计算机开发与制作实例丛书

即插即用
即学即会

Visual C++ 6.0

数据库与网络开发实例

清汉计算机工作室　编著

机械工业出版社
China Machine Press

计算机开发与制作实例丛书

Visual C++ 6.0 数据库与网络开发实例

清汉计算机工作室　编著

机 械 工 业 出 版 社

本书是以实例的方式来介绍 Visual C++6.0 网络和数据库应用程序开发和技巧的图书。书中给出了 17 个非常有实用价值的开发实例，内容基本上覆盖了 ODBC API、DAO、对话框数据库查询、超级链接的建立、COM、ATL、HTTP、SMTP、WinSoket、MAPI、电子邮件和网络浏览器的开发、编程方法和技巧。

本书的最大特点是实用性强，书中每个实例基本上都提供了可以直接引用的 C++类，把对网络和数据库的操作封装起来，以方便使用 Visual C++6.0 的读者直接引用，开发自己需要的多媒体应用程序。

本书可供所有从事 Visual C++6.0 网络和数据库应用程序开发的技术人员阅读和使用，也可供广大计算机用户及学习网络和数据库应用程序开发的读者参考。

机械工业出版社（北京市百万庄大街 22 号　邮政编码 100037）
责任编辑：王听讲　李铭杰　　封面设计：姚　毅
责任印制：路　琳
北京市密云县印刷厂印刷·新华书店北京发行所发行
2000 年 8 月第 1 版第 2 次印刷
787mm×1092mm 1/16·　30印张·738 千字
5 001— 7 000　册
定价：52.00 元　　（1CD，含配套书）

新出音管[1999]438 号
ISBN 7-980039-75-0/TP·20

前　言

随着我国计算机应用的普及与发展，各种不同用途的应用软件不断被开发出来，并迅速为人们掌握和使用。目前，除了专业软件开发人员外，许多一般的计算机用户也已经开始自己动手开发一些应用程序。计算机的应用开发工作包括很多方面，例如：图形图像处理开发、动画制作、专项应用程序开发、通用应用程序开发、游戏软件的制作等等。开发这些项目需要使用各种各样的开发工具，以便能够大大地缩短开发周期并减少开发费用。因此，很多从事计算机应用开发工作的人员迫切需要了解和掌握各种计算机编程和制作的方法。但目前市场上的这类图书主要是介绍开发工具的使用，并没有大量的应用实例与之配套，这与只有课本而没有习题集的情况类似。为适应广大读者在学习软件的同时需要参考大量实例的要求，我们特地组织了这套“计算机开发与制作实例丛书”。

这套丛书主要是针对应用程序开发者而编写的，它以各种实例的方式，向读者介绍了计算机编程和制作方面的知识，力求使读者能够轻松地学习开发的方法并熟练地掌握、使用相关的开发工具。对于需要编制应用程序或利用计算机进行开发的用户，这套丛书不仅是一个开发指南，而且还给出了其他图书很少涉及的开发技巧。通过本丛书中介绍的一个个具体的例子，用户便可以比较容易地掌握各种开发软件的功能和使用技巧。

这套丛书力求通过实例来介绍开发工具的功能，由浅入深地讲述编程和开发内容，每个实例都包括：目标和要点、实现步骤、归纳和注释、小结或练习几大部分，不仅适用于那些刚刚从事编程或开发制作的计算机使用者，并且对已经有较多开发经验的计算机用户也同样有较大的帮助。易学易用且实用性强是本丛书的特点，书中每一个例子里的程序或方法，读者都可以直接引用，相信会使广大读者受益匪浅。

这套丛书目前拟包括以下内容：

1．Office 等办公软件实例（Word、Excel、PowerPoint 和 Access、Microsoft Project98、Microsoft Money98 等）；

