8 Übergreifendes

8.1 Sprachenabhängigkeit

Die Mehrsprachigkeit ist grundsätzlich vorgesehen in unserem kleinen Tool, doch leider aus Schlamperei ein wenig eingeschränkt. Bei vielen Aufrufen der

$itasks.//4FF//.$GiveText()

ist die Sprache nicht berücksichtigt worden, obwohl sie im entsprechenden Objekt als Klassen- oder Instanzvariable zur Verfügung stand. Mit dem FIND&REPLACE können diese fehlerhaften Vorkommen gefunden und ergänzt werden, beispielsweise im Menu **m_4fF**-Class, **$SetLanguage**:

FIND&REPLACE

Calculate c_LST(1,1) as $itasks.//4FF//.$GiveText('NavOpen',p_Language)

Im **w_Navigator** fehlt die Sprache bei fast allen Zugriffen, dort ist die entsprechende Variable **c_Language**, im Report **r_File** , in der **Startup_Task**, **w_Fields** und **w_Files** ist es dagegen die Instanzvariable **i_Language**, die anzufügen ist. Sogar im PROPERTY MANAGER der zweiten Seite des **w_4fF**-Fensters ist die Textermittlung fehlerhaft.

Bei den individuellen Datei-Methoden fehlt die aktuelle Sprache nur an zwei Stellen, in der **$DoEigenschaft** und der **$DoTaskRel** der **4fF**-Methoden, wo **p_Show** ermittelt wird. Hier muss freilich zuvor diese aktuelle Sprache erst einmal über einen neuen Parameter mitgegeben werden.

Das wiederum heißt, dass alle Vorkommen von **$DoEigenschaft** und der **$DoTaskRel** der **4fF**-Methoden überprüft werden müssen, weil ihnen nun als neuer, letzter Parameter die Sprache mitgegeben wird. Doch das ist nur notwendig, wenn **p_Show** tatsächlich verlangt wird, wenn also **p_Show_Line** größer 0 ist. Die **Startup_Task**, **$RefreshCalculation** entfällt damit sowohl für **$DoTaskRel** als für die **$DoEigenschaft**, geändert muss nur das **w_4fF**-Fenster in den Methoden **$SetDistanceField** und **RefreshCalculation** werden, wo **i_Language** am Ende des Aufrufs der **$DoTaskRel** erforderlich ist. Für die **$DoEigenschaft** hat in der **RefreshCalculation** dasselbe zu erfolgen, weiterhin in der **$DoJobs** und der **$CheckEigenschaft**. Die Aufrufe im **w_Files**-Fenster erfolgen indessen wieder ohne die Bearbeitung der Anzeigeliste **i_LST_Show** und können deshalb vernachlässigt werden.

Um die Sprache „OnTheFly„ zu ändern, können Sie dann beispielsweise im Menü **m_4fF** in der **$SetLanguage** noch eine Zeile einfügen:

m_4fF,
$SetLanguage

Calculate c_LST(1,7) as $itasks.//4FF//.$GiveText('ChangeLang',p_Language)

Calculate c_LST(2,7) as '$itasks.//4FF//.$ChangeLanguage()'

Damit es diese Listenzeile Nr. 7 überhaupt gibt, müssen Sie dies noch in der **$construct** berücksichten:

m_4fF,
$construct

```
For l_L from 1 to 7 step 1
    Calculate #F as c_LST.$add()    ;; empty list
End For
```

Gespeichert wird diese Sprachauswahl über den Navigator, wie Sie sich vielleicht noch erinnern. Damit diese Speicherung nicht bei jedem Start der Anwendung vernachlässigt wird, sollten Sie in der **$construct** der **Startup_Task** die Vorbelegung der Sprache entsprechend anpassen:

Startup_Task,
$construct

Do code method w_Navigator.$RememberValues

Um die kleine Änderungsroutine zu aktivieren, ist jedoch in den **$SetLanguage** aller Fenster eine weitere Schlamperei zu korrigieren. Nach der anfänglichen Bestimmung der jeweiligen Sprache muss noch folgende Zuordnung geschehen:

