变，而绿线则会变，这是一种不正确的用法。

显示值的颜色在一定程度上是可控制的，它用FillDirection的第7、8位来控制，具体来说，第7位：0－使用前景色，1－使用本条曲线的画笔的颜色，第8位如果为1，则使用第7位指示的颜色的反色。

注意：在填充模式为Solid时，7、8位无效，此时将使用Solid模式的填充色的反色来显示纵坐标值。

### void SetVisibleCoorRange(DATE MinTime, DATE MaxTime, float MinValue,float MaxValue, short Mask);

### void GetVisibleCoorRange(DATE\* pMinTime, DATE\* pMaxTime,float\* pMinValue, float\* pMaxValue);

坐标显示范围，超出范围的坐标不显示，默认显示所有坐标值。Mask的低字节用于确定前面四个参数的有效性，高字节则用于取消已经设置了的范围，顺序与前面四个参数的顺序一样，比如：

m\_ST\_Curve.SetVisibleCoorRange(.0, .0, .0f, .0f, 1);

//让横坐标显示大于等于.0的值

m\_ST\_Curve.SetVisibleCoorRange(.0, .0, .0f, 1.0f, 0xC);

//让纵坐标显示.0f到1.0f之间的值

m\_ST\_Curve.SetVisibleCoorRange(.0, .0, .0f, .0f, 0xF00); //取消所有范围

注意：这与SetBeginTime2等函数的作用是完全不一样的，SetBeginTime2是设置原点的坐标，SetVisibleCoorRange是设置显示范围，显示不一定非要从原点开始。

如果小值大于大值，则坐标将完全不被显示，这是隐藏坐标的唯一方法，但是如果要隐藏的是横坐标，并且横坐标显示为时间的话，推荐的方法是使用SetShowMode函数，这个函数可以控制横坐标是否显示日期和时间，当日期和时间都不显示的时候，就隐藏了横坐标。

### void SetBenchmark(DATE Time, float Value);

### void GetBenchmark(DATE\* pTime, float\* pValue);

设置基值，一般不会使用这两个函数，请阅读《ST\_Curve升级报告》，特别是你要使用这两个函数的时候。

### short GetPower(long Id);

获取指定曲线的幂次，幂次这个属性为只读，控件自动求所有曲线的幂次。返回1或者2（0次和1次统称为1次，大于2次统称为2次），如Id果曲线未找到则返回-1。

### BOOL ChangeId(long Id, long NewId);

更改曲线的Id。

### BOOL CloneCurve(long Id, long NewId);

复制一条曲线，从Id复制到NewId，复制结束后不会删除Id曲线。

### BOOL UniteCurve(long DesId, long nInsertPos, long Id, long nBegin,long nCount);

### BOOL UniteCurve2(long DesId, long nInsertPos, long Id,DATE BTime, DATE ETime, short Mask);

### BOOL UniteCurve3(long DesId, DATE fInsertPos, long Id, long nBegin,long nCount);

### BOOL UniteCurve4(long DesId, DATE fInsertPos, long Id,DATE BTime, DATE ETime, short Mask);

将Id曲线的指定范围之内的点，插入到DesId曲线里面，范围的确定方法有两种，一种是从某个点开始的一定数量的点，一种是两个时间，这两种范围的确定，在前面的方法中不仅一次解释过了，以后不再说明，大家应该一看到就明白。

插入有两种方法：

一：对所有范围之内的点，以bAddTrail为假调用AddMainData2函数，结果是点被插入到了按时间从小到大排列的队列的合适位置上。要执行这种插入，将nInsertPos置为-1即可，只有UniteCurve和UniteCurve2函数能执行此种插入。

二：直接将范围之内的点插入到指定位置，这种插入，上面四个函数都可以，此时nInsertPos应该大于等于0（如果等于曲线的点数量或者更大，都将插入到曲线末尾），注意是插入到nInsertPos的前面，所以如果等于0的话，是插入到DesId的最前面，插入的次序按点在Id曲线里面的次序为准。至于fInsertPos参数，意思是插入到DesId曲线的第一个横坐标大于等于fInsertPos的点的前面，如果没有大于等于fInsertPos的点，则将插入到曲线末尾，如果所有点都大于等于fInsertPos（二次曲线），或者第一个点就已经大于等于fInsertPos（一次曲线），则将插入到曲线的最前面。

注意，所有这四个函数，后面确定的范围，都指的是Id曲线的范围，而非DesId曲线的范围。插入结束后，不会自动删除Id曲线。

在demo的演示“方案一”里面，有使用这些函数的例子（可能注释掉了）。

### BOOL OffSetCurve(long Id, DATE Time, float Value, short Operator);

偏移曲线，Operator的高字节代表如何使用TimeSpan参数，低字节代表如何使用ValueStep参数，他们的具体意义是一样的：

'+'－加，'\*'－乘，其它值无效，如果横坐标显示为时间，则只能对横坐标做加操作，因为时间乘时间是无意义的。

举个例子，下面的程序将14曲线的每一个点的横坐标加1.0 / 24 / 2（半小时），纵坐标乘以0.95：

m\_ST\_Curve.OffSetCurve(14, 1.0 / 24 / 2, .95f, ('+' << 8) + '\*');

### long ArithmeticOperate(long DesId, long Id, short Operator);

对两条曲线做算术运算，具体实现方法是找到两条曲线中横坐标相同的点，然后对这两个点的纵坐标做算术运算，包括加减乘除运算，注意只支持一次曲线。

本函数只对DesId和Id曲线遍历一次，找到横坐标相等的点，然后对这两个点的纵坐标执行Style指定的操作，并将结果赋给DesId的相应点的纵坐标。这也是本函数只支持一次曲线的原因之一，如果要支持二次曲线，则对于每一个DesId里面的点，都将要遍历Id曲线一次，效率太差。而且，如果某一个DesId里面的点，其横坐标对应多个Id里面的点（二次曲线才会出现这样的情况），那么如何对这些点的纵坐标（多余两个）作Operator指定的操作呢？

Operator：'+'－加，'-'－减，'\*'－乘，'/'－除。

本函数返回执行过算术运算的点数量。

下面我说一下本函数的内部实现，其实和数据结构里面的合并两个有序序列，并让合并后的序列仍然有序的操作方法完全一样，如果大家对这种实现原理有什么疑问，可以查阅数据结构方面的书籍。本函数加上CloneCurve函数和OffSetCurve函数，可以完成一些较为复杂的曲线处理过程，特别是在统计的时候，可能会有用，比如每个月的收入是一条曲线，支出是另一条曲线，那么这两条曲线相减就是净收入曲线了。再复杂一点，比如要将曲线1加上曲线2和曲线3的乘，再给曲线1的每一个点的横坐标加上1.0，给纵坐标乘以1.5，可以这样操作：

CloneCurve(2, 100); //从曲线2复制出临时曲线100

ArithmeticOperate(100, 3, '\*'); //临时曲线100与曲线3求乘

ArithmeticOperate(1, 100, '+'); //将临时曲线100加到曲线1上

OffSetCurve(1, 1.0, 1.5f, ('+' << 8) + '\*'); //偏移曲线1

DelRange(100, 0, -1, FALSE, TRUE);

### void ClearTempBuff();

本函数清除控件内部使用的缓存，该缓存专用于Polygon和Polyline函数，当然控件中使用的缓存远不止这些，但其它的缓存不允许控制，本着效率优先的思路（STL也是这种思想，比如vector，缓存不够时它将重新分配缓存，但缓存在因为删除了某些点而变得富余的时候，它不会回收，直到自己被析构），本控件在缓存有富余的情况下，并不会去压缩缓存，因为控件无法知道下一个时刻，缓存是否仍然还有富余，但二次开发者可能会知道什么时候压缩缓存，所以提供了此方法，让二次开发者对控件的内部缓存有一个初步控制。

注意，清除了缓存并不代表控件内部就不使用缓存，其实这个函数是压缩缓存，控件会在需要的时候再次分配缓存，如果再次分配的大小和调用本函数之前一样大，那么这是一次错误的缓存压缩，只有缓存中有很多未使用的空间时，压缩才有意义。

那么什么时候压缩缓存最好呢，比如在某一时刻，添加了一条点特别多的曲线，比如有一万个点，而在另一个时刻，删除了这条曲线，并且以后或者以后很长一段时间内，不会再有如此长的曲线被添加了，那么此时可以压缩缓存，以释放内存空间。

控件内部默认会申请512个POINT的临时缓存空间。

### BOOL PreMallocMem(long Id, long size);

为曲线预分配空间，size指的是点的个数，如果你能预测某曲线的最大点数量（或者在添加删除点操作非常频繁的时候），则使用这个函数可以让控件达到最高的效率，因为在添加点的过程中，不需要再分配缓存。

本接口需要在曲线已被添加之后调用，所以最好在成功调用一次AddMainData(2)函数之后调用，当然你也可以随时调用，但应该越早越好，如果曲线已经很长了再调用本函数，会涉及到大量的内存拷贝。

如果size小于等于曲线已经分配的缓存，则设置将失败，比如先设置某条曲线的预分配空间为100个点，后来又更改为50个点，则更改不会成功，换句话说，缓存只能是增大不能减小（STL的vector就是这样做的）。

如果size小于等于0，则压缩曲线的预分配空间为刚好装下曲线所有点大小的空间，如果你觉得某条曲线以后不会或者在很长一段时间之内不会有点增加，可以调用本函数，以减少内存占用量。

注意：预分配与曲线实际长度无关（只是预分配不能比曲线实际长度更短），如果对这一点有疑惑，建议看看STL方面的书籍。

删除曲线点（参看DelRange(2)接口）不会释放缓存，除非整条曲线被删除，这也是本着效率优先的原则（思想跟STL一样）。

### long GetMemSize(long Id);

返回指定曲线所分配的缓存（注意不是长度，长度用GetCurveLength接口获得），单位是点的个数（具体一个点占用多少字节，参看后面第六节：关于内存使用量）。

### void GetMemInfo(long FAR\* pTempBuffSize, long FAR\* pAllBuffSize,

### float FAR\* pUseRate, long FAR\* pId);

获取缓存详细信息，pTempBuffSize用于接收临时缓存分配量（参看ClearTempBuff接口），单位是POINT的个数（所以乘以8就是字节数）；pAllBuffSize用于接收所有曲线的总的缓存分配量，单位是点的个数；pUseRate用于接收总的缓存利用率（小于等于1，比如分配了10个点的缓存，结果只有5个点，利用率就是0.5）；pId用于接收缓存利用率最低的曲线的Id，二次开发者如果想要压缩缓存的话（参看PreMallocMem接口），这就是最应该被压缩的那条曲线。

那么，如何得到指定的曲线的缓存利用率呢？答案是：GetCurveLength/GetMemSize，由此可见，对于后三个参数，二次开发者其实是可以通过遍历所有曲线而自己计算出来的，之所以提供本接口，是考虑还有第一个参数无法获取的问题，另外也是为了方便二次开发。

### BOOL IsCurve(long Id);

查看Id是不是一条曲线，非常有讽刺意味的是，我为控件添加了这么多的接口，认为接口已经很全面了，却忽略了上面这个接口，也就是说，在这之前，二次开发者连是否存在某条曲线（Id）这样简单的要求都得不到满足！

### void SetSorptionRange(short Range);

### short GetSorptionRange();

设置吸附效应，吸附效应就像AutoCAD自动找圆心的功能差不多，鼠标会自动跳到附近的一个点上，以便观看其坐标值，控件内部实现办法是，找到一个在吸附范围之内的点即告成功，将鼠标跳到这一点之上，并没有去求在吸附范围之内的所有点中的最佳点（离鼠标点最近的点），这考虑到效率问题，我想AutoCAD也是这样做的，只是他不会在文档里说明，我这个文档面对于二次开发者，所以我要说明一下。其实鼠标在正常移动的时候，总会有一个点最先移动到吸附范围之内（除非点有重合），不可能同时出现多个点都在吸附范围之内，但有一点，鼠标在启用加速后，移动非常快时，或者系统忙时，鼠标都会跳跃前进，此时鼠标完全有可能一下子跳到一个存在多个点的吸附范围之内，比如有2个点，这2个点离鼠标点的距离可能有远有近，由于控件在查询吸附点的时候，采用最先匹配算法（为了效率，没有查找最佳点），所以判断的结果可能是鼠标被吸附到一个与自己相隔较远的点（这个点是在吸附范围之内的），这在人用眼睛看来，好像是控件出现了BUG，实际设计即是这样。这样的现象其实是很难遇得上的。

函数的Range参数即吸附范围，如果鼠标的坐标为x和y，则吸附范围为这样一个正方形：(x - Range, y - Range), (x + Range, y + Range)。

