# ST\_Curve开发文档

2009-Apr 对应版本1.3.1.3（2009-3-1 21:31:22）

[1 ST\_Curve开发文档 1](#_Toc226454330)

[1.1 主要特点： 7](#_Toc226454331)

[1.1.1 属性： 9](#_Toc226454332)

[1.1.2 函数： 10](#_Toc226454333)

[1.1.2.1 坐标轴相关（4个）： 10](#_Toc226454334)

[1.1.2.2 坐标相关(20个)： 10](#_Toc226454335)

[1.1.2.3 单位相关（4个）： 10](#_Toc226454336)

[1.1.2.4 图例相关： 10](#_Toc226454337)

[1.1.2.5 添加曲线点： 11](#_Toc226454338)

[1.1.2.6 删除曲线点： 11](#_Toc226454339)

[1.1.2.7 翻页相关： 11](#_Toc226454340)

[1.1.2.8 缩放相关： 11](#_Toc226454341)

[1.1.2.9 模式（移动模式、显示模式、网格模式）相关： 11](#_Toc226454342)

[1.1.2.10 背景图相关： 11](#_Toc226454343)

[1.1.2.11 导出图片或导出导入文件： 12](#_Toc226454344)

[1.1.2.12 标题脚注： 12](#_Toc226454345)

[1.1.2.13 枚举（并修改）某条曲线： 12](#_Toc226454346)

[1.1.2.14 枚举所有曲线： 12](#_Toc226454347)

[1.1.2.15 枚举图例： 12](#_Toc226454348)

[1.1.2.16 联动相关： 13](#_Toc226454349)

[1.1.2.17 Z-Order相关： 13](#_Toc226454350)

[1.1.2.18 范围相关： 13](#_Toc226454351)

[1.1.2.19 移动曲线： 13](#_Toc226454352)

[1.1.2.20 辅助函数： 13](#_Toc226454353)

[1.1.2.21 曲线长度控制相关： 13](#_Toc226454354)

[1.1.2.22 页数量变化事件（消息）： 14](#_Toc226454355)

[1.1.2.23 三维相关函数： 14](#_Toc226454356)

[1.1.2.24 基点相关： 14](#_Toc226454357)

[1.1.2.25 填充方向相关： 14](#_Toc226454358)

[1.1.2.26 缓存控制相关： 14](#_Toc226454359)

[1.1.2.27 曲线操作函数： 14](#_Toc226454360)

[1.1.2.28 其它函数： 15](#_Toc226454361)

[1.1.2.29 限制坐标相关： 15](#_Toc226454362)

[1.2 属性及函数详细介绍： 15](#_Toc226454363)

[1.2.1 属性（８个） 15](#_Toc226454364)

[1.2.2 函数（172个） 16](#_Toc226454365)

[1.2.2.1 BOOL **SetVInterval**(short VInterval); 16](#_Toc226454366)

[1.2.2.2 BOOL **SetHInterval**(short HInterval); 16](#_Toc226454367)

[1.2.2.3 short **GetScaleInterval**(); 16](#_Toc226454368)

[1.2.2.4 long **GetScaleNums**(); 16](#_Toc226454369)

[1.2.2.5 BOOL **SetBeginValue**(float fBeginValue); 16](#_Toc226454370)

[1.2.2.6 float **GetBeginValue**(); 16](#_Toc226454371)

[1.2.2.7 double **GetEndTime**(); 16](#_Toc226454372)

[1.2.2.8 float **GetEndValue**(); 16](#_Toc226454373)

[1.2.2.9 BOOL **SetBeginTime**(LPCTSTR pBeginTime); 17](#_Toc226454374)

[1.2.2.10 BOOL **SetBeginTime2**(double fBeginTime); 17](#_Toc226454375)

[1.2.2.11 CString **GetBeginTime**(); 17](#_Toc226454376)

[1.2.2.12 double **GetBeginTime2**(); 17](#_Toc226454377)

[1.2.2.13 BOOL **SetValueStep**(float ValueStep); 17](#_Toc226454378)

[1.2.2.14 float **GetValueStep**(); 17](#_Toc226454379)

[1.2.2.15 BOOL **SetTimeSpan**(double TimeStep); 17](#_Toc226454380)

[1.2.2.16 double **GetTimeSpan**(); 17](#_Toc226454381)

[1.2.2.17 BOOL **SetVPrecision**(short Precision); 17](#_Toc226454382)

[1.2.2.18 short **GetVPrecision**(); 17](#_Toc226454383)

[1.2.2.19 BOOL **SetHPrecision**(short Precision); 18](#_Toc226454384)

[1.2.2.20 short **GetHPrecision**(); 18](#_Toc226454385)

[1.2.2.21 void **TrimCoor**(); 18](#_Toc226454386)

[1.2.2.22 void **EnableAutoTrimCoor**(BOOL bEnable); 18](#_Toc226454387)

[1.2.2.23 void **SetVisibleCoorRange**(double MinTime, double MaxTime, float MinValue,float MaxValue, short Mask); 18](#_Toc226454388)

[1.2.2.24 void **GetVisibleCoorRange**(double\* pMinTime, double\* pMaxTime,float\* pMinValue, float\* pMaxValue); 18](#_Toc226454389)

[1.2.2.25 BOOL **SetUnit**(LPCTSTR pUnit); 18](#_Toc226454390)

[1.2.2.26 CString **GetUnit**(); 18](#_Toc226454391)

[1.2.2.27 BOOL **SetHUnit**(LPCTSTR pHUnit); 18](#_Toc226454392)

[1.2.2.28 CString **GetHUnit**(); 18](#_Toc226454393)

[1.2.2.29 short **AddLegend**(long Address, LPCTSTR pSign, OLE\_COLOR PenColor, short PenStyle, short LineWidth, OLE\_COLOR BrushColor, short BrushStyle, short CurveMode, short NodeMode, short Mask, BOOL bUpdate); 19](#_Toc226454394)

[1.2.2.30 short **AddLegendHelper**(long Address, LPCTSTR pSign, OLE\_COLOR PenColor, short PenStyle, short LineWidth, boolean bUpdate); 20](#_Toc226454395)

[1.2.2.31 BOOL **GetLegend**(LPCTSTR pSign, OLE\_COLOR\* pPenColor, short FAR\* pPenStyle, short FAR\* pLineWidth, OLE\_COLOR\* pBrushColor, short FAR\* pBrushStyle, short FAR\* pCurveMode, short FAR\* pNodeMode); 20](#_Toc226454396)

[1.2.2.32 BOOL **IsLegend**(LPCTSTR pSign); 20](#_Toc226454397)

[1.2.2.33 CString **QueryLegend**(long Address); 20](#_Toc226454398)

[1.2.2.34 BOOL **DelLegend**(long Address, BOOL bAll, BOOL bUpdate); 20](#_Toc226454399)

[1.2.2.35 BOOL **DelLegend2**(LPCTSTR pSign, BOOL bUpdate); 20](#_Toc226454400)

[1.2.2.36 BOOL **SetLegendSpace**(short LegendSpace); 21](#_Toc226454401)

[1.2.2.37 short **GetLegendSpace**(); 21](#_Toc226454402)

[1.2.2.38 BOOL **ShowLegend**(LPCTSTR pSign, BOOL bShow); 21](#_Toc226454403)

[1.2.2.39 BOOL **ShowCurve**(long Address, BOOL bShow); 21](#_Toc226454404)

[1.2.2.40 short **AddMainData**(long Address, LPCTSTR pTime, float Value, short State, short VisibleState, BOOL bAddTrail); 21](#_Toc226454405)

[1.2.2.41 short **AddMainData2**(long Address, double Time, float Value, short State, short VisibleState, BOOL bAddTrail); 21](#_Toc226454406)

[1.2.2.42 long **AddMemMainData**(OLE\_HANDLE pMemMainData, long MemSize, BOOL bAddTrail); 21](#_Toc226454407)

[1.2.2.43 void **DelRange**(long Address, double BTime, double ETime,short Mask, BOOL bAll, BOOL bUpdate); 23](#_Toc226454408)

[1.2.2.44 void **DelRange2**(long Address, long nIndex, long nCount, BOOL bAll, BOOL bUpdate); 23](#_Toc226454409)

[1.2.2.45 BOOL **FirstPage**(BOOL bLast, BOOL bUpdate); 23](#_Toc226454410)

[1.2.2.46 short **GotoPage**(short RelativePage, BOOL bUpdate); 23](#_Toc226454411)

[1.2.2.47 BOOL **SetZoom**(short Zoom); 24](#_Toc226454412)

[1.2.2.48 short **GetZoom**(); 24](#_Toc226454413)

[1.2.2.49 void **EnableZoom**(BOOL bEnable); 24](#_Toc226454414)

[1.2.2.50 BOOL **SetShowMode**(short ShowMode); 24](#_Toc226454415)

[1.2.2.51 short **GetShowMode**(); 24](#_Toc226454416)

[1.2.2.52 BOOL **SetMoveMode**(short MoveMode); 24](#_Toc226454417)

[1.2.2.53 short **GetMoveMode**(); 24](#_Toc226454418)

[1.2.2.54 BOOL **SetGridMode**(short GridMode); 25](#_Toc226454419)

[1.2.2.55 short **GetGridMode**(); 25](#_Toc226454420)

[1.2.2.56 BOOL **AddImageHandle**(LPCTSTR pFileName, BOOL bShared); 25](#_Toc226454421)

[1.2.2.57 void **AddBitmapHandle**(OLE\_HANDLE hBitmap, BOOL bShared); 25](#_Toc226454422)

[1.2.2.58 BOOL **AddBitmapHandle2**(OLE\_HANDLE hInstance, LPCTSTR pszResourceName, BOOL bShared); 25](#_Toc226454423)

[1.2.2.59 BOOL **AddBitmapHandle3**(OLE\_HANDLE hInstance, long nIDResource, BOOL bShared); 25](#_Toc226454424)

[1.2.2.60 BOOL **RemoveBitmapHandle**(OLE\_HANDLE hBitmap, BOOL bDel); 25](#_Toc226454425)

[1.2.2.61 BOOL **RemoveBitmapHandle2**(short nIndex, BOOL bDel); 25](#_Toc226454426)

[1.2.2.62 BOOL **SetBkBitmap**(short nIndex); 25](#_Toc226454427)

[1.2.2.63 short **GetBkBitmap**(); 25](#_Toc226454428)

[1.2.2.64 long **GetBitmapCount**(); 26](#_Toc226454429)

[1.2.2.65 OLE\_HANDLE **GetBitmap**(short nIndex); 26](#_Toc226454430)

[1.2.2.66 short **GetBitmapState**(short nIndex); 26](#_Toc226454431)

[1.2.2.67 short **GetBitmapState2**(OLE\_HANDLE hBitmap); 26](#_Toc226454432)

[1.2.2.68 BOOL **SetCanvasBkBitmap**(short nIndex); 26](#_Toc226454433)

[1.2.2.69 short **GetCanvasBkBitmap**(); 26](#_Toc226454434)

[1.2.2.70 BOOL **SetBkMode**(short BkMode); 26](#_Toc226454435)

[1.2.2.71 short **GetBkMode**(); 26](#_Toc226454436)

[1.2.2.72 BOOL **SetCanvasBkMode**(short CanvasBkMode); 26](#_Toc226454437)

[1.2.2.73 short **GetCanvasBkMode**(); 26](#_Toc226454438)

[1.2.2.74 BOOL **ExportImage**(LPCTSTR pFileName); 26](#_Toc226454439)

[1.2.2.75 long **ExportImageFromPage**(LPCTSTR pFileName, long Address, long nStartPage, long nCount, BOOL bAll, short Style); 26](#_Toc226454440)