$SetLanguage

```
Calculate i_Language as l_Language
```

Auch sollte in all diesen Methoden der Wiederaufbau des betreffenden Fensters noch durch die Berücksichtigung der Hintergrundobjekte ergänzt werden:

$SetLanguage

```
Calculate #F as $cinst.$redraw()

Calculate #F as $cinst.$objs.Pane.$bobjs.$sendall($ref.$redraw)
```

Im Verlauf der Testerei verlor ich gelegentlich die Sprache, deshalb fügte ich noch in der **$RememberValues** des **w_Navigator**-Fensters nach dem Einlesen der Klassenvariablen die Absicherung ein:

w_Navigator, $RememberValues

```
If len(c_Language)=0

    Calculate c_Language as 'de'

End If
```

Der Navigator erfordert auch noch andere Ergänzungen, um „OnTheFly„ auf Sprachenänderung zu reagieren. Seine **$SetLanguage** muss um die Berücksichtigung der Extra-Texte erweitert werden, am besten am Ende .

w_Navigator, $SetLanguage

```
If i_LST_RFile.$linecount>0

    Do method RightMouse

    Do method RightMouse (1)

    Do method RightMouse (2)

    Do method SetTitle

End If
```

Die Methode zur Sprachenänderung selbst ist dann eigentlich furchtbar niedlich. Das Umständlichste dran ist noch die Auswahl der Sprachen anzubieten. Beispielsweise können Sie dazu ein **Popup Menu from list** wählen wie ich hier, doch natürlich wäre ein hierarchisches Menü bei einer Menüauswahl adretter. Sollte die Sprachauswahl dagegen in ein Fenster verlegt werden, ist diese Art der Auswahl, eine **Popup List**, sicher nicht schlecht, wobei auf der Hand liegt, dass mehrere Sprachen nicht mehr in dieser primitiven, „hardcoded“ Programmierung in die Auswahl gelangen dürfen.

Gefällt Ihnen auch die Einfachheit des **$appendlist** und des **$sendall**?