据个人经验，Range应该要大于等于8（象素），否则吸附效果不明显，如果Range小于等于0，则取消吸附效应。

如果按钮shift或者ctrl移动鼠标，则不会产生吸附效应。

### BOOL GetActualPoint(long x, long y, DATE\* pTime, DATE\* pValue);

根据屏幕坐标得到实际坐标，坐标为客户区坐标，参看MouseDown(Up)事件，如果坐标不在画布里面，则返回假（但此时仍然可以得到实际坐标，实际坐标与屏幕坐标只是一种对应关系）。对于pTime和pValue，如果不需要某个值，将指针赋一个空指针即可。

由于本函数没有曲线Id这样一个参数，所以无法考虑Z坐标影响，如果要计算xy点相对于某条曲线来说的实际坐标，需要二次开发者对xy坐标进行一个处理，说是处理，其实也就加减一个偏移量，非常简单（需要用到GetZOffset接口），具体请看demo的OnMouseUpStcurvectrl函数里面，有通过鼠标点击增加点的例子（我注释掉了）。

### long GetPointFromScreenPoint(long Id, long x, long y, short MaxRange);

获取屏幕坐标指向的真实坐标在曲线中的序号，有了这个序号，可以调用DelRange2函数轻松的删除掉这个点。此时一定要开启吸附效应，否则很难刚好点中想要的点。如果返回-1，说明没有点中屏幕点，或者点中的屏幕点不属于Id曲线。MaxRange参数很重要，它的作用是确定一个范围，和吸附范围一样，只要在这个范围之内Id曲线有点，就会返回这个点在曲线中的序号。大家要问，开启吸附效应后，鼠标不是自动跳到点上去了吗，按道理来说，xy应该确定了一个真正的屏幕点，不需要范围了，其实不然，在开启吸附效应后，当鼠标从远处向某个屏幕点靠近，近到一定程度后，十字架的确会自动跳到屏幕点上，但相反，当鼠标渐渐向屏幕点远去的时候，也要远到一定程度，十字架才会从屏幕点上脱离，说得更通俗一点，开启吸附效应后，鼠标点（鼠标被隐藏了，所以看不到）与十字架的中心点可能不是重合的。比如我举个例子：

当十字架刚刚跳到屏幕点之上时，此时十字架与真正的鼠标位置（鼠标是隐藏的）是重合的，后来操作员轻微的动了下鼠标，此时鼠标位置偏离了十字架的中心，但由于仍然在吸附范围之内，所以十字架不会跟着鼠标一起移动，此时操作员点击鼠标，得到的鼠标坐标与十字架是不重合的，所以要找到一个在曲线上的点，与此时的鼠标坐标完全重合，是找不到的。这个问题让人想不明白的原因是，鼠标是隐藏的，十字架并不是鼠标，所以操作员看到十字架就在点上，但是得到的屏幕点（MouseDown事件）可能并不在十字架的位置上。

这个MaxRange最好是等于吸附范围，这个接口之所以没有直接使用吸附范围，而是让二次开发者传入一个范围值，是考虑到灵活性的要求。

本函数不用考虑Z坐标的影响，具体请看demo的OnMouseUpStcurvectrl函数里面，有通过鼠标点击删除点的例子（我注释掉了）。

### BOOL GetPixelPoint(DATE Time, float Value, long\* px, long\* py);

根据实际坐标得到客户区坐标（像素单位），定点缩放时有用。

### void EnableFullScreen(BOOL bEnable);

全屏控件，这里的全屏，意思是将控件的全部窗口用于绘制曲线，标题、坐标、图例这些统统隐藏。

### DATE GetEndTime();

### CString GetEndTime2();

### float GetEndValue();

获取当前坐标的结束值，其实可以通过GetZoom函数得到缩放比，再通过GetTimeSpan和GetValueStep函数跟一个公式就可以得到当前的坐标间隔，再通过GetScaleNums函数得到刻度个数，就可以算出来当前坐标的结束值来，只是显得麻烦，所以提供了这两个辅助函数。得到这两个值的用途比如可以删除某条曲线不在画布中的所有点，比如：

m\_ST\_Curve.DelRange2(1, .0, m\_ST\_Curve.GetBeginTime2() - .000000000000001, 2, FALSE, TRUE);

m\_ST\_Curve.DelRange2(1, m\_ST\_Curve.GetEndTime() + .000000000000001, .0, 1, FALSE, TRUE);

注意加减.000000000000001，这样可以防止将刚好在画布边沿的点删除掉（如果你希望他们被保留的话，从这里可以看出来，DelRange2函数在时间相等的时候，仍然执行删除）。

### void SetZLength(short ZLength);

### short GetZLength();

设置Z轴的长度（大于等于0），单位为网格的一个格子，一般两三个格子就好了。

有一点很重要，需要指出来，翻页时，Z轴会缩小一个页面的大小，也就是说，在计算翻页大小时，是按有网格的区域来的，只有在Z轴长度为0的时候，有网格的区域才与画布是重合的，否则小于画布的面积。具体将会影响ReportPageInfo、ExportImageFromPage、FirstPage、GotoPage等接口。

### void SetLeftBkColor(OLE\_COLOR Color);

### OLE\_COLOR GetLeftBkColor();

三维画布左边的壁的颜色，如果不想使用，可以将Color参数的低位起第31位置1。如果第32位为1，则弹出颜色选择框让用户选择，由于这是个方法而不是属性，所以不必像属性那样将当前的颜色放在Color的低三个字节里面，像下面这样调用即可：

m\_ST\_Curve.SetLeftBkColor(0x80000000);

如果31和32位同时为1，则31位有效。

### void SetBottomBkColor(OLE\_COLOR Color);

### OLE\_COLOR GetBottomBkColor();

三维画布下边的壁的颜色，参看上面的左壁颜色的说明部分。

### BOOL SetZOffset(long Id, short nOffset, BOOL bUpdate);

### long GetZOffset(long Id);

设置指定曲线在Z轴上的偏移量（屏幕坐标，大于等于0），一个格子（参看SetZLength函数）的长度为29.698484809834996（大家一定记住这个值，如果你需要刚好偏移到某个格子上的时候需要），控件将根据nOffset量，计算真正的偏移量——X分量和Y分量，然后在绘制曲线的时候，执行这个偏移。注意，这个偏移与OffSetCurve有本质区别，这是在Z轴上的偏移，当曲线画到画布之外的时候，不同的Z坐标，穿出画布的位置将不一样，具体看demo里面的最后一个演示方案。

GetZOffset函数得到的是X分量和Y分量（X占高16位，Y占低16位），而不是SetZOffset函数的nOffset参数，控件没有保存nOffset参数，所以无法取得。

下面介绍一下SetZOffset函数的另外一个用途，大家也许注意到了，控件无法隐藏指定的某一条曲线，而只能按图例来隐藏曲线，可是一个图例可以包含多条曲线，这样，一隐藏图例，所有这个图例包涵的曲线都被隐藏了。现在有了SetZOffset函数，可以将指定的曲线偏出Z轴的范围来达到隐藏某条曲线的目的，具体看demo里面的最后一个演示方案，它隐藏了12曲线。注意一点是，在通过nOffset计算X分量和Y分量时候，涉及到开方运算，最后还要取整，所以误差肯定会有的，要想隐藏曲线，nOffset要设置的大一点，比如当前的Z轴长度为1个格式，那么nOffset取30按理来说已经在Z轴之外了，但由于计算过程中的误差，可能结果仍然在画布之内，所以应该设置大一点，比如31、32什么的，或者干脆设置成0x7FFF。

但有一点需要注意，通过Z轴隐藏的曲线，控件并不当它是隐藏的（因为要防止这一点非常的麻烦，所以没有实现），它与普通曲线唯一的不同只是它不显示。

那么如何知道某条曲线被Z轴隐藏了呢，大家记住一个数字——21，假设Z轴长度为nZLength，那么当GetZOffset得到的X轴分量大于nZLength \* 21的时候，就被隐藏了（由于这个判断太简单，所以没有提供接口）。

从接口的签名不难看出，一条曲线只有一个Z坐标值，所以严格来说，控件所实现的三维效果并非真正的三维曲线（任何曲线永远只会在一个与XY平面平行的平面里面延伸，任意两条曲线不可能在Z轴上相交，除非他们处于同一个XY平面），这里的Z坐标更像是用来在曲线与曲线之间产生层次关系，而真正的三维曲线，每个坐标都是要有XYZ三个分量的。

曲线的Z坐标将影响到CalcActualPoint函数，具体请看开发文档。

### void EnableFocusState(BOOL bEnable);

当控件获得焦点时，是否绘制一个边框示意。

### BOOL SetReviseToolTip(short Type);

### short GetReviseToolTip();

控制曲线对于坐标提示的处理方式：

0－始终按Z坐标等于0来校正坐标提示，其实就是不校正（默认）

1－始终按选中曲线的Z坐标来校正坐标提示，如果没有选中曲线，则等效于0

2－如果选中的曲线在画布中，则等效于1，如果没有选中曲线，或者选中曲线不在画布中，则等效于0

3－只在曲线点上显示坐标提示，这个功能在三维显示状态下有用，如果没有使用三维效果，推荐设置为0。注：如果按3来处理坐标提示的话，一定要开启吸附效应，否则不会弹出坐标提示（哪怕鼠标刚好移动到了某个点之上，因为不吸附时根本不会判断鼠标点在不在曲线点之上，这是一个优化）。

### void LimitOnePage(BOOL bLimit);

限制曲线在一页之内，在绘制实时曲线时也有效。以同一值多次调用本接口，后面算为重复调用，控件不响应。

### BOOL FixCoor(DATE MinTime, DATE MaxTime,

### float MinValue, float MaxValue, short Mask);

### short GetFixCoor(DATE\* pMinTime, DATE\* pMaxTime,

### float\* pMinValue, float\* pMaxValue);

限制曲线的纵横坐标的开始结束值，Mask代表参数的有效性，次序以参数的次序为准，按位算，从低位起。GetFixCoor函数用于获取这些参数，不需要者传入一个空指针即可，返回的short数据，就是FixCoor的Mask参数。如果在调用FixCoor的时候，Mask的相位位置上为无效，而GetFixCoor的相应参数又提供了一个有效的地址，则控件仍然会给这个地址写入值，只是这个值可能是随机值或者是上上次调用FixCoor传入的值，所以有效性还是要按返回的Mask为准。

如果最大、最小值同时有效，则最大值必须大于最小值，所以FixCoor返回布尔型来代表设置的成败。

FixCoor这种坐标限制与LimitOnePage是互斥的，在设置了其中一种后，自动清除另外一种，如果两种都存在，则LimitOnePage有效。

不同于LimitOnePage，FixCoor接口在多次按完全一样的参数调用时，仍然会响应，所以FixCoor可以当成RefreshLimitedOrFixedCoor来用，只是每次要提供众多的参数而已，这样做是因为FixCoor接口参数众多，不方便判断是否是重复调用。

注意：在打印时，一定不要坐标限制！在限制一页时打印，效果不会达到限制一页，而是按原来的曲线位置来打印（由于打印窗口比屏幕大，所以理论上应该在打印时放大曲线，以达到限制一页的效果，但实际上控件并没有这么做）；在其它坐标限制模式下，可能让打印陷入死循环！

### BOOL RefreshLimitedOrFixedCoor();

按照LimitOnePage或者FixCoor函数设置的样子，强行让控件复位。

至于为什么要提供这个接口，因为就算调用过LimitOnePage或者FixCoor函数以后，画布的原点仍然是有可能被修改的，原点一修改，就必须要调用这个函数来复位，原点被修改可能有如下一些情况：

一：移动曲线（包括鼠标键盘移动、二次开发者调用接口移动，控件自动移动）；

二：纵坐标精度改变，从而引起画布大小改变；

三：图例长度改变，从而引起画布大小改变；

四：定点缩放。

注：对于因控件窗口大小的改变而引起的画布大小的改变不用关心。

现在问题的焦点是，二次开发者比较难于知道“控件自动移动曲线”是何时发生，建议二次开发者在界面上放一个按钮，或者设置一个热键，让操作者来控制对本接口的调用。

一般在如下这些情况，会引起“控件自动移动曲线”这个操作（假设曲线移动已被禁止，二次开发者也不会调用接口移动曲线，调用任何可控制曲线移动的接口，比如AddMainData2接口的VisibleState参数，都已经设置了不让曲线移动）：

一：删除曲线；

二：字体改变；

三：选中曲线（EnableSelectCurve接口可以防止曲线被选中，参看开发文档）。

二次开发者只要注意以上几条，在适当的地方调用一下RefreshLimitedOrFixedCoor接口，应该是可以始终控制曲线的坐标的。

在FixCoor函数的内部，其实仍然是调用SetBeginTime2等等函数的，所以也有可能会失败，二次开发者必须要保证MinTime、MaxTime、MinValue、MaxValue这些参数确定的区域里面存在曲线。