[1.2.2.76 long **ExportImageFromTime**(LPCTSTR pFileName, long Address, double BTime, double ETime, BOOL bAll, short Style); 26](#_Toc226454441)

[1.2.2.77 void **BatchExportImage**(LPCTSTR pFileName, long nSecond); 26](#_Toc226454442)

[1.2.2.78 long ImportFile(LPCTSTR pFileName, short Style, BOOL bAddTrail); 26](#_Toc226454443)

[1.2.2.79 long **ExportMetaFile**(LPCTSTR pFileName, long Address, long nBegin, long nCount, BOOL bAll, short Style); 26](#_Toc226454444)

[1.2.2.80 void **SetCurveTitle**(LPCTSTR pCurveTitle); 28](#_Toc226454445)

[1.2.2.81 CString **GetCurveTitle**(); 28](#_Toc226454446)

[1.2.2.82 void **SetFootNote**(LPCTSTR pFootNote); 28](#_Toc226454447)

[1.2.2.83 CString **GetFootNote**(); 28](#_Toc226454448)

[1.2.2.84 OLE\_HANDLE **GetDataHeadPosition**(long Address); 28](#_Toc226454449)

[1.2.2.85 double **GetTimeData**(OLE\_HANDLE hCurve, long nIndex); 28](#_Toc226454450)

[1.2.2.86 float **GetValueData**(OLE\_HANDLE hCurve, long nIndex); 28](#_Toc226454451)

[1.2.2.87 short **GetState**(OLE\_HANDLE hCurve, long nIndex); 28](#_Toc226454452)

[1.2.2.88 BOOL **InsertMainData**(OLE\_HANDLE hCurve, long nIndex,LPCTSTR pTime, float Value, short State, short Position); 28](#_Toc226454453)

[1.2.2.89 BOOL **InsertMainData2**(OLE\_HANDLE hCurve, long nIndex,double Time, float Value, short State, short Position); 28](#_Toc226454454)

[1.2.2.90 BOOL **CanContinueEnum**(OLE\_HANDLE hCurve); 28](#_Toc226454455)

[1.2.2.91 void **DelPoint**(OLE\_HANDLE hCurve, long nIndex); 28](#_Toc226454456)

[1.2.2.92 long **GetCurveCount**(); 30](#_Toc226454457)

[1.2.2.93 long **GetCurve**(long nIndex); 30](#_Toc226454458)

[1.2.2.94 long **GetLegendCount**(); 30](#_Toc226454459)

[1.2.2.95 BOOL **GetLegend2**(long nIndex, OLE\_COLOR\* pPenColor, short FAR\* pPenStyle,short FAR\* pLineWidth, OLE\_COLOR\* pBrushColor,short FAR\* pBrushStyle, short FAR\* pCurveMode,short FAR\* pNodeMode); 30](#_Toc226454460)

[1.2.2.96 long **GetLegendAddressCount**(long nIndex); 30](#_Toc226454461)

[1.2.2.97 long **GetLegendAddress**(long nLegendIndex, long nAddressIndex); 30](#_Toc226454462)

[1.2.2.98 BOOL **SetBuddy**(long hBuddy, short State); 30](#_Toc226454463)

[1.2.2.99 short **GetBuddyCount**(); 30](#_Toc226454464)

[1.2.2.100 long **GetBuddy**(short nIndex); 30](#_Toc226454465)

[1.2.2.101 void **EnableAdjustZOrder**(BOOL bEnable); 31](#_Toc226454466)

[1.2.2.102 short **GetCurveIndex**(long Address); 31](#_Toc226454467)

[1.2.2.103 BOOL **SetCurveIndex**(long Address, short nIndex); 31](#_Toc226454468)

[1.2.2.104 BOOL **GetSelectedCurve**(long FAR\* pAddress); 31](#_Toc226454469)

[1.2.2.105 BOOL **SelectCurve**(long Address, BOOL bSelect); 31](#_Toc226454470)

[1.2.2.106 short **DragCurve**(short xStep, short yStep, BOOL bUpdate); 31](#_Toc226454471)

[1.2.2.107 BOOL **GetOneTimeRange**(long Address, double FAR\* pMinTime, double FAR\* pMaxTime); 31](#_Toc226454472)

[1.2.2.108 BOOL **GetOneValueRange**(long Address, float FAR\* pMinValue, float FAR\* pMaxValue); 31](#_Toc226454473)

[1.2.2.109 BOOL **GetOneFirstPos**(long Address, double FAR\* pTime, float FAR\* pValue,BOOL bLast); 31](#_Toc226454474)

[1.2.2.110 BOOL **GetTimeRange**(double FAR\* pMinTime, double FAR\* pMaxTime); 32](#_Toc226454475)

[1.2.2.111 BOOL **GetValueRange**(float FAR\* pMinValue, float FAR\* pMaxValue); 32](#_Toc226454476)

[1.2.2.112 void **GetViableTimeRange**(double FAR\* pMinTime, double FAR\* pMaxTime); 32](#_Toc226454477)

[1.2.2.113 BOOL **VCenterCurve**(long Address, BOOL bUpdate); 32](#_Toc226454478)

[1.2.2.114 BOOL **GotoCurve**(long Address); 32](#_Toc226454479)

[1.2.2.115 BOOL **IsSelected**(long Address); 32](#_Toc226454480)

[1.2.2.116 BOOL **IsLegendVisible**(LPCTSTR pSign); 32](#_Toc226454481)

[1.2.2.117 BOOL **IsCurveVisible**(long Address); 32](#_Toc226454482)

[1.2.2.118 BOOL **IsCurveInCanvas**(long Address); 32](#_Toc226454483)

[1.2.2.119 void **EnableHelpTip**(BOOL bEnable); 32](#_Toc226454484)

[1.2.2.120 long **CheckUpdate**(BSTR\* pHomePage, BSTR\* pVersion, BSTR\* pModifyTime); 32](#_Toc226454485)

[1.2.2.121 short **GetPower**(long Address); 32](#_Toc226454486)

[1.2.2.122 BOOL **IsCurve**(long Address); 33](#_Toc226454487)

[1.2.2.123 void **SetSorptionRange**(short Range); 33](#_Toc226454488)

[1.2.2.124 short **GetSorptionRange**(); 33](#_Toc226454489)

[1.2.2.125 BOOL **GetActualPoint**(long x, long y, double\* pTime, double\* pValue); 33](#_Toc226454490)

[1.2.2.126 long **GetPointFromScreenPoint**(long Address, long x, long y, short MaxRange); 33](#_Toc226454491)

[1.2.2.127 BOOL **SetMaxLength**(long MaxLength, long CutLength); 34](#_Toc226454492)

[1.2.2.128 long **GetMaxLength**(); 34](#_Toc226454493)

[1.2.2.129 long **GetCutLength**(); 34](#_Toc226454494)

[1.2.2.130 long **GetCurveLength**(long Address); 34](#_Toc226454495)

[1.2.2.131 void **EnablePageChangeEvent**(BOOL bEnable); 34](#_Toc226454496)

[1.2.2.132 long **ReportPageInfo**(); 34](#_Toc226454497)

[1.2.2.133 void **SetZLength**(short ZLength); 34](#_Toc226454498)

[1.2.2.134 short **GetZLength**(); 34](#_Toc226454499)

[1.2.2.135 BOOL **SetZOffset**(long Address, short nOffset, BOOL bUpdate); 35](#_Toc226454500)

[1.2.2.136 long **GetZOffset**(long Address); 35](#_Toc226454501)

[1.2.2.137 void **SetLeftBkColor**(OLE\_COLOR Color); 35](#_Toc226454502)

[1.2.2.138 OLE\_COLOR **GetLeftBkColor**(); 35](#_Toc226454503)

[1.2.2.139 void **SetBottomBkColor**(OLE\_COLOR Color); 36](#_Toc226454504)

[1.2.2.140 OLE\_COLOR **GetBottomBkColor**(); 36](#_Toc226454505)

[1.2.2.141 BOOL **SetReviseToolTip**(short Type); 36](#_Toc226454506)

[1.2.2.142 short **GetReviseToolTip**(); 36](#_Toc226454507)

[1.2.2.143 void **SetBenchmark**(double Time, float Value); 36](#_Toc226454508)

[1.2.2.144 void **GetBenchmark**(double\* pTime, float\* pValue); 36](#_Toc226454509)

[1.2.2.145 BOOL **SetFillDirection**(long Address, short FillDirection, BOOL bUpdate); 36](#_Toc226454510)

[1.2.2.146 short **GetFillDirection**(long Address); 36](#_Toc226454511)

[1.2.2.147 void **ClearTempBuff**(); 37](#_Toc226454512)

[1.2.2.148 BOOL **PreMallocMem**(long Address, long size); 37](#_Toc226454513)

[1.2.2.149 long **GetMemSize**(long Address); 37](#_Toc226454514)

[1.2.2.150 long **TrimCurve**(long Address, short State, long nBegin, long nCount,short nStep, BOOL bAll); 37](#_Toc226454515)

[1.2.2.151 long **TrimCurve2**(long Address, short State, double BTime, double ETime,short Mask, short nStep, BOOL bAll) 37](#_Toc226454516)

[1.2.2.152 short **PrintCurve**(long Address, double BTime, double ETime, short Mask, short LeftMargin,short TopMargin,short RightMargin,short BottomMargin, LPCTSTR pTitle, LPCTSTR pFootNote, short Flag, BOOL bAll); 38](#_Toc226454517)

[1.2.2.153 BOOL **OffSetCurve**(long Address, double Time, float Value, short Operator); 39](#_Toc226454518)

[1.2.2.154 BOOL **ChangeAddress**(long Address, long NewAddr); 39](#_Toc226454519)

[1.2.2.155 BOOL **CloneCurve**(long Address, long NewAddr); 39](#_Toc226454520)

[1.2.2.156 BOOL **UniteCurve**(long DesAddr, long nInsertPos, long Address, long nBegin,long nCount); 39](#_Toc226454521)

[1.2.2.157 BOOL **UniteCurve2**(long DesAddr, long nInsertPos, long Address,double BTime, double ETime, short Mask); 39](#_Toc226454522)

[1.2.2.158 BOOL **UniteCurve3**(long DesAddr, double fInsertPos, long Address, long nBegin,long nCount); 39](#_Toc226454523)

[1.2.2.159 BOOL **UniteCurve4**(long DesAddr, double fInsertPos, long Address,double BTime, double ETime, short Mask); 39](#_Toc226454524)

[1.2.2.160 long **ArithmeticOperate**(long DesAddr, long Address, short Operator); 40](#_Toc226454525)

[1.2.2.161 void **SetFont**(OLE\_HANDLE hFont); 41](#_Toc226454526)

[1.2.2.162 CString **GetCopyrightInfo**(); 41](#_Toc226454527)

[1.2.2.163 void **Refresh**(); 41](#_Toc226454528)

[1.2.2.164 void **EnableFullScreen**(BOOL bEnable); 41](#_Toc226454529)

[1.2.2.165 void **EnableFocusState**(BOOL bEnable); 41](#_Toc226454530)

[1.2.2.166 void **EnablePreview**(BOOL bEnable); 41](#_Toc226454531)

[1.2.2.167 void **SetWaterMark**(LPCTSTR pWaterMark); 41](#_Toc226454532)

[1.2.2.168 long **GetSysState**(); 41](#_Toc226454533)

[1.2.2.169 void **LimitOnePage**(BOOL bLimit); 42](#_Toc226454534)

[1.2.2.170 BOOL **FixCoor**(double MinTime, double MaxTime,float MinValue, float MaxValue, short Mask); 42](#_Toc226454535)

