Ich schätze, wenn ich weiter in solch einem ausführlichen Tempo voranschreite, erwürgen Sie mich? Ich werde also, schon aus reinem Überlebenstrieb, nun nicht mehr jede Einzelheit beschreiben. Falls Ihnen dennoch etwas unbekannt ist - und Sie finden es nicht im Abbildungs- oder Indexverzeichnis dieses Buches, so finden Sie es sicher in der Hilfe oder in den verschiedenen Tools, die Omnis Ihnen so bereitwillig zur Verfügung stellt. Auch die Handbücher sollten Sie nicht wirklich vergessen!

Zum Abschluss löschen Sie bitte noch per Rechter Maus auf die Task-Klasse, Auswahl „Close Task„ alle Instanzen. Sie können dies auch per Menü **INSTANCES** tun, das Sie in der Methodendarstellung der Task **t_File** vorfinden - denn eine Instanz ist eine dynamische Sache und „einfach so“ verwenden ist nicht ganz so einfach. Also warum auf einen fahrenden Zug springen, wenn Sie ihn anhalten und neu starten können? Sie können sie übrigens nur einzeln löschen mithilfe dieses Menüs - Sie müssen sie nur vorab aktivieren, erst dann kennt Omnis den genauen Zustand und kann sie löschen.

2.4 Das erste SQL

Sie haben schon Erfahrung mit SQL, nicht wahr? Das ist sehr gut, dann macht Ihnen der Zugang, den Omnis dazu zur Verfügung stellt, sicher Spaß! Für arbeitssparende Menschen ist es ein Vergnügen - doch wenn Sie sich gerne Mühe machen, dürfen Sie das natürlich auch, Omnis schreibt Ihnen da nichts vor. Sie müssen sich nur an die **TABLES** erinnern, doch da kann ich Ihnen nicht mehr helfen - ich bin ein arbeitssparendes Wesen ganz extremen Ausmaßes und meide unnütze Mühen!

Der nächste Schritt muss jetzt sein, unsere Dateien irgendwie zur Verfügung zu stellen, wir müssen sie also an den SQL-Host anknüpfen, das allseits bekannte „Login“ durchführen und den Datensatz der betreffender Datei aus **Files** einlesen.

Mit dem **Calculate** berechnen wir dabei nicht nur unsere neue Instanzenvariable **i_File** aus dem übergebenen Parameter, sondern führen auch die Methoden in unserer eigenen Instanz aus. Dazu stellt die Notation das Konstrukt **$cinst** zur Verfügung: „current instance“. Dabei gibt es eine interessante Unterscheidung zwischen dem kennzeichnenden Zugriff auf die Instanz **$cinst** und dem Objekt der Instanz **$cinst()** selbst. Wenn Sie diese verwechseln, können Sie nicht immer mit dem erwarteten Ergebnis rechnen: „$cinst.$name“ ist nichts weiter als heiße Luft, dieser Begriff ist zwar gültig, wartet aber sozusagen noch auf eine Vervollständigung.

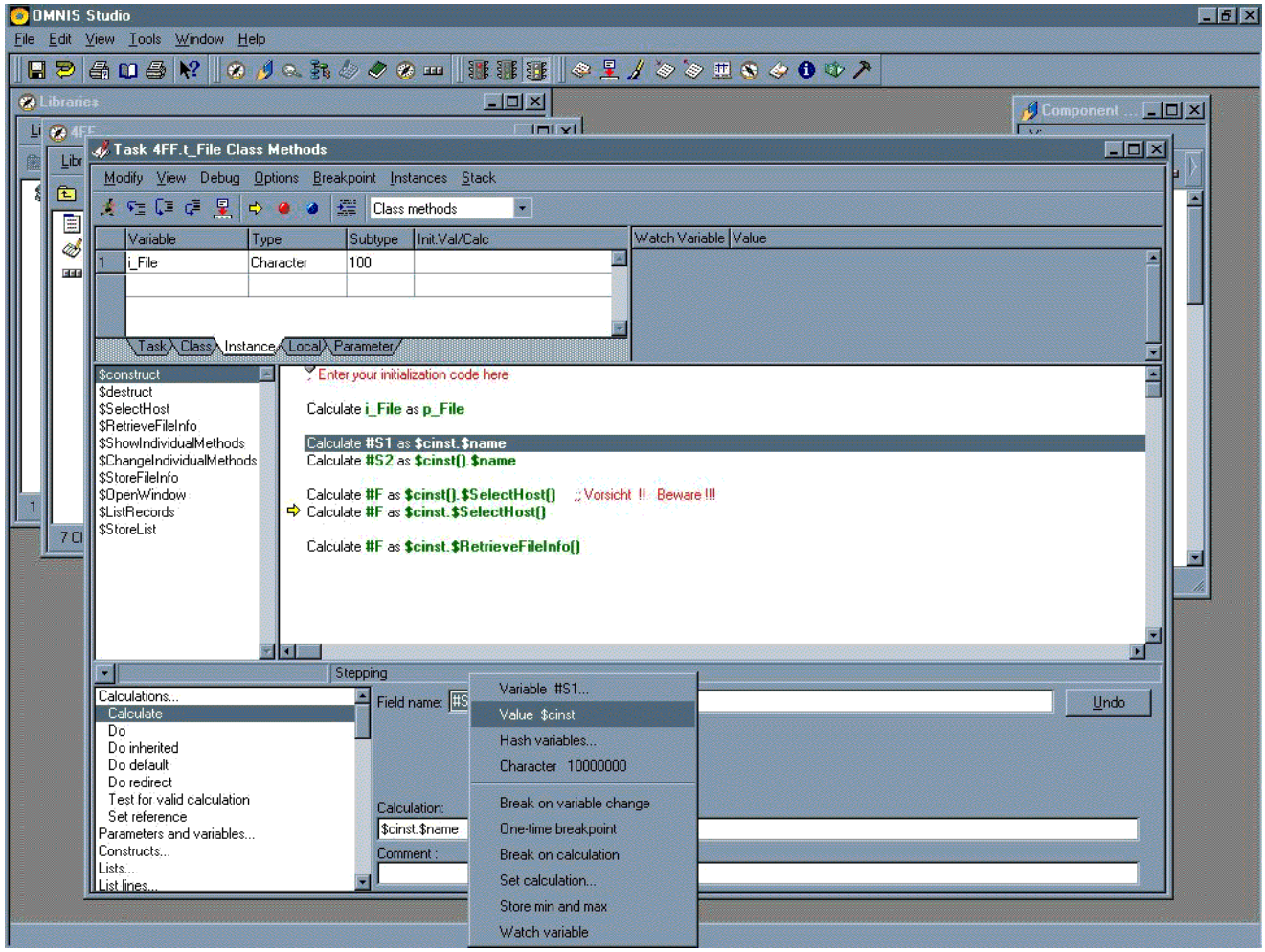

Abbildung 44 Verwendung von $cinst und $cinst()