如果MinTime和MaxTime不全有效，则限制坐标采用的是移动曲线，对于MinValue和MaxValue同理。如果同时有效，限制坐标采用的是SetTimeSpan或者SetValueStep函数，此时一定要注意，控件原有的TimeSpan和ValueStep将丢失，如果二次开发者准备在以后的某个时候，恢复到限制坐标以前的状态，必须要自己记录当前的TimeSpan和ValueStep值，参看GetTimeSpan和GetValueStep接口，以便在再需要的时候恢复，参看SetTimeSpan和SetValueStep接口。

在缩放的时候，限制坐标可能会有误差，以TimeSpan为例，控件根据当前的TimeSpan（缩放后的TimeSpan），生成一个新的TimeSpan（以便让横坐标从MinTime开始，并且结束于MaxTime），再在这个新的TimeSpan上反向应用缩放，然后在调用SetTimeSpan函数，该函数再正向应用缩放，理论上结果是相等的，但在计算机的世界里，就可能不相等，比如：

1.0f / 0.4f \* 0.4f应该是不等于1.0f的，这就是浮点数误差。

### BOOL SetLimitOnePageMode(short Mode);

### short GetLimitOnePageMode();

限制一页模式：

0-永远充满一页（除非只有一个点，默认模式）；

1-当点绘制到画布之外时，把相应的坐标间隔增至原来的2倍，纵横坐标一样处理（如果只在某一个方向比如X方向上出了画布，则只会对该方向增加间隔），如果增加坐标间隔之后，点还是在画布之外，则继续增加，也就是相当于原来的4倍；

2-同上，只是坐标间隔增至原来的3倍；

n-同上，只是坐标间隔增至原来的n+1倍；

取值0-16，其余值保留以后扩展。

### void EnablePreview(BOOL bEnable);

显示或者隐藏全局位置预览窗口，这个窗口里面有一个小窗口，代表当前窗口在整个画布中的位置，可以在这个预览窗口里面点击鼠标左键，以达到快速移动曲线的目的。操作员可以在原点处的小矩形框内点击鼠标或者按快捷键F6来显示或者隐藏这个预览窗口，二次开发者则是通过此操作来操作。注，在窗口处于显示状态下时，如果在这个窗口里面点击鼠标右键，也会隐藏本窗口。

控件在默认情况下，会开启这个功能。

### void SetWaterMark(LPCTSTR pWaterMark);

设置水印内容，水印是一行文字，居中显示在背景中，颜色在前面说过了，和帮助的颜色一样，参看PrintCurve接口的说明文档的最后部分。

### long GetSysState();

获取Enable系列函数设置的状态，大家注意到Enable系列函数与Set/Get系列函数是不一样的，Enable系列函数无法获取当前状态，只能修改当前状态，这样设计的目的是为了减少接口数量，因为Enable系列函数都是开关量（只能取0或者1），非常简单，设计成Set/Get系列函数没有多大的必要性，但有些时候，难免还是会遇到需要获取这些开关量的时候，此时使用GetSysState接口就刚好满足要求了。返回值按位看，从低位起依次是：

EnableZoom、EnablePageChangeEvent（已废除，改为GetEventMask实现，恒为0）、EnableAdjustZOrder、EnableAutoTrimCoor、EnableHelpTip、EnableFullScreen、EnableFocusState、EnablePreview、EnableHZoom、EnableSelectCurve。

### void SetTension(float Tension);

### float GetTension();

绘制平滑曲线的张力，如果张力等于0，就是折线图了。这个张力是控制所有曲线的，至于绘制时按平滑曲线还是普通曲线是通过图例来控制的，参看AddLegend和AddLegendHelper函数。

如果大家不想显示平滑曲线，则一定要使用图例来控制，不要通过将本值设置为0来达到目的，因为如果图例里面的CurveMode等于3，不管张力等于多少，都将采用GDI+来绘制，这样不但是速度慢了，达到的效果和GDI还一样，吃力不讨好！

本值默认为0.5，这也是GDI+的默认值。

### long LoadPlugIn(LPCTSTR pFileName, short Type, long Mask);

加载插件，插件是一个动态库，通过pFileName指定其名字。Type是插件的类型，目前只支持1种插件，所以这个字段目前必须为1。Mask用于控制加载具体哪些函数，按位算，每一位代表一个插件接口，比如Mask等于2，则加载第二个接口，如果为3，则加载第一二两个接口。接口的序号根据在头文件ST\_Curve\_PlugIn.hpp里面的定义顺序而定，所以千万不要改动ST\_Curve\_PlugIn.hpp里面的函数签名及函数之间的顺序。

注意：控件通过LoadLibrary加载插件，只要是LoadLibrary成功，则上一次调用LoadPlugIn加载的插件将自动被FreeLibrary，比如上次加载插件装载了接口1，再次加载另外一个插件，想装载接口2并且保持接口1，这是办不到了。如果接口1和2都想要，必须放在同一个dll里面，且在一次调用LoadPlugIn里面同时添加这两个接口。

如果相应的接口加载成功，控件会置Mask的相应位为0，如果返回值等于传入的Mask的说，说明一个接口也没加载成功。

关于怎么使用插件，在ST\_Curve\_PlugIn.hpp里面有详细的说明，demo里面也做了一个简单的插件（带源代码），大家可以试一试。

大家千万注意不能修改ST\_Curve\_PlugIn.hpp里面的接口定义，因为控件是按函数的名字去加载（调用GetProcAddress）插件接口的，如果签名错了，加载的时候不会有问题，一但调用就会崩溃。

如果想卸载已加载的插件怎么办呢？用Mask等于0去调用这个接口，就是卸载（但只能卸载通过LoadPlugIn加载的接口）。

更多信息请参看本文档第五个大类：插件。

### long LoadLuaScript(LPCTSTR pFileName, short Type, long Mask);

这个接口与LoadPlugIn作用及用法一样，只不过是加载Lua脚本。在demo源代码里面，已经包含了一个写好的示例Lua脚本，里面有和ST\_Curve\_PlugIn.hpp里面申明类似的函数（其实就是相同的函数，只是Lua语法不一样，所以写法上就不一样），其签名是不能修改的（当然，非接口函数可以修改，比如GETSTEP这个函数，它仅仅是个辅助函数，不是必须的），至于函数里面的实现，二次开发者就尽情发挥吧（你也可以增加任意多的函数或者变量）；当然，他们的返回值还是有点要求，就是要是ST\_Curve\_PlugIn.hpp里面要求的返回值一样（由于Lua语言里面变量类型很弱，返回什么都没有语法错误，但可能到了控件里面就出问题了），如果返回多个值，则只取第一个（这是Lua的特性，说的有点偏了）。

注意，Lua脚本不像dll插件，它是允许从多个文件里面加载的（前面我们讲过，所有插件接口必须都从同一个dll插件里面加载），你可以从dll插件里面加载一个接口，再从Lua脚本里面加载另外一个接口（demo里面有示例），甚至从一个Lua脚本里面加载接口一，再从另一个Lua脚本里面加载接口二。

同样，卸载的时候，用Mask等于0去调用本接口（但只能卸载通过LoadLuaScript加载的接口）。

如果一前一后加载同一个接口，不管是从什么地方加载，都是后者覆盖前者。

脚本的好处显而易见，修改行为甚至连控件都不用重启，也不需要像dll那样重新编译，重新加载一下脚本即可。

### CString GetLuaVer();

返回控件里面使用的Lua解释器的，不排除将来会随着Lua的升级而升级，知道Lua的版本，对于二次开发者很重要，以免二次开发者写出这个版本不支持的Lua脚本。

### BOOL AppendLegendEx(LPCTSTR pSign,

### OLE\_COLOR BeginNodeColor, OLE\_COLOR EndNodeColor,

### OLE\_COLOR SelectedNodeColor, short NodeModeEx);

### BOOL GetLegendEx(LPCTSTR pSign,

### OLE\_COLOR\* pBeginNodeColor, OLE\_COLOR\* pEndNodeColor,

### OLE\_COLOR\* pSelectedNodeColor, short\* pNodeModeEx);

### BOOL GetLegendEx2(short nIndex, OLE\_COLOR\* pBeginNodeColor,

### OLE\_COLOR\* pEndNodeColor, OLE\_COLOR\* pSelectedNodeColor,

### short\* pNodeModeEx);

### long GetSelectedNodeIndex(long Id);

### BOOL SetSelectedNodeIndex(long Id, long NewNodeIndex);

起始、结束、选中点标识是通过在绘制节点的时候，采用不用的颜色来区分的，这样做的好处是当曲线点很多的时候（特别是绘制二次曲线的时候，起始点不一定在屏幕的最左边，结束点同样不一定在屏幕的最右边），仍然可以找到目前绘制到哪个位置了。

起始、结束、选中点标识属于图例范畴，所以需要提供一个图例标识，至于为什么不添加到AddLegend(Helper)，原因在S(G)etXYFormat里面说得很清楚了。

要想标识起始、结束、选中点，则相应的曲线的节点一定是处于显示状态的，否则调用本接口无效果。至于如何才能让曲线的节点显示出来，参看AddLegend函数。

NodeModeEx按位算，从低位起：1-BeginNodeColor是否有效；2-EndNodeColor是否有效；3-SelectedNodeColor是否有效。

关于选中点，还有一个操作方法，就是按住ctrl键再按方向键，这样选中点就会像火炬传递一样一个一个传递下去，大家可以实际操作一下。但问题是，操作哪一条曲线呢，答案是选中的曲线，如果没有选中曲线，则按住ctrl和不按ctrl效果是一样的，都是移动所有曲线。这个可以在我的demo里面做实际操作看看效果，操作方法是选中黄色曲线，然后操作（注意选中其它曲线则不行，因为图例不支持，大家看看其它曲线的图例就知道了）。

### void SetShortcutKeyMask(long ShortcutKey);

### long GetShortcutKeyMask();

开启禁止快捷键，按位算，从低位起，为1表示开启（注意：有些快捷键只能按组开启或者禁止，对于缩放键（-+），虽然其也是快捷键，但通过EnableZoom来开启与禁止）；想要绝对保持画面位置不变的朋友，需要禁止快捷键，因为快捷键也可能引起画面位置改变。

1 -F4

2 -F5

3 -F6

4 -F7

5 -home/page up/page down/end

6 -上下方向键

7 -左右方向键

8-16 -数字1键-9键

### OLE\_HANDLE GetFrceHDC()

获取控件内部核心变量——双缓冲使用的内存兼容DC句柄（HDC类型）。二次开发者可以通过它做一些非常极端的事，比如在自己控制打印的页面的某个地方，嵌入控件所绘制的曲线（当然是以图形的形式）；或者针对控件界面截图（此时直接从这个DC句柄里面BitBlt出来即可，相信大家都会吧）；或者通过GetCurrentObject函数从这个DC里面获取其它的GDI句柄，比如位图（位图只能选择到一个内存兼容DC里面，这是GDI的限制，所以二次开发者得到的位图句柄不能直接使用，但可以复制这个位图来使用）、画笔、画刷等；总之想怎么操作都可以，只要你能保证不出错，不过不推荐修改这个DC句柄，读取一般不会有什么问题。

### BOOL SetBottomSpace(short Space);

### short GetBottomSpace();

控件横坐标的显示位置（与控件窗口底部的距离），Space的单位为行，默认为2，即横坐标距离控件窗口底部为3（加上一行脚注）行距离。如果设置为1，则横坐标只显示为一行（如果有两行的话，后一行不显示），设置为0，则不显示。所以说，隐藏横坐标就有两种方式了（另外一种是显示模式，参看SetShowMode接口）。

至于说为什么要添加这个接口，是因为考虑到横坐标只显示一行的情况下（通过插件、Lua脚本、SetShowMode接口等，都可以让控件的横坐标只显示一行），坐标刻度值与控件窗口底部间距太远，浪费画布空间，且不美观。后来我干脆一不做二不休，让这个参数支持从0-127之间。

注：虽然本接口也可隐藏横坐标，但用SetShowMode效率会更高，所以推荐使用SetShowMode，本接口建议只是用来消除调用SetShowMode后，在横坐标刻度与控件窗口底部之间的间距。本来在调用SetShowMode后，控件是知道是否需要调用SetBottomSpace接口的，但考虑到还有插件和Lua脚本，所以干脆一并交给二次开发者去调用。

### Long GetEventMask();

### void SetEventMask(long Event);

开启或者关闭事件，Event按位算，从低位起以次是：

PageChange、BeginTimeChange、BeginValueChange、TimeSpanChange、

ValueStepChange、ZoomChange、SelectedCurveChange、LegendVisableChange、

SorptionChange、CurveStateChange、ZoomModeChange、HZoomChange。

所以，所有事件的定义为：#define ALL\_EVENT\_MASK 0xFFF，至于三个鼠标按下弹起移动事件，则不能开启或者关闭，那是MFC框架自己处理的。