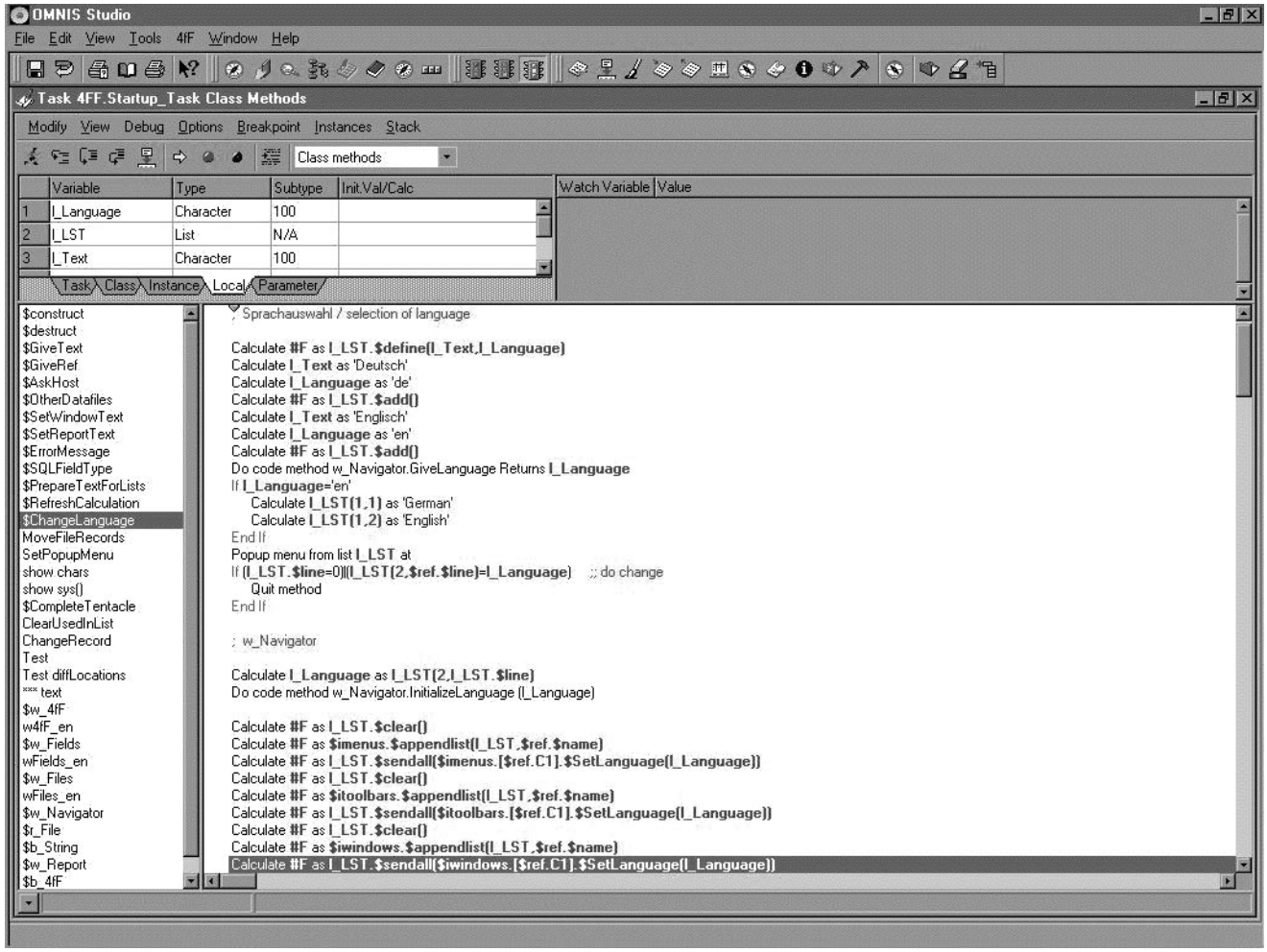

Abbildung 397 Sprachenänderung OnTheFly

8.2 Apple Macintosh-Anpassung

Bei Tests auf dem Apple Macintosh fiel auf, dass gar nichts ging. Oh nein, es hing nicht an Omnis oder der Graphik-Komponente, die wohl leider für Linux nicht verwendbar ist. Es hing an dieser Namenswahl **4fF**, die zu Syntax-Problemen führte. Und wie es scheint, ist der Apple Macintosh hier wesentlich strenger als die Windows-Seite.

Die Anpassungen sind deshalb ein wenig umfangreicher als die Fehlerbeseitigung bei der Sprachenabhängigkeit. Hier müssen sogar Namen geändert werden, denn einer der Fehler ist die Notation **$itasks.//4FF//**, weil die Instanz der **Startup_Task** denselben Namen wie die Library erhält. Die Library möchte ich natürlich nicht umbenennen, also muss ich es bei dem Instanzennamen tun, doch bevor ich dies durchführe, ändere ich zuerst einmal die individuellen Methoden. Das mache ich mir recht leicht. Ich verändere die gesicherten Sourcen der individuellen Datei-Methoden in der **SAVE**-Library mit **FIND&REPLACE**, füge die so korrigierten Methoden insgesamt über mein Werkzeug-Button des **w_Files**-Fensters in die Dateifelder **d_FIL_Methods** für meine Systemdateien ein und speichere erst einmal alles ab.

So wird dann natürlich gar nichts mehr funktionieren, also muss ich sofort danach dieselbe Ersetzungsaktion wie in den individuellen Methoden im ganzen restlichen Code bewerkstelligen. Warum ich das nicht gleichzeitig

tue? Weil ich das Abspeichern der individuellen Methoden nicht gefährden möchte.Was ersetzt wird?