Aber **$cinst().$name** gibt Ihnen den Namen des Objektes zurück.

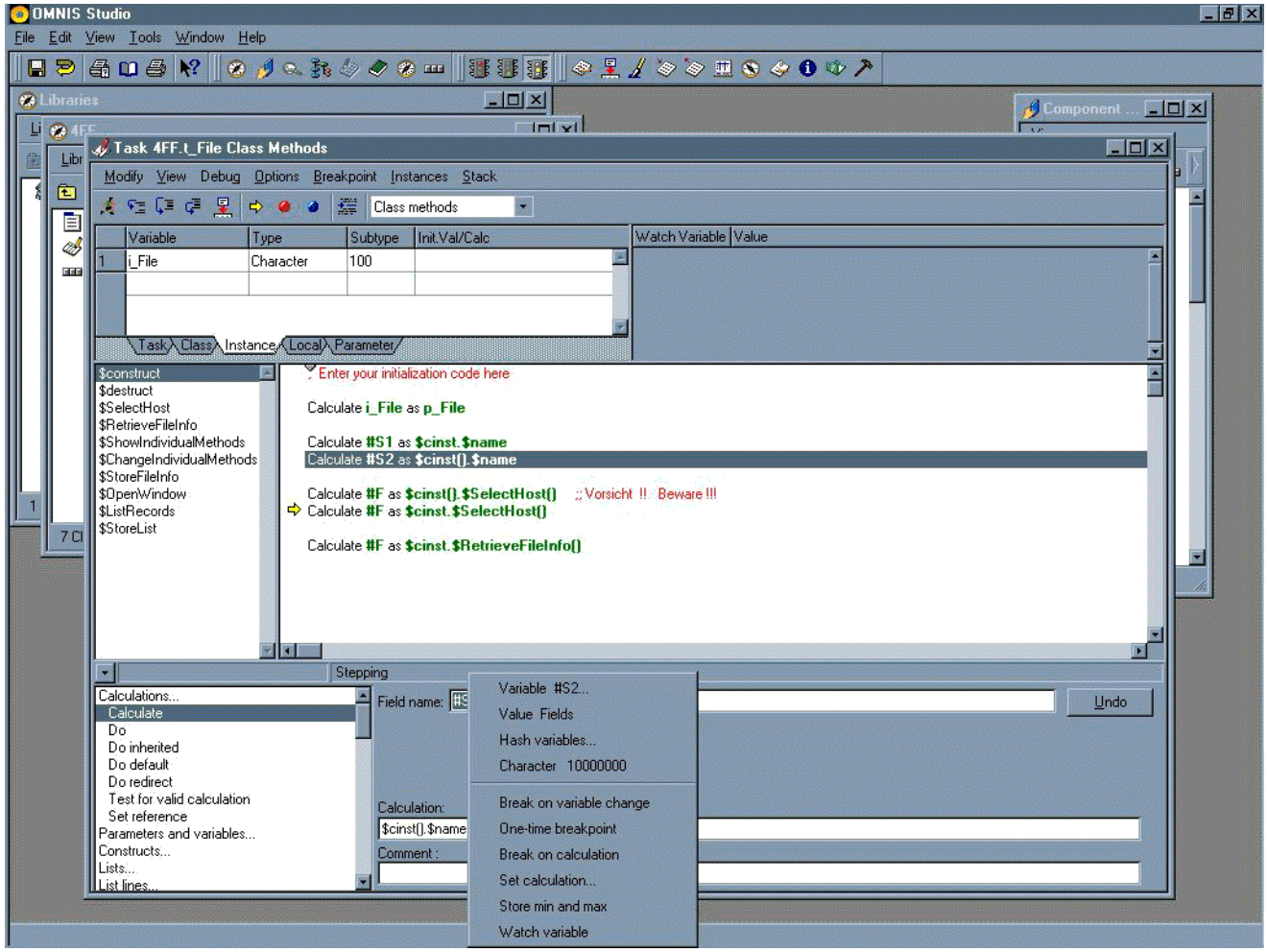

Abbildung 45 Verwendung von $cinst und $cinst()

Wenn Sie mit „$cinst().$MeineMethode“ innerhalb einer von vielen Instanzen agieren wollen, kann es schon mal vorkommen, dass Sie die Instanz wechseln. Sollten Sie also Probleme mit Instanzen haben - prüfen Sie Ihren $cinst-Befehl: eine Methode der eigenen Instanz rufen Sie viel sicherer mit **$cinst.$MeineMethode** auf! Das lässt sich nämlich nicht mit so einfachen Beispielen feststellen, da funktionieren leider beide Schreibweisen. Probieren Sie es - und glauben Sie mir: wenn Sie in mehreren Instanzen, möglichst noch in meinen geliebten EXTERNAL SCRIPTS herumhüpfen, dann können Sie mit der Klammerschreibweise Probleme kriegen!