2．Microsoft Developer Studio 实例（包括 VC、VB、VFP、VJ、ActiveX）；

3．多媒体实例（包括 CorelDRAW、PhotoShop、3DMAX、Director、Authorware、Freehand、Illustrate 等）；

4．网页制作实例（包括 FrontPage、Pagemaker、Net Object Fusion、VBScript、JavaScript、ASP 等）；

5．图形开发实例（例如 OpenGL）；

6．数据库及网络开发实例（例如 PowerBuilder、Oracle、Sybase）。

目前本丛书共计 22 本，随着软件版本的不断更新和新型软件的出现，我们还将不断充实更新这套丛书。

清汉计算机工作室

编 者 的 话

随着网络的发展，计算机和网络必会合二为一，而网络的开发，又是和数据库紧密结合在一起的。如果希望在当今的软件市场取得成功，就必须熟悉网络编程和数据库编程技术。本书以实例的形式向读者介绍了通信、因特网（Internet）和相关的数据库程序设计。

Visual C++ 6.0 是一个功能强大、灵活性好和完全可扩展的 Windows 开发系统。如今，作为一种通用且功能强大的编程语言，Visual C++的地位不可动摇。它提供的完全集成性以及可视化用户界面驱动的特性，不仅适用于传统的 C/C++开发过程，更充分优化了对面向对象技术的支持。一方面，它完美地与 Windows 平台进行结合，从而保证了程序具有强大的功能；另一方面，其无可比拟的与 Windows 同步更新的优势对程序员也具有极大的吸引力。

在本书中读者可以学到如何使用底层的 API，像 Windows Sockets（套接字）和高层的 API，像 Win32 Inter API 和 Message API，同时还可以学会如何通过 ActiveX 控件激活万维网（World Wide Web）、ActiveX 容器、ActiveX 服务器以及使用电子邮件向 MFC 应用程序发送文档等等。另外，在 Visual C++中，包含了开发数据库应用程序的全方位的支持。通过 Visual C++提供的多种多样的访问技术如 ODBC API、MFC ODBC、DAO、OLE DB、ADO 等，我们可以访问当今主流的所有数据格式。上述这些技术各有特点，共同组成了强大的 Visual C++数据库开发环境。

在 Visual C++ 6.0 中，不仅提供传统的 ODBC 接口和 DAO 访问多种数据库，而且在 OLE DB 方面的功能又有很大的增强。使用 COM 技术和 ATL，Visual C++ 6.0 为我们编制网络环境下的数据库应用程序提供了极大的方便。

在网络和数据库的开发过程中开发经验是非常重要的，编程技巧是开发高质量应用程序必不可少的条件之一。本书以实例的形式，详细地介绍了 Visual C++ 6.0 的应用程序开发方法和高级应用技巧。书中的实例都以已经封装好的 C++类的形式给出，读者可以在自己的网络和数据库应用程序开发中，直接引用或者重新定制这些 C++类，从而大大简化了开发的过程、缩短开发的时间，使应用程序更加生动漂亮。

无论读者是 Visual C++ 6.0 的初学者还是专业的程序员，都可通过学习本书的内容，熟悉 Visual C++ 6.0 多媒体开发的过程，提高开发的水平。

本书主要由吴斌、李玉忠和赵有珍编著，参加本书编写的还有薛风武、史义霞、毕丽蕴等。另外本书的编写是集体劳动的结晶。

限于水平和时间，书中出现的错误和不妥之处，敬请读者批评指正。

编　　者

目　录

实例 1　创建超级链接

目标和要点

实例目标

本实例将介绍如何创建超级链接（包括邮箱的链接和 Internet 资源的链接）的示例。程序运行的显示结果如图 1-1 所示。

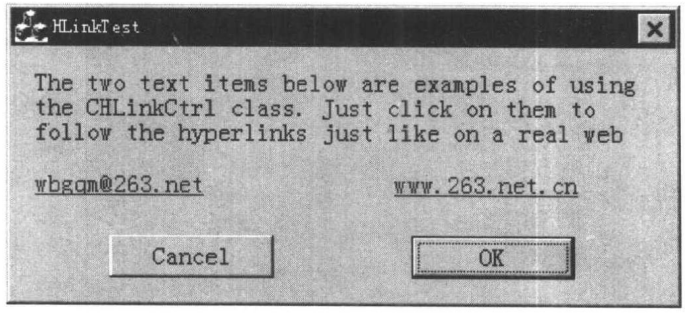

图 1-1　具有超级链接的窗口

本实例介绍了如何创建功能强大的热点链接(HotLink)。该热点链接初始为蓝色，当鼠标移动到这些热点链接上时，它们会变成绿色，并且鼠标变为小手形状；当我们使用鼠标点击这些热点链接时，能够跳转到发送邮件程序（例如 Outlook），给该热点链接的人发送邮件，或者访问与该热点链接相对应的 Internet 资源（缺省使用 IE 进行浏览），如果点击这些热点链接后，它们会变成粉红色，表示这些热点链接已经被点击过。