Syntax-Problem entfernen

//4FF// durch p_4fF

//4fF// durch d_4fF

Diese exakten Ersetzungen kriegen Sie mit den Optionen des **FIND&REPLACE**

Match Case

Match Whole Words Only

Danach suche ich noch „4fF“ und „4FF“ mit denselben Optionen auf verbleibende Reste und finde nur zwei Stellen, die ich zu bedenken habe. In der **$CheckHas** der individuellen **Files**-Methoden werfe ich die Syntax-Bereinigung heraus, die nach dem Dateinamen abfragen musste:

individuelle Files-Methoden, $CheckHas

If p_File='4fF'

Hier kann ich nun, nachdem ich mir ja den „vernünftigen“ Dateinamen **d_4fF** ausgedacht habe, auf die Umschreibung mit der Schreibweise „//“ verzichten. Was ich vor dem Abspeichern in die binären Dateifelder weiterhin dringend machen muss, ist den Wert von **i_File** für die individuelle **4fF**-Methoden zu ändern auf **d_4fF**. Das mache ich einfach beim Kopieren des geänderten Codes aus der **SAVE**-Library in die **External Scripts** über die Rechte Maus auf dieser Variablen, wie gehabt.

Das zweite Vorkommen vom verbliebenen „4FF“ finde ich in der **w_Files**-Methode **$NewRecord**, dort wird auf das Template-Objekt meiner Library verwiesen. Da ich diese nicht umbenennen möchte, belasse ich erst einmal alles beim Alten. Sollten Probleme auftreten, muss dies natürlich trotzdem bereinigt werden.

Nachdem die individuellen Methode alle abgespeichert sind, führe ich diese Änderungen und Ersetzungen auch im restlichen Source-Code durch. Bitte beachten Sie, dass die Optionen von Omnis bei jedem Schließen des **FIND&REPLACE** wieder zurückgesetzt werden, damit Ihnen dieses praktische Tool nicht Chaos erzeugt beim Ersetzen.

Nun, nachdem also überall die Zugriffe auf die **TASK**- und **SCHEMA**-Namen abgeändert sind, müssen diese Namensänderungen dann auch tatsächlich durchgeführt werden. Der Name der **TASK**-Instanz wird rasch und als allererstes in der **$construct** der **Startup_Task** korrigiert:

Startup_Task, $construct

Calculate $cinst.$name as 'p_4fF'

Auch der **SCHEMA**-Name kann über den Browser bequem abgeändert werden, doch der Name des entsprechenden **Tables** auf dem Host ist nicht einfach über Menü oder Rechter Maus zu ändern. Glücklicherweise ist es ja nicht nur ein SQL-Dateisystem, sondern auch ein Omnis-Dateisystem, also kann ich den **DATA FILE BROWSER** verwenden und der erlaubt es mir, Namen zu ändern. Den **SERVER TABLE NAME** im **SCHEMA** und meine eigenen Daten in der **Files** und der **4fF** hätte ich zu guter Letzt fast vergessen zu ändern. **w_Files** muss ich mit SQL ändern, Dateinamen sollten nach meinem Konzept ja nicht einfach so manipulierbar sein.

interaktives SQL

update Files set d_FIL_File = 'd_4fF' where d_FIL_File = '4fF'

update d_4fF set d_4fF_File='d_4fF' where d_4fF_File='4fF'

Da ich sehr gespannt bin, wo ich jetzt stolpere, setze ich mir direkt zu Beginn der **$construct** der **Startup_Task** noch einen Befehl **Breakpoint**, um die ganze Start-Prozedur beim nächsten Mal verfolgen zu können.

Damit stelle ich fest, dass ich eine Ersetzung zuviel durchgeführt habe.

Startup_Task, $construct

Set reference l_Ref to $root.$libs.//4FF//.$schemas

muss solange stehen bleiben, wie ich die Library nicht umbenenne.

Als nächstes fällt mir auf, dass ich noch das Fenster **w_4fF** mit dem neuen Namen **w_d_4fF** versehen sollte. Hätte ich mich jetzt schön an meine eigenen Regeln gehalten, alle Zugriffe indirekt zu halten, müsste ich nun nicht den **FIND&REPLACE** bemühen. Bei einer Kernel-Technologie wäre es auch nicht nötig gewesen, da sich der Klassenname einer Schnittstellenobjektes wie einem Fenster nur in Ausnahmefällen nach den Dateien richtet und der Instanzenname sowieso auf den aktuellen Dateinamen zugreift.