Zurück zu unserem Job, unsere Datei als richtige SQL-Datei in einen Host zu pflanzen, damit der uns mit seinen Dienstleistungen beglücken kann. Da momentan kein SQL-System zur Verfügung steht, benutzen wir Omnis selbst. Wie üblich speichert Omnis seine Daten unabhängig von den Programmen. Während die ausführbaren Objekte in den Libraries untergebracht sind, die auf unseren Standard-Betriebssystem für europäische Verhältnisse (MS) die Objekttypisierung „lbs“ aufweisen, werden die datenenthaltenden Objekte unterlassungsmäßig gespeichert mit der Endung „df1“, aber mit dem gleichen Namen. Finden Sie schon sowas auf Ihrem Rechner?

Kein Problem - Omnis ist auch datenbankmäßig wirklich freundlich! Erstellen Sie einfach eines - mit dem DATA FILE BROWSER, der zweite Kompass in Ihrer höchstrangigen, der Haupt-Buttonleiste, auch über das Menü VIEW zu erreichen. Dieses Tool kontrolliert die Omnis-nativen Dateien und ist damit exakt der richtige Ort für uns im Augenblick. Sollten Sie dort ein Datafile sehen, schließen Sie es bitte erst mit dem Menü DATAFILE

des DATA FILE BROWSERS. Dann erstellen Sie ein neues Datafile mit dem üblichen Namen 4ff.

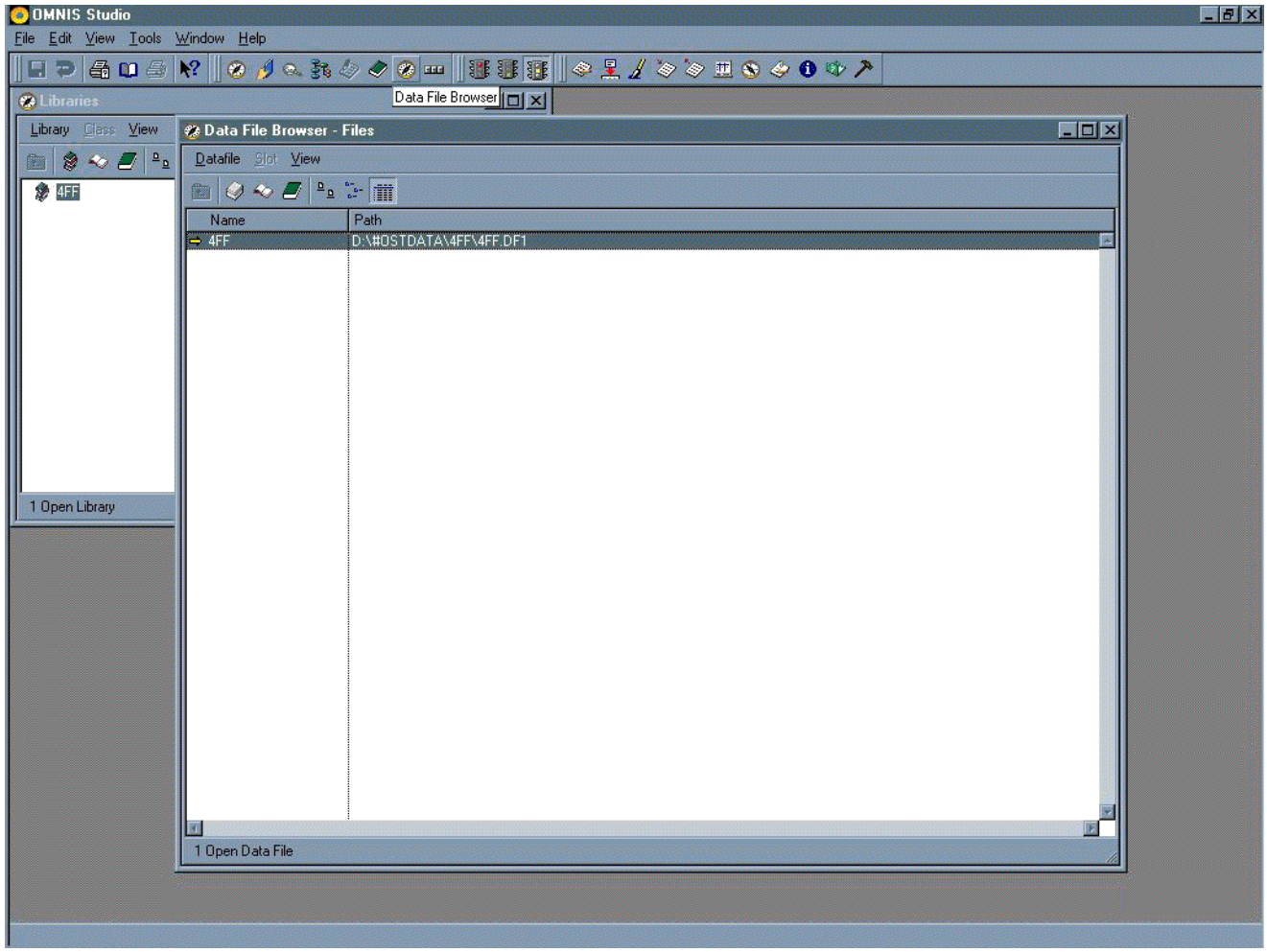

Abbildung 46 Data File Browser

Das Nächste, was ich Ihnen empfehle - da es ein neues Arbeitsgebiet für Sie ist - sehen Sie sich zuerst die Beispiele an, die als EXAMPLES in der WELCOME-Anwendung von Omnis angeboten werden, unter NEW USER zu erreichen. Sie finden darin einige Demonstrationen über SQL-Probleme. Die Schriftgröße können Sie übrigens mit diesem kleinen Schiebregler in der linken unteren Ecke einstellen.

