



Mit VC++ programmieren

Der eigene Web-Browser



Die Standard-Browser sind **nie perfekt** auf die eigenen Bedürfnisse abgestimmt. Im folgenden Beitrag zeigen wir Ihnen, wie Sie Ihren Browser mit dem **WebBrowser Control** von Microsoft individuell einrichten können.

THOMAS WÖLFER

Der Internet Explorer ist nicht nur ein sehr umfangreiches Programm, sondern auch eine sehr umfangreiche Programm-Bibliothek. Praktisch alle Elemente dieses Browsers lassen sich in eigenen Anwendungen weiterverwenden. Dazu zählt nicht zuletzt der Teil des Browsers, der die eigentliche Anzeige der HTML-Seiten besorgt. Um diesen Teil wiederzuverwenden, gibt es sogar eine eigene MFC-Klasse. Diese macht die grundlegende Arbeit beim Programmieren eines eigenen Browsers sehr einfach.

Es gibt dabei noch viel mehr Möglichkeiten, als einfach nur ein Programm zu schreiben, das Webseiten anzeigen kann. Eine davon ist, sich Webseiten anzuzeigen, die gar nicht als HTML auf der Festplatte oder einem Server vorliegen, sondern als Ressourcen direkt in einer .EXE- oder .DLL-Datei eingebunden sind. Das ist nur eine der Funktionen, die Sie im Folgenden für die Programmierung des eigenen Browsers verwenden werden.

■ Das Projekt

Wenn Sie sich die Programmierarbeit sparen möchten, können Sie den Quell-Code und das fertige Programm von der Heft-CD dieses Sonderheftes laden. Interessiert Sie der Aufbau des Programms, folgen Sie den Einzelschritten in der anschließenden Erläuterung.

Zunächst brauchen Sie ein neues MFC-Projekt. Das legen Sie am besten mit dem AppWizard an. Als Typ des

Projekts wählen Sie *Multiple Document mit Document/View Support*. Das macht die Programmierung hinterher angenehmer, da der Browser mehrere Webseiten gleichzeitig anzeigen kann.

■ Schnellstart zum Browser

Bei der Anfrage, ob die Anwendung eine „MFC Standard“- oder eine „Windows Explorer“-Anwendung werden soll, verwenden Sie die Option *MFC Standard*. Wenn der AppWizard wenige Schritte später die zu verwendenden Klassen anzeigt, müssen Sie eine wichtige Veränderung vornehmen: Die Basis-Klasse für die View der Anwendung setzen Sie auf *CHtmlView*. Diese View-Klasse ist die MFC-Klasse, die das Browser-Control kapselt.

Wenn Sie das Projekt übersetzen und das resultierende Programm starten, gibt es eine kleine Überraschung: Sie haben jetzt bereits einen voll einsatzfähigen Browser. Dieser kann alle Webseiten anzeigen, die dem Internet Explorer keine Probleme machen. Für einen „echten“ Browser ist das Programm etwas eingeschränkt, denn es gibt noch kein Benutzer-Interface, mit dem Sie navigieren können. Voreingestellt navigiert das Programm zur MSDN-Entwicklerseite bei Microsoft.

Einen Vorteil gegenüber dem echten Internet Explorer hat das Programm schon jetzt:

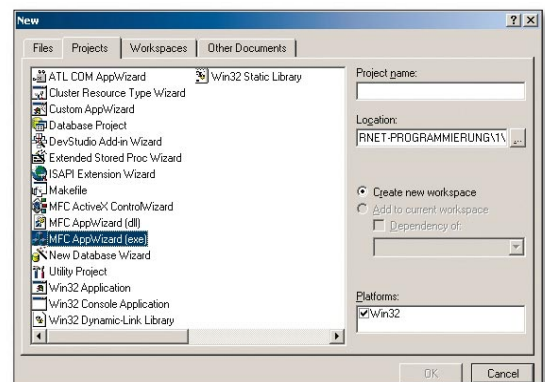
Es kann mehrere Webseiten gleichzeitig anzeigen. Das können Sie leicht überprüfen, indem Sie ein neues Fenster öffnen und die Fenster nebeneinander anzeigen lassen.

■ Ressourcen extrahieren

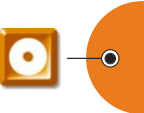
Nach diesem Schnellstart wird es komplizierter. Der eigene Browser soll mit einer eigenen Startseite versorgt werden. Da es unpraktisch wäre, diese immer mitzuliefern, bietet es sich an, diese Seite sowie alle darauf enthaltenen Elemente, wie etwa Bilder, im VC++ Projekt als Resource mit einzubinden. Das ist nicht nur für eigene Browser eine praktische Möglichkeit. Sie können damit zum Beispiel komplette eigene HTML-Präsentationen in Form eines ausführbaren Programms verteilen.