Der neue Dateiname erfordert weiterhin, dass ich auch die externeText-Routine der **Startup_Task** neu benamsen muss in **$w_d_4fF**. So, jetzt sieht alles wieder aus wie zuvor. Wird's der Mac vertragen?

Ooops - zuerst lasse ich nochmal meine Routine zur Löschung der Verwendungsliste **ClearUsedInList** der **Startup_Task** laufen, doch nur für die betroffene Systemdatei **Fields**, die auf Datenfelder der **4fF**-Datei verweist. Schließlich ist die Erstellung der Verwendungsliste **d_4fF_UsedInList** nicht zur Beseitigung nicht mehr existierender Dateifelder vorgesehen und die **4fF**-Datei, die darin eben bei manchen **Fields**-Feldern verwendet wird, gibt es nun ja nicht mehr unter diesem Namen. Die Auflistung ihrer Dateifelder würde damit also nicht mehr rückgängig gemacht werden können.

Die alte Abfrage in dieser Bereinigungsprozedur kommentiere ich zur späteren Reaktivierung nur aus und füge dafür ein:

Startup_Task, ClearUsedInList

If l_File='Fields'

Schließlich brauche ich die anderen Dateien ja nicht zu manipulieren, sie sind nicht betroffen von der Namensänderung. Bevor ich jedoch das Neu-Durchrechnen aufrufe, kontrolliere ich noch alle Vorgänger und Nachfolger der **Fields**-Datei und korrigiere sie per erneuter Zuordnung zur jetzt umbenannten Datei, falls sie auf **4fF**-Datenfelder zugreifen. Dieses Neu-Durchrechnen führt zu neuen Typgewichten bei allen drei Systemdateien, offenbar war die vormalige Berechnung doch noch nicht auf dem besten Stand gewesen.