技术要点

本实例的技术要点主要包括：

1. 光标通过控件时改变光标的形状
2. Windows 字体的设置

当鼠标光标通过某个特定控件时，为了立即给用户提供反馈信息，需要改变鼠标光标的类型。例如，当用户移动鼠标通过一个按钮时如何将鼠标光标改为十字形。改变光标的最简单的方法是响应 WM_SETCURSOR 消息。当光标移动到一个窗口内并且还没有捕捉到鼠标时，Windows 向窗口发送 WM_SETCURSOR 消息，MFC 应用程序通过 OnSet Cursor 方法处理这个消息。此处使用 OnSetCursor 方法响应子窗口控件的光标设置。完成本实例中对静态控件的所有操作后，可以看到光标在移到按钮上时会发生变化。

老式的字符方式（charecter_mode）的应用程序只能在屏幕上显示单调的系统字体，而 Windows 则提供了多种多样的、与设备无关的、各种尺寸的字体。只要有效地利用 Windows

的字体，那么我们用不着在编程上下多大的功夫，就可以极大地增强各种应用程序的功能。而在 Windows3.1 中首次引入的 TureType 字体要比以前的与设备无关的字体效果更好，而且更加易于编程。

实 现 步 骤

☞ 建立一个新工程

启动 Visual C++ 6.0，使用 AppWizard 新建一个【MFC AppWizard】工程，该工程的名称为 HLinkTest，该工程的最终配置如图 1-2 所示。

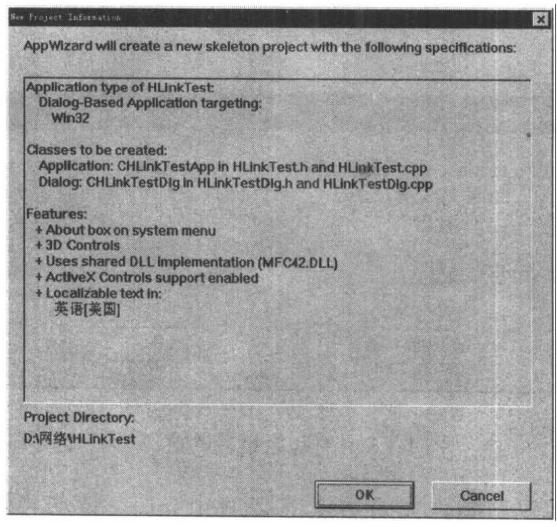

图 1-2 HLinkTest 工程的配置

☞ 建立超级链接类

为了实现超级链接，我们创建了一个超级链接类 CHyperLink。

1．超级链接类 CHyperLink

使用 ClassWizard 来创建超级链接类 CHyperLink，结果如下：

派生类：CHyperLink

基类：CStatic

类声明文件： HyperLink.h

类实现文件： HyperLink.cpp

2．类 CHyperLink 声明文件

创建了超级链接类 CHyperLink 后，对该类映射如表 1-1 所示的消息函数。

表 1-1 类 CHyperLink 的消息函数

消 息 函 数	消 息	说 明
CtlColor	=WM_CTLCOLOR	指定将重画控件
OnKeyDown	WM_KEYDOWN	当按下非系统键时响应
OnKillFocus	WM_KILLFOCUS	指定窗口是否失去输入焦点

（续）

消息函数	消　息	说　明
OnLButtonDown	WM_LBUTTONDOWN	当鼠标左键按下时响应
OnLButtonUp	WM_LBUTTONUP	当鼠标左键抬起时响应
OnMouseMove	WM_MOUSEMOVE	当鼠标移动时响应
OnNcHitTest	WM_NCHITTEST	指定鼠标移动时的形状
OnSetCursor	WM_SETCURSOR	显示合适的鼠标形状
OnSetFocus	WM_SETFOCUS	指示窗口已获得了输入焦点
PreSubclassWindow	PreSubclassWindow	在对象连接到一个窗口时调用该函数
PreTranslateMessage	PreTranslateMessage	在发出消息后，调用该函数来过滤消息