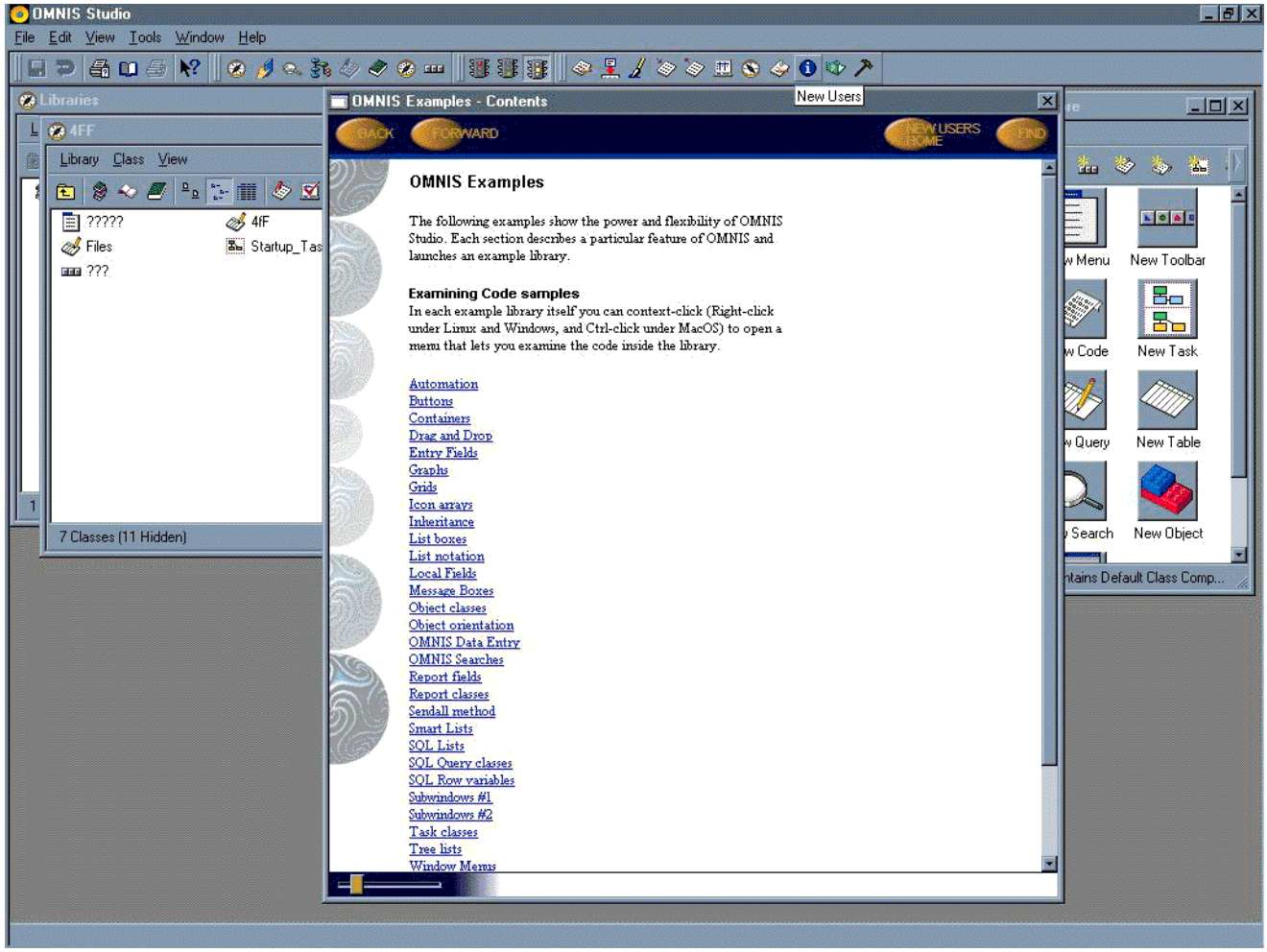

Abbildung 47 Examples

Schauen Sie sich ruhig die Beispiele an, spielen Sie sie durch und öffnen Sie im **BROWSER** die jeweiligen Libraries, um sich einen Überblick über die beteiligten Objekte zu verschaffen. Mit Doppelklick und Rechter Maus kennen Sie sich ja schon aus.