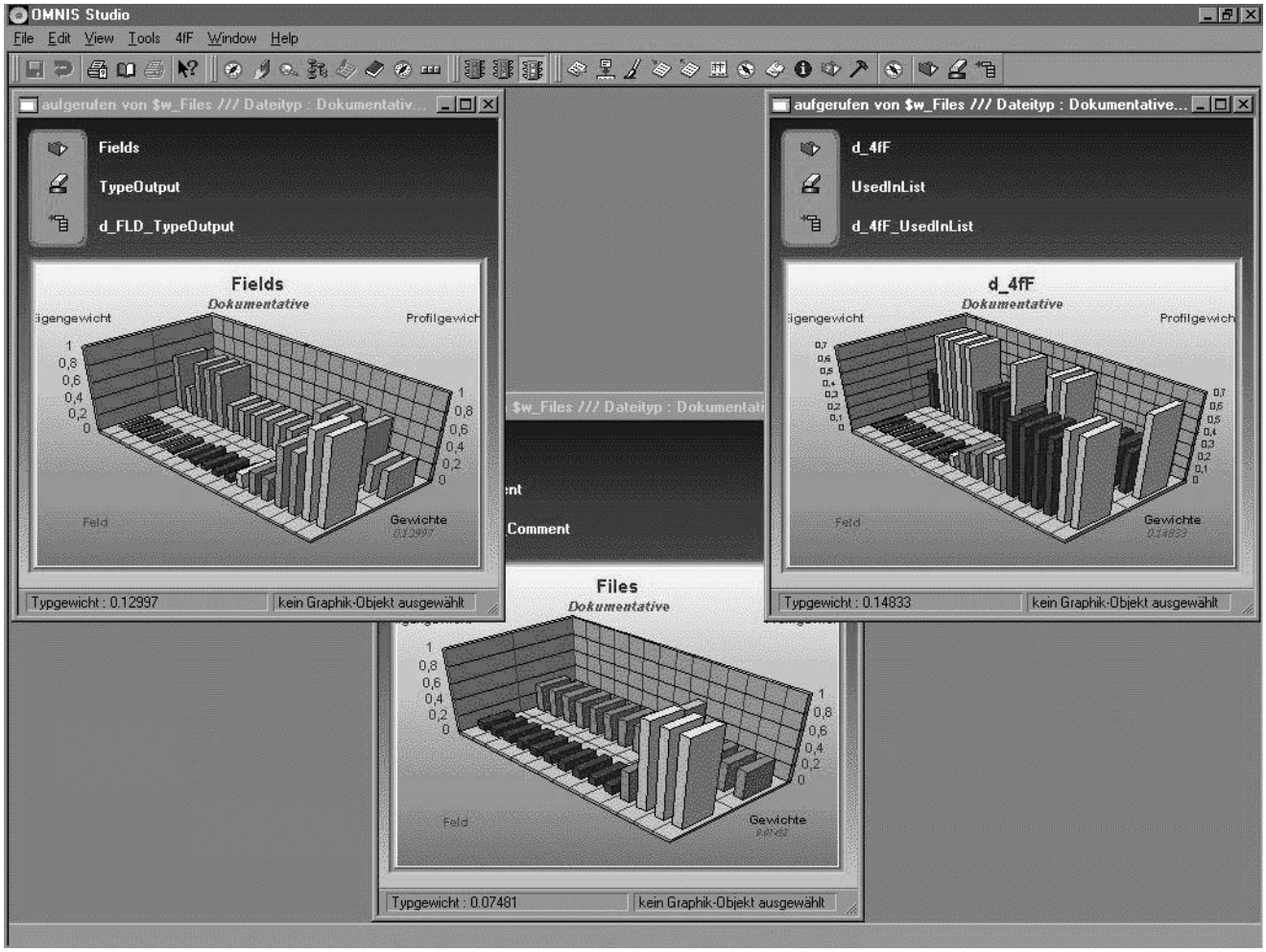

Abbildung 398 Dateien, Eigenschaften und Felder

8.3 Studio Version 3.0

Im Verlauf der Erstellung der kleinen Applikation ist Studio 3.0 auf dem Markt erschienen. Nach den Gerüchten, dass der ganze SQL-Zugriff überarbeitet werden sollte, bin ich natürlich ein wenig gespannt, ob irgendetwas auffällig ist, doch die Konvertierung läuft wie omnisgewohnt problemlos, es gibt keinerlei Schwierigkeiten.

Die alten SQL-Befehle (im Befehlseditor unter PreV3 SQL Commands zu finden) arbeiten noch immer einwandfrei, doch wenn Sie mehr Möglichkeiten hinsichtlich der SQL-Zugriffe haben wollen, beschäftigen Sie sich mit den neuen Session-Objekten (Hilfe-Topic: SQL Statement Object Methods, SQL Session Object Methods) von der Art **External Components**. Sie erlauben nun einen parallelen Zugriff auf die SQL-Server, das sogenannte Multithreading im Falle, dass Sie Omnis als Server verwenden. Genaueres erfahren Sie von Omnis selbst: OMNIS Software GmbH, Langhorner Chaussee 40, D-22335 Hamburg, www.omnis.net.