[1.2.2.171 short **GetFixCoor**(double\* pMinTime, double\* pMaxTime,float\* pMinValue, float\* pMaxValue); 42](#_Toc226454536)

[1.2.2.172 BOOL **RefreshLimitedOrFixedCoor**(); 42](#_Toc226454537)

[1.2.2.173 MouseDown(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y); 44](#_Toc226454538)

[1.2.2.174 MouseUp(short Button, short Shift, OLE\_XPOS\_PIXELS x, OLE\_YPOS\_PIXELS y); 44](#_Toc226454539)

[1.2.2.175 void PageChange(long wParam, long lParam); 44](#_Toc226454540)

[1.2.3 导出方法（3个） 44](#_Toc226454541)

[1.2.3.1 Extern "C" BOOL \_\_stdcall ExportImage(HBITMAP hBitmap, const unsigned short\* pFileName); 44](#_Toc226454542)

[1.2.3.2 Extern "C" LPBITMAPINFO \_\_stdcall GetDIBFromDDB(HDC hDC, HBITMAP hBitmap); 44](#_Toc226454543)

[1.2.3.3 Extern "C" int \_\_stdcall CheckUpdate(BSTR\* pHomePage, BSTR\* pVersion, BSTR\* pModifyTime); 45](#_Toc226454544)

[1.2.4 关于内存使用量 46](#_Toc226454545)

[1.2.5 请初学者或是对编程不太精通者往下看 46](#_Toc226454546)

[1.2.6 更新记录（从2009年1月份开始记录，以前的记录删除掉了） 48](#_Toc226454547)

[1.3 ST\_Curve升级报告 49](#_Toc226454548)

[1.3.1 ST\_Curve计算屏幕坐标的内部原理详解 49](#_Toc226454549)

[1.3.2 此次升级，控件增加的功能概述 51](#_Toc226454550)

[1.3.3 关于一些网友的建议 52](#_Toc226454551)

[1.4 控件内部GDI部分技术文档（部分） 53](#_Toc226454552)

[1.5 FAQ 54](#_Toc226454553)

[1.5.1 什么是ST\_Curve？ 54](#_Toc226454554)

[1.5.2 ST\_Curve面向对象？ 54](#_Toc226454555)

[1.5.3 ST\_Curve有什么使用与分发限制？ 54](#_Toc226454556)

[1.5.4 ST\_Curve的特点与优势（对操作者） 54](#_Toc226454557)

[1.5.5 ST\_Curve的特点与优势（对开发者，只说说几个重点，其它的请看开发文档） 54](#_Toc226454558)

[1.5.6 ST\_Curve的开发环境与运行平台 55](#_Toc226454559)

[1.5.7 下载页面说明 55](#_Toc226454560)

[1.5.8 初学者请关注这一栏 55](#_Toc226454561)

[1.5.9 更新事宜及开发举例： 55](#_Toc226454562)

[1.5.10 位操作 56](#_Toc226454563)

[1.6 鸣谢（排名不分先后） 56](#_Toc226454564)

[1.7 联系方式及留言本 57](#_Toc226454565)

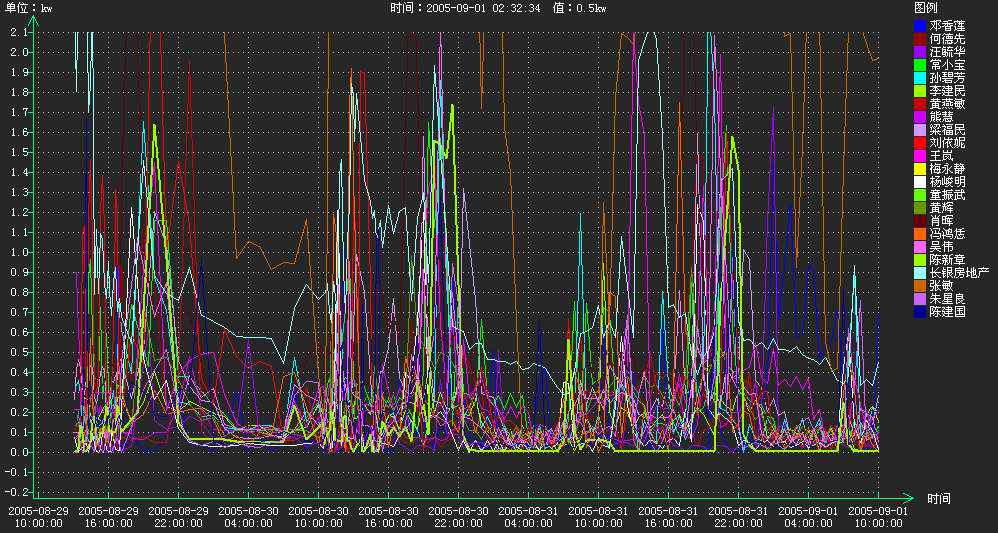
本文档描述了ST\_Curve控件的开发使用说明，包括了每一个属性、每一个函数的意义及其使用方法。

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）；

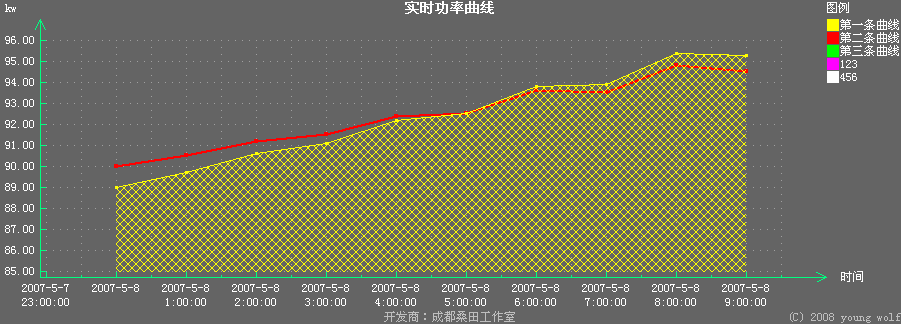
## 主要特点：

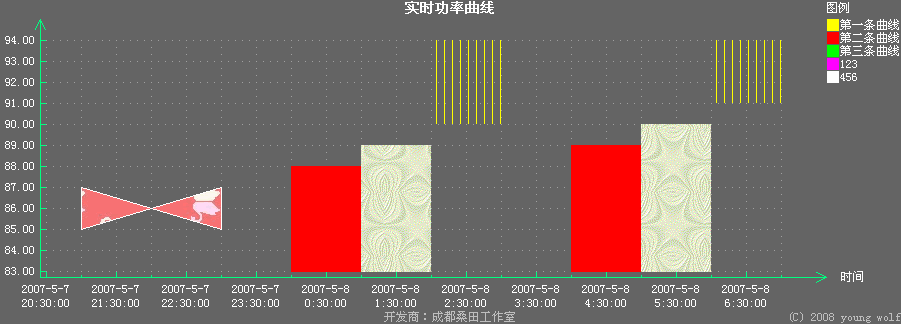
1. 强大的自定义界面，主要的可自定义属性有：背景色、坐标轴色、文字色、网格色、背景图（其中又包括拉伸、平铺、居中三种显示模式）、是否显示网格、单位、各曲线的颜色、横纵坐标值的显示精度、横纵坐标的开始值及每刻度步长值、刻度间隔数、原点位置、跟踪显示鼠标坐标、横坐标自定义显示（值或者时间）等；
2. 功能性：
   1. 任意上下左右移动曲线;
   2. 任意（这里的任意，都不是绝对的，因为计算机只考虑有限的数，是数就会有溢出，所以这里的任意不要钻牛角尖）倍数放大缩小曲线;
   3. 以任意点为中心缩放（鼠标取点）;
   4. 任意隐藏某些曲线;
   5. 按任意曲线居中;
   6. 打印任意曲线或者全部;
   7. 导出任意曲线或者全部为图片或者图元文件;
   8. 自动批量导出图片；
   9. 导出指定页面的曲线，填充曲线等;
   10. 支持三维显示;
   11. 可显示全局位置预览窗口(鹰眼功能)，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）；
   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）；
   4. 选中的曲线将变宽，且提到所有曲线的前面，以示醒目；
   5. 按数字键可按序号选择曲线（如果有的话）；
   6. 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。

**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

### 函数：

#### 坐标轴相关（4个）：

* [SetVInterval](#_BOOL_SetVInterval(short_VInterval);)
* [SetHInterval](#_BOOL_SetHInterval(short_HInterval);)
* [GetScaleInterval](#_short_GetScaleInterval();)
* [GetScaleNums](#_long_GetScaleNums();_1)

#### 坐标相关(20个)：

* [SetBeginValue](#_BOOL_SetBeginValue(float_fBeginValu)
* [GetBeginValue](#_float_GetBeginValue();)
* [GetEndValue](#_float_GetEndValue();)
* [SetBeginTime](#_BOOL_SetBeginTime(LPCTSTR_pBeginTim)
* [SetBeginTime2](#_BOOL_SetBeginTime2(double_fBeginTim)
* [GetEndTime](#_double_GetEndTime();)
* [GetBeginTime](#_CString_GetBeginTime();)
* [GetBeginTime2](#_double_GetBeginTime2();)
* [SetValueStep](#_BOOL_SetValueStep(float_ValueStep);)
* [GetValueStep](#_float_GetValueStep();)
* [SetTimeSpan](#_BOOL_SetTimeSpan(double_TimeStep);)
* [GetTimeSpan](#_double_GetTimeSpan();)
* [SetVPrecision](#_BOOL_SetVPrecision(short_Precision))
* [GetVPrecision](#_short_GetVPrecision();)
* [SetHPrecision](#_BOOL_SetHPrecision(short_Precision))
* [GetHPrecision](#_short_GetHPrecision();)
* [TrimCoor](#_void_TrimCoor();)
* [EnableAutoTrimCoor](#_void_EnableAutoTrimCoor(BOOL_bEnabl)
* [SetVisibleCoorRange](#_void_SetVisibleCoorRange(double_Min)
* [GetVisibleCoorRange](#_void_GetVisibleCoorRange(double*_pM)

#### 单位相关（4个）：

* [SetUnit](#_BOOL_SetUnit(LPCTSTR_pUnit);)
* [GetUnit](#_CString_GetUnit();)
* [SetHUnit](#_BOOL_SetHUnit(LPCTSTR_pHUnit);)
* [GetHUnit](#_CString_GetHUnit();)

#### 图例相关：

* [AddLegend](#_short_AddLegend(long_Address,_LPCTS)
* [AddLegendHelper](#_short_AddLegendHelper(long_Address,)
* [GetLegend](#_BOOL_GetLegend(LPCTSTR_pSign,_OLE_C)
* [IsLegend](#_BOOL_IsLegend(LPCTSTR_pSign);)
* [QueryLegend](#_CString_QueryLegend(long_Address);)
* [DelLegend](#_BOOL_DelLegend(long_Address,_BOOL b)
* [DelLegend2](#_BOOL_DelLegend2(LPCTSTR_pSign,_BOOL)
* [SetLegendSpace](#_BOOL_SetLegendSpace(short_LegendSpa)
* [GetLegendSpace](#_short_GetLegendSpace();)
* [ShowLegend](#_BOOL_ShowLegend(LPCTSTR_pSign,_BOOL)
* [ShowCurve](#_BOOL_ShowCurve(long_Address,_BOOL b)

#### 添加曲线点：

* [AddMainData](#_short_AddMainData(long_Address,_LPC)
* [AddMainData2](#_short_AddMainData2(long_Address,_do)
* [AddMemMainData](#_long_AddMemMainData(OLE_HANDLE_pMem)