### short AddComment(DATE Time, float Value, short Position, short nBkBitmap, short Width, short Height, OLE\_COLOR TransColor, LPCTSTR pComment, OLE\_COLOR TextColor, short XOffSet, short YOffSet, BOOL bUpdate);

### BOOL DelComment(long nIndex, BOOL bAll, BOOL bUpdate);

### long GetCommentNum();

### BOOL GetComment(long nIndex, DATE\* pTime, float\* pValue, short\* pPosition, short\* pBkBitmap,Short\* pWidth, short\* pHeight, OLE\_COLOR\* pTransColor, BSTR\* pComment, OLE\_COLOR\* pTextColor, short\* pXOffSet, short\* pYOffSet);

### short SetComment(long nIndex, DATE Time, float Value, short Position, short nBkBitmap, short Width, short Height, OLE\_COLOR TransColor, LPCTSTR pComment, OLE\_COLOR TextColor, short XOffSet, short YOffSet, short Mask, BOOL bUpdate);

以上注解相关，你可以用任意文字在画布的任意位置做注解，这也是应网友要求而增加的功能。

AddComment添加一个新的注解，SetComment修改一个已经存在的注解（nIndex从0开始）。

二次开发者需要提供一个坐标点用来确定显示位置（一般来说，二次开发者应该为注解提供一张位图，这样显得好看，位图设置方法跟背景位图一样）。Time和Value这个坐标就是用来确定位图的显示位置的（先姑且这样说，其实还需考虑Position参数，请往后看）。

Position指Time Value这个坐标的位置：

0-左上角 1-左下角 2-右上角 3-右下角 4-中心

这个参数难以理解，举个例子，假设Time Value这个坐标最终对应屏幕上的x y点，此时如果Position等于3，则得到矩形：x-Width y-Height x y；如果Position等于1，则得到矩形：x y-Height x+Width y。矩形的长为Widht，高为Height，如果他们等于0，则取位图的长和高（它们可以某一个为0，而另外一个不为0，也可同时为0，负数非法），如果既不提供Width和Height，也不提供位图，则Position当成0处理。得到这个矩形之后，位图从left top开始绘制（原始大小绘制，不拉伸压缩，为了效率），至于文字的绘制，则从left top偏移XOffSet YOffSet，偏移可正可负（单位为像素）。

注意，这里虽然我们得到一个矩形，但控件不会把注解限制在这个矩形之内，这个由二次开发者去保证，控件只是在规定的点绘制文字罢了（支持换行符，可任意多行）。

上面得到的这个矩形还有一个用处，在非默认坐标系之下，有了它，可以保证注解的绘制更到位（跟默认坐标系下表现一样）。

至于TransColor，控件可以对位图里面等于这个颜色的点做透明处理，要是没有这个，注解的背景就是个矩形，很难看，具体效果还是看demo吧。如果TransColor的最高位为1，则不做透明处理（控件内部使用TransparentBlt来做透明处理，某些打印机是不支持这个函数的，此时必须设置为不做透明处理，否则位图无法显示）。如果不想做透明处理，一定要设置该位，不要通过给TransColor指定一个在位图中不存在的点去达到不透明的效果，这样会损失效率。注意，如果图片的格式是png，则只有在默认坐标下，才会有透明效果，原因未知，如果图片是jpg的，则完全不能表现透明效果，bmp和gif则完美支持。

pComment当然就是注解的真正内容了（可以为空），长度限制在63个字符以内，TextColor则是注解的显示颜色。

SetComment返回的是Mask，在什么位置上为1，则说明相应的设置没有成功（跟以前众多接口一样），入参Mask指明了哪些位置需要修改（从Time以次开始），AddComment返回的值的意义与SetComment相同，你就理解为AddComment是用0x7FF去调用SetComment就好了。

但SetComment和AddCommnet有一点不同，对于SetComment，返回的Mask与入参Mask比较，凡是从1变成了0的位置，设置都是成功的（所以可能出现部分成功）；而对于AddComment，返回的Mask只要不等于0（有失败），则所有的设置都是失败的，即注解没有被添加。

至于GetComment里面的指针参数，如果不需要，就置空，这个在以前的接口里面都是这样的，控件会尽量保持风格，以方便大家理解，以后类似的功能在开发文档中可能就一笔带过了，以节省我编写文档的时间和二次开发者的阅读时间。

注解的绘制速度远远低于曲线，所以不要添加过多的注解。

由于我个人时间的原因，注解没有导出到文件或者从文件导入，但控件提供了读取注解功能，二次开发者可以轻松实现导出导入功能。

### BOOL SwapCommentIndex(long nIndex, long nOldIndex, BOOL bUpdate);

交换两个注解，用处不是很大，后面的注解由于最后绘制，所以会出现在最上层，如果两个注解相交叠，则后面的（nIndex大者）注解覆盖前面的注解。之所以说用处不大，是二次开发者不容易知道被覆盖的注解到底是谁（它的nIndex）。每次AddComment的时候，都向后添加（nIndex增大），不会对其排序。

### BOOL ShowComment(long nIndex, BOOL bShow, BOOL bUpdate);

### BOOL IsCommentVisiable(long nIndex);

显示或者隐藏注解。

### BOOL SetCommentPosition(short Position);

### short GetCommentPosition();

设置注解的位置，注解只有两个位置，0-置于所有曲线的最下面，1-置于所有曲线的最上面。

### BOOL SetFixedZoomMode(short ZoomMode);

### short GetFixedZoomMode();

### BOOL FixedZoom(short ZoomMode, short x, short y,

### BOOL bHoldMode);

定点缩放，ZoomMode参数只能取值字符'-'、'+'和数字0，代表缩小、放大和取消缩放状态（等同于点击鼠标右键），重复设置同一个状态，也认为是取消（等同于按键操作）。

重点说说FixedZoom函数，它用来真正的执行缩放（相当于在缩放状态下点击鼠标左键），它也带一个ZoomMode参数（不可为数字0），所以它也可以完成SetFixedZoomMode函数的工作，bHoldMode参数如果为真，则保持住ZoomMode状态，本函数的内部实现大致是：先判断当前缩放状态，如果状态与ZoomMode相同，则也认为是合法的，此时把bHoldMode置真（防止取消缩放状态，因为缩放状态是早就存在了的，不应取消，哪怕是调用者指定要取消），否则调用SetFixedZoomMode函数，接着执行定点缩放，最后根据bHoldMode决定是否取消ZoomMode状态。xy参数指定了定点缩放需要的点（客户区坐标，像素），如果你想按照DATE float这样的坐标去定点缩放，需要先转换成像素，参看GetPixelPoint函数。跟鼠标点击一样，xy确定的点必须在画布之内，但有一点突破，FixedZoom函数可以接受处于全局位置窗口（原点处小窗口）里面的点，而通过鼠标点击则不能。如果xy不在画布之内，则定点缩放失败，但ZoomMode的状态是设置成功的（因为SetFixedZoomMode调用在定点缩放之前），如果bHoldMode为真，控件仍将保持缩放状态。

### BOOL SetAutoRefresh(short TimeInterval, short NumInterval);

### long GetAutoRefresh();

自动刷新，考虑到很多开发者将本控件用于数据高速刷新的情况下，此时绘制实时曲线，会比较耗费CPU资源，比如每秒1000个点，很多人问我怎么办，我推荐的做法是：没必要每个点都刷新，10（或者你认为合适的数量）个点刷新一次即可，刷新太快了，就跟直接跳过去是一样的了（人眼是有延迟的）。可是，仍然有人不知道如何实现，或者是不想去实现，于是我添加了这两个接口，用于按时间间隔或者点个数间隔自动刷新。TimeInterval用于按时间间隔自动刷新，单位是0.1秒，当成无符号来看，所以最长间隔为6553.5秒（最短当然就是0.1秒了，用有符号表达无符号最大数，大家都知道吧，对了，就是-1，以后我不再重复这样的基础知识了）；NumInterval用于按点个数间隔自动刷新，它取值只能是2-32767，为什么不能为1呢，因为为1时，就和普通没有自动刷新时绘制实时曲线的效果一样了（效率上还会差于普通实时曲线）。

当TimeInterval和NumInterval都不为0时，后者有效，当两者都为0时，则取消自动刷新，否则取非0者为有效。GetAutoRefresh当然就是返回当前的自动刷新参数了，如果是按点个数自动刷新，则个数放在高二字节的低15位之内（第16位永远为0，其实就是TimeInterval左移16位的结果），如果是按时间间隔自动刷新，则这个刷新频率放在低二字节（就是TimeInterval）。

在自动刷新开启的情况下，二次开发者调用AddMainData(2)时，还是按照和原来的调用方式完全不变，控件内部会做处理，比如每两个点刷新一次，则两次调用AddMainData(2)时，仍然按普通实时曲线一样调用，控件内部会在每两次刷新请求时，只做一次。总之，刷新仍然只发生在VisibleState里面指定要刷新的时候（唯一的变化就是，以前指定了刷新就肯定要刷新，现在不一定了；如果采用记点个数自动刷新，当VisibleState没有指定要刷新时，不记数），这就带来一个问题，比如每两个点刷新一次，如果AddMainData(2)了3个点，那么第3个点就不会被刷新，因为它在等第4个点（好达到每两个点刷新一次的要求），怎么办呢，这个就需要二次开发者，在所有数据都添加完之后，手动调用一次Refresh，只要有数据不停的添加，就不需要调用Refresh，因为自动刷新的条件一定达得到。

### void EnableSelectCurve(BOOL bEnable);

是否允许曲线被选中，如果不允许，则不能通过点击图例或者按数字键来选中曲线（不用再禁止1-9数字键了），但二次开发者仍然可以通过SelectCurve来选中曲线（一贯风格了）。再重申一下，Enable系列接口，如果想获取当前状态但又没有找到对应的Get类似的接口的话，看看GetSysState接口，说不定就在其返回值的某个位中。

### void SetToolTipDelay(short Delay);

### short GetToolTipDelay();

设置坐标提示的延迟时间，就是当鼠标停下来之后，延迟多久开始显示坐标提示（如果有显示的话），这个数当成无符号处理，单位为毫秒，所以最长为65.535秒，默认是100毫秒；如果等于0，则不显示坐标提示。

### BOOL AddInfiniteCurve(long Id, DATE Time, float Value, short State, BOOL bUpdate);

### BOOL DelInfiniteCurve(long Id, BOOL bAll, BOOL bUpdate);

无限曲线，所谓无限曲线，其实是无限直线，什么叫无限直线呢，就是无论画布怎么拖动，它总是贯通画布。无限曲线分两类，一类是垂直贯通画布，另一类是水平贯通画布。无限直线也有一个Id，所以可以通过图例设置它的属性。

State分成两部分看，低字节取值0和1，0代表水平无限曲线，所以只有Value有效，1代表垂直无限曲线，所以只有Time有效。高字节只能设置低4位，否则失败，其取值与SetFillDirection接口里面的FillDirection一样，但只有低4位有效，所以要少些功能，具体请看SetFillDirection接口。

如果要添加的无限曲线已经存在，则修改它。

注意，最好为无限曲线增加单独的图例，因为：

一：无限直线一般来说，颜色上应该与普通曲线有所区别；

二：可能会有效率损失；

要达到最佳效率，无限曲线的图例请关闭如下图例基本属性（参看AddLegend接口）：

CurveMode = 0; NodeMode = 0;

和如下图例扩展属性（参看AppendLegendEx接口）：

BeginNodeColor = 0; EndNodeColor = 0; SelectedNodeColor = 0; NodeModeEx = 0;

和如下图例坐标标记属性（参看SetXYFormat接口）：

Format = 0;

具体的做法是：

若调用AddLegend接口，请指定CurveMode和NodeMode均为0；对于AppendLegendEx和SetXYFormat接口，你不调用它，默认就是上面的情况；另外还有一个简单的方法，如果不需要填充的话，调用AddLegendHelper添加出来的图例，就满足上面的要求。

无限曲线有什么用：

一：简单的添加一个值，就可以绘制一条直线；

二：可模拟固定坐标轴，比如在某个点绘制两条无限曲线，模拟坐标轴；

三：无限曲线支持填充，所以可以绘制区域，比如画一条60分，向下填充的无限曲线，那么一看就知道有多少人不及格了，同理，绘制一条90分的，向上填充的无限曲线，一看就知道有多少人成绩优秀了；