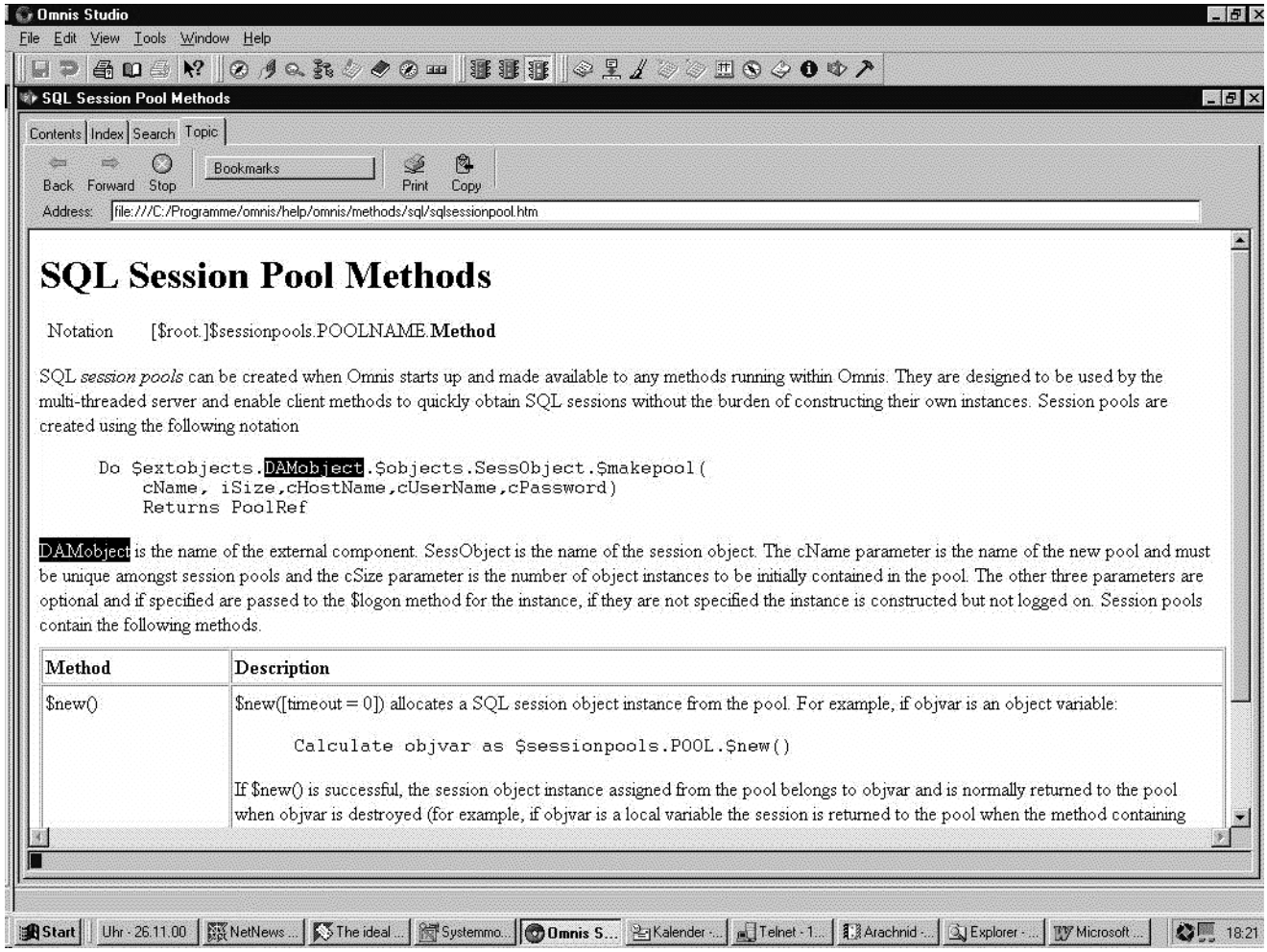

Abbildung 399 Hilfetext zu SQL Session Pool Methods

Was sonst noch neu ist? Vor allem sind es WebClient-Erweiterungen, die Studio V3 vorweisen kann. Da ich Ihnen dieses höchst begehrte Tool gar nicht vorstellte, überspringe ich auch ihre Verbesserungen. Last not least ist die Oberfläche, Studios IDE noch komfortabler geworden. Der Methods Editor, die Methodendarstellung, ist noch flexibler und vor allem, die Notation bietet nun die omnisgewohnten Hilfsbereitschaft. Dafür ist die Hilfe hinsichtlich der Notation augenscheinlich sehr gekürzt worden, doch wegen NOTATION INSPECTOR, INTERFACE MANAGER und PROPERTY MANAGER ist das kein unerträglicher Verlust.