在 HyperLink.h 文件中，我们定义了超级链接类 CHyperLink。下面是实现的源代码：

```
//得到/设置超级链接的颜色
typedef struct tagHYPERLINKCOLORS
{
    COLORREF   crLink;
    COLORREF   crActive;
    COLORREF   crVisited;
    COLORREF   crHover;
} HYPERLINKCOLORS;

/////////////////////////////////////////////////////////////////////////////
// CHyperLink window 下面的代码创建超级链接窗口

class CHyperLink : public CStatic
{
    DECLARE_DYNAMIC(CHyperLink)

public:
// 超级链接类型
    static const DWORD StyleUnderline;
    static const DWORD StyleUseHover;
    static const DWORD StyleAutoSize;
    static const DWORD StyleDownClick;
    static const DWORD StyleGetFocusOnClick;
    static const DWORD StyleNoHandCursor;
    static const DWORD StyleNoActiveColor;

    //构造/析构函数
    CHyperLink(),
    virtual ~CHyperLink(),
```

```
// Attributes
public:

// Operations
public:
    static void GetColors(HYPERLINKCOLORS& linkColors);

    static HCURSOR GetLinkCursor();
    static void SetLinkCursor(HCURSOR hCursor);

    static void SetColors(COLORREF crLinkColor, COLORREF crActiveColor,
            COLORREF crVisitedColor, COLORREF crHoverColor = -1);
    static void SetColors(HYPERLINKCOLORS& colors);

    void SetURL(CString strURL);
    CString GetURL() const;

    DWORD GetLinkStyle() const;
    BOOL ModifyLinkStyle(DWORD dwRemove, DWORD dwAdd, BOOL
            bApply=TRUE);

    void SetWindowText(LPCTSTR lpszText);
    void SetFont(CFont *pFont);

    BOOL IsVisited() const;
    void SetVisited(BOOL bVisited = TRUE);

    // 如果需要 subclass 和设置不同的 URL，使用下面的函数。
    BOOL SubclassDlgItem(UINT nID, CWnd* pParent, LPCTSTR
            lpszURL=NULL)
    {
        m_strURL = lpszURL;
        return CStatic::SubclassDlgItem(nID, pParent);
    }

    // Overrides
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CHyperLink)
public:
    virtual BOOL PreTranslateMessage(MSG* pMsg);
```

```
    protected:
    virtual void PreSubclassWindow();
    //}}AFX_VIRTUAL

// Implementation
protected:
    static void SetDefaultCursor();
    static LONG GetRegKey(HKEY key, LPCTSTR subkey, LPTSTR retdata);
    static void ReportError(int nError);
    static HINSTANCE GotoURL(LPCTSTR url, int showcmd);

    void AdjustWindow();
    void FollowLink();
    inline void SwitchUnderline();

// Protected attributes
protected:
    static COLORREF g_crLinkColor;      // 连接的正常颜色
    static COLORREF g_crActiveColor;    // 链接激活时的颜色
    static COLORREF g_crVisitedColor;   // 访问链接时的颜色
    static COLORREF g_crHoverColor;     // 巡航时颜色
    static HCURSOR  g_hLinkCursor;      // 链接时的鼠标形状

    BOOL     m_bLinkActive;             // 是否已经激活链接
    BOOL     m_bOverControl;            // 鼠标是否在控件上
    BOOL     m_bVisited;                // 是否已访问链接
    DWORD    m_dwStyle;                 // 连接形状
    CString  m_strURL;                  // 连接 URL 字符
    CFont    m_Font;                    // 下画线字型
    CToolTipCtrl m_ToolTip;             // 链接的 tooltip 窗口

    // Generated message map functions
protected:
    //{{AFX_MSG(CHyperLink)
    afx_msg HBRUSH CtlColor(CDC* pDC, UINT nCtlColor);
    afx_msg BOOL OnSetCursor(CWnd* pWnd, UINT nHitTest, UINT message);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnSetFocus(CWnd* pOldWnd);
    afx_msg void OnKillFocus(CWnd* pNewWnd);
```

```
    afx_msg void OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags);
    afx_msg UINT OnNcHitTest(CPoint point);
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};
```