Ganz so einfach wie es klingt, ist das nicht. Die Entwicklungsumgebung von VC++ unterstützt zwar das benötigte Vorgehen, aber es liegt keine Dokumentation oder sonstige Hilfe vor. Folgendes ist zu tun:

Zunächst gehen Sie in die Ressourcen-Ansicht vom Projekt-Workspace. Dort klicken Sie mit der rechten Maustaste auf die Ressourcen und wählen den Befehl *Insert Resource*. VC++ zeigt die passende Dialogbox an. Auf dieser gibt es unter anderen einen Button mit der Beschriftung *Custom....* Diesen Button



FÜR DEN EIGENEN BROWSER erzeugen Sie ein neues MFC-Projekt mit dem AppWizard.



brauchen Sie nun zweimal. Beim ersten Mal erzeugen Sie eine *Custom-Resource* mit dem Namen „HTML“, beim zweiten Mal eine mit der Bezeichnung „GIF“. Danach ist Ihr Ressourcen-Fenster um zwei Ordner reicher. Dass „HTML“ keine „Custom“-Resource ist, erkennen Sie an der Schreibweise. Im Gegensatz zum „GIF“-Ordner steht das HTML nicht in Hochhaken.

Nun können Sie die Ressourcen hinzufügen. Das geht am einfachsten, wenn Sie mit der rechten Maustaste auf den „GIF“-Ordner klicken und dann *Import* auswählen. VC++ öffnet einen ganz normalen *File-Open*-Dialog, mit dem Sie das gewünschte Bild auswählen können. Beim Beispielprogramm ist das die Datei *mybrowse1.jpg*. Es handelt sich nicht um ein „GIF“-Bild und das ist auch gar nicht notwendig, denn es ist nur ein Name, mit dem VC++ keine bestimmte Eigenschaft verbindet.

Genauso importieren Sie die benötigten HTML-Dateien in den HTML-

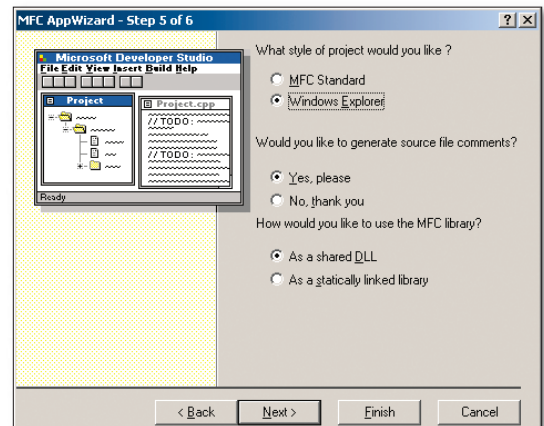
Ordner. Beim Importieren vergibt VC++ automatisch IDs für die importierten Dateien. Sie sollten jedoch die Ressourcen innerhalb des Ressourcen-Ordners so umbenennen, dass sie den ursprünglichen Dateinamen widerspiegeln. Das vereinfacht die spätere Bearbeitung der HTML-Dateien. Sie legen die zu einer Datei gehörenden Bilder in einen Ordner mit dem Namen „GIF“ ab, der sich innerhalb des Ordners befindet, in dem auch die HTML-Dateien liegen. Wenn Sie dies tun, können Sie die Bilder im HTML ganz normal referenzieren. Im anderen Fall müssen Sie die Dateien mit dem *Res:// Protokoll* einbinden. Das ist unpraktisch, weil Sie immer erst eine neue Ressourcen-DLL erzeugen müssen, um ein geändertes Bild im HTML begutachten zu können.

Beim Beispiel-Quell-Code finden Sie die Seite „start.html“. Diese Seite ist als Resource im Projekt eingebunden. Für den eigenen Browser soll das die Startseite werden.

Das ist relativ einfach. Wie Sie beim ersten Test des erzeugten Programms bereits gesehen haben, navigiert Ihr Browser „von selbst“ auf die MSDN-Webseite. In der Methode *OnInitialUpdate()* wurde dafür ein entsprechender Code vom AppWizard eingefügt. Den müssen Sie nun ändern:

```
void CMyBrowseView::
OnInitialUpdate()
{
    CHtmlView::
    OnInitialUpdate();
    //Navigate2(_T("http://www.microsoft.com/visualc/"),
    //NULL, NULL);
    LoadFromResource
    ( "START.HTML" );
}
```

Der Original-Code verwendet die Methode *Navigate2()*, um auf die angegebene MSDN-Seite zu wechseln. Diesen Funktionsaufruf können Sie löschen oder auskommentieren. Statt dessen fügen Sie einen Aufruf von



AUCH MIT DEM „Windows Explorer“-Style können Sie schon beim Erzeugen des Projekts IE-Ähnlichkeit sicherstellen. Das Beispielprogramm bietet das nicht.

LoadFromResource() ein. Wählen Sie als Ressourcen-Namen den Namen der zu ladenden HTML-Seite, bzw. den Namen, den Sie in der Resource-View dafür vergeben haben. (Praktischerweise sollten Sie, wie oben beschrieben, immer den Namen der Datei verwenden.)