四：请发挥想像……

无限曲线特点：

一：它是直线；

二：绘制无限曲线的效率比普通曲线稍差；

三：所有无限曲线均绘制在所有普通曲线的下面；

四：无限曲线的Id可与普通曲线的Id重复；

五：无限曲线不参与计算所有曲线所占的空间大小，比如说有多少页的曲线，打印和按图片导出曲线时，无限曲线有效，导出曲线到图元文件时，无限曲线无效；

六：如果设置注解在最底层，则次序以次是：注解->无限曲线->普通曲线；如果设置注解在最上层，则次序以次是：无限曲线->普通曲线->注解。

### void SetMouseWheelMode(short Mode);

### short GetMouseWheelMode();

鼠标滚轮模式，下面介绍一下默认情况下，滚动鼠标时，控件的行为：

一：如果按住了shift，则执行缩放，相当于调用SetZoom接口；

二：如果按住了ctrl，则执行水平移动；

三：如果按住了alt，则执行水平缩放，相当于调用SetHZoom接口；

四：前面都不是时，则执行垂直移动；

如果你想修改这种行为，比如你经常用水平滚动而不是垂直滚动，又不想按住ctrl（这样至少要两只手操作），可以通过本接口来修改。

Mode只使用了低8位，从低位起，每两位为一个单位，共四个事件，分别代表：shift按下、alt按下、ctrl按下和不按键，顺序是写死的。另外还有四种行为，分别是：0-垂直移动、1-缩放、2-水平缩放、3-水平移动，通过修改相应位置上的值来修改控件的行为，举个例子，控件默认行为用上面的表达方式表达就是：

1 + (2 << 2) + (3 << 4) + (0 << 6)

其中1是第1，2位，代表按住了shift，其值为1，说明是执行缩放；(2 << 2)是第3，4位，代表按住了ctrl，其值为2，说明是执行水平移动，以次类推。

如果想把默认的垂直移动与水平移动交换一下，则是（交换相应位置上的值）：

1 + (2 << 2) + (0 << 4) + (3 << 6)，如果只想垂直移动曲线，则是：

1 + (2 << 2) + (0 << 4) + (0 << 6)

可以把所有事件都指向同一种行为，比如指定Mode为0，则所有行为都是垂直滚动，不管按住shift、ctrl还是alt都不会改变。

### BOOL SetMouseWheelSpeed(short Speed);

### short GetMouseWheelSpeed();

设置鼠标滚动速度（正常速度的倍数），取值0-255（0的意思是取消鼠标滚轮支持，这个功能也是之前没有的），速度太快会出现大步跳跃，而且可能出现滚动失败（因为全部滚出了画布，此时控件不会自动减小速度，因为实现太麻烦）。

### void Refresh();

刷新曲线，凡是调用了有bUpdate作为参数的函数（比如DragCurve），而又没有将该参数设置为真的话，如果想要马上得到更改效果，都需要调用本函数。如果调用了某些函数会更改曲线（比如InsertMainData2），但又不带bUpdate参数的话，如果想要马上得到更改效果，也需要调用本函数。

## 事件（15个）

### MouseDown(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y);

### MouseMove(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y);

### MouseUp(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y);

这三个函数的意义看名字即懂，如果你想在控件窗口上响应鼠标并弹出菜单什么的，你会爱上这些事件的。这些事件返回的坐标是客户区坐标，如果要弹出菜单什么的，需要自己转换成屏幕坐标，比如：

POINT point = {x, y};

m\_ST\_Curve.ClientToScreen(&point);

Button用于标识左中右键，自己试一下就知道了。

### void PageChange(long wParam, long lParam);

参数与发送PageChangeMSG消息时的wParam和lParam参数意义完全一样。

### void BeginTimeChange(DATE NewTime);

原点横坐标改变

### void BeginValueChange(float NewValue);

原点纵坐标改变

### void TimeSpanChange(double NewTimeSpan);

横坐标间隔改变

### void ValueStepChange(float NewValueStep);

纵坐标间隔改变

### void ZoomChange(short NewZoom);

缩放改变

### void SelectedCurveChange(long NewId);

选中曲线改变，当NewId等于0x7FFFFFFF时，说明取消了选中曲线

### void LegendVisableChange(long nIndex, short State);

图例可见状态改变，nIndex为序号，从0开始，State为1可见，为0不可见

### void SorptionChange(long Id, long nIndex, short State);

吸附点改变，State为1处于吸附状态，此时Id和nIndex共同表明了是哪条曲线的哪个点被吸附了；为0离开吸附状态，此时nIndex无意义（恒等于-1），Id意义和前面一样。

### void CurveStateChange(long Id, short State);

曲线状态改变，1-曲线被添加；2-曲线被删除；3-曲线被裁剪（曲线设置了长度限制并且超过了限制，此时将删除前面的一些点）。

注意：当曲线被复制的时候（参看CloneCurve接口），不会触发本事件，原因是二次开发者可以精确的知道这一行为（控件不会自己复制曲线）；当曲线Id被修改的时候（参看ChangeId接口），也不会触发本事件，原因同上。

### void ZoomModeChange(short NewMode);

定点缩放状态改变，NewMode取值和SetFixedZoomMode的ZoomMode参数完全一样。注意这个状态与(H)ZoomChange完全不同，后者指的是缩放率的改变。

### void HZoomChange(short NewZoom);

水平缩放改变。

### BatchExportImageChange(long FileNameIndex);

批量导出报告（关于批量导出，参看BatchExportImage接口），每导出一张图片，这个事件都会触发一次，参数就是导出的序号，比如导出格式是c:\\image\*\*\*.jpg，那么本事件将按照1、2、3、...、999、0这样的顺序触发（当然，如果中间某个序号对应的文件已经存在，则跳到下一个继续判断，也就不报告这个序号了，这在BatchExportImage接口里面已经解释过了）；注意最后报告的0很重要，这代表导出结束了（在批量导出的时候，就算二次开发者没有指定结束，控件也会在某种情况下自动结束，比如可用序号用完了等，正是存在这种情况，才添加了这个事件，以便通知到二次开发者，批量导出已经结束了）。

在demo中调试事件，只需要运行Debug(U)版本即可，此时会有一个控制台窗口弹出来，里面有各种事件发生的信息输出；在对话框任意空白位置双击右键，会清屏控制台窗口。

## 导出方法（3个）

这里所谓的导出方法，与上面说的方法不一样，上面的方法叫着接口，是需要创建COM组件后才能调用的，这里说的导出方法，和dll里的导出方法一样，不需要创建控件，此时就当成一个规则dll来使用即可，即可显示加载也可隐式加载（隐式加载时，需要一个头文件ST\_Curve.hpp以及ST\_Curve.Lib库文件）。

### extern "C" BOOL \_\_stdcall ExportImage(HBITMAP hBitmap, const unsigned short\* pFileName);

导出位图到文件，hBitmap为要导出的位图句柄；pFileName为文件名，如果为空指针或空字符串，则本控件弹出文件选择框让用户输入文件名及存放位置。

注：前面说控件的方法的时候，也有一个ExportImage函数，那里的ExportImage函数只能导出本控件显示的内容，其实里面的实现也是调用了这里的ExportImage函数。

同样，这个函数也是根据文件的扩展名来确定要导出文件的格式（bmp、png、jpg、gif）。

### extern "C" LPBITMAPINFO \_\_stdcall GetDIBFromDDB(HDC hDC, HBITMAP hBitmap);

从DDB位图获取DIB数据，类似于GetDIBits函数，hBitmap为位图句柄；hWnd为窗口句柄，本控件将获取这个窗口的DC，并以此为依据（比如颜色深度）生成DIB数据。有些时候生成DIB数据是有用的，比如调用StretchDIBits函数时。

注意：返回的LPBITMAPINFO一定要记得释放，格式如下：

LocalFree((HLOCAL) lpbi); //lpbi即为GetDIBFromDDB的返回值

具体用法举例：

DWORD dwPaletteSize = lpbi->bmiHeader.biBitCount > 8 ? 0 : sizeof(RGBQUAD) \* ((1 << lpbi->bmiHeader.biBitCount) - 1);

StretchDIBits(hPrintDC, 0, 0, WinWidth, WinHeight, 0, 0, WinWidth, WinHeight,

(LPBYTE) lpbi + sizeof(BITMAPINFO) + dwPaletteSize, (LPBITMAPINFO) lpbi, DIB\_RGB\_COLORS, SRCCOPY);

LocalFree((HLOCAL) lpbi);

# 插件

本控件支持两种插件，一种是用dll提供的，其签名方式放在了ST\_Curve\_PlugIn.hpp

里面（其中还有更详尽的每一个接口的说明，二次开发者应该把它当成开发文档来阅读），至于怎么使用请参看LoadPlugIn接口。二次开发者只要按要求在dll里面导出控件所需的方法（不一定要实现所有方法，这要看LoadPlugIn的加载要求），就可以加载插件了，注意函数除了签名必须符合要求外，名字也必须符合要求，因为控件是通过GetProcAddress去获取函数地址的。具体参看TestST\_Curve2工作空间里面的ST\_Curve\_PlugIn工程。

第二种插件是Lua脚本，请参看LoadLuaScript接口。在TestST\_Curve2工程里面，也有示例，大家可以参看一下。关于Lua脚本，要求和dll一样，除了必须的接口之外，其它完全可以由二次开发者尽情发挥。

目前，控件里面的Lua解释器版本为5.2.1，二次开发者不应该写出这个版本不支持的代码。更多Lua信息，请访问www.lua.org网站。

# 关于内存使用量

每条曲线都有一个头结构，考虑到曲线的条数不会很多（画几百条曲线是没办法观看的，因为无法设置那么多的图例，而且曲线几乎都叠在一起了），这些头结构所占内存可以认为是固定的；还有一种结构就是图例结构，图例就更少了，二三十个不得了了，所以也认为是固定的。而占内存线性增长的，是点的多少，每个点占用24个字节（默认对齐方式），这24个字节是与点的数量成正比的，所以要计算内存的使用情况，就计算点数量即可。如果可用内存太小，就要通过SetMaxLength来限制曲线的长度。

# 致初学者或是对编程不太精通者

1.注册ST\_Curve：

将ST\_Curve.ocx拷贝到一个你认为合适的目录，比如C:\Windows，然后打开“运行”对话框（从桌面上的开始菜单里进入），输入regsvr32 C:\Windows\ST\_Curve.ocx，其中C:\Windows根据实际情况而定，点运行，控件即完成了安装。或者可以直接从官方主页下载安装包来安装，这样最省事。

2.添加控件到工程：

在工程中选择Project->Add To Project->Components and Controls，然后打开Registered ActiveX Controls文件夹，再选择ST\_Curve Control，再选择Insert。这样VC将为ST\_Curve控件生成一个包装类CST\_Curve，这个包装类提供了对上面的属性及方法的调用。

3.关于数据类型：

通过第二步操作后，在生成的包装类中，数据类型有所变化，具体表现为：

OLE\_COLOR、OLE\_HANDLE等全部变成了（unsigned）long型(本文档里面，有些地方使用OLE\_HANDLE，有些地方使用long，其实他们并不区别)，这是因为COM中的数据类型比较少（比如COM中没有int数据类型），考虑到跨语言要求，其实并不影响使用，强制转换一下即可（仅在出现警告或编译错误的时候需要），比如:

SetMaxLength(100L, 50L); //100本来是int型，这里强制转换成了long型。

另外，所有的属性也被包装成了类似方法的函数调用，ClassWizard就是这样处理的，我也无能为力，比如ForeColor属性将被包装成：

unsigned long GetForeColor();和void SetForeColor(unsigned long);

4.关于字体句柄：

初学者一定注意了，如果你用API创建字体，则一定不要调用::DeleteObject，本控件会负责释放。如果你使用CFont类来创建字体，则一定要使用CFont的Detach函数，而不是使用CFont的HFONT操作符，举例如下（本例用通用字体选择框来选择字体，其中m\_ST\_Curve为控件包装类的一个实例）：

方法一，用API：

CFontDialog dlg;

if (IDOK == dlg.DoModal())

{

LOGFONT l;

dlg.GetCurrentFont(&l);

HFONT h = ::CreateFontIndirect(&l);

m\_ST\_Curve.SetFont((long) h); //设置字体

// ::DeleteObject(h); //这行一定不能要

}

原因是控件需要多次使用字体h。

方法二，用MFC：

CFontDialog dlg;

if (IDOK == dlg.DoModal())

{

LOGFONT l;

dlg.GetCurrentFont(&l);

CFont font;

font.CreateFontIndirect(&l);

m\_ST\_Curve.SetFont((long) font.Detach()); //应该这样设置字体

//m\_ST\_Curve.SetFont((long) (HFONT) font); //不可这样设置字体

}

原因是CFont在被析构时，将调用DeleteObject函数，其实和方法一的道理一样。

同样的问题也会出现在添加位图句柄上（AddBitmapHandle），原则一样，即如果用API创建句柄，则不能调用DeleteObject函数，如果使用MFC，则要用Detach函数。