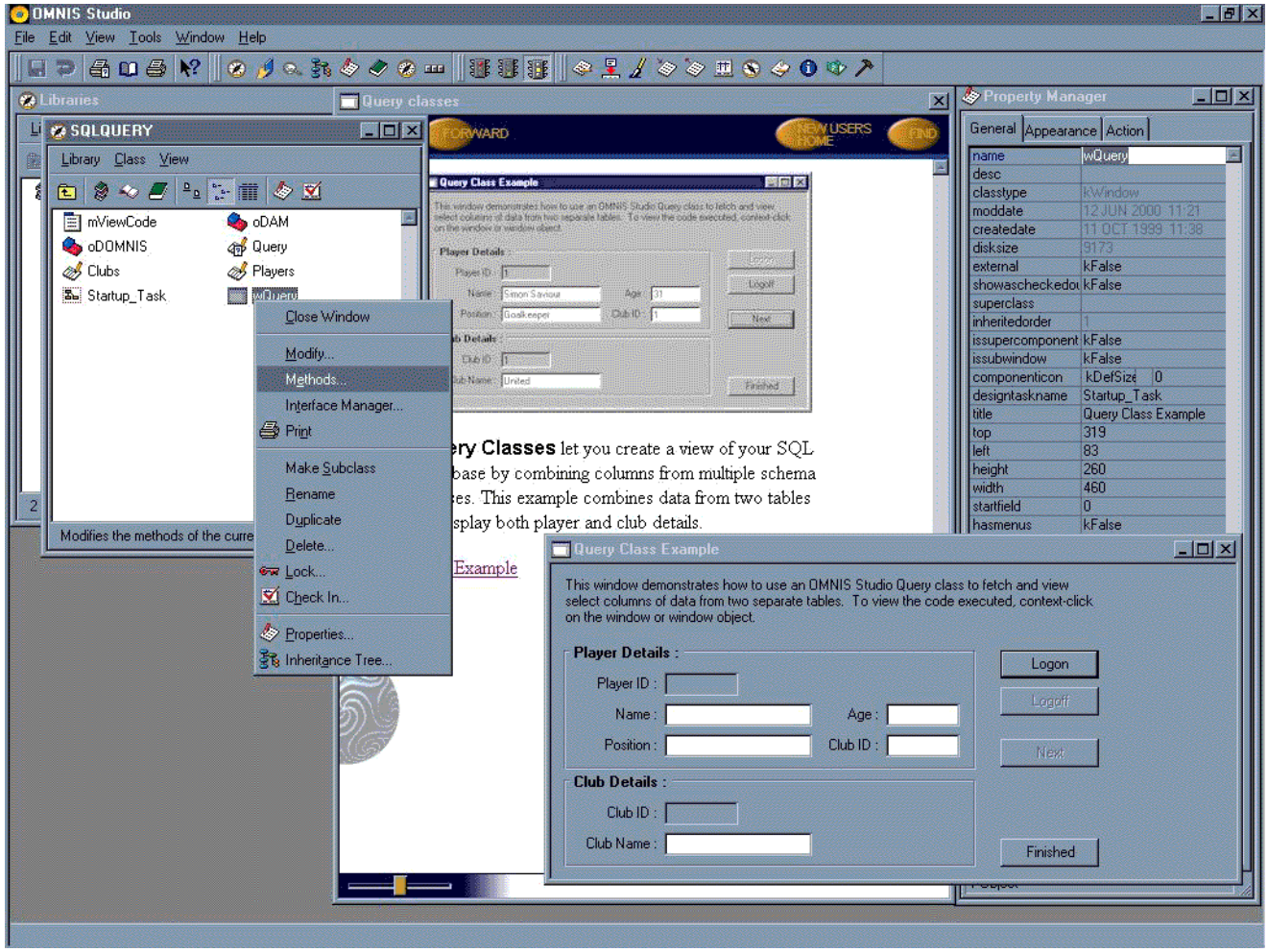

Abbildung 48 Example mit Beschreibung, Vorführung und vollständiger Programmierung

Ist Ihnen ein Unterschied zwischen den Beispielen aufgefallen? Haben Sie schon herausgefunden, warum ich das Beispiel „SQL Query Classes“ auswählte? Nun, es weist zwei Objekte auf, die die anderen in ihrer Library nicht kennen: **oDAM** und **oDOMNIS**. Dies ist ein untrüglicher Hinweis darauf, dass dieses Beispiel die moderne SQL-Version von OMNIS STUDIO® verwendet, denn auch OMNIS STUDIO® wurde nicht über Nacht aus dem Zauberhut gezogen. Es hatte äußerst respektable und zu ihrer Zeit unschlagbare Vorgängerprodukte, ohne die seine Mächtigkeit heute nicht denkbar wäre. Auch der so bequeme und intuitive Umgang mit SQL, den Studio heute anbietet, hatte bereits in seinem Vorgängerprodukt ein Pendant, das jedoch zwangsläufig nicht mit den neuen Objekten von Studio agieren konnte. Eine interessante Erfahrung meinerseits ist nun, dass es verständlicherweise zu Unvorhersehbarkeiten führen kann, wenn die alten Befehle auf die neuen Objekte angewandt werden - und Unvorhersehbarkeiten hasst wohl jeder Programmierer, schätze ich. Deshalb ein guter Rat: benutzen Sie im Umgang mit SQL ausschließlich die Notation, die auf die modernen Objekte zugreift! Oder noch allgemeiner: suchen Sie sich in Beispielen oder Hilfetexten, in denen das gleiche Problem auf verschiedene Weise gelöst wird, immer dasjenige mit der Notation heraus.