#### 删除曲线点：

* [DelRange](#_void_DelRange(long_Address,_double )
* [DelRange2](#_void_DelRange2(long_Address,_long n)

#### 翻页相关：

* [FirstPage](#_BOOL_FirstPage(BOOL_bLast,_BOOL bUp)
* [GotoPage](#_short_GotoPage(short_RelativePage,_)

#### 缩放相关：

* [SetZoom](#_BOOL_SetZoom(short_Zoom);)
* [GetZoom](#_short_GetZoom();)
* [EnableZoom](#_void_EnableZoom(BOOL_bEnable);)

#### 模式（移动模式、显示模式、网格模式）相关：

* [SetMoveMode](#_BOOL_SetMoveMode(short_MoveMode);)
* [GetMoveMode](#_short_GetMoveMode();)
* [SetShowMode](#_BOOL_SetShowMode(short_ShowMode);)
* [GetShowMode](#_short_GetShowMode();)
* [SetGridMode](#_BOOL_SetGridMode(short_GridMode);)
* [GetGridMode](#_short_GetGridMode();)

#### 背景图相关：

* [AddImageHandle](#_BOOL_AddImageHandle(LPCTSTR_pFileNa)
* [AddBitmapHandle](#_void_AddBitmapHandle(OLE_HANDLE_hBi)
* [AddBitmapHandle2](#_BOOL_AddBitmapHandle2(OLE_HANDLE_hI)
* [AddBitmapHandle3](#_BOOL_AddBitmapHandle3(OLE_HANDLE_hI)
* [RemoveBitmapHandle](#_BOOL_RemoveBitmapHandle(OLE_HANDLE_)
* [RemoveBitmapHandle2](#_BOOL_RemoveBitmapHandle2(short_nInd)
* [SetBkBitmap](#_BOOL_SetBkBitmap(short_nIndex);)
* [GetBkBitmap](#_short_GetBkBitmap();)
* [SetCanvasBkMode](#_BOOL_SetCanvasBkMode(short_CanvasBk)
* [GetCanvasBkMode](#_short_GetCanvasBkMode();)
* [SetBkMode](#_BOOL_SetBkMode(short_BkMode);)
* [GetBkMode](#_short_GetBkMode();)
* [GetBitmapCount](#_long_GetBitmapCount();)
* [GetBitmap](#_OLE_HANDLE_GetBitmap(short_nIndex);)
* [GetBitmapState](#_short_GetBitmapState(short_nIndex);)
* [GetBitmapState2](#_short_GetBitmapState2(OLE_HANDLE_hB)
* [SetCanvasBkBitmap](#_BOOL_SetCanvasBkBitmap(short_nIndex)
* [GetCanvasBkBitmap](#_short_GetCanvasBkBitmap();)

#### 导出图片或导出导入文件：

* [ExportImage](#_BOOL_ExportImage(LPCTSTR_pFileName))
* [ExportImageFromPage](#_long_ExportImageFromPage(LPCTSTR_pF)
* [ExportImageFromTime](#_long_ExportImageFromTime(LPCTSTR_pF)
* [BatchExportImage](#_void_BatchExportImage(LPCTSTR_pFile)
* [ImportFile](#_long_ImportFile(LPCTSTR_pFileName,_)
* [ExportMetaFile](#_long_ExportMetaFile(LPCTSTR_pFileNa)

#### 标题脚注：

* [SetCurveTitle](#_void_SetCurveTitle(LPCTSTR_pCurveTi)
* [GetCurveTitle](#_CString_GetCurveTitle();)
* [SetFootNote](#_void_SetFootNote(LPCTSTR_pFootNote))
* [GetFootNote](#_CString_GetFootNote();)

#### 枚举（并修改）某条曲线：

* [GetDataHeadPosition](#_OLE_HANDLE_GetDataHeadPosition(long)
* [GetTimeData](#_double_GetTimeData(OLE_HANDLE_hCurv)
* [GetValueData](#_float_GetValueData(OLE_HANDLE_hCurv)
* [GetState](#_short_GetState(OLE_HANDLE_hCurve,_l)
* [InsertMainData](#_BOOL_InsertMainData(OLE_HANDLE_hCur)
* [InsertMainData2](#_BOOL_InsertMainData2(OLE_HANDLE_hCu)
* [CanContinueEnum](#_BOOL_CanContinueEnum(OLE_HANDLE_hCu)
* [DelPoint](#_void_DelPoint(OLE_HANDLE_hCurve,_lo)

#### 枚举所有曲线：

* [GetCurveCount](#_long_GetCurveCount();)
* [GetCurve](#_long_GetCurve(long_nIndex);)

#### 枚举图例：

* [GetLegendCount](#_long_GetLegendCount();)
* [GetLegend2](#_BOOL_GetLegend2(long_nIndex,_OLE_CO)
* [GetLegendAddressCount](#_long_GetLegendAddressCount(long_nIn)
* [GetLegendAddress](#_long_GetLegendAddress(long_nLegendI)

#### 联动相关：

* [SetBuddy](#_BOOL_SetBuddy(long_hBuddy,_short St)
* [GetBuddyCount](#_short_GetBuddyCount();)
* [GetBuddy](#_long_GetBuddy(short_nIndex);)

#### Z-Order相关：

* [EnableAdjustZOrder](#_void_EnableAdjustZOrder(BOOL_bEnabl)
* [SetCurveIndex](#_BOOL_SetCurveIndex(long_Address,_sh)
* [GetCurveIndex](#_short_GetCurveIndex(long_Address);)
* [SelectCurve](#_BOOL_SelectCurve(long_Address,_BOOL)
* [GetSelectedCurve](#_BOOL_GetSelectedCurve(long_FAR*_pAd)

#### 范围相关：

* [GetOneTimeRange](#_BOOL_GetOneTimeRange(long_Address,_)
* [GetOneValueRange](#_BOOL_GetOneValueRange(long_Address,)
* [GetOneFirstPos](#_BOOL_GetOneFirstPos(long_Address,_d)
* [GetTimeRange](#_BOOL_GetTimeRange(double_FAR*_pMinT)
* [GetValueRange](#_BOOL_GetValueRange(float_FAR*_pMinV)
* [GetViableTimeRange](#_void_GetViableTimeRange(double_FAR*)

#### 移动曲线：

* [DragCurve](#_short_DragCurve(short_xStep,_short )
* [VCenterCurve](#_BOOL_VCenterCurve(long_Address,_BOO)
* [GotoCurve](#_BOOL_GotoCurve(long_Address);)

#### 辅助函数：

* [IsSelected](#_BOOL_IsSelected(long_Address);)
* [IsLegendVisible](#_BOOL_IsLegendVisible(LPCTSTR_pSign))
* [IsCurveVisible](#_BOOL_IsCurveVisible(long_Address);)
* [IsCurveInCanvas](#_BOOL_IsCurveInCanvas(long_Address);)
* [EnableHelpTip](#_void_EnableHelpTip(BOOL_bEnable);)
* [CheckUpdate](#_long_CheckUpdate(BSTR*_pHomePage,_B)
* [GetPower](#_short_GetPower(long_Address);)
* [IsCurve](#_BOOL_IsCurve(long_Address);)
* [SetSorptionRange](#_void_SetSorptionRange(short_Range);)
* [GetSorptionRange](#_short_GetSorptionRange();)
* [GetActualPoint](#_BOOL_GetActualPoint(long_x,_long y,)
* [GetPointFromScreenPoint](#_long_GetPointFromScreenPoint(long_A)

#### 曲线长度控制相关：

* [GetCurveLength](#_long_GetCurveLength(long_Address);)
* [SetMaxLength](#_BOOL_SetMaxLength(long_MaxLength,_l)
* [GetMaxLength](#_long_GetMaxLength();)
* [GetCutLength](#_long_GetCutLength();)

#### 页数量变化事件（消息）：

* [EnablePageChangeEvent](#_void_EnablePageChangeEvent(BOOL_bEn)
* [ReportPageInfo](#_long_ReportPageInfo();)

#### 三维相关函数：

* [SetZLength](#_short_GetZLength();)
* [GetZLength](#_short_GetZLength();)
* [SetZOffset](#_BOOL_SetZOffset(long_Address,_short)
* [GetZOffset](#_long_GetZOffset(long_Address);)
* [SetLeftBkColor](#_void_SetLeftBkColor(OLE_COLOR_Color)
* [GetLeftBkColor](#_OLE_COLOR_GetLeftBkColor();)
* [SetBottomBkColor](#_void_SetBottomBkColor(OLE_COLOR_Col)
* [GetBottomBkColor](#_OLE_COLOR_GetBottomBkColor();)
* [SetReviseToolTip](#_BOOL_SetReviseToolTip(short_Type);)
* [GetReviseToolTip](#_short_GetReviseToolTip();)

#### 基点相关：

* [SetBenchmark](#_void_SetBenchmark(double_Time,_floa)
* [GetBenchmark](#_void_GetBenchmark(double*_pTime,_fl)

#### 填充方向相关：

* [SetFillDirection](#_BOOL_SetFillDirection(long_Address,)
* [GetFillDirection](#_short_GetFillDirection(long_Address)

#### 缓存控制相关：

* [ClearTempBuff](#_void_ClearTempBuff();)
* [PreMallocMem](#_BOOL_PreMallocMem(long_Address,_lon)
* [GetMemSize](#_long_GetMemSize(long_Address);)

#### 曲线操作函数：

* [TrimCurve](#_long_TrimCurve(long_Address,_short )
* [TrimCurve2](#_long_TrimCurve2(long_Address,_short)
* [PrintCurve](#_short_PrintCurve(long_Address,_doub)
* [OffSetCurve](#_BOOL_OffSetCurve(long_Address,_doub)
* [ChangeAddress](#_BOOL_ChangeAddress(long_Address,_lo)
* [CloneCurve](#_BOOL_CloneCurve(long_Address,_long )
* [UniteCurve](#_BOOL_UniteCurve(long_DesAddr,_long )
* [UniteCurve2](#_BOOL_UniteCurve2(long_DesAddr,_long)
* [UniteCurve3](#_BOOL_UniteCurve3(long_DesAddr,_doub)
* [UniteCurve4](#_BOOL_UniteCurve4(long_DesAddr,_doub)
* [ArithmeticOperate](#_long_ArithmeticOperate(long_DesAddr)

#### 其它函数：

* [SetFont](#_void_SetFont(OLE_HANDLE_hFont);)
* [GetCopyrightInfo](#_CString_GetCopyrightInfo();)
* [Refresh](#_void_Refresh();)
* [EnableFullScreen](#_void_EnableFullScreen(BOOL_bEnable))
* [EnableFocusState](#_void_EnableFocusState(BOOL_bEnable))
* [EnablePreview](#_void_EnablePreview(BOOL_bEnable);)
* [SetWaterMark](#_void_SetWaterMark(LPCTSTR_pWaterMar)
* [GetSysState](#_long_GetSysState();)

#### 限制坐标相关：

* [LimitOnePage](#_void_LimitOnePage(BOOL_bLimit);)
* [FixCoor](#_BOOL_FixCoor(double_MinTime,_double)
* [GetFixCoor](#_short_GetFixCoor(double*_pMinTime,_)
* [RefreshLimitedOrFixedCoor](#_BOOL_RefreshLimitedOrFixedCoor();)

## 属性及函数详细介绍：

### 属性（８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：

OLE\_HANDLE PageChangeMSG：

当曲线页数发生变化时发送此消息给父窗口，不为0即认为有效，页数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：

lParam等于0时wParam参数无效，表示没有可显示的曲线。

lParam等于1时wParam的低2字节为当前页前面的页数，高2字节为当前面页后面的页数，总页数为(wParam & 0xFFFF) + (wParam >> 16) + 1。