# 小窍门及FAQ

1. 绘制实时曲线的时候，如果被绘制的曲线在当前页中处于最上层（即不会被任何曲线覆盖），则绘制过程将会被大大的优化。至于如何让曲线显示在最上层呢，可以参看SetCurveIndex接口。当然，也不一定非要在最上层，只要是在当前页里面是最上层即可，比如有两条曲线，被绘制实时曲线的，在当前页里面有显示，而另外一条没有，则无论如何，被绘制实现曲线的曲线将被认为在最上层，所以绘制优化生效（绘制平滑曲线时无效）；
2. 通过按住ctrl再加上方向键这种方式来移动曲线的选中点的时候，情况和上面第1点完全一样。好好利用这些窍门，可以让绘制更节省CPU资源（绘制平滑曲线时也有效）；
3. 记不住快捷键，或者是鼠标与键盘的组合操作怎么办：记住F4，按F4会显示一些最常用的快捷键和鼠标与键盘的组合键，再按一次消失；
4. 在添加曲线非常频繁的时候，可以先把数据缓存起来，然后通过AddMemMainData调用一次行传递给控件。但是如果是绘制实时曲线，则不建议缓存过多的数据，否则会让使用者感觉不实时。比如每秒50个数据，可以每0.2秒调用一次AddMemMainData，每次传递10个数据，这样绘制效率和实时性都兼顾到了；
5. 好些朋友问为什么一个图例里面有多条曲线，我把曲线大部份设置都放在了图例里面，但又有一些设置是关联于曲线的，比如SetFillDirection接口。换句话说，两条曲线同属于一个图例，有相同的图例属性，但还允许有自己的私有属性，虽然目前除了FillDirection之外还没有其它的私有属性，但这种结构为将来的扩展提供了支持。如果不喜欢一个图例包涵多条曲线，你可以为每一个曲线添加一个图例；
6. 关于一些没有提供接口的界面元素的修改，比如“图例”“时间”这些文字，是没有接口可修改的，想修改的朋友，可以去修改ST\_Curve.ocx的资源，本身我在写代码时，所有文字都没有硬编码，这些文字要么来自于接口，要么来自于资源文件。关于如何修改ST\_Curve.ocx的资源，推荐使用ExtraRes软件，在VC7及其以后的版本中，也支持对资源的修改，操作方法是直接把ST\_Curve.ocx拖到VC窗口即可。

8.如何获取到一个没有使用的Id，这在复制临时曲线时非常有用，控件没有提供接口给二次开发者调用以便生成一个临时曲线Id，但二次开发者可以用下面一段很简单的代码来自己获取，之所以控件本身没有提供，太简单也是原因之一：

long TempId = 0x7FFFFFFF; //最大正值

While (IsCurve(++TempId)); //从0x10000000（最小负数）开始轮询

到这里后（退出了while循环），TempId即为所求。当然，从哪里开始轮询并不重要，我上面只是举了个例子而已。注：最好不要使用0x7fffffff这个Id，这在插件里面可能需要使用。

1. 如何知道控件中有没有处于显示状态下的曲线，注意，GetCurveCount接口返回所有曲线的条数，包括隐藏的和显示的。

if (-1 != m\_ST\_Curve.ReportPageInfo())

; //有处于显示状态下的曲线

这里我多说几句，有些功能的接口，表面上控件并没有提供，其实是可以通过其它相关接口去实现的，这也是为了尽量让控件少一些接口，请相信，效率上肯定是没有影响的，否则我一定会增加专门的接口去处理，我一直是效率优先的原则。类似的还有比如删除一条曲线，这个接口是没有的，但可以通过删除曲线点（DelRange2）去做到。

10.如何从一个DATE数据转换成一个表示日期时间的字符串。

如果能使用COleDateTime类，则:

DATE Time = ...; //被转换的值

COleDateTime OleTime(Time);

CString str = OleTime.Format(); //从DATE转换为字符串完成

如果不能使用COleDateTime类，则：

VarBstrFromDate

1. 如何从一个表示日期时间的字符串转换成一个DATE数据。

如果能使用COleDateTime类，则：

CString str = ...; //被转换的字符串

COleDateTime OleTime;

OleTime.ParseDateTime(str);

DATE Time = OleTime; //从字符串转换为DATE完成

如果不能使用COleDateTime类，则：

VarDateFromStr

1. 如何阅读demo源代码

大家已经看到过demo的界面了，上面摆满了按钮，可就是这样，也还有好多的控件接口没有测试到（因为按钮不够），在做新功能的时候，为了调试，我经常会随便找一个按钮来测试（因为已经没地方添加新按钮了），测试完以后，有时会注释掉新的代码，以回到原来的样子，有时候就忘记了。所以可能会出现按钮上的文字与点下按钮实际做的事情并不一样的情况，推荐大家还是以代码为准，按钮上的文字只是辅助。另外，光从界面上去找的话，可能会发现，有些功能好像根本没有演示，我推荐的做法是，去源代码里面查找接口的调用，因为前面我已经说过了，有时我会注释掉已经测试通过的接口，这些接口的调用，虽然注释掉了，但调用方法还是在的，在二次开发者不清楚怎么用的情况下，也是一个好的帮助。另外，也可以直接问我。

1. 如何编译demo源代码

demo源代码是在vc6下面写的，如果你在vc6中编译，却发现连接的时候出错，很有可能是字符版本的问题（错误基本上是找不到mfc42ud.lib），此时说明你没有完整的安装vc6，解决办法有二，一是切换到release或者debug版本再编译，二是完全安装vc6；原因是万恶的vc6默认不安装unicode版本库。如果你想在高于vc6的环境下编译，则打开时会询问是否转换工程格式，选择是，打开之后，就可以编译了（vc6之后的版本，都会默认的安装unicode版本库）。

demo共有四个版本，Win32 Release、Win32 Debug、Win32 ReleaseU、Win32 DebugU，

它们分别包括了ansi和unicode版本的release和debug版本。

1. 对于带bUpdate参数的接口，什么情况下传入TRUE，什么时候传入FALSE

如果你要调用一连串的，都带有bUpdate参数的接口，则可以全部以FALSE调用，最后Refresh一下即可，如果你就调用一个（总之是很少，比如两个）带有bUpdate参数的接口，则最好用TRUE调用，因为用FALSE调用可能反而影响效率，原因我解释如下：

当控件收到FALSE时，会在内部置一个标志，代表需要刷新，至于刷新什么地方，不会记（要记的话，可能太多，需要用动态数组），那么下次调用Refresh刷新的时候，不得不完全刷新整个控件界面，如果按TRUE调用，则可以精确的知道刷新哪些区域（因为它每次都马上刷新，不需要记录刷新区域），所以用FALSE调用，只适合于连续调用很多次的情况，这一点希望大家理解。

1. 什么是曲线的Id

它仅仅用于区分不同曲线而已。

16.已知问题

在像vb和.net这样的高级语言里面，所有控件接口里面的DATE类型（idl里面描述

为VT\_DATA），将被译成DateTime对象（换句话说，在vc里面，仍然还是DATE类型，DATE就是double，它们完全等效），这是一个时间对象，它与DATE（也就是double类型）虽然可以相互转换（通过FromOADate和ToOADate），但也有其缺陷，就是所表达的范围不一样，DateTime对象的范围只能是-657434.0至2958465.0（这个信息来源于MFC的COleDateTime类的开发文档，vb和.net里面，我不确定是否还是这个范围，但我相信它应该仍然还是double范围的子集），而double的范围就是c++标准的范围，好像是 -1.79E+308 至 1.79E+308。这就带来一个问题，如果我让控件横坐标显示为数字，此时理论上，横坐标的范围就是double的范围了，其实不然，因为必须得构造一个DateTime对象，而DateTime应该是有范围限制的，大家可以给FromOADate传一个不在-657434.0和2958465.0之间的值，看看能否成功，我估计是失败。如果你真的需要横坐标的范围超出DateTime的范围，那么请像我索取double版本控件（已经在2.2.0.2版本里面放出），我将把所有控件接口里面的DATE类型，修改为double类型。在double版本下，如果你的横坐标数据是DateTime怎么办呢？此时可以调用其ToOADate把DateTime转成double，换句话，double版本的控件是全能的，在什么情况下都能用。

# 版权

ST\_Curve (C)2008-2017 young wolf. All rights reserved.

Home: http://www.st-curve.cn

Email: [mail2tao@163.com](mailto:mail2tao@163.com)

QQ: 676218192 QQ群号：132339384

Mobile: 15520720170（四川成都）

本控件可自由传播，但必须要保证所有文件（包括但不限于二进制文件、开发文档等）的完整性。目前最新版本：2.2.0.7。

# 更新记录（从2010年开始记录）

1/17/2010

增加了按位图打印功能，用于解决平滑曲线的打印问题，当然这不可避免的带来了打印效果相对粗糙，具体请看PrintCurve函数。建议把这个函数的开发文档再从头到尾的读一遍，因为不但增加了位图打印功能，还增加了一个错误返回值；

升级控件版本到1.3.2.3。

3/14/2010

增加了插件支持功能，目前只支持1类型插件，关于这种类型的插件能定制什么，参看LoadPlugIn接口，大致是一些坐标显示的定制问题；

修改了一个BUG，此BUG会造成在显示坐标点值的时候，如果选择了不显示单位，并且横坐标是按值显示的时候，显示错乱；

此次更新没有版本的更新。

3/21/2010

修改了一个BUG，此BUG会造成在绘制平滑曲线的时候内存泄漏；

增加了起始结束点标识功能，二次开发者可以通过不同的颜色来标识曲线头和曲线尾，以便于观看曲线；

修改了横坐标每个刻度之间的距离（像素），这样方便二次开发者对调XY轴而曲线不会变形（只产生旋转变形）。这一修改会影响到S(G)etZOffset接口，请大家回去再看看这两个接口的开发文档（我已经更新了它们的开发文档）；

修改了在不允许改变曲线层次结构（参看EnableAdjustZOrder接口）的状态下，左键点击图例时控件的行为：和允许改变曲线层次结构的处理一样（仍然选中了曲线，之前是不能选中曲线），唯一不同之处是不把选中曲线提到所有曲线的最前面；

升级控件版本到2.0.0.0。

4/30/2010

修改了AppendLegendEx GetLegendEx GetLegendEx2接口，这里向大家道歉，老用户如果想使用新版本控件，不得不重新编译老的工程（如果没使用上面三个接口则不需要）；

为网格增加了实线风格，参看SetGridMode接口（与老版本控件接口向下兼容）；

优化了实时曲线的绘制，如何利用这些优化机制，请参看“小窍门”一节；

增加了选中曲线点功能，你可以通过ctrl加方向键去移动曲线的选中点，这样在曲线很多很复杂的时候，可以用来看曲线是如何走向的，参看SetSelectedNodeIndex接口；

增加了“小窍门”一节，大家以后多多关注这一节，它将告诉你如何才能更好的使用本控件；

升级控件版本到2.0.0.1。

5/2/2010

修改了一个BUG，此BUG会造成在全屏显示状态下，曲线位置错误；

修改了一个BUG，此BUG会造成在限制坐标状态下，切换到全屏时，被限制的坐标没有更新（比如限制一页，在切换到全屏时，曲线并没有放大以适应新的画布大小）；

增加了F7快捷键用于全屏的切换；

升级控件版本到2.0.0.2。

5/4/2010

为坐标限制相关接口增加了更为详细的说明及使用注意事项，参看FixCoor等接口；

此次更新没有版本的更新。

5/11/2010

为控件增加了Lua脚本支持，恳请大家仔细阅读一下LoadLuaScript接口；

增加了开启禁止快捷键功能，参看S(G)etShortcutKeyMask接口；

修改了一个BUG，此BUG会造成在通过ctrl键加方向键移动选中点时，无法刷新画布以外的区域，比如坐标轴等，造成界面混乱；

修改了一个BUG，此BUG可能会造成在动态创建控件的时候，如果窗口大小指定了一个空矩形（创建时指定空矩形，以后随着容器窗口的变化而变化，此现象在多文档里面容易出现），则会出现图例显示不完整（自动计算图例占用宽度出错）；

扩展坐标刻度个数到65535个，原来仅127个，当屏幕分辨达到两千七八的时候，控件内部保存的刻度个数将溢出，绘制刻度时会产生崩溃；

升级控件版本到2.0.1.0。

6/1/2010

修改了一个BUG，此BUG会造成在画布大小改变时（比如全屏、控件窗口改变等），没有报告页面信息，参看PageChangeMSG属性及EnablePageChangeEvent方法；

实时曲线绘制优化（以前的绘制在有移动曲线的时候存在冗余）；

控件内部的某些实现做了一些小的优化；

完善了联动相关接口的开发文档，因为有使用者反应写的不清楚；

升级控件版本到2.0.1.1。

6/20/2010

暴露了控件内部核心变量——双缓冲使用的内存兼容DC句柄，有了这个句柄，二次开发者可以灵活的做很多事情。当然，使用不当，危险也是并存的，具体参看GetFrceHDC接口开发文档；

修改了插件接口签名，增加了一个参数——曲线Id，这在多坐标系坐标提示功能方面非常有用，一般情况下不需要使用，具体参看我的demo里面的头文件定义，以及Lua脚本里面的例子；