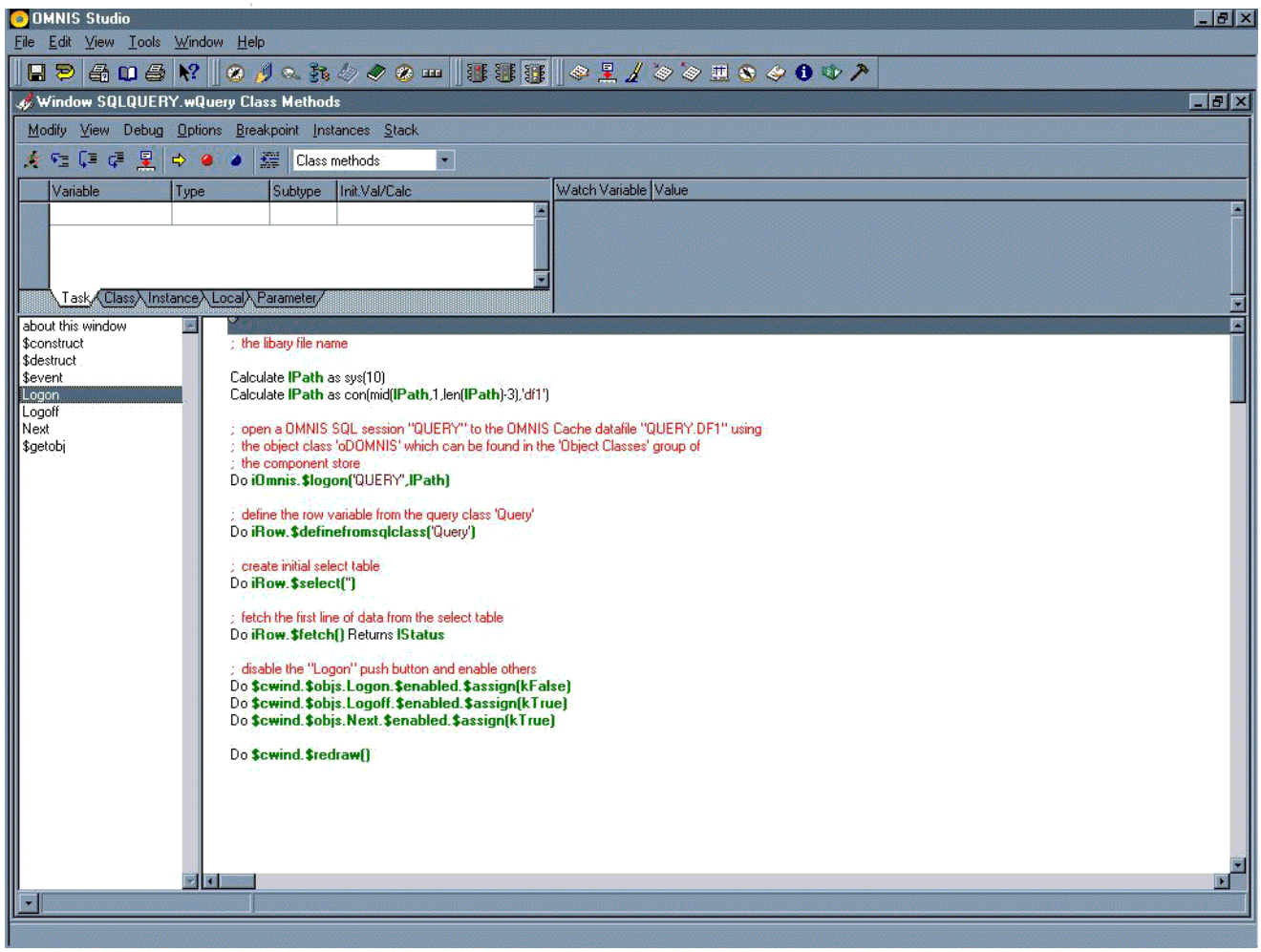

Abbildung 49 Example für Login

Dieses Beispiel passt prächtig für unsere Zwecke, nicht wahr? Das einzige, was wir nicht benötigen, sind die fensterabhängigen Aktionen, denn unsere Datei-Task soll den Login automatisch durchführen, Passworte sind kein Problem in unserem Pflichtenheft, deshalb dürfen wir sie problemlos vorbelegen.

Gefällt Ihnen das Beispiel? Ist das eine kurze und prägnante Schreibweise? So kurz, dass eine Methode **$RetrieveFileInfo** nur deshalb Sinn macht, damit sie später unabhängig vom Logon-Verfahren benutzt werden kann, nicht jedoch aus Gründen der Übersichtlichkeit und Wartungsfreundlichkeit der Anwendung.

Ok - dieses Beispiel nehmen wir, wir passen nur die Namensgebung später noch an unsere bisherige an. Gewöhnungseffekte als Hilfen bei der Arbeit soll man nicht unterschätzen, auch wenn es sich nur um die Ansicht von Feldnamen handelt! Wie Sie dieses Beispiel kopieren können? Ganz einfach! Wenn Sie das gesamte Beispiel kopieren wollen, aktivieren Sie einfach die linke Methodenübersicht und markieren alle Prozeduren oder lassen sich über das Menü **EDIT** mit der Auswahl „Select All“ das erledigen. Dann kopieren Sie mit demselben Menü **EDIT**, Auswahl „Copy“ oder dem Kurzbefehl „Str C“ bzw. „Ctrl C“ den ausgewählten Inhalt und fügen ihn in einer Methodendarstellung Ihrer Wahl. Wollen Sie nur eine einzige Methode kopieren, selektieren Sie eben nur diese in der linken Übersicht, Omnis wird dann auch nur diese Methode berücksichtigen. Beabsichtigen Sie gar, nur einzelne Befehle oder Teile von Befehlen zu übernehmen, so aktivieren Sie den Methodentext rechts und selektieren dort den gewünschten Bereich, wie Sie es auch in anderen Programmen tun.

In jedem Fall wird Omnis Ihnen nicht nur den sichtbaren Bereich kopieren, sondern auch alle Variablendefinitionen. Bei vererbten Instanzen sollten Sie da jedoch vorsichtig sein und die Instanzenvariablen genau überprüfen, wenn diese Variable nur aus der SUPERCLASS, also der Klasse, aus der die Vererbung hergeleitet wurde, stammen soll. Denn eine Instanzenvariable, die in einer vererbten Klasse überschrieben wurde, ist nicht dasselbe wie die Variable in der SUPERCLASS - so können Ihnen gelegentlich schon ganze Prozesse nur deshalb nicht mehr funktionieren, weil irgendeine wesentliche Instanzvariable aus der SUPERCLASS nun nicht mehr zur Verfügung steht, überschrieben durch einen Kopiervorgang! Sie sehen dies jedoch ganz einfach - Omnis stellt die Methodendarstellung mit Farben dar, wie es üblich ist in modernen Entwicklungsumgebungen, und tut dies auch für Vererbungsstrukturen. Wenn Sie nichts an der Farbauswahl geändert haben, erkennen Sie die vererbten Methoden und Variablen an der Farbe „blau“ - zeigt eine Ihrer Methoden oder Variablen nicht diese Farbe, dann ist sie nicht vererbt, also entweder neu oder überschrieben.