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

**特别注意：**

对于OLE\_COLOR型属性（注意，只能是OLE\_COLOR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，即后面3个字节，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，此时如果用户指定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。比如，我有很长一段时间在做服务器，服务器是随系统一起启动的，一般是无人值守的，在这样的情况下，最好不要弹出模式对话框。

### 函数（172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，纵坐标取值0到100，横坐标取值1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。

#### 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。

#### 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。

#### double **GetEndTime**();

#### 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面边沿的点删除掉（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）。

#### BOOL **SetBeginTime**(LPCTSTR pBeginTime);

#### BOOL **SetBeginTime2**(double fBeginTime);

#### CString **GetBeginTime**();

#### double **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double型数据来保存时间的，它与double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double，最后直接显示double型数据。

#### 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 **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 / 24 / 60 / 60;

//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 **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 **SetHPrecision**(short Precision);

#### short **GetHPrecision**();

当横坐标显示为值的时候，其显示精度，参看SetVPrecision。

#### void **TrimCoor**();

#### void **EnableAutoTrimCoor**(BOOL bEnable);

修整横纵坐标，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TrimCoor函数）。

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

#### void **GetVisibleCoorRange**(double\* pMinTime, double\* 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。

#### 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 Address, 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Address、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：

把Address地址添加到图例中（如果它不存在并且Mask最低位为1的话），并根据Mask来决定哪些值用来更新图例。

当要添加的图例（pSign）不存在时：

如果Address已存在（在别的图例中），则首先把Address从它所在的图例中删除，再以pSign为图例名称新建图例，并将Address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。

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函数，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，不在范围之内是不会出错的），参看CreateHatchBrush函数，颜色为BrushColor；

128-254－pattern brush样式，参看CreatePatternBrush函数，(BrushStyle - 128)即为位图序号（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。

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。

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 Address, 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Address、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 **GetLegend**(LPCTSTR pSign, OLE\_COLOR\* pPenColor, short FAR\* pPenStyle, short FAR\* pLineWidth, OLE\_COLOR\* pBrushColor, short FAR\* pBrushStyle, short FAR\* pCurveMode, short FAR\* pNodeMode);

获取指定图例的所有属性，如果不需要某些属性，则传入空指针即可。

#### 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）。

#### CString **QueryLegend**(long Address);

获取指定地址所属的图例标志，就是AddLegend时的pSign参数。

#### BOOL **DelLegend**(long Address, 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，Address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地址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 **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，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 **ShowLegend**(LPCTSTR pSign, BOOL bShow);

#### BOOL **ShowCurve**(long Address, BOOL bShow);

显示、隐藏指定的图例，或者曲线。其实ShowCurve函数也是根据Address找到相应的图例，然后调用ShowLegend函数的。

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

#### short **AddMainData2**(long Address, double 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。

特别注意这里的**Address**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。Address与AddLegend中的Address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)。

**Time**为时间，即横坐标。

**State**为该点状态，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。

**VisibleState**从低位起：

1－是否马上绘制添加的点（绘制实时曲线），如果当前只有一条曲线，并且这个点是该函数的第一个点，则自动设置该位为1，即当成实时曲线来处理

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）

如果正在绘制实时曲线（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，控制将自动更改为普通点（State等于0）。

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：

地址（4）+ 时间（8）+ 值（4）+ State（1）

这里的State与AddMainData函数的State是相同的，只是一个是BYTE型，一个是short型，因为COM接口里面没有BYTE这种类型，所有AddMainData(2)函数只能采用short型，而AddMemMainData函数既然是接收一个内存地址，则只需要为State分配一个字节的空间就足够了。该函数返回成功添加的点数量。

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17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。

下面，我将花一大篇幅来解释一个问题：如果当前横坐标开始值是2008-1-1，然后在添加曲线的时候（实时曲线），点的横坐标是200-1-1，会有什么结果呢（意思是当新添加点的坐标与控件当前坐标开始值相差很远时，会怎样）？结果就是，控件将自动移动自己的横坐标，以便让添加点能显示出来，这个功能是有用的，控件本身也是这样做的。但有些时候，好事没做好就成了错误，为了移动坐标，控件需要计算移动量（移动量是long型），这下大家明白了吧，当需要的移动量超过long型的范围时，移动将是不确定的（有可能移动方向都反了）！这个问题因为涉及到数据的存储容量的限制（就算用\_\_int64也会有溢出的时候），所以不太好解决，只能是大家在二次开发的时候去避免它，推荐的方法就是：当移动跨度很大的时候，调用完AddMainData2函数后，这样调用一下FirstPage函数：

FirstPage(TRUE, TRUE);

注意，频繁调用FirstPage也没什么好处，会影响效率，应该只在需要时调用。

同样的问题也会出现在纵坐标上，但大家要相信，要想让long型溢出，那跨度可能大到无法想象，或者是坐标间隔小到无法想象，实际使用时我认为极少见，可以不关心这个问题，等问题表现出来后再想办法解决。

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，这是我做人的原则：

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 Address, double BTime, double ETime,short Mask, BOOL bAll, BOOL bUpdate);

#### void **DelRange2**(long Address, 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Address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(Address, 6, 1, FALSE, FALSE);

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

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

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

M\_Curve.DelRange2(Address, 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**();

设置放大率，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。

#### void **EnableZoom**(BOOL bEnable);

是否允许缩放曲线。

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）。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）。所以在设置为禁止缩放时，以前的缩放大小仍然是有效的，二次开发者随时可以通过SetZoom(0)来恢复到不缩放状态。

#### 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－默认坐标系，X轴向上，Y轴向右，原点在左下角；

1－X轴向上，Y轴向左，原点在右下角；

2－X轴向下，Y轴向右，原点在左上角；

3－X轴向下，Y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位的设置无效）。

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（默认值）。

注意，显示为手型鼠标时，坐标提示将失效，除此之外，没有其它办法让坐标提示失效，也就是说，要让坐标提示失效，只能让鼠标在移动过程中显示手型。

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），这样设计的好处不言而喻，大家都是开发者，这个问题我不再多说。禁止曲线被移动，实际上是禁止了以下功能键：F5键、方向键、水平滚动、垂直滚动、鼠标拖动。

#### 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。

#### 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。

#### 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。

#### 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 **SetCanvasBkBitmap**(short nIndex);

#### short **GetCanvasBkBitmap**();

设置画布背景位图，就是绘制网格的区域。

#### BOOL **SetBkMode**(short BkMode);

#### short **GetBkMode**();

#### BOOL **SetCanvasBkMode**(short CanvasBkMode);

#### short **GetCanvasBkMode**();

背景位图（后面两个为画布的背景位图模式）的显示模式，取值如为：0－平铺；1－居中；2－拉伸；

#### BOOL **ExportImage**(LPCTSTR pFileName);

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

#### long **ExportImageFromTime**(LPCTSTR pFileName, long Address, double BTime, double ETime, BOOL bAll, short Style);

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

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

#### long **ExportMetaFile**(LPCTSTR pFileName, long Address, long nBegin, long nCount, BOOL bAll, short Style);

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，成功返回真。

第2－4个函数用于导出一部分图片到文件中。如果bAll为假，则只导出Address这一条曲线，否则导出所有曲线，此时Address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。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，后面所有函数生成的文件的文件名都依赖于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，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。

上面说的这一些，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，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，会默认的选择中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。

注意：ImportFile函数对于读到的点，将按VisibleState等于0调用AddMainData2函数，即该函数是无法绘制实时曲线的，因为批量添加点的时候，就算按实时曲线处理，也是一闪而过直接跳到了最后一页，根本还是看不到绘制过程，反而带来极大的开销，所以如果想绘制实时曲线的话，还不如调用ImportFile后，直接调用一下FirstPage函数，让曲线直接跳到末页，这样达到的效果是一样。

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，导入的时候也一样，但后面两种方法速度相差甚小，只有第一种方式与后面两方式在速度上有数量级的差别。

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，以免费内存占用无限的增长。如果让控件自动删除点（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。

#### void **SetCurveTitle**(LPCTSTR pCurveTitle);

#### CString **GetCurveTitle**();

设置曲线标题。

#### 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。

#### OLE\_HANDLE **GetDataHeadPosition**(long Address);

#### double **GetTimeData**(OLE\_HANDLE hCurve, long nIndex);

#### float **GetValueData**(OLE\_HANDLE hCurve, long nIndex);

#### short **GetState**(OLE\_HANDLE hCurve, long nIndex);

#### BOOL **InsertMainData**(OLE\_HANDLE hCurve, long nIndex,LPCTSTR pTime, float Value, short State, short Position);

#### BOOL **InsertMainData2**(OLE\_HANDLE hCurve, long nIndex,double Time, float Value, short State, short Position);

#### BOOL **CanContinueEnum**(OLE\_HANDLE hCurve);

#### void **DelPoint**(OLE\_HANDLE hCurve, long nIndex);

以上函数组用于枚举指定曲线中的所有点，并在枚举过程中操作曲线，具体如下：

GetDataHeadPosition获取曲线句柄，这是枚举的第一步，以后的函数均需要这个函数的返回值。

GetTimeData、GetValueData、GetState获取nIndex位置的时间、值及状态，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；

0：更改nIndex点，此时State的高字节有意义：按位算，从低位起，9－Time有效，10－Value有效，11－State有效（低字节）；

1：插入到nIndex之后。

注意，这两个函数不会对插入点按时间排序，不管是几次曲线。

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。

枚举过程中不要删除或者添加曲线点，在删除点的时候，可能会让迭代器失效，这是非常严重的（迭代器就是GetDataHeadPosition函数返回的那个句柄），还有不太严重的情况，就是在删除或者添加点之后，函数的nIndex参数可能错乱，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，另有甚者，如果你正要枚举第100个值的时候，第100个值及其以后的所有点被删除，此时枚举将出现严重错误，这种错误和迭代器失效的错误一样严重。

对于迭代器失效和nIndex越界，本控件内部是可以检测到的（nIndex错乱的情况检测不到），但控件不应该这样做，我所见过的所有的枚举，基本上都是要调用者去保证枚举的正确性的，可能是效率上的考虑吧。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，删除点操作不是不允许，只是要小心，使用对了就没事。

long pos = m\_ST\_Curve.GetDataHeadPosition(11);

if (pos)

{

if (!m\_ST\_Curve.CanContinueEnum(pos))

AfxMessageBox(\_T("枚举器已经失效！"));

else

{

CString str;

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

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

{

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

str.Format(\_T("%s, %f"),

((COleDateTime) m\_ST\_Curve.GetTimeData(pos, i)).Format(),value);

AfxMessageBox(str);

if (value > 92.0f)

{

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

--i; //当前i的位置删除了，所以要继续在这个位置上进行枚举

--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);

获取曲线的条数，GetCurve用于获取按序号表示的曲线的地址，枚举曲线会有用。

#### long **GetLegendCount**();

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

#### long **GetLegendAddressCount**(long nIndex);

#### long **GetLegendAddress**(long nLegendIndex, long nAddressIndex);

获取图例的个数，获取指定图例控制的地址个数，最后一个函数用于枚举地址，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。

GetLegendAddress的nLegendIndex参数就是GetLegendAddressCount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，有一个曲线被指定为服务器，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，这一组曲线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，任何时间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，并关闭与所有相关的联动客户机的连接

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

如果State == 0，则hBuddy看成联动客户机，并将其添加到当前控件。

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链表里删除。

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任意）。

删除联动服务器只能是对联动服务器调用SetBuddy（此时hBuddyo 0，State任意）这一种方法。

#### 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来调用本接口的话，任何曲线将无法被选中，如果已经有选中的曲线，则该曲线只能被取消选中，取消后，也无法再次被选中。

#### short **GetCurveIndex**(long Address);

#### BOOL **SetCurveIndex**(long Address, 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 **GetSelectedCurve**(long FAR\* pAddress);