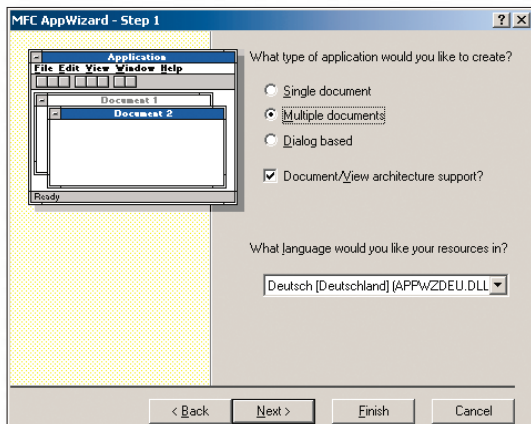
LoadFromResource() führt eine Menge Vorgänge aus, im Wesentlichen extrahiert die Funktion die angegebene Resource aus dem Resource-Directory des ausführbaren Programms und übergibt sie an das WebBrowser Control. Wenn es sich bei der Resource um ein Element handelt, mit dem der Web-Browser etwas anfangen kann – und bei HTML-Seiten ist das der Fall – zeigt der Browser die Datei an. Schon haben Sie Ihre ganz persönliche Browser-Startseite.

Wenn Sie das Beispiel-HTML verwendet haben, können Sie leicht überprüfen, dass es sich bei der Browser-View wirklich um einen vollwertigen Browser handelt, der mit Programmen wie JavaScript umgehen kann. Dazu gibt es bei dem Beispiel „start.html“ extra einen kleinen Button mit der Aufschrift *Info über*. Wenn Sie auf diesen Button klicken, öffnet sich ein weiteres Browser-Fenster. Das wird per JavaScript angezeigt. Die Resource für dieses Fenster ist direkt im Programm enthalten:

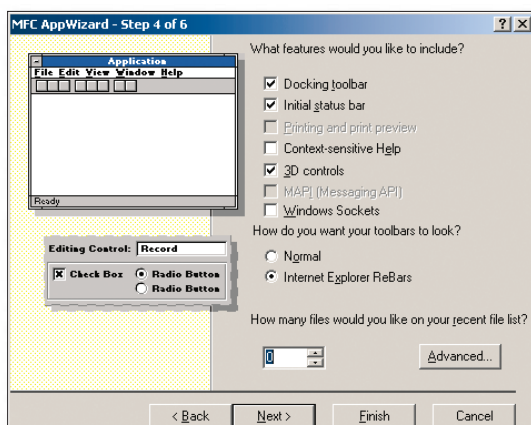
```
<script language=javascript>
function AboutFxn()
{
    window.showModalDialog
    ( "about.html" );
}
</script>
```

Selber navigieren

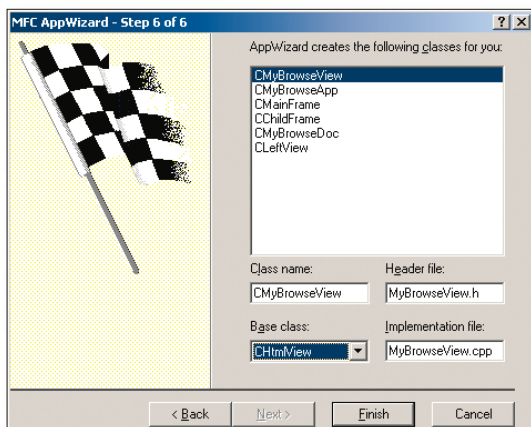
Bisher ist das Programm ganz ordentlich, aber für einen echten Browser wür-



ALS ANWENDUNGSTYP ENTScheiden Sie sich für ein MDI-Projekt mit *Document/View Support*. Das brauchen Sie deshalb, damit die Änderung der View-Klasse im AppWizard möglich wird.



MIT INTERNET EXPLORER REBARS wird das Look&Feel Ihres eigenen Browsers IE ähnlicher.



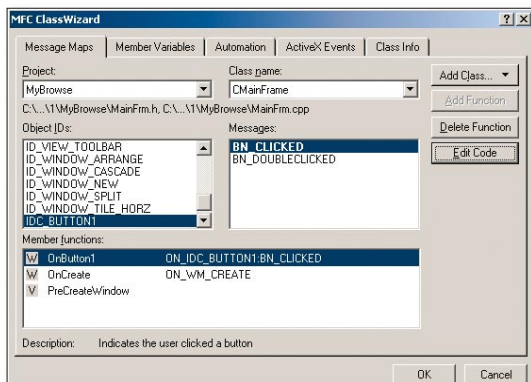
WICHTIG BEIM ANLEGEN des Projekts: Geben Sie als Basis-Klasse Ihrer View die CHtmlView an.

de es nicht schaden, wenn man irgendwo eine URL eintippen könnte. Auf Dauer ist die Anzeige der eigenen Startseite etwas langweilig. Außerdem sollten ein paar der elementaren Navigationsmöglichkeiten vorhanden sein. Dazu zählt etwa ein Zurück- und ein Vor-Button, sowie ein Stop- und ein Reload-Button. Alle diese Elemente implementieren Sie im nun folgenden Schritt.

Die vom AppWizard erzeugte CMDIFrameWnd-Klasse mit dem Na-



SCHON NACH DEM ERSTEN ÜBERSETZEN des Projekts – ohne dass Sie auch nur eine Zeile programmiert hätten – haben Sie einen funktionstüchtigen Browser.