为“小窍门”一节增添了一些内容；

升级控件版本到2.0.1.2。

7/3/2010

修改了一个BUG，此BUG会造成在设计模式下，复制的控件刷新有问题，大家可以这样重现这个BUG，打开vc，新建一个对话框，在对话框上拖一个控件，然后选中这个控件，按ctrl+c复制，再按ctrl+v粘贴，此时被复制出来的控件的刷新将出现问题，成了几乎一块黑屏，当然这并不影响程序的运行，甚至也不影响在vc中的开发，在ide中关闭对话框，再打开，一切恢复正常；

此次更新没有版本的更新，问题很小，大家更不更新都可以。

7/4/2010

为坐标提示增加了画布自适应功能，当提示窗口无法完整的显示的时候（比如提示点在画布边沿），控件将自动移动提示窗口的位置，以求完全显示出来；

此次更新没有版本的更新。

7/18/2010

修改了一个BUG，此BUG会造成在横坐标刻度值显示超出坐标范围（坐标范围从原点到箭头结束）时，屏幕上可能留下脏数据（刷新不完全）；

此次更新没有版本的更新。

7/25/2010

修改了一个BUG，此BUG会造成在修改控件字体后，曲线偏离原来的位置；

增加了两个接口用于控制横坐标刻度值到控件窗口底部的距离，以便节省画布空间，具体参看S(G)etBottomSpace接口；

升级控件版本到2.0.1.3。

8/8/2010

将所有表示时间的double改成了DATE，老工程可直接替换控件，不需要重新编译；

做了一些内部优化，主要是字符串转换方面；

考虑完整性，增加了GetEndTime2和GetTimeData2接口，用于按字符串返回时间；

修改了修整坐标相关功能，参看TrimCoor接口；

为插件（包括Lua脚本）增加了一个Action：8，用于自定义修整坐标；

完善了TrimCoor接口的开发文档，完善了本文档的第五大类（插件）的开发文档，在小窍门一节增加了一些内容（关于DATE与字符串之间的转换问题）；

此次更新没有版本的更新。

8/15/2010

修改了一个BUG，此BUG会造成在删除位图时（RemoveBitmapHandle），如果被删除的位图被选入了图例（用于填充曲线），则图例不刷新（曲线会刷新）；

为控件增加了注解功能，参看AddComment等接口；

在小窍门里面增加了了一条：如何阅读demo源代码；

升级控件版本到2.0.1.4。

8/21/2010

修改了注解相关的接口签名，因为功能性的增加（比如显示隐藏注解，调整注解显示顺序等），这里向已经在使用注解功能的朋友道歉，对于你们的老工程，不得不重新添加控件到工程并且重新编译，不能简单的替换控件；

升级控件版本到2.0.1.5。

8/26/2010

修复了一批BUG，这些BUG的共同表现是，凡是调用带有DATE\*类型参数的接口，都会失败，这是8/8/2010那次更新遗留下来的BUG；

优化了设置基点；

此次更新没有版本的更新。

8/29/2010

修复了SetCurveIndex接口的一个BUG：没有对入参nIndex的合法性做判断；

删除了EnablePageChangeEvent接口（其它接口位置不变，如果老工程中没有使用这个接口，则不需要重新编译），该接口的功能转为由SetEventMask接口来实现；

增加了9个事件（加上原来的三个，一个为自定义事件，两个系统预定义事件，共12个事件），具体参看SetEventMask接口；

由于EnablePageChangeEvent接口被删除，GetSysState接口的返回值有所修改，这部分文档已经更新；

为曲线枚举系列接口完善了说明文档（增加了一些使用上的注意事项），具体参看GetDataHeadPosition接口；

升级控件版本到2.0.1.6。

8/31/2010

修复了注解显示位置上的一个小BUG，此BUG发生需要如下条件：

1.未使用背景位图；

2.未指定矩形大小；

3.指定了偏移；

4.采用了非默认坐标系。

BUG表现为，没有执行偏移；

此次更新没有版本的更新。

9/12/2010

优化了按住ctrl或者shift控制鼠标只在水平或者垂直上移动功能；

修复了一个小BUG，此BUG表现为，在吸附成功之后，按住ctrl或者shift移动鼠标，鼠标仍然可能在不应该的方向（比如按住ctrl移动，垂直方向就是不应该的方向）上移动几个像素；

增加了从DATE float坐标转换到屏幕坐标的接口，参看GetPixelPoint接口；

增加了对注解位置的控制，参看S(G)etCommentPosition接口；

增加了定点缩放功能，参看S(G)etFixedZoomMode、FixedZoom接口；

升级控件版本到2.0.1.7。

9/21/2010

修复了一个BUG，此BUG表现为，在通过SetFixedZoomMode接口做定点缩放的时候，如果当前已经是想要的缩放状态时，缩放不会成功；

修改了定点缩放接口在某些情况下的行为表现，具体参看SetFixedZoomMode接口，已经对这个接口的开发文档进行了更新；

增加了一个事件：ZoomModeChange，用于在缩放状态改变时，通知容器；

此次更新没有版本的更新。

9/24/2010

修复了一个BUG，此BUG表现为，当以一个当前状态相同的状态调用SetFixedZoomMode接口时（以达到取消缩放状态的目的），ZoomModeChange事件报告错误；

修复了一个BUG，此BUG表现为，当用户通过鼠标右键取消缩放状态时，不会报告ZoomModeChange事件；

修复了控件内部一些小BUG，这些BUG不会表现给用户，但会做一些不必要的计算；

修改了控件的如下行为：

在缩放状态下删除或者隐藏掉所有曲线时，以前的操作是删除掉缩放状态，现在改为保留（但鼠标仍然会变成标准的指针形状），当有曲线被添加或者回到显示状态时，马上会得到以前的缩放状态（同时鼠标形状也会变成缩放的样子）；

在没有任何可见曲线的情况下（比如根本就没有曲线或者所有曲线都是隐藏的），不能通过键盘操作设置缩放状态（这一点保持不变），但二次开发者可以通过SetFixedZoomMode接口去设置缩放状态（此时鼠标不会表现出缩放状态，这也是为什么在这种情况下不支持用户键盘操作的原因，因为操作没有表面上的反应，这会让操作者感到沮丧），如果设置成功，控件内部将保持住这个状态，一但有曲线被添加或者回到显示状态时，将马上得到之前设置的缩放状态；

此次更新没有版本的更新。

10/6/2010

修改了SetReviseToolTip接口的开发文档（以前的表述有些问题），请大家回去再看看这个接口的表述；

修改了内部默认的校正坐标提示模式为0（参看SetReviseToolTip接口），原来默认为3，这（再加上SetReviseToolTip接口表述有些问题）导致了大家几乎没有使用过坐标提示功能，还以为控件没有此功能；

优化了绘制实时曲线的时候的一些不必要的移动；

增加了绘制实时曲线的两个功能——在X方向上最少移动和在Y方向上最少移动（这也是应某些网友要求而增加的功能，达到的效果类似于Windows任务管理器的性能栏的CPU使用记录），具体参看AddMainData(2)接口，由于是通过VisibleState参数去扩展这个功能的，所以向下兼容，老工程只需要直接替换控件（如果你不使用新功能的话）；

升级控件版本到2.0.1.8。

11/6/2010

完善了CurveStateChange事件的说明文档；

完善了缓存管理相关接口（ClearTempBuff、PreMallocMem、GetMemSize）的说明文档；

增加了缓存管理接口：GetMemInfo，用于获取更多详细的控件内部的缓存使用情况和监视控件内存使用量，以及帮助二次开发者检测内存泄漏（比如重复添加数据，表面上看曲线的外观没有变化，实际上点数量多了）等；

增加了一个辅助接口：IsCurveClsed，用于查检曲线是否是封闭的（注：这里的封闭是指屏幕坐标相等，实际坐标不一定相等）；

增加了一个用于枚举曲线时，获取屏幕坐标的接口：GetPosData，具体参看开发文档；

修复了一个BUG，此BUG表现为，当坐标提示框处于显示状态时移动曲线（比如滚动鼠标），坐标提示框消失，并且不再显示（除非移动一下鼠标）；

升级控件版本到2.0.1.9。

11/28/2010

修复了一个BUG，此BUG会造成当控件获得焦点时，按alt键，再移动鼠标，再按一下鼠标左键，曲线出现跳动；

修复了一个BUG，此BUG会造成当控件的MoveMode（参看SetMoveMode接口）指定限制垂直移动时，按住ctrl键（水平移动）加滚轮无法执行水平移动；

修复了SetMoveMode接口开发文档中的错误说明；

完善了SetEventMask接口的说明文档；

修改了控件对鼠标滚动的响应行为，判断的次序为（优先级）：shift键（缩放）、alt键（水平缩放）、ctrl键（水平移动）、最后执行垂直移动。如果前一个条件判断失败（条件为假）或者执行失败，都接着判断下一个条件（以前如果是执行失败，就直接返回了）。举个例子就明白了，虽然你按着shift滚动鼠标滚轮，可能仍然执行的是垂直移动曲线（而非缩放，因为缩放执行失败）；

引入了一个新的概念——水平缩放，即只在水平方向上做的缩放，为此新增加了三个接口：EnableHZoom、SetHZoom、GetHZoom，用于控制这个特性，为此GetSysState接口将受到影响，不过仍向前兼容，具体参看开发文档；

新增加了一个事件——HZoomChange，这是由于引入水平缩放功能而附带的，为此SetEventMask接口将受到影响，不过仍向前兼容，具体参看开发文档；

在2.0.1.9的基础上，对一次曲线做了一些小优化，仍然是尽量减少绘制点的数量；

对控件的绘制做了优化（系统层面上的），一是设置了“未剪辑的设备上下文”，二是设置了“无闪烁激活”，这些优化以前都没引起我的注意。关于更多的这方面资源，参看MSDN的文章——MFC控件优化；

升级控件版本到2.1.0.0。

12/1/2010

修复了一个BUG，此BUG会造成当控件未获得焦点时，把鼠标移动到控件画布之内，按下alt键，再移动鼠标，再按一下鼠标左键，曲线出现跳动；

修改了在开启水平缩放时，按下alt键的行为，参看EnableHZoom接口（有更新）；

增加了帮助信息（按F4显示）内容（水平缩放快捷键）；

本次更新没有版本更新。

1/1/2011

为插件增加了另外四个接口，具体参看ST\_Curve\_PlugIn.hpp，大家应该把它也当成开发文档来阅读；

将插件中修整坐标接口从FormatXCoordinate和FormatYCoordinate中分离了出来，所以以上两个接口中的Action等于8的情况将不再存在，如果已有工程中已经使用了，则不得不重新编译你的插件（增加相应的接口）和工程（因为这影响到了LoadPlugIn和LoadLuaScript的Mask参数），这里向二次开发者道歉；

为插件增加了自定义缩放功能，二次开发者可以轻松的通过插件控制控件的缩放速度、缩放加速度等，或者让缩放完全失效，什么都可以实现，只要你想得到；

增加了一个事件BatchExportImageChange，用于在批量导出图片过程中，把当前导出序号或者导出自动结束了等信息通知到二次开发者；

升级控件版本到2.1.0.1。

3/17/2011

增加了MouseMove事件，具体参看“事件”一栏；

修改了AddMainData和InsertMainData的pTime参数的解析方式，参看AddMainData接口；

增加了新的“帮我改进控件”栏，专门用于讨论在vb和c#这样的高级语言下使用本控件所遇到的调用问题（比如如何调用带指针的接口）以及解决办法，我本人在vb和c#方面非常的不给力，几乎没用过，开辟出这一栏，只是起一个中间人的作用，主要收集和汇总问题以及解决办法，并及时更新文档，以方便大家查阅，希望大家积极参与；

升级控件版本到2.1.0.2。

6/16/2011

修复了一个BUG，此BUG会造成在使用debug版本控件情况下，在ide中拖放控件时，出现非法操作；

修复了一个BUG，此BUG表现在绘制平滑曲线时，如果只有两个点（连着的，离散的不算），则会造成控件使用错误的画笔（上一条曲线的画笔），或者非法画笔；

修复了属性重置时的一个小BUG；

完善了ST\_Curve\_PlugIn.hpp里面的说明性文档；

升级控件版本到2.1.0.3。

7/2/2011

修复了一个BUG，此BUG会让删除图例时（比如调用DelLegend接口），控件窗口里面留下残留的图例画面；

修复了一个BUG，此BUG表现在，当调用AddLegend接口时，如果图例已经存在，而且Id已经属于另外的图例的话，这条曲线将同时出现在两个图例里面；

增加了两个接口MoveCurveToLegend和ChangeLegendName，前者用于把指定的曲线从其所在图例中，移动到指定的图例中，后者用于修改图例名；

增加了QueryLegend接口的开发文档，之前一直写漏掉了；

第八章里面增加些内容，有兴趣者可以看看；

升级控件版本到2.1.0.4。

7/12/2011

为网格增加了一种风格——只绘制主刻度上的网格，具体参看S(G)etGridMode接口；

修复ExportImage方法的一个BUG（指的是导出方法，而非ExportImage接口，具体参看第四大节——导出方法），此BUG会错误的将入参hBitmap释放掉（DeleteObject）；