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 **SelectCurve**(long Address, 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 **GetOneTimeRange**(long Address, double FAR\* pMinTime, double FAR\* pMaxTime);

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

#### BOOL **GetOneFirstPos**(long Address, double FAR\* pTime, float FAR\* 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**(double FAR\* pMinTime, double FAR\* pMaxTime);

#### BOOL **GetValueRange**(float FAR\* pMinValue, float FAR\* 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**(double FAR\* pMinTime, double FAR\* 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。

#### BOOL **VCenterCurve**(long Address, BOOL bUpdate);

按指定的曲线为中心垂直移动曲线，让Address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 **GotoCurve**(long Address);

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 Address);

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 Address);

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 Address);

判断曲线是否在画布上有显示。

#### 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。

#### long **CheckUpdate**(BSTR\* pHomePage, BSTR\* pVersion, BSTR\* pModifyTime);

检测控件是否有更新，这个函数直接调用后面在“导出方法”一栏说到的CheckUpdate函数，具体请往后看。

#### short **GetPower**(long Address);

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），如果曲线未找到则返回-1。

#### BOOL **IsCurve**(long Address);

查看Address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（Address）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，则吸附范围为如下一个正方形：

(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。

#### BOOL **GetActualPoint**(long x, long y, double\* pTime, double\* 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地址这样一个参数，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 Address, 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Address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Address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 **SetMaxLength**(long MaxLength, long CutLength);

#### long **GetMaxLength**();

#### long **GetCutLength**();

#### long **GetCurveLength**(long Address);

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。

#### void **EnablePageChangeEvent**(BOOL bEnable);

是否允许PageChange事件，如果PageChange是按消息方式发送的，则是否发送消息由m\_pageChangeMSG和m\_MSGRecWnd决定，只要m\_pageChangeMSG和m\_MSGRecWnd合法，就会发送消息，本函数只控制是否发送PageChange事件。

#### 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），返回 -1表示没有可显示的曲线，否则返回值的意义同PageChangeMSG消息的wParam参数。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，该事件的参数请参看后面对PageChange事件的说明。

#### 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，Z轴会缩小翻页时，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。具体影响表现在：ReportPageInfo、ExportImageFromPage、FirstPage、GotoPage等。

#### BOOL **SetZOffset**(long Address, short nOffset, BOOL bUpdate);

#### long **GetZOffset**(long Address);

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40.816663263917103（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涵多条曲线，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41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42、43什么的，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——35，假设Z轴长度为nZLength，那么当GetZOffset得到的X轴分量大于nZLength \* 35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）。

显示坐标提示的时候，按Z轴长度等于0处理，所以在三维状态下，最好取消坐标提示功能，否则本来坐标相同的点，显示出来的坐标将不相等，因为坐标提示里面没有Z轴的信息。

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（比如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，每个坐标必须是要有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 **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 **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

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，则按选中曲线的Z坐标来校正坐标提示，当没有选中曲线，或者选中曲线不在画布中时，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，默认为3。由此可见，如果按方法3来处理坐标提示的话，一定要开启吸附效应，否则很难刚好移动到某个点之上，坐标提示也就经常的不会弹出来。

#### void **SetBenchmark**(double Time, float Value);

#### void **GetBenchmark**(double\* 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。

#### BOOL **SetFillDirection**(long Address, short FillDirection, BOOL bUpdate);

#### short **GetFillDirection**(long Address);

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，默认设置填充方向向下填充，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 **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。

#### BOOL **PreMallocMem**(long Address, long size);

#### long **GetMemSize**(long Address);

为曲线预分配空间，size指的是点的个数，如果你预计某曲线的点数量不会超过100个点，则使用这个函数会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。

注意：PreMallocMem函数需要在曲线已被添加之后调用，由于本控件没有提供接口用于添加一条曲线，只提供了添加曲线点的接口，添加点的时候，如果曲线不存在，将添加曲线，所以只要在成功调用一次AddMainData(2)函数之后，就可以调用本函数了，当然你随时都可以调用本函数，但是应该越早调用越好，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小于当前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，而控件里面也是使用的vector）。

如果size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。

#### long **TrimCurve**(long Address, short State, long nBegin, long nCount,short nStep, BOOL bAll);

#### long **TrimCurve2**(long Address, short State, double BTime, double 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Address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Address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 Address, double BTime, double ETime, short Mask, short LeftMargin,short TopMargin,short RightMargin,short BottomMargin, LPCTSTR pTitle, LPCTSTR pFootNote, short Flag, BOOL bAll);

打印曲线，Address为要打印的曲线的地址，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。

如果bAll为真，则Address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。

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。

#### BOOL **OffSetCurve**(long Address, double 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) + '\*');

#### BOOL **ChangeAddress**(long Address, long NewAddr);

更改曲线的地址。

#### BOOL **CloneCurve**(long Address, long NewAddr);

复制一条曲线，从Address复制到NewAddr，复制结束后不会删除Address曲线。

#### BOOL **UniteCurve**(long DesAddr, long nInsertPos, long Address, long nBegin,long nCount);

#### BOOL **UniteCurve2**(long DesAddr, long nInsertPos, long Address,double BTime, double ETime, short Mask);

#### BOOL **UniteCurve3**(long DesAddr, double fInsertPos, long Address, long nBegin,long nCount);

#### BOOL **UniteCurve4**(long DesAddr, double fInsertPos, long Address,double BTime, double ETime, short Mask);

将Address曲线的指定范围之内的点，插入到DesAddr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据的点，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DesAddr的最前面，插入的次序按点在Address曲线里面的次序为准。至于fInsertPos参数，意思是插入到DesAddr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），所有第一个点就大于等于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Address曲线的范围，而非DesAddr曲线的范围。插入结束后，不会自动删除Address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）。

#### long **ArithmeticOperate**(long DesAddr, long Address, 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DesAddr和Address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DesAddr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DesAddr里面的点，都将要遍历Address曲线一次，效率太差。而且，如果某一个DesAddr里面的点，其横坐标对应多个Address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); //删除临时曲线，这一步与上一步可对调

下面说说如何获取到一个没有使用的地址，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地址，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 TempAddr = 0x7FFFFFFF; //最大正值

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

到这里后（退出了while循环），TempAddr即为所求。

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

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颜色或字体，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不方便，最好是程序直接生成合法的字体。

#### CString **GetCopyrightInfo**();

获取版权信息，一个字符串。

#### 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。

#### 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。

#### void **EnableFocusState**(BOOL bEnable);

默认情况下，控件在获得焦点的时候，会绘制一个边框以示意，二次开发者可以去掉这个边框，这样做至少有一个用处，就是二次开发者想将控件和另外一个控件紧靠在一起的时候（达到界面上的某些视觉要求），有边框显得有隔阂。

#### 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、EnableAdjustZOrder、EnableAutoTrimCoor、EnableHelpTip、EnableFullScreen、EnableFocusState、EnablePreview。

只使用了低8位。

#### 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**(double MinTime, double MaxTime,float MinValue, float MaxValue, short Mask);

#### short **GetFixCoor**(double\* pMinTime, double\* 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，所以有效性还是要接返回的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。

#### 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（EnableAdjustZOrder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。

三：事件（3个）

#### MouseDown(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);

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

前面两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，你会爱上这两个事件的。这两个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\_Curve.ClientToScreen(&point);

Button用于标识左中右键，自己试一下就知道了。

对于PageChange事件，其参数与发送PageChangeMSG消息时的wParam和lParam参数意义完全一样。至于为什么要通过消息和事件两种方式来触发PageChange，是考虑到容器没有消息泵的情况。

### 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

#### Extern "C" int \_\_stdcall CheckUpdate(BSTR\* pHomePage, BSTR\* pVersion, BSTR\* pModifyTime);

检查控件是否有新版本，返回：

-2 未知故障，官方主页可能被黑了；

-1 网络故障，比如说电脑没有连接互联网；

0 成功，但无更新；1 成功，且有更新。

控件根据修改时间来判断控件是否有新版本，只要本地控件的修改时间与官方主页上的修改时间相差大于等于1小时，就认为是有更新，并没有限制非要本地控件的修改时间早于官方主页上的修改时间才认为是有更新，因为考虑到本地控件可能被人为的修改（修改了控件的修改日期）。目前暂时不支持自动下载更新，所以在有更新的时候，需要二次开发者从我的官方主页上去手动下载。

注意，修改时间是UTD时间，所以，如果你在中国，通过CheckUpdate得到的修改时间会比从windows浏览器得到的修改时间早8小时。返回的BSTR，调用者要负责释放，比如：

BSTR HomePage, Version;

long re = m\_ST\_Curve.CheckUpdate(&HomePage, &Version, 0); //不需要者传入空

if (HomePage) //对于取不到的值，控件会置空，所以这里只需要判断是否为空即可

{

CString str = HomePage;

SysFreeString(HomePage);

AfxMessageBox(str);

}

if (Version)

{

CString str = Version;

SysFreeString(Version);

AfxMessageBox(str);

}

if (re > 0)

AfxMessageBox(\_T("有更新，请从www.st-curve.cn下载最新控件。"));

else if (re < 0)

AfxMessageBox(\_T("网站或者网络故障！"));

如果要将ST\_Curve.ocx当成一个动态库来使用（以便调用上面说到的三个方法），我必须多说几句，这些函数和dll的导出函数完全一样，如果是通过ST\_Curve.Lib隐式加载，则一定要注意路径问题，加载过程是windows标准加载过程，首先看容器所在的目录里面有没有ST\_Curve.ocx，如果有就加载，没有就在系统目录和环境变量里面找，如果仍然没有，则会失败。而控件的加载是从注册表里面获取路径的，比如将ST\_Curve.ocx放到C盘根目录下，只要注册成功，那么控件就是可加载的。但当成dll来加载ST\_Curve.ocx的时候，情况就不一样了，如果容器没有在C盘根目录下，C盘根目录又不在环境变量里面，则加载ST\_Curve.ocx会失败。要解决加载问题，大家可能马上想到，将ST\_Curve.ocx拷贝到容器的相同目录之下，这个方法的确可行，但对于CheckUpdate函数就要小心了，这个函数可以检测是否有更新，检测的依据大家都知道，就是将自己的修改时间与官方主页上的最新修改时间做比较，那么细心的读者可能会发现，这个“自己”到底是谁呢？注册过的ST\_Curve.ocx可能会与当成dll加载的ST\_Curve.ocx不是同一个文件（大家都知道，dll是可以存在多个的，只要不在同一个目录下，而控件只能存在一个，因为它要注册，后面注册的会覆盖前面注册的），本控件在处理这个问题的时候，步骤是这样的：如果控件已注册，则将这个已注册的ST\_Curve.ocx作为是否有更新的对比依据（此时不管它与当成dll来加载的这个ST\_Curve.ocx是否是同一个文件），若控件未注册，则将当成dll加载的这个ST\_Curve.ocx作为是否有更新的对比依据。

### 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，因为要追求速度），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。

### 请初学者或是对编程不太精通者往下看

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。(C) 2009 young wolf. All rights reserved.