MIT DEM CLASSWIZARD definieren Sie die Event-Handler für die Buttons Ihres Browsers.

men CMainFrame enthält dafür bereits alles was benötigt wird. Das ist eine Toolbar und eine Dialog-Bar-Leiste. Die Toolbar-Leiste kann dabei die Buttons für das Navigieren aufnehmen, während der Dialog-Bar ein Feld fürs Eintippen von URLs erhalten soll.

Die Buttons behandeln Sie wie mit VC++ gewohnt. Legen Sie zunächst einen Button für die Zurück-Navigation an. Definieren Sie dann für diesen Button einen Event-Handler für den *OnClick*-Event. Diesen Handler platzieren Sie in der CMainFrame-Klasse. Im Quell-Code zum Handler müssen Sie einen Zeiger auf die aktuelle View ermitteln. Das geht bei den MDI-Anwendungen mit zwei hintereinander aufzurufenden Funktionen. Ermitteln Sie mit *MDIGetActive()* das aktive MDI Child-Frame und fragen Sie mit *GetActiveView()* nach der aktuell aktiven View:

```
void CMainFrame::OnBack()
{
    CView* pView =
        MDIGetActive()->Get
        ActiveView();
    if( pView)
    {
        if( pView->
            IsKindOf( RUN
            TIME_CLASS( CMy
            BrowseView)))
        {
            ((CMyBrowse
            View*)pView)-
            > GoBack();
        }
    }
}
```

```
void CMainFrame::NavigateActive
View(const char* pszURL)
{
    CView* pView = MDIGet
    Active()->GetActiveView();
    if( pView)
    {
        if( pView->
            IsKindOf( RUNTIME_CLASS
            (CMyBrowseView)))
        {
            MDIGetActive()-
            >SetWindow
            Text( pszURL);
            ((CMyBrowseView*)pView)-
            >Navigate2
            ( pszURL, 0, NULL);
        }
    }
}
```

Dabei ist es möglich, dass momentan gar keine View aktiv ist. Dies erkennen Sie, indem Sie den gelieferten Zeiger prüfen. Ist er nicht NULL, gibt es eine View. Vielleicht bauen Sie zu einem späteren Zeitpunkt zusätzliche Views in Ihren Browser ein. Wenn Sie das tun, besteht die Gefahr, dass diese View und nicht die Browser-View die aktive ist. Um die aktive View zu erkennen, müssen Sie den Typ der gelieferten View ermitteln. Nun haben Sie einen Zeiger auf der Browser-View und können diesen verwenden, um die richtige Funktion für den Back-Button aufzurufen. Die

anderen Buttons behandeln Sie ähnlich, dabei verwenden Sie am Ende die richtige Member-Funktion der *CHtmlView*-Klasse, bzw. Ihrer abgeleiteten Klassen. Das sind zum Beispiel *GoForward()*, *Stop()* und *Refresh()*.

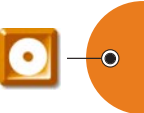
Für das URL-Fenster wird der Vorgang etwas schwieriger. Hier verwenden Sie am besten eine Combo-Box, denn nun können Sie die einmal eingetippten Adressen direkt in dieser Combo-Box speichern, sodass diese danach nicht mehr eingetippt werden brauchen. Für die Behandlung der Combo-Box benötigen Sie zwei Event-Handler: einen, der *IDOK* behandelt und einen, der „SELENDOK“ behandelt. *IDOK* wird ausgelöst, wenn im Edit-Control-Teil der Combo-Box *Return* gedrückt wird, also dann, wenn eine neue Adresse per Tastatur eingegeben wurde. *SELENDOK* wird dann ausgelöst, wenn aus dem Listbox-Teil der Combo-Box eine Adresse ausgewählt wurde.

In beiden Fällen soll natürlich eine Navigations-Aktion ausgeführt werden. Dazu definieren Sie eine Hilfsfunktion, mit der die aktuelle View zu einer gegebenen Adresse navigiert:

Implementieren Sie nun noch die beiden genannten Event-Handler. Der Handler für *SELENDOK* hat im Beispiel den Namen *OnSelendokAddress*:

```
void CMainFrame::OnSelendok
Address()
{
    CString str;
    CComboBox* pB = (CCombo
    Box*)m_wndDlgBar.GetDlgItem
    ( IDC_ADDRESS);
    pB->GetLabelText( pB->Get
    CurSel(), str);
    NavigateActiveView( str);
}
```

Dieser Handler ermittelt zunächst die aktuelle Adresse aus der Combo-Box und verwendet die zuvor definierte Hilfsfunktion zur Navigation.