3．初始化成员变量

在 CHyperLink 的构造函数中，初始化其成员变量：

```
CHyperLink::CHyperLink()
{
    m_bOverControl      = FALSE,    // 初始时鼠标未在控件上方
    m_bVisited          = FALSE,    // 初始时超级链接未被访问
    m_bLinkActive       = FALSE;    // 初始时控件不是操作焦点
    m_strURL.Empty();               // 初始时设置 URL 为空字符串
    // 设置缺省的风格
    m_dwStyle = StyleUnderline|StyleAutoSize|StyleGetFocusOnClick;
}
```

4．释放内存

在 CHyperLink 的析构函数中，释放字型所占用的内存：

```
CHyperLink::~CHyperLink()
{
    m_Font.DeleteObject();
}
```

5．实现消息映射函数

在消息 WM_CTLCOLOR 的 CtlColor ()函数中，得到 WM_CTLCOLOR 的消息句柄。

```
HBRUSH CHyperLink::CtlColor(CDC* pDC, UINT nCtlColor)
{
    ASSERT(nCtlColor == CTLCOLOR_STATIC),

    if (m_bOverControl && BITSET(m_dwStyle,StyleUseHover))
        pDC->SetTextColor(g_crHoverColor);
    else if (!BITSET(m_dwStyle,StyleNoActiveColor) && m_bLinkActive)
        pDC->SetTextColor(g_crActiveColor);
    else if (m_bVisited)
```

```
            pDC->SetTextColor(g_crVisitedColor);
        else
            pDC->SetTextColor(g_crLinkColor);

        //设置透明的绘图模式
        pDC->SetBkMode(TRANSPARENT);
        return (HBRUSH)GetStockObject(NULL_BRUSH);
    }
```

在函数 OnKeyDown()中，系统响应非系统键盘的按下某个键的消息：

```
    void CHyperLink::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
    {
        if (nChar == VK_SPACE)
            FollowLink();
        else
            CStatic::OnKeyDown(nChar, nRepCnt, nFlags);
    }
```

在函数 OnKillFocus()中，使链接区失去输入焦点：

```
    void CHyperLink::OnKillFocus(CWnd* /*pNewWnd*/)
    {
        // Assume that control lost focus = mouse out
        // this avoid troubles with the Hover color
        m_bOverControl = FALSE;
        m_bLinkActive = FALSE;
        //重新绘制控件，设置输入焦点
        Invalidate();
    }
```

在函数 OnLButtonDown()中，设置输入焦点，激活链接：

```
    void CHyperLink::OnLButtonDown(UINT /*nFlags*/, CPoint /*point*/)
    {
        if (BITSET(m_dwStyle,StyleGetFocusOnClick))
            SetFocus();                    // 设置链接焦点，激活超级链接
        if (BITSET(m_dwStyle,StyleDownClick))
            FollowLink();
        m_bLinkActive = TRUE;
```

```
}
```

在函数 OnLButtonUp()中，激活链接：

```
void CHyperLink::OnLButtonUp(UINT /*nFlags*/, CPoint /*point*/)
{
    if (m_bLinkActive && !BITSET(m_dwStyle,StyleDownClick))
        FollowLink();
}
```

在上面的两个消息函数中，使用的激活链接的函数为:FollowLink()。其源代码如下：

```
void CHyperLink::FollowLink()
{
    int result = (int) GotoURL(m_strURL, SW_SHOW);
    if (result <= HINSTANCE_ERROR)
    {
        MessageBeep(MB_ICONEXCLAMATION);      // 无法建立链接
        ReportError(result);
    }
    else
    {
        // 标志链接为可访问和可刷新窗口
        m_bVisited = TRUE;
        Invalidate();
    }
}
```

移动鼠标时，在 OnMouseMove()函数中得到当前鼠标的位置，改变鼠标的形状：

```
void CHyperLink::OnMouseMove(UINT nFlags, CPoint point)
{

    if (m_bOverControl)          // 当鼠标移动到控件上时
    {
        CRect rect;
        GetClientRect(rect);

        if (!rect.PtInRect(point))
        {
```

```
                m_bOverControl = FALSE;
                ReleaseCapture();
                Invalidate();
                return;
            }
        }
        else                        // 当鼠标离开控件时
        {
            m_bOverControl = TRUE,
            Invalidate();
            SetCapture();
        }
    }
```