Home: http://www.ST-Curve.cn

Email: [Mail2tao@163.com](mailto:Mail2tao@163.com) [web@ST-Curve.cn](mailto:web@ST-Curve.cn.cn)

QQ: 676218192

Mobile: 13290958750（福建福州）

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：1.3.1.3。

### 更新记录（从2009年1月份开始记录，以前的记录删除掉了）

2/19/2009

增加了快捷键F6，用于显示或者隐藏全局位置窗口；

在全局位置窗口里面增加了对鼠标右键点击的响应（隐藏窗口）；

升级控件版本到1.3.1.3。

2/4/2009

为控件增加了GetSysState接口，由于获取由Enable系列接口设置的属性值，具体看开发文档。

2/2/2009

为控件增加了简单的水印功能，参看SetWaterMark接口；

修改了一个BUG，此BUG会造成在更改前景颜色时，帮助信息和水印颜色得不到更新。

2/1/2009

修改了打印函数，让其更合理，以前的做法是：标题和脚注通过PrintCurve函数用参数来传递，现在改为控件本身的标题与脚注照打，而PrintCurve函数传入的标题与脚注的处理方法没有改变。另外，打印时增加了文字颜色与白色（白色是打印纸颜色）的相似度的判断，如果太相似，就取反色来打印，这样不管如何设置颜色，打印结果都不至于不可见；

修改了一个BUG，此BUG会造成在全屏打印时，误将控件的标题与脚注打印了出来，这与屏幕上的显示效果不一致；

将AutoTrimCoor接口更名为EnableAutoTrimCoor，将SetFullScreen接口更名为EnableFullScreen，这主要是为了接口名字的统一，不属于接口改变（接口只是一个地址，具体的名字用处不大，可以认为是给程序员用的），所以老的工程不需要重新编译即可使用最新版本的控件；

升级控件版本到1.3.1.2，这纯粹是为了在门户网站上更新控件，其实改动非常小。

1/29/2009

完善了开发文档；

增加了EnablePreview接口，用于显示或者隐藏全局位置预览窗口，全局位置预览窗口是一个在画布中的小窗口，里面用一个小方块来代表当前窗口在整个画布里面的位置，操作员还可以通过在全局位置预览窗口中点击鼠标左键来快速移动曲线；

升级控件版本到1.3.1.1。

## ST\_Curve升级报告

至2008年7月24日，控件自诞生以来第二次大的升级基本完成（第一次是将所有数据的内存管理交给了STL），这次升级的主要内容是增加了曲线的绘制速度，以及减少了CPU资源的占用量，当然内存占用会更多一些。ST\_Curve自编写第一行代码之前，我就知道，这个控件的优势将是随心所欲的拖动曲线，缩放曲线，以及绘制速度。在曲线的拖动过程中，曲线不得不一次又一次的重绘，以前的做法是，每次重绘的时候，都去计算屏幕坐标，这样当然效率就会差一些，于是我不得不从其它一些地方来弥补（这些弥补，不一定眼睛可以看到，但至少可以减少占用的CPU资源，比如CPU从原来的占用%2到占用1%，用眼睛是感觉不到的，但提高效率是每一位程序员的责任），比如我将每条曲线在画布中的第一个可见点的位置保存了下来，这样重绘的时候，就从这一点开始即可；我又将曲线分为1次曲线和2次曲线，如果是1次曲线，则绘制到第一个画布之外的点的时候，马上退出绘制（2次则要绘制到曲线结束），于是更新每一条曲线在画布中的第一个可见点变得非常麻烦，而且更新非常的频繁（曲线经常被拖动，那么第一个可见点经常要改变）。为了解决这一系列问题，在这一次升级中，我将曲线的屏幕坐标也保存了下（这就带来了每一点8个字节的额外开销），这样曲线在重新绘制的时候，不需要计算屏幕坐标，而曲线的移动过程中，可以直接对保存下来的屏幕坐标做一个偏移，这些都是整数运算，相对于以前的double型加减乘除，速度自然提高许多，具体请往下看。

### ST\_Curve计算屏幕坐标的内部原理详解

一：

首先找到一个基点（double, float），（参看SetBenchmark函数，控件自动将第一条曲线的第一个点的坐标当成基点，基点最好不要修改，但有些时候又必须修改，请往后看），然后对于当前坐标原点（double, float），根据每个刻度所代表的间隔（参看SetTimeSpan和SetValueStep函数），以及每个刻度所代表的屏幕像素个数（横坐标35个，纵坐标21个，不能修改），就可以得到原点到基点的屏幕距离（long, long）；

二：

对于所有曲线的所有点，按上面的方法，得到每个点到基点的屏幕距离（long, long），将得到的距离（long, long）当成坐标来看待，于是可以映射到画布坐标系（结合原点到基点的屏幕距离，映射过程只有对整数的加减），得到每个点在画布中的坐标（long, long），并将所有这些坐标保存下来，在绘制曲线的时候，直接取出来使用，不需要做任何处理；

三：

当移动曲线的时候（参看DragCurve函数），对当前坐标原点，以及所有曲线的点的屏幕坐标，直接偏移一个距离即可（均为整数的加减）。

四：

当改变当前坐标原点的时候（参看SetBeginTime、SetBeginTime2、SetBeginValue），那么对原点做第一步操作，并记录原点到基点的距离（long, long）在原点改变后，相对于改变之前的偏移量（均为整数），有了这个偏移量之后，对所有曲线的所有点，直接应用这个偏移（均为整数的加减），就得到了所有曲线的新的绘制位置（屏幕坐标）；

五：

当画布（画曲线的地方，不包括坐标区域）大小发生改变时（比如控件窗口大小改变，纵坐标显示精度改变，纵坐标显示范围改变等都有可能引起画布大小的改变），记录画布改变后相对于改变前的偏移量，对所有曲线的所有点，直接应用这个偏移（均为整数的加减），就得到了曲线的新的绘制位置（此时原点到基点的距离不用重新计算）；

六：

当坐标的一个刻度所代表的间隔发生改变时（参看SetTimeSpan和SetValueStep函数，SetZoom也会产生相同的效果），必须重新做第一、二步；

七：

大家可以看到，对曲线的缩放（调用SetTimeSpan和SetValueStep函数也是缩放的一种特例），要远远慢于对曲线的移动，知道了这些特性，有助于大家更好的使用本控件；

八：

溢出处理，从上面的坐标计算可以看出，溢出是可能的，但要想让long溢出，是很难的，前提是基点要找好，一个理想的基点是容纳所有曲线的那个最小的矩形的中心点。实际使用时并没有必要去找这个理想的基点，但至少要在曲线附近。溢出只是理论上的，现实中可以不用关注这个问题。如果基点选择没有问题，而仍然发生了溢出，那么曲线的长度已经超过了本控件的允许范围，就无能为力了！

九：

关于SetBenchmark函数，提供它只是让控件尽量的完整，其实基本上不使用，控件会自动将第一条曲线的第一个点的坐标当成基点，这个函数的唯一用处就是，如果曲线长度超过控件的允许范围，但还没有超过2倍范围的时候，将基点设置成“理想基点”（参看上面第八条），可以消除溢出。但当长度超过曲线允许范围的2倍时，也就再也无能为力了。

本函数会重复第一、二步，没事可千万别调用。

十：

曲线的允许长度（在屏幕上的长度，而不是指有多少个点）可以由下面的公式大概计算：

maxwidth = (maxtime - Benchmarktime) / timespan / (zoom > 0 ? (zoom \* .25f + 1) : (-zoom \* .25f + 1)) \* 35;

maxheight = (maxvalue - Benchmarkvalue) / valuestep / (zoom > 0 ? (zoom \* .25f + 1) : (-zoom \* .25f + 1)) \* 21;

只要上面的maxwidth和maxheight不超过long数据类型的限制，就可行。

十一：

上面八提到的long数据溢出可解决吗？表面上看，只要记下发生溢出的点（发生了溢出的点，一定不在画布之内），等显示到它的时候（通过翻页或者拖动曲线），再计算屏幕坐标不就行了？其实并不是这么简单，比如：如何知道溢出的点已经处于画布之中了呢？计算得到的屏幕坐标，因为已经溢出了，显然是不能用的，那么只能通过原始坐标来判断，此时又退化成了double数据的加减乘除了！另外还有一个技术难题，这涉及到控件的内部实现原理，控件保存了每一条曲线的最小坐标和最大坐标（参看GetOneFirstPos、GetOneTimeRange和GetTimeRange函数），以及他们的屏幕坐标，这两个屏幕坐标可以迅速的判断曲线在画布中有没有显示，如果没有显示，绘制的时候直接跳过，以加快绘制速度。控件还保存了所有曲线最小坐标和最大坐标，以及他们的屏幕坐标，这两个屏幕坐标可以防止曲线被完全拖出画布。现在回到说溢出的问题，如果有溢出发生，则上面提到的那些最小坐标和最大坐标会最先溢出，他们一但溢出，所有的判断将会出现问题，曲线的拖动也会因为判断错误而出问题。

### 此次升级，控件增加的功能概述

一：

填充方面有所增强，可向四个方向填充，而且同时可向多个方向填充。对于封闭的2次曲线（封闭的曲线一定是2次的），可对其内部填充。

二：

去掉了所有关于曲线次数的接口，但控件内部仍然是区分1次曲线和2次曲线的。0次和1次曲线都归为1次曲线，2次及其以上的曲线归为2次曲线。区分曲线次数是为了绘制速度，比如一条曲线有100年长，而显示在屏幕上的总是那么几个点，对于1次曲线，在绘制过程中，当出现第一个画布之外的点时，就退出绘制；而2次曲线必须要绘制到曲线结尾，因为曲线会折回，这样显然效率不如1次曲线。

现在去掉了关于幂次的接口，不但方便了二次开发者，同时也避免了出错的可能，因为开发者有可能误将1次曲线设置成了2次曲线，那样将影响绘制速度，或者误将2次曲线设置成了1次曲线，那样显示将不正常。

三：

去掉了AddMinDataOver和AllAddDataOver接口，表面上看只是减少了两个接口而已，其实远没有这么简单，这要涉及到控件内部的一些东西。上面的《ST\_Curve计算屏幕坐标的内部原理详解》中第十一条，大家还记得吧，控件内部保存了每一条曲线占的矩形区域，以及所有曲线占的矩形区域，AddMinDataOver和AllAddDataOver接口正是用于计算这些区域的，如果让开发者来调用它们，那么开发者如果忘记了呢？这将带来严重的问题，比如看见曲线在那里，就是拖不动，翻不了页等。为此，此次升级，控件将在内部自动调用这两个函数（去掉了它们的接口属性，现在只是两个内部函数），那么控件内部是如何做到的呢？

控件内部用了一个链表来保存需要调用AddMinDataOver函数的曲线的地址（那么哪些曲线需要调用AddMinDataOver函数呢？答案是，按VisibleState为0调用过AddMainData和AddMainData2函数的曲线，调用过InsertMainData和InsertMainData2函数的曲线），在曲线需要被绘制时，对链表里面的所有曲线调用AddMinDataOver函数，完了后再调用一次AllAddDataOver函数，当然这样做以后，会清空链表，不用担心效率问题，只会执行一次。

我之所以透漏这么多控件内部的实现原理，是要大家明白一点，在曲线没有得到首次刷新之前，调用一些范围函数会出现问题，比如：GetOneTimeRange、GetOneValueRange、GetTimeRange、GetValueRange、FirstPage、GotoPage、ReportPageInfo等等函数，大家可以看看demo里面的演示“方案五”的代码，在代码后面，特别添加了Refresh函数调用，就是为控件创造机会来调用AddMinDataOver和AllAddDataOver函数，否则最后面的FirstPage将达不到想要的结果，那么“方案一”至“方案四”为什么不需要Refresh呢？因为“方案五”里面的曲线是2次的，最小时间不在曲线头，调用Refresh就是要找到最小时间（当然还包括最大时间），以便翻到首页。而“方案一”至“方案四”由于都是1次曲线（至少第一条曲线是1次的，也是屏幕位置最靠前的，这就足够了），最小时间刚好是曲线头，所以歪打正着，也就刚好达到要求，如果要翻页到最后一页，则“方案一”至“方案五”都将达不到要求（不调用Refresh函数，而曲线又长于一页的的话），大家不防一试。