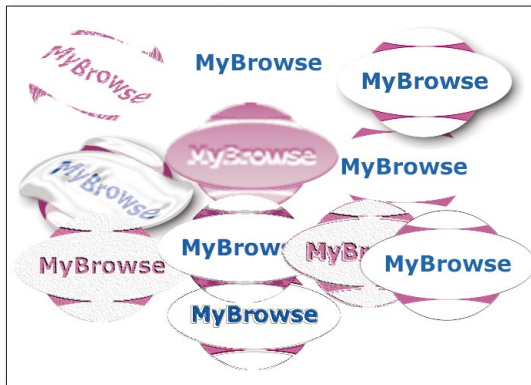
Der Handler für IDOK ist ein bisschen aufwändiger, denn dieser soll die URLs in der Combo-Box speichern:

```
void CMainFrame::OnAddressOK()
{
    CString str;
    CComboBoxEx* pB = (CComboBoxEx*)m_wndDlgBar.GetDlgItem(
        IDC_ADDRESS);
    CEdit* pEdit = pB->GetEditCtrl();
    pEdit->GetWindowText(str);
    NavigateActiveView( str);
    // neue adresse in
    // combo-box eintragen
    COMBOBOXEXITEM item;
    item.mask = CBEIF_TEXT;
    item.iItem = -1;
    item.pszText = (LPTSTR)
        (LPCTSTR)str;
    pB->InsertItem(&item);
}
```

Dazu lesen Sie den Text aus dem Edit-Control-Teil der Combo-Box aus und verwenden wieder die Hilfsfunktion fürs Navigieren. Nun tragen Sie den Text in der Combo-Box ein.

Die eigene Suchseite

Im nun folgenden Schritt verwenden Sie das bisher angesammelte Wissen und bündeln es für eine neue Funktion im Browser. Sie bauen eine eigene Suchseite ein.



DIE QUAL DER WAHL: Welches Logo nimmt man für den eigenen Browser?

Es wäre natürlich möglich, einfach die vorgefertigte Suche der *ChtmlView*-Klasse zu verwenden, aber diese bietet nur die Suchfunktion von Microsoft, also die MSN-Suchseite in neueren Versionen des IE. Wenn Sie den Browser aber konfektionieren möchten, etwa als „Spezial“-Browser für Ihr eigenes Web-Angebot, ist das nicht ausreichend. Statt dessen wäre es schön, wenn Sie eine eigene Seite bauen könnten.

Als Beispiel wird hier die Suche auf www.nickles.de verwendet, die die Datenbank der Site durchsucht. Diese setzt

sich hauptsächlich mit Computer-nahen Themen auseinander und umfasst ca. 500.000 Datenbankeinträge.

Für den Einbau der Suchseite verwenden Sie den Mechanismus, den Sie zu Beginn dieses Artikels schon einmal verwendet haben. Sie schreiben eine kleine HTML-Seite, die Sie als Resource im Projekt einbinden. Diese Suchseite muss zumindest ein Eingabefeld für die Suchworte enthalten. Zudem legen Sie einen Button an, mit dem die Suche gestartet werden kann:

```
Suchworte: <INPUT id=text1
name=text1>
<INPUT id=button1 type=submit
value=Suchen name=button1
LANGUAGE=javascript onclick=
return button1_onclick()>
```

Der Event-Handler in JavaScript setzt die Ziel-URL mit den Suchworten zusammen und navigiert zu den Suchresultaten:

```
function button1_onclick()
{
    var url="http://search.
    nickles.de/index.php3?
    sw=" +text1.value +
    "&sw=&cmd=snormal";
    window.navigate( url);
}
```

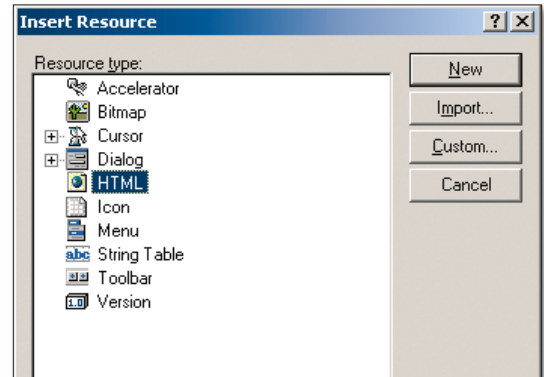
Schließlich brauchen Sie noch einen Button mit passendem Event-Handler, der mit *LoadFromResource()* Ihre Suchseite anzeigt.

Browsen mit COM

Bisher haben Sie nur die mehr oder minder fertig zur Verfügung gestellte Funktionalität des Browser Controls verwendet. Es ist aber wesentlich mehr möglich, denn der Browser

hat ein breit angelegtes COM Interface mit jeder Menge Funktionalität. Wie Sie diese nutzen, erfahren Sie im folgenden Schritt.

Für die Nutzung des COM Interface des Browsers müssen Sie zwei zusätzliche Header-Dateien inkludieren. Dabei handelt es sich um *ATLBase.h* und *MSHTML.H*. Die *ATLBase* enthält die „ATL“-Basis-Funktionalität. Deren Nutzung macht

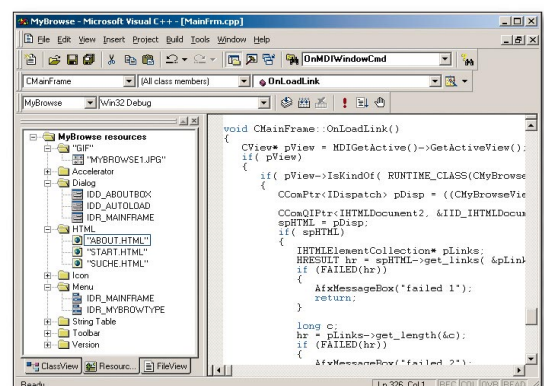


HTML IST EIN RESOURCE-TYP, den VC++ von Haus aus kennt, auch wenn das nicht immer von vornherein klar wird.

die Verwendung des COM Interface des Browsers deutlich einfacher. *MSHTML.H* enthält die benötigten GUIDs und IIDs für die COM-Schnittstelle des Browsers, bzw. für dessen HTML-Interface.