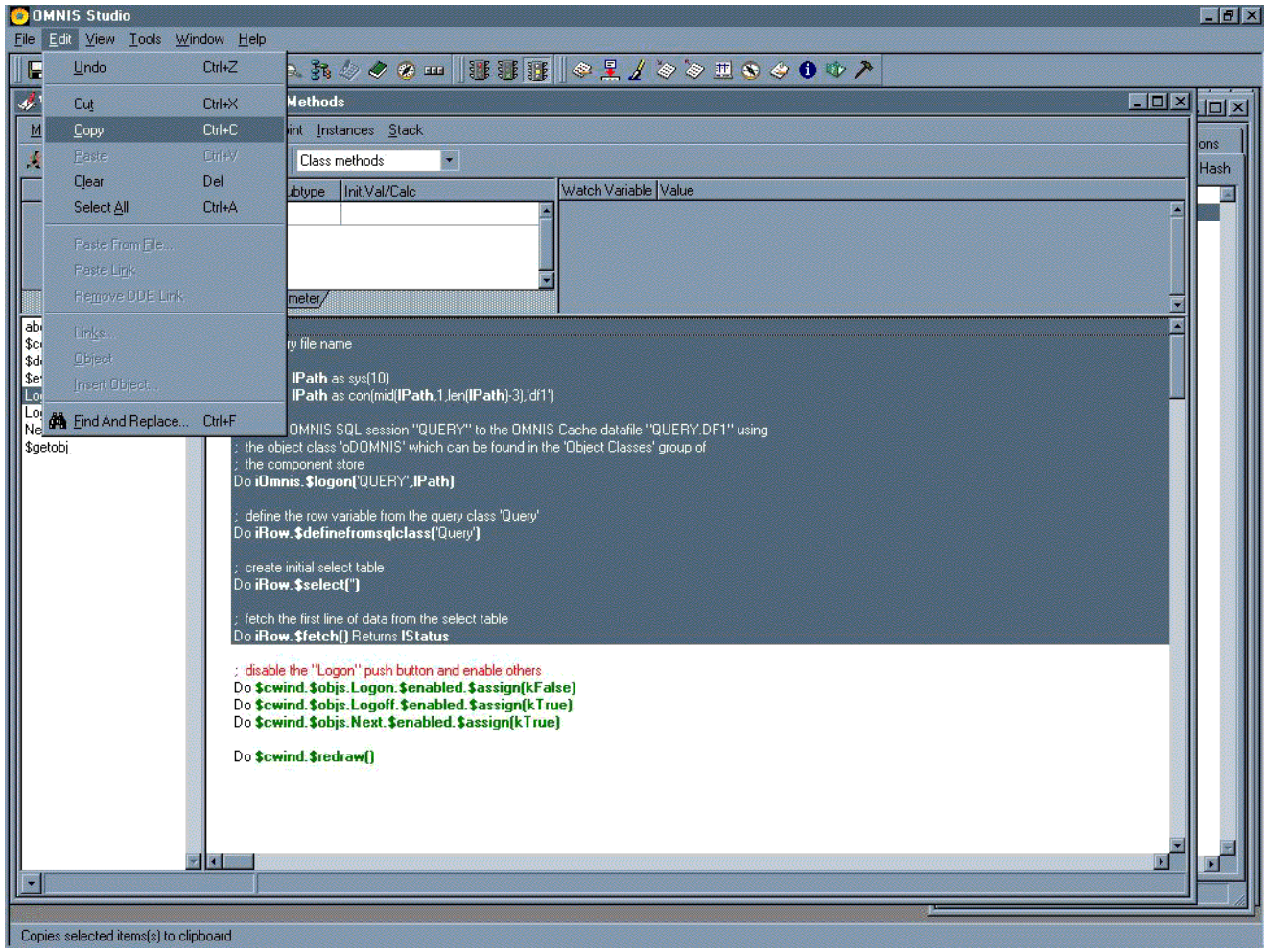

Abbildung 50 Kopieren von Methoden oder Methodenteilen

Leider ist hier die Rechte Maus nicht sonderlich hilfreich in der Methodenübersicht, dem linken Teil der Methodendarstellung, doch im Methodentext können Sie auch über die Rechte-Maus-Funktionalität „Kopieren“ und „Einfügen“ verwenden. Öffnen Sie also bitte mit dem Browser nun die Library **4fF** und die Methodendarstellung der Datei-Task **t_File**, aktivieren Sie die Methode **$SelectHost** und positionieren Sie den Methodentext an der Stelle, an der der kopierte Text eingefügt werden soll.