当然，去掉了AddMinDataOver和AllAddDataOver接口，也可以防止开发者忘记对他们的调用，对开发者也方便了，现在问题顶多是在必要的时候，要调用一下Refresh函数而已，其实要让控件刷新，有很多方法，比如将控件覆盖一下也可，因为大家都是开发者，这个问题我就不多说了。

四：

导出图元文件时，对于文本格式，里面的时间的表达方式增加了标准日期时间格式，以前只能按浮点数格式导出。

### 关于一些网友的建议

当然首先我要感谢那些给我提出意见建议，发现BUG的网友们。下面我将对于这些意见建议，不管我实现与否，都做一个解答：

一：

只让坐标显示一定范围之内的值，这个建议已经实现，开始我不大愿意做这个功能，因为我觉得控件的优势之一就是拖动，现在却要固定住坐标，有点不情愿，但后来想想，这在某些时候是还是有用的。

当然，在使用过程中，由于全部是浮点数比较，最好将范围设置得比想要的范围稍微大一点点（往两边都拓展一点点），只要不大到下一刻度的位置不小到上一刻度的位置，效果将是一样。

二：

让标题、脚注、正文、图例的字体与颜色独立，这一点我没有做，非常的抱歉。因为管理如此多的字体句柄和颜色的确是太麻烦，特别是多个字体的时候，字体的变化会影响到所有不以（0, 0)坐标为参照物的文字的绘制，比如横坐标，它是以画布底部为参照物的，目前我的做法是，只设置一个字体，对于标题，适当的增大字体，并用粗体，颜色不变。对于脚注，字体不变，颜色为前景色的四分之三，对于帮助信息，字体不变，颜色为前景色的二分之一。这样在灵活性与复杂度之间，我认为是找到了一个平衡。也许，正是由于控件在字体以及颜色方面的粗糙，说明它不是一个文字编辑器，而是一个绘制曲线的工具！

三：

控件在非XP样式下弹不出来坐标提示框，这个问题我至今还没有解决，为此我使用了另外一种方法，由我自己来实现一个类似tooltip的东西，目前看来效果还不错。

young wolf

2008年7月26日

## 控件内部GDI部分技术文档（部分）

一：

采用双缓冲，两个内存兼容DC，一个前景，一个背景。每当有刷新的时候，会将刷新应用到前景DC（结合背景DC），那么在窗口需要刷新的时候，只需要一条BitBlt函数，下面是控件的OnDraw函数（就一行代码）：

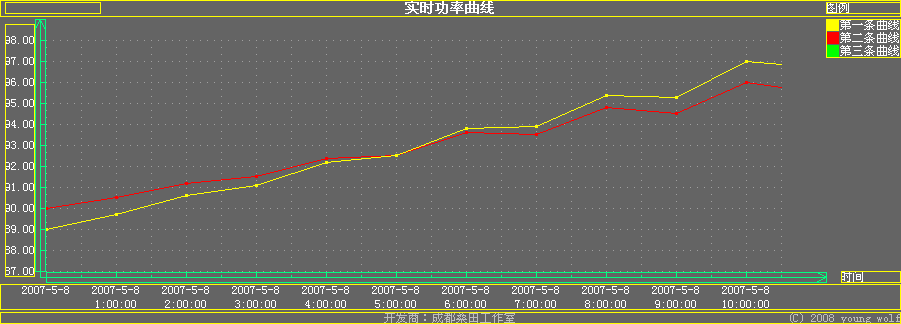
BitBlt(pdc->m\_hDC, rcInvalid.left, rcInvalid.top,

rcInvalid.Width(), rcInvalid.Height(),

hFrceDC, rcInvalid.left, rcInvalid.top, SRCCOPY);

二：

区域刷新技术，下图为分的区域，可见已经很详细了，区域刷新是为了节省CPU，对防止闪烁帮助不大。



关于区域刷新，请看我在CSDN中的技术文章：

http://blog.csdn.net/yang79tao/archive/2008/07/31/2743184.aspx

三：

尽量使用能够连续绘制的GDI函数，这也是为了节省CPU资源，具体为：

在绘制坐标值时，采用了PolyTextOut函数；在绘制坐标刻度时，采用了PolyPolyline函数；在绘制曲线的时候，采用了Polyline函数，在填充的时候，使用Polygon函数以及ExtTextOut函数（后者用于填充节点、填充图例前面的颜色示意块）。这些函数大多都能一次调用，执行一批绘制命令，所以节省了CPU资源，比如本控件中，一次调用PolyTextOut函数，就可以将横坐标或者纵坐标的刻度值全部显示出来，一次调用PolyPolyline函数，就可以将横坐标或者纵坐标的刻度全部显示出来。

当然，一个永远的事实是，要提高效率，要么提高运行速度，要么以空间换取时间，我们不可能去提高CPU的运算速度，所以只能以空间换取时间，所有这些一次调用执行多条GDI指令的函数（姑且这样说吧），是要付出更多的内存空间的。

## FAQ

### 什么是ST\_Curve？

ST\_Curve是一个专业的曲线绘制控件，只要是xy坐标系的曲线，都可绘制，纵坐标只能显示为值，横坐标可以显示为值或者时间。

### ST\_Curve面向对象？

二次开发者，即你必须要对ST\_Curve进行开发，它才能使用，就像现在流行的诸多界面库一样。

### ST\_Curve有什么使用与分发限制？

使用无任何限制，分发时，必须保证控件的完整性，包括但不限于主控件、开发文档等。

### ST\_Curve的特点与优势（对操作者）

鼠标任意拖动曲线，任意缩放曲线，定点（鼠标取点）缩放曲线，数字键快速找到曲线，F5键居中曲线，F4键显示隐藏帮助（帮助显示于背景图之上），支持方向键，翻页键等，可显示三维效果，可显示全局位置预览窗口并通过它快速移动曲线。

### ST\_Curve的特点与优势（对开发者，只说说几个重点，其它的请看开发文档）

自定义各种颜色，字体，背景，画笔，填充模式等，可切换坐标原点到窗口四个角的任意一个，可导出图片（bmp、png、jpg、gif），可批量导出图片，以及定时导出图片，可导入图片作为背景或者用于填充曲线，可导出曲线到文件（支持二进制文件，它速度最快，还支持文本文件，它的好处是可以手动编辑，支持的文本格式包括：ansi、unicode、unicode big endian、utf8，这些文本格式都被windows支持），可绘制实时曲线（曲线自动滚动），支持在各个方向上对曲线进行填充，支持打印，打印或者导出图片时，可指定只对某一条曲线，或某一段数据。

### ST\_Curve的开发环境与运行平台

控件的开发环境为vc7.1，unicode（所以控件无法加载时最可能的原因就是缺少mfc71u.dll），二次开发环境则是所有可以使用控件的地方，运行平台为win2000及其以上的操作系统，如果需要在win98下面使用，请向我索取mbcs版本的控件，如果要在纯英文操作系统下使用，请向我索取英文版控件，都不需要付费，但只提供release版本。

### 下载页面说明

注意：如果控件做的改动较小，则一般不会更新版本号，所以确定你自己的控件是否是最新，应该是看文件的修改日期。下载的时候，直接用下载工具下载，或者直接点击连接下载，不要在连接上右键选择“另存为”来下载，这样网站的防盗链功能将生效，下载下来的结果是一个gif图片，这个问题我也觉得很烦，但我的域名供应商就是这样做的，我也没有办法！

### 初学者请关注这一栏

考虑到初学者可能会缺少一些运行库，或者无法编译demo源代码，我特意做了一个安装包，大家下载下来后，只要安装即可，完了后会在开始菜单里创建一个指向开发文档和demo工程文件的快捷方式。在卸载的时候，可以卸载完全的，包括注册表里面的注册项，不用担心留下垃圾。

下载：安装包 <http://www.st-curve.cn/ST_Curve.exe> （注意，并没有包括升级报告，因为升级报告以后不会再有更新）

### 更新事宜及开发举例：

请大家定时调用控件的接口CheckUpdate来确定当前是否有最新的控件，如果有，强烈建议下载并覆盖现有的控件，也建议大家重新将控件添加到工程，因为有可能接口会改变，大家多注意一下开发文档后面的修改日志，接口修改肯定会有说明的。

下面我简单讲一下控件的开发流程，以帮助大家迅速的看到效果，更详细的开发需要你仔细的阅读开发文档。 首先是调用AddLegendHelper函数添加图例，然后调用AddMainData函数添加曲线点（这两步的先后关系无所谓），比如：

[c/c++ code]

m\_ST\_Curve.AddLegendHelper(11, \_T("第一条曲线"), (long) RGB(255, 255, 0), PS\_SOLID, 1, FALSE);

COleDateTime Time = COleDateTime::GetCurrentTime();

for (int i = 0; i < 5; i++)

{

m\_ST\_Curve.AddMainData2(11, (double) Time, 90 + .6f \* i, 0, 0, TRUE);

Time += 1.0 / 24 / 2;

}

[c/c++ code]

至此，如果你可以看到曲线，那么ST\_Curve控件就是我写的，如果看不到，我也不知道是谁写的！

考虑到初学者老是问如何用COleDateTime和COleDateTimeSpan表达浮点数，我这里给一个统一的解释：

COleDateTimeSpan代表一个时间间隔，它是用浮点数来表达的，所以他们可以隐式转换（COleDateTimeSpan重载了operator double），表达的方式是1.0代表一天，其它时间间隔以此类推，比如：

1小时 = 1.0 / 24

1分钟 = 1.0 / 24 / 60

1秒钟 = 1.0 / 24 / 60 / 60

0.5秒钟 = 1秒钟的二分之一

……

COleDateTime代表一个日期，从1899.12.30开始，即1899.12.30等于0，如果是1899.12.31呢，当然就加一天，而一天刚才说过了，等于1.0，所以1899.12.31就是1.0。

可以直接将double数据赋给COleDateTimeSpan或者COleDateTime，可隐式转换。

### 位操作

所谓位操作，是指的是二进制的位，本文档都是从低位起算，比如：

第几位  对应的二进制  对应的十六进制  对应的十进制

1        0001          0x1             1

2        0010          0x2             2

3        0100          0x4             4

4        1000          0x8             8

如果是1、3位为1呢，就是：0101  0x9  9

如果是2、4位为1呢，就是：0x1010  0xA  10

## 鸣谢（排名不分先后）

感谢北京的Ordie大哥，不管是在控件的编制过程中还是在我平常的工作中，都给予我很大的帮助，在此表示感谢；

感谢屋只同学帮我发现一些BUG，并向自己的同学推荐本控件；

感谢友情水帮我发现一些BUG，并提出一些宝贵意见，比如将曲线数据保存到图元文件就是他的主意；

感谢海之子提出一些意见；

感谢北京的白大庆帮我做了这个网页；

感谢北京的程威帮我发现一些打印过程中的BUG，也提出一些意见；

感谢天空、华军、中关村、硅谷动力和霏凡下载站为大家提供免费的软件发布空间；

感谢北京的DaLiang朋友对开发文档做了重新排版；

感谢所有正在使用或者即将要使用本控件的人。

## 联系方式及留言本



Email：[web@st-curve.cn](mailto:web@st-curve.cn)  
                [mail2tao@163.com](mailto:mail2tao@163.com)  
      QQ：676218192

ass-icon3 132 9095 8750  
   邮编：350001

留言本： <http://hi.baidu.com/yangl79>