Das COM Interface des Browser Controls vollständig zu behandeln, ist in den wenigen Seiten eines solchen Beitrages nicht möglich. Die folgenden beiden Beispiele sollten Sie aber auf den Weg bringen. Mit der ersten Beispielfunktion erweitern Sie Ihren eigenen Browser um eine Möglichkeit, den HTML-Quellcode der momentan angezeigten Seite anzuzeigen. Das geht mit einem Klick auf die rechte Maustaste. Allerdings programmieren Sie dies hier im Beispiel selbst. Dadurch haben Sie die Möglichkeit, Ihren eigenen Browser um eine *Copy*-Funktion zu erweitern, mit der Sie Teile einer Webseite in ein Ihnen genehmes Format kopieren und umwandeln könnten.

Die zweite Beispielfunktion ist etwas aufwändiger. Dabei programmieren Sie eine Funktion für den Browser, mit dem Sie eine „Linkverfolgung“ implementieren. Das hat folgenden Sinn: Wenn Sie

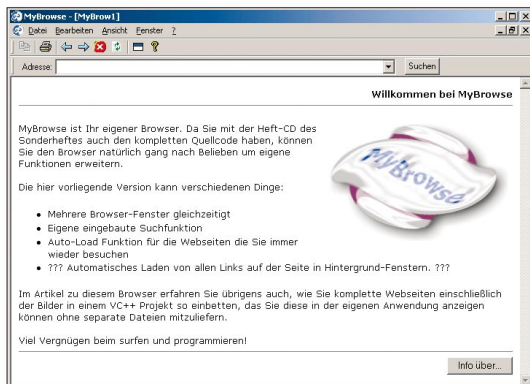


DIE RESOURCEN für die GIF- und JPG-Bilder legen Sie als „Custom Resource“ unter einem passenden Namen ab.

einen Suchmotor verwenden, liefert dieser normalerweise eine Resultatseite zurück, auf der sich Links zu einer Reihe von möglichen „richtigen“ Fundstellen befinden. Nun wäre es sehr praktisch, wenn man auf Knopfdruck für jeden dieser Links ein Fenster öffnen könnte, das automatisch zum Link navigiert. Genau das tut diese Funktion.

HTML anzeigen

Ein einfaches Beispiel ist das Anzeigen des HTML-Quell-Codes. Die *Chtmlview*-Klasse hat neben den einfachen Methoden wie *Navigate2()* und *Stop()* eine besondere Methode: *GetHtmlDocument()*. Diese Methode liefert nicht wie der Name erwarten lässt einen String



DER EIGENE BROWSER IN AKTION: Die Startseite wird aus den Ressourcen des Programms geladen.

oder Ähnliches mit dem Inhalt des HTML-Dokuments, sondern einen Zeiger auf das *IDispatch*-Interface dieses Dokuments. Damit können Sie alle Funktionen, die über dieses Interface zur Verfügung gestellt werden, nutzen. Die Funktion zur Anzeige des Quell-Codes hat folgenden Aufbau:

```
#include <atlbase.h>
#include <mshtml.h>

void CMyBrowseView::OnGet
↳Dispatch()
{
    CComPtr<IDispatch> pDisp =
    ↳GetHtmlDocument();
    CComQIPtr<IHTMLDocument2,
    ↳IID_IHTMLDocument2> spHTML;
    spHTML = pDisp;
    if( spHTML)
    {
        IHTMLElement*
        ↳pBody;
        HRESULT hr = spHTML-
        ↳>get_body(&pBody);
        if (FAILED(hr))
        {
            Afx↳MessageBox
            ↳("failed 1");
            return;
        }
        BSTR bstrHTMLText;
        hr = pBody->get_ou
```

```
terHTML(&bstrHTMLText);
if (FAILED(hr))
{
    Afx↳MessageBox
    ↳("failed 2");
    return;
}
Afx↳MessageBox
↳( CString(bstrHTMLText));
}
else
{
    Afx↳MessageBox("failed");
}
}
```

Zunächst rufen Sie *GetHtmlDocument()* auf, speichern aber den gelieferten *LPDISPATCH* nicht in einer Variable vom Typ *LPDISPATCH*, sondern in der Instanz einer *ATL Template*-Klasse, die dazu in der Lage ist. Dabei handelt es sich um einen *CComPtr* (C COM Pointer) mit dem Namen *pDisp*. Einer der praktischen Aspekte von ATL ist in diesem Zusammenhang die Tatsache, dass die benötigten Typenumwandlungen automatisch stattfinden. Das zeigt sich auch in den nächsten beiden Zeilen des Quell-Codes.

Eigentlich interessiert Sie das *IDispatch* Interface gar nicht. Sie benötigen es nur, um einen Zeiger auf ein *IHTML Document* zu ermitteln. Normalerweise müssten Sie das mit der *QueryInterface()*-Methode des *IDispatch* tun, aber bei ATL führen das die Zuweisungs-Operatoren für Sie durch.