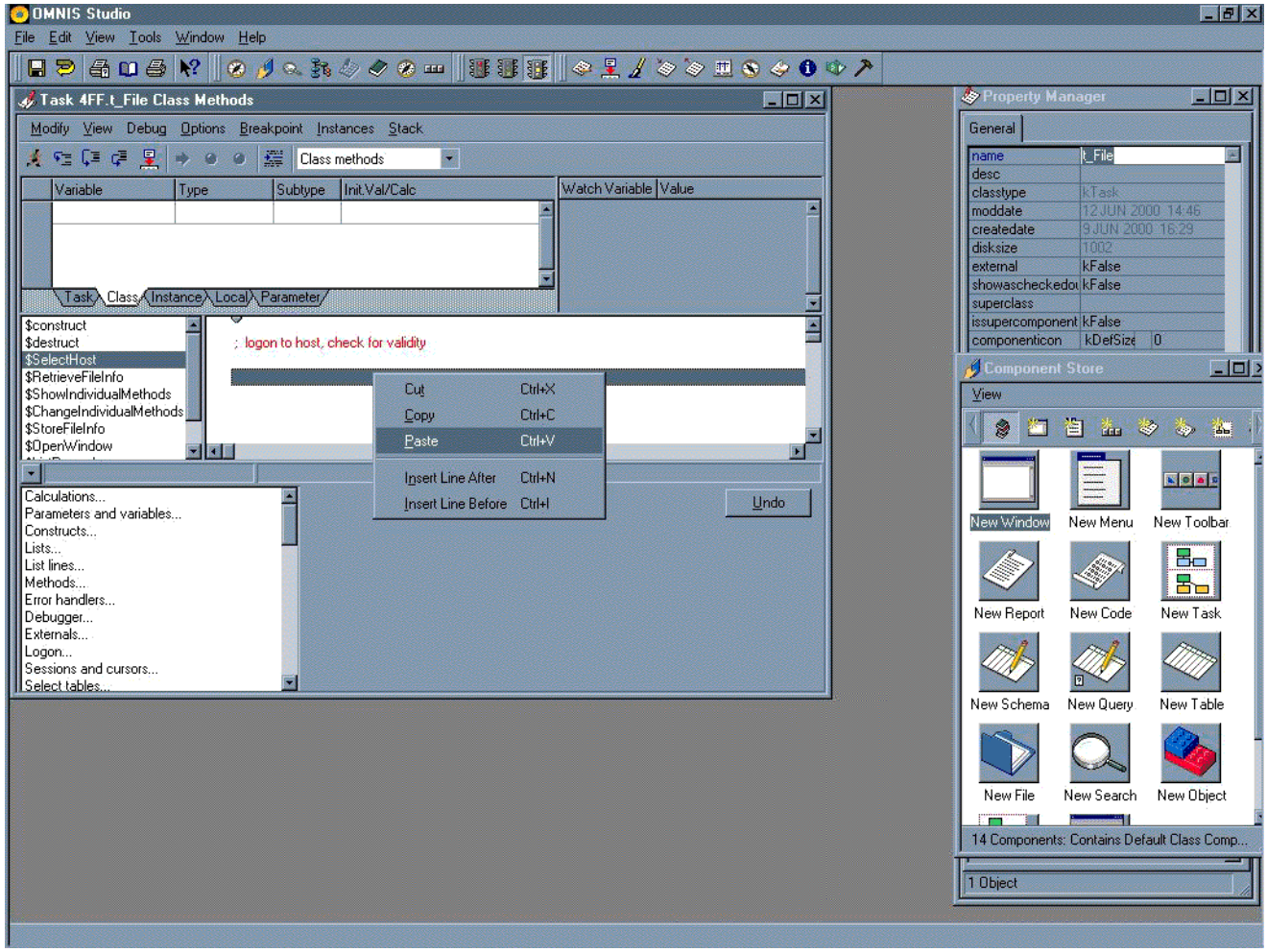

Abbildung 51 Kopieren von Methoden oder Methodenteilen

Alle Variablen, ob lokal oder Instanz, sind mit vollständiger Definition übernommen worden. Praktisch, nicht wahr?

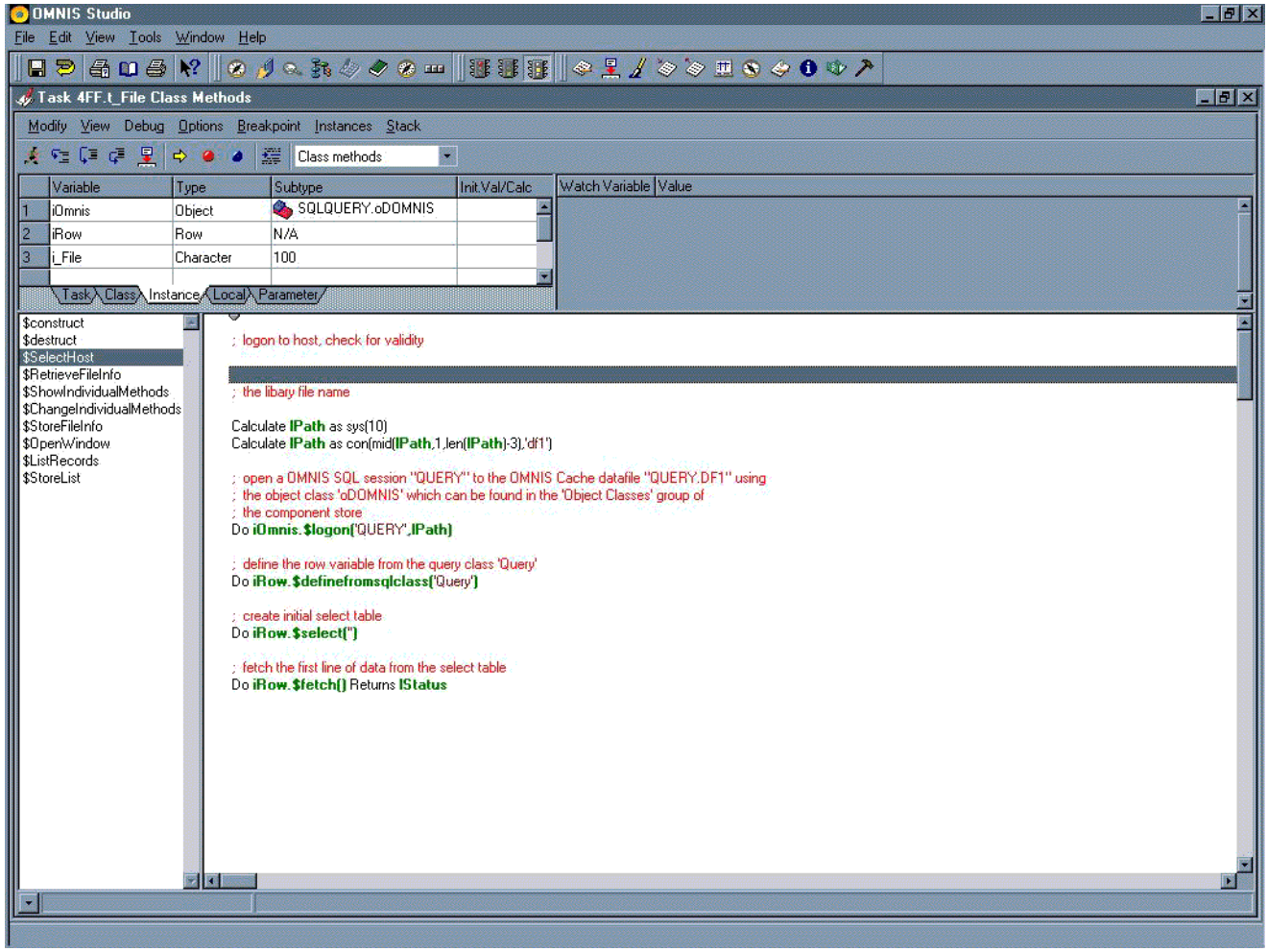

Abbildung 52 Einfügen von Methoden oder Methodenteilen

Das interessanteste Objekt in diesem Beispiel ist das Objekt **iOmnis**, es regelt den gesamten Zugriff auf den Host, mehr brauchen wir nicht als diese Objekt - aber wir haben es nicht! Es befindet sich immer noch in der Beispiel-Library **SQLQUERY**, wie uns die Variablenübersicht anzeigt. Aber da exakt dieses Objekt für uns interessant ist, suchen Sie es bitte im **COMPONENT STORE**, den Sie auch mit der Funktionstaste „F3“ in den Vordergrund Ihrer Bearbeitung holen können. Ein Tipp - Sie können die Masse der Objekte über die kleine Toolbar im **COMPONENT STORE** einschränken, wählen Sie dort einfach „Object Classes“ aus. Omnis bietet Ihnen dann alle bekannten SQL-Datenbanken an! Soweit Sie natürlich Zugang zu diesen Datenbanken haben, selbstverständlich. Da uns im Moment nur die Omnis-eigene Datenbank bedient, holen wir uns nur das Objekt **dOMNIS** in unsere Library.