在正常情况下，除非是 SS_NOTIFY 风格，否则静态控件并不接受鼠标移动的事件。为了 达到和 SS_NOTIFY 相同的效果，可以使用下面的消息函数，它比在 OnCtlColor()中使用 SS_NOTIFY 所编写的代码更少，也更加可靠，这是因为 Windows 并不给图形静态控件发送 WM_CTLCOLOR 消息。

```
UINT CHyperLink::OnNcHitTest(CPoint /*point*/)
{
        return HTCLIENT;
}
```

在函数 OnSetCursor()中，设置鼠标形状。注意该鼠标变化情况均为 Windows 系统鼠标形状。其源代码如下：

```
BOOL CHyperLink::OnSetCursor(CWnd* /*pWnd*/, UINT /*nHitTest*/,
                             UINT /*message*/)
{
    if (g_hLinkCursor)
    {
        ::SetCursor(g_hLinkCursor);
        return TRUE;
    }
    return FALSE;
}
```

在函数 OnSetFocus()中，设置输入焦点窗口。当设置输入焦点窗口时，重新刷新该窗口：

```
void CHyperLink::OnSetFocus(CWnd* /*pOldWnd*/)
{
    m_bLinkActive = TRUE;
    Invalidate();                                    // 刷新窗口，设置焦点
}
```

在函数 PreSubclassWindow()中，将控件和一定的窗口链接：

```
void CHyperLink::PreSubclassWindow()
{
    // 如果 URL 字符串为空，重新设置该窗口文字
    if (m_strURL.IsEmpty())
        GetWindowText(m_strURL);

    // 检查该窗口文字是否为空。如果不为空，将其设置为 URL 字符。
    CString strWndText;
    GetWindowText(strWndText);
    if (strWndText.IsEmpty())
    {
        // 下面将 URL 字符串设置为窗口文本
        ASSERT(!m_strURL.IsEmpty());      //窗口文本和 URL 皆为 NULL!
        CStatic::SetWindowText(m_strURL);
    }

    // 获得当前的窗口字型
    CFont* pFont = GetFont();

    if (pFont != NULL)
    {
        LOGFONT lf;
        pFont->GetLogFont(&lf);
        lf.lfUnderline = BITSET(m_dwStyle, StyleUnderline);
        if (m_Font.CreateFontIndirect(&lf))
            CStatic::SetFont(&m_Font);
        // 为 URL 适当调整窗口
        AdjustWindow();
    }
    else
    {
        //如果 GetFont()函数返回值为 NULL
```

```
        //那么该静态控件可能不是文本类型
        //最好将该静态控件的 auto-resizing 属性关闭
        CLEARBITS(m_dwStyle,StyleAutoSize);
    }

    if (!BITSET(m_dwStyle,StyleNoHandCursor))
        SetDefaultCursor();          // 装载“小手”鼠标

    // 建立提示（tooltip）窗口
    CRect rect;
    GetClientRect(rect);
    m_ToolTip.Create(this);

    m_ToolTip AddTool(this, m_strURL, rect, TOOLTIP_ID);

    CStatic .PreSubclassWindow();
}
```

在函数 PreTranslateMessage()中，延迟出现提示窗口的消息：

```
BOOL CHyperLink::PreTranslateMessage(MSG* pMsg)
{
    m_ToolTip.RelayEvent(pMsg);
    return CStatic::PreTranslateMessage(pMsg);
}
```

6．定义全局变量和常量

在 HyperLink.cpp 文件中，定义全局变量和常量，其源代码如下：

```
#define TOOLTIP_ID 1

#define SETBITS(dw, bits)    (dw |= bits)
#define CLEARBITS(dw, bits)      (dw &= ~(bits))
#define BITSET(dw, bit)       (((dw) & (bit)) != 0L)

// Underline bit
const DWORD CHyperLink::StyleUnderline = 0x00000001;
// Hand over coloring bit
const DWORD CHyperLink::StyleUseHover = 0x00000002;
// Auto size bit
```