In der nächsten Zeile legen Sie eine Zeiger-Variable vom gewünschten Typ an. Das geht unter Verwendung der *ATL*-Klasse *CComQIPtr* (C COM Query Interface Pointer) mit dem richtigen Template-Argument, in diesem Fall mit einem *IHTMLDocument*.

Diesem Zeiger weisen Sie nun den Zeiger mit dem *IDispatch* Interface zu. Unterstützt das Objekt, auf das der *IDispatch*-Zeiger zeigt, das erfragte Interface (also *IHTMLDocument*), enthält der *CComQIPtr* nach der Zuweisung einen Zeiger auf das richtige Interface. Wird das erfragte Interface nicht unterstützt, wird der Zeiger auf *NULL* gesetzt.

Nun haben Sie einen Zeiger auf das *IHTMLInterface* und können dessen Methoden verwenden. Eine davon ist *get_body*. Diese Funktion lie-

fert einen Zeiger auf ein Objekt vom Typ *IHTMLElement*, das den „Body“-Teil der HTML-Seite kapselt.

```
IHTMLElement* pBody;
spHTML->get_body(&pBody);
```

Mit diesem Zeiger können Sie nun alle Eigenschaften eines *HTML*-Elements erfragen. Eine davon ist der *HTML*-Quell-Code. Den erhalten Sie mit einem Aufruf von *get_outerHTML*, der seinerseits Daten liefert, die sich in einem *BSTR* (einer weiteren *ATL*-Klasse, die Strings kapselt) speichern lassen.

```
BSTR bstrHTMLText;
hr = pBody-
↳>get_outerHTML(&bstrHTMLText);
```

Nun bleibt noch das Problem, diesen *BSTR* in irgend etwas umzuwandeln, dass auch von *MFC* aus verwendet werden kann. Dafür gibt es eine einfache Lösung, denn auch *MFC* hat eine String-Klasse mit Namen *CString*. Diese Klasse wiederum hat einen passenden Konstruktor, der *ATL*-Strings konvertieren kann. Sie können also den Text mit

```
Afx↳MessageBox
↳( CString(bstrHTMLText));
```

anzeigen.

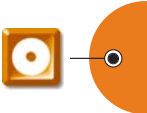
Links verfolgen

Das Beispiel zur Verfolgung der Links finden Sie im Quell-Code der *CMainFrame*-Klasse in der Funktion *OnLoadLink()*. Hier werden im Prinzip die gleichen Mechanismen verwendet wie zuvor – nur eben in etwas umfangreicherer Form. Zunächst benötigen Sie einen Zeiger auf die aktuelle View, um damit einen Zeiger auf das *IDispatch* Interface der *Chtmlview*-Klasse zu ermitteln:

```
CView* pView = MDIGetActive()-
↳>GetActiveView();
if( pView)
{
```



DIE „ABOUT“-BOX DES BROWSERS programmieren Sie in JavaScript und HTML und laden diese aus den Programm-Ressourcen.



```
if( pView->IsKindOf(
    RUNTIME_CLASS(CMyBrowse
    View)))
{
    CComPtr<IDispatch> pDisp =
        ((CMyBrowseView*)pView)-
        >GetHtmlDocument();
    // ... mehr code
}
```

Mit dessen und ATLS Hilfe ermitteln Sie einen Zeiger auf das IHTMLDocument-Objekt:

```
CComQIPtr<IHTMLDocument2,
    &IID_IHTMLDocument2> spHTML;
spHTML = pDisp;
if( spHTML)
{
    // mehr code...
}
```

Beachten Sie, dass Sie immer dann, wenn Sie das IHTMLDocument nach einer Ansammlung von Elementen fragen, etwa nach allen Links, das IHTMLCollection Interface verwenden müssen. Sie benötigen einen Zeiger auf eine IHTMLCollection:

```
IHTMLCollection*
    pLinks;
```

Nun können Sie mit Hilfe des Interface-Pointers auf das IHTMLDocument nach den Links auf der Seite fragen:

```
HRESULT hr = spHTML-
    >get_links( &pLinks);
```

Sie haben nun eine IHTML-ElementCollection und diese unterstützt verschiedene Methoden, von denen zunächst nur zwei von Interesse sind: `get_length()` liefert die Anzahl der Elemente in der Collection, und `item()` liefert ein Element aus dieser Collection. Mit beiden Methoden können Sie über die Collection iterieren:

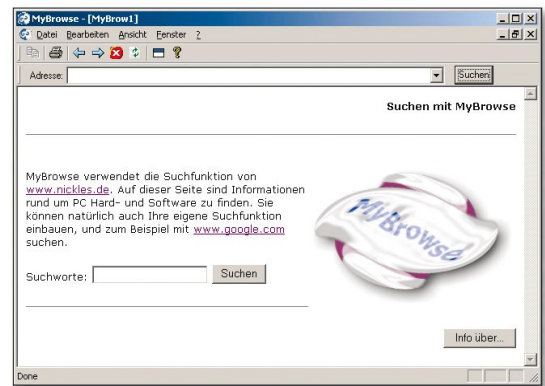
```
long c;
pLinks->get_length(&c);
for( long il=0; il<c; il++)
{
```

```
    CComPtr<IDispatch>
        pDispE;
    CComVariant v1( il);
    CComVariant v2( il);
    pLinks->item( v1, v2,
        &pDispE);
    // ... mehr code
}
```

Die `item()`-Funktion erhält drei Parameter: Die ersten beiden sind vom Typ VARIANT und dienen der Angabe des Indizes des zu liefernden Elements. Das bedeutet, Sie können eine CComVariant-Variable erzeugen, die den Index des aktuellen Schleifendurchlaufs enthält.