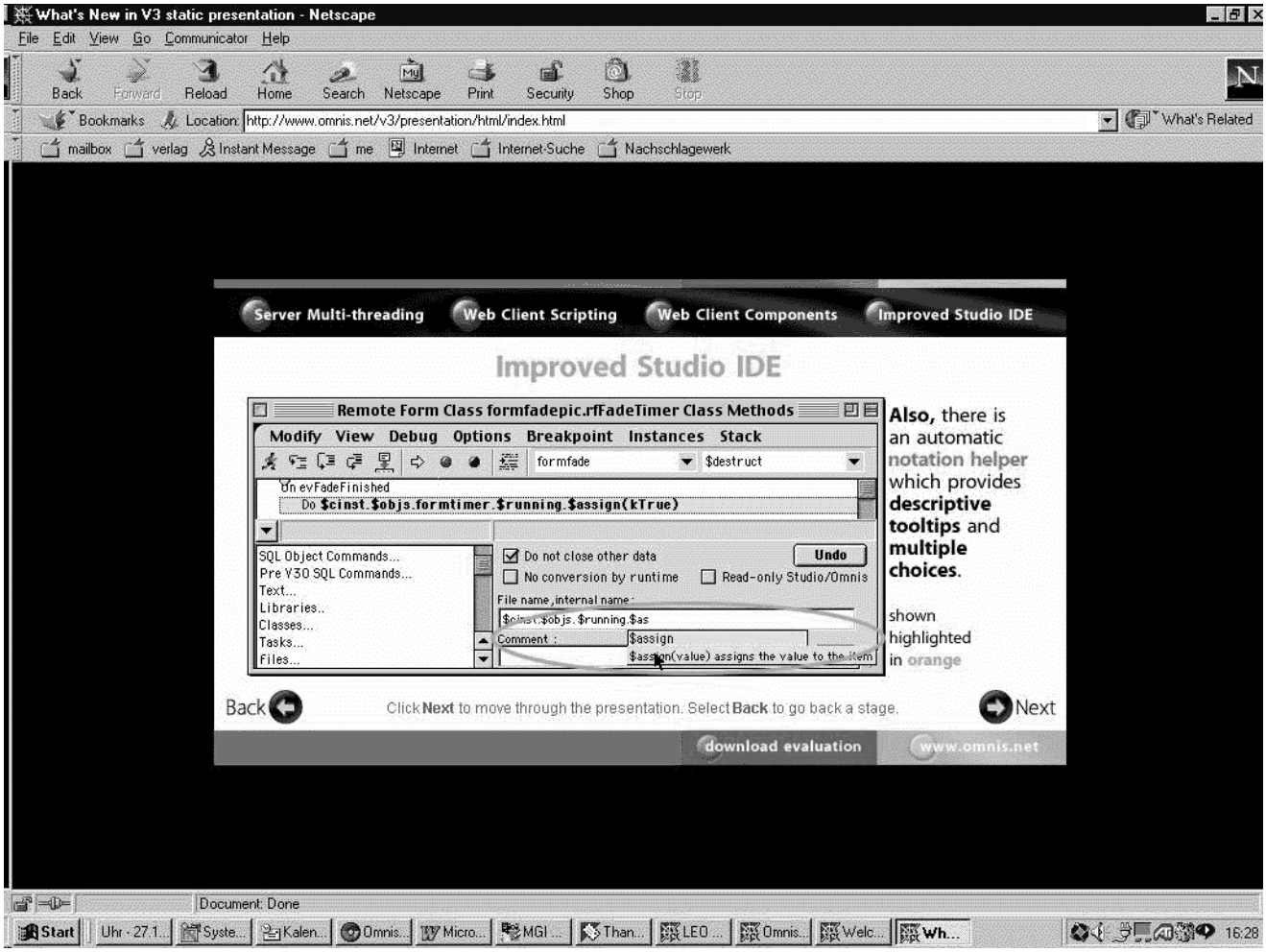

Abbildung 400 Notation mit Vorschlagsfunktion

Die WebSite zum Zeitpunkt von Studio V3 lautet http://www.omnis-software.com, dort sind die Neuerungen adrett, aber zeitraubend vorgestellt. Auch insgesamt wird Studio V3 noch gefälliger als sein Vorgänger, bietet noch unauffälliger und bequemer seine weitgespannte Funktionalität auf.

Also packen Sie's an!