升级控件版本到2.1.0.5。

9/1/2011

升级编译器到vc2010版本，以方便之后的x64版本编译，同时升级控件内部所有crt函数至相应的安全版本，以减少潜在的写内存越界等问题；

当然，编译器的升级不会影响到使用（这就是COM要解决的问题），老用户只需要覆盖老版本控件即可（当然，mfc71库就不再使用了，改而使用mfc100库，可从我网站下载）；

在TestST\_Curve的OnMouseUpStcurvectrl函数里面，我写了一些示例代码，用以实现问得比较多的如何鼠标点击删除某点，鼠标点击添加点，插入点等问题（注释状态）；

升级控件版本到2.1.1.0。

9/17/2011

修复了一个BUG，此BUG会造成在绘制坐标时，如果之前绘制过填充值，那么绘制出来的坐标的颜色不正确（不是指定的颜色）；

修复了一个BUG，此BUG会造成绘制坐标时，Y坐标第一个字符无法绘制出来；

修复了一个BUG，此BUG会造成通过插件绘制坐标时，如果返回了空字符串，则绘制出现问题，比如绘制不出来、绘制位置不正确等；

修复了解析通配符的BUG（通配符参看BatchExportImage），当字符串的最后一个字符也是通配符时，控件将无法解析这个通配符，造成了生成的文件名里面带有星号（\*），这样的文件名是不合法的（windows规定），所以也就无法创建文件更谈不上导出了；

升级控件版本至2.1.1.1。

9/30/2011

修复了CanContinueEnum接口的一个BUG，此BUG可能造成判断错误，为此，修改了所有枚举相关的接口：GetDataHeadPosition、GetTimeData、GetTimeData2、GetValueData、GetState、GetPosData、InsertMainData、InsertMainData2、CanContinueEnum、DelPoint。其中除了GetDataHeadPosition被删除之外，其它的只是修改了参数类型，老工程如果想要使用新版本控件（用到了上面的接口的），将不得不重新编译工程，为此我向大家道歉！

枚举相关接口的开发文档有更新，特别是在枚举过程中有删除或者增加点的情况下；

增加了GetLuaVer接口，用于返回控件内部集成的Lua解释器的版本；

升级控件版本至2.1.1.2。

11/12/2011

修复了一个BUG，此BUG会造成在绘制实时曲线时，如果通过VisibleState限制了曲线的移动，则全局位置预览窗口没有得到刷新；

修复了一个BUG，此BUG表现在，当二次开发者交替的对同一条曲线绘制实时曲线和非实时曲线时（比如调用AddMainData(2)10次，5次VisibleState为1，5次为0），控件内部一些状态会出现不正确的情况（主要是位置及大小状态，比如一条曲线所占的矩形大小，所有曲线所占的矩形大小），这可能会造成在移动曲线、报告页数信息、打印等情况下，出现错误或者操作不完全；

增加了两个接口SetAutoRefresh和GetAutoRefresh，用于自动刷新（这个属于实时曲线的概念，简单说来，就是按一定的间隔自动刷新，而不完全依赖于VisibleState参数），用好自动刷新，可显著提高CPU利用率，让绘制实时曲线时，刷新即平滑，又不至于过于频繁而耗CPU资源。具体请参看开发文档；

在本开发文档中，增加了一栏：捐助（第十大栏），如果你想为ST\_Curve捐助，请参看第十栏；

升级控件版本至2.1.1.3。

1/2/2012

修复了一个BUG，此BUG会造成某些接口失效，这是升级vc2010编译器造成的，影响到的接口有：EnableAutoTrimCoor、EnableHelpTip、EnableFullScreen、EnableFocusState、EnablePreview；影响到的快捷键有：F4、F7、F6。此BUG从2.1.1.0开始一直存在；

修改了注解文档说明，以便于更好的理解；

增加了注解的位图居中功能，具体请参看AddComment等相关接口；

升级内部Lua解释器版本到5.2.0；

优化了控件内部对于等宽和不等宽字体的处理；

升级控件版本至2.1.2.0。

2/10/2012

修复了一个BUG，此BUG会造成控件无法在win2k、xp、xp sp1系统下面运行，其实这也不能算是控件的BUG，这是用vc2010编译器编译的结果，所以在官网上增加了一个“兼容版本”下载，用以解决这个问题，其实就是把编译器换到vc2003，xp sp2及其以上的系统，推荐下载正规版本，虽然兼容版本也行；

增加了一个接口EnableSelectCurve，用于设置曲线是否可以被选中；

增加了一个接口S(G)ToolTipDelay，用于设置坐标提示的延迟时间，本接口还可让坐标提示失效，具体请参看开发文档；

升级控件版本至2.1.2.1。

3/1/2012

修复了兼容版本一个小BUG，此BUG会造成限制坐标时（包括限制一页），重复刷新；

为限制一页功能增加了一个模式，此模式只会在有点出了画布之外时，才生效，具体参看SetLimitOnePageMode接口，这个功能非常有用，据我了解需要的人很多，请大家认真看看开发文档；

增加本控件的第一个图元对象——无限曲线，有两种无限曲线，一种是水平的，一种是垂直的，只需要添加一个坐标，就能绘制出来。它特别适合画边界，比如学生成绩曲线，你可以画一条60分的无限曲线，很直观的看出来他在什么时候考试不及格；还有一种用法就是可以模拟固定坐标轴（正常的坐标轴是不固定的），比如在0 0处画两条无限直线，然后把真正的坐标轴隐藏起来（全屏），具体请参看AddInfiniteCurve接口；

增加了double版本，该版本把所有接口里面的时间类型，全部修改成了double类型，这个修改在控件里面根本不是一个修改，因为DATE本身就是double，但在vb和.net中，将解决DateTime范围太小的问题（如果横坐标显示为数字，则DateTime的范围显得太小，最好能长到double的范围，为此必须修改idl接口定义），具体请参看第八章：小窍门及FAQ的第16条；

升级控件版本至2.2.0.0。

7/14/2012

修复了控件在win7下面，绘制十字架闪烁的问题；

增加了一个属性——Register1，用于解决与64位版本控件之间的64位数据传递的问题，在之前的32位控件中，不存在64位整数，当升级到64位版本之后，很多数据类型，比如HANDLE、HFONT等都变成了64位，此时按道理来说，64位控件应该将接口中的这些数据类型由32位增加至64位，可是带来一个问题：控件接口在32位和64位下不一致，这造成的问题是无法容忍的，比如要使用64位控件的话，需要64位编译器等很多不可理解的限制（高版本vc是有交叉编译功能的，所以32位编译器编译64位代码是天经地义的，但用在64位控件上却不行了），关于64位控件的使用问题，请参看我的文章《[vc2010下使用64位控件](http://blog.csdn.net/yang79tao/article/details/7705754)》；

修改了ReportPageInfo接口、PageChangeMSG消息和PageChange事件的行为（返回数据格式有修改），这里向已经使用了这些功能的二次开发者道歉，具体请参看开发文档；

关于64位版本控件哪些接口和属性需要使用寄存器1，请参看Register1属性；

升级控件版本至2.2.0.1。

8/15/2012

修复BUG，此BUG会把水印显示在控件及画布背景的下面，从而被遮挡；

修复BUG，此BUG会造成在非默认坐标系下，画布背景贴图方向与控件背景不一致；

为控件背景增加了一种模式——裁剪掉画布背景，具体请参看S(G)BkMode接口；

修改属性和接口中所有OLE\_COLOR和OLE\_HANDLE（生成的包装类中为unsigned long）类型为long类型（包括指针，相应的就是long\*），也就是把所有无符号类型改成了有符号类型，宽度不变（只要宽度不变，数据在有符号和无符号之间传递就不会有损失，这一点大家放心），如果大家在使用控件时无法编译，请把原来的unsigned long改为long，至于为什么要这样，请参看我的文章《[MFC ActiveX 接口数据类型，伤不起！](http://blog.csdn.net/yang79tao/article/details/7821796)》，这是vc2010所带来的新BUG；

优化控件在默认坐标系下面的绘制效率；

升级控件版本至2.2.0.2。

11/17/2012

增加了修改控件刻度在屏幕上的间隔大小的功能，刻度间隔一般是不变的（想像一下老式秤杆上的刻度），只是一个刻度代表的数值有变化（想像一下重量不同的秤砣用在老式秤上面），增加的这个功能打破了这种传统的认知，有什么用呢，具体还是请参看S(G)etGraduationSize接口；

升级控件版本至2.2.0.3。

2/16/2013

增加了运行时修改鼠标滚轮行为（包括按下shift、ctrl、atl）具体请参看S(G)etMouseWheelMode接口；

增加了修改鼠标滚轮速度功能，具体请参看S(G)etMouseWheelSpeed接口；

增加了禁用鼠标滚轮功能，也由SetMouseWheelSpeed接口实现；

升级内部Lua解释器版本到5.2.1；

申请了一个QQ交流群——132339384；

升级控件版本至2.2.0.4。

4/6/2013

修改GetLegendAddressCount、GetLegendAddress、ChangeAddress接口分别为GetLegendIdCount、GetLegendId、ChangeId，参数保存不变；

提供了对win8的支持；

本次更新无版本升级。

4/20/2013

重新排版开发文档；

整理了开发包里面的目录结构，以及《说明.txt》；

本次更新无版本升级。

9/8/2013

增加了对数字键盘上的数字和加减键的支持（作用与大键盘上相应键的作用完全一样）；

点状态（参看AddMainData(2)中的State参数）增加了一个状态，于是扩展为short类型（之前需要参数是short，但只使用了低字节），为此受影响的接口有：AddMainData(2)、AddMemMainData和InsertMainData(2)，其中第3组接口增加了一个参数，第2组接口没有改变，但参数的内容有改变，第1组接口没有改变，且向下兼容，具体请参看它们的接口说明，老工程如果想要升级控件，对于第2、3组接口来说，必须重新编译，对此造成的不便，我深表歉意；

增加了水平图例支持，水平图例显示在横坐标Label下面，只能显示文字，用于显示一些说明性的信息，支持多行显示，具体请参看S(G)etHLegend接口；

升级内部Lua解释器版本到5.2.2；

升级控件版本至2.2.0.5。

7/5/2014

增加了在横坐标显示为值的情况下，可隐藏横坐标（之前只能在横坐标显示为时间时才能隐藏），具体请参看SetShowMode接口；

修复BUG，此BUG会造成在同一条曲线上设置了向多个方向填充，并且在填充区域内有显示坐标的情况下，控件崩溃。关于填充和显示坐标，请参看SetFillDirection接口；

从文件中导入曲线的时候，bAddTrail参数被忽略，原因是顺序导出，当然应该顺序导入。为了接口的一致性，本参数保留，只是不再具有意义，具体请参看ImportFile接口；

升级内部Lua解释器版本到5.2.3；

本次更新无版本升级。

4/25/2015

修复BUG，此BUG会造成从文件导入曲线的时候，可能错误的把以浮点数格式导出的数据认为是时间格式，从而导入错误的数据；

在绘制实时曲线的时候，如果新添加的这个点的前面那个点不在画布中，则不会自动移动曲线（绘制实时曲线失效），那么如何恢复呢，答案是把上一个点拖到画布中来。这样的设计是基于一个逻辑：如果没有人为的拖动，实时曲线总是会自动滚动的，因为总是能保证上一个点处于画布之中，在人为拖动的情况下，控件理解为，操作者需要查看曲线的某一段历史情况，此时如果自动滚动，显然违背了操作者的意图，所以实时曲线会停止不动（当然实时曲线还是在不停绘制的），直到操作者解除这种状态（把上一个点拖入画布）；

升级内部Lua解释器版本到5.3.0；

本次更新无版本升级。

5/31/2015

修复BUG，此BUG会造成在通过InsertMainData(2)修改某个点的时候，实际坐标修改了，但屏幕坐标没有重新计算（所以看起来感觉修改没有成功）；

回退了上一次更新中，绘制实时曲线的行为，但功能保留，只是默认不开启。所以扩展了AddMainData(2)的VisibleState的第6位，开发者需要显式地指定这位为1，才会达到上次更新的绘制实时曲线的行为（上一个点不在画布中时，绘制实时曲线失效）；

升级控件版本至2.2.0.6。

6/17/2017

修复BUG，此BUG会造成在绘制平滑曲线时内存泄漏；

升级内部Lua解释器版本到5.3.4；

升级编译器至VC2015；

升级控件版本至2.2.0.7。

5/7/2018

修复BUG，此BUG会造成CloneCurve时新曲线不可见（随机Z轴）；

开源本控件（https://github.com/youngwolf-project/ST\_Curve）。