Der dritte Parameter ist ein Zeiger auf einen IDispatch-Zeiger. Darin liefert die Funktion den Zeiger auf das IDispatch Interface des erfragten Elements zurück.

Nun liegt Ihnen erneut ein IDispatch-Interface-Zeiger vor – und zwar erneut einer, der zu einem IHTML-Element gehört. Im Falle von Links unterstützt dieses Element nicht nur das IHTML-Element Interface, sondern auch das IHTMLLinkElement Interface und einige andere. Wenn Sie in der online nach „Links“ suchen, ist das IHTMLLinkElement das Interface, das Sie sofort finden werden. Wenn Sie dieses Interface zum Erfragen von Informationen verwenden, werden Sie enttäuscht. Dieses



EIGENER BROWSER – EIGENE SUCHSEITE: Welche Suche Sie verwenden, steht Ihnen offen. Das Beispielprogramm verwendet die Suche von www.nickles.de.

Interface liefert nur den Text, der zum Link gehört. Zur automatischen Navigation benötigen Sie die URL, und die erhalten Sie mit dem IHTMLAnchorElement Interface.

Dieses Interface wird vom IDispatch Interface, das Ihnen zur Verfügung steht, unterstützt, sodass Sie nun den IDispatch-Zeiger in einen IHTMLAnchorElement-Zeiger umwandeln können:

```
CComQIPtr<IHTMLAnchorElement,
    &IID_IHTMLAnchorElement> spE;
spE = pDispE;
if( spE)
{
    // ... mehr code
}
```

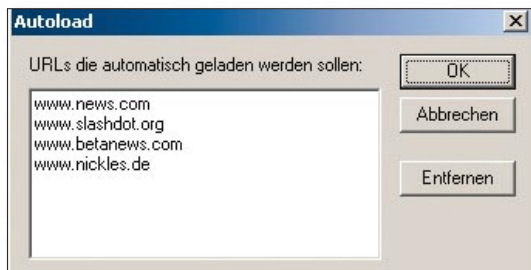
Das IHTMLAnchorElement hat unter anderem die Methode `get_href()`. Die liefert das gewünschte Ergebnis: die URL, die zum Link auf der Seite gehört.

```
BSTR bstr;
spE->get_href(&bstr);
```

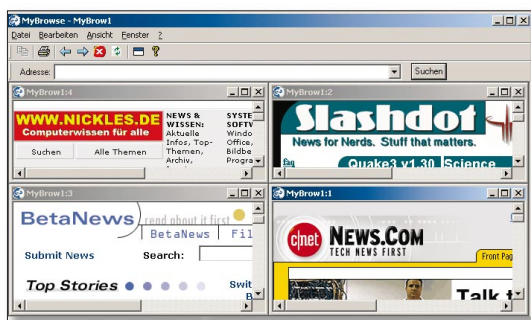
Den gelieferten ATL-String konvertieren Sie in einen MFC-String und verwenden die früher definierte Hilfsfunktion zum Navigieren. Zu dem müssen Sie zuvor eine neue MDIView anlegen, sonst würden alle Navigations-Schritte nacheinander im gleichen Fenster stattfinden:

```
OnWindowNew();
NavigateActiveView(CString(bstr));
```

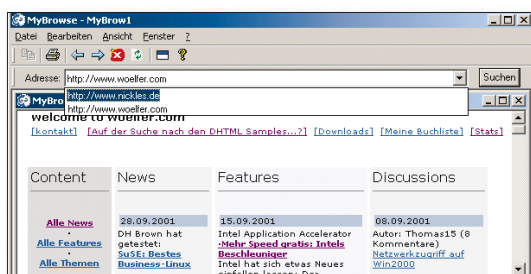
In diesem Beitrag haben Sie erfahren, wie Sie mit der `ChtmlView`-Klasse und der richtigen Verwendung von HTML und GIFs als Ressourcen, einen eigenen Browser zaubern können. Mit dem IHTMLDocument Interface können Sie sehr komplexe Funktionen rund um HTML-Seiten programmieren. Die richtig eingesetzte Verwendung der ATL-Hilfsklassen macht die sonst so aufwändige Kommunikation per COM recht einfach. UR



MIT DER AUTO-LOAD-FUNKTION können Sie mehrere Seiten auf einmal öffnen.



SEHR PRAKTISCH: Sie können mehrere Seiten gleichzeitig anzeigen. Das ist für die automatische Link-Verfolgung auch notwendig.



EINMAL EINGEGEBENE URLs merken Sie sich zur späteren Auswahl in der Combo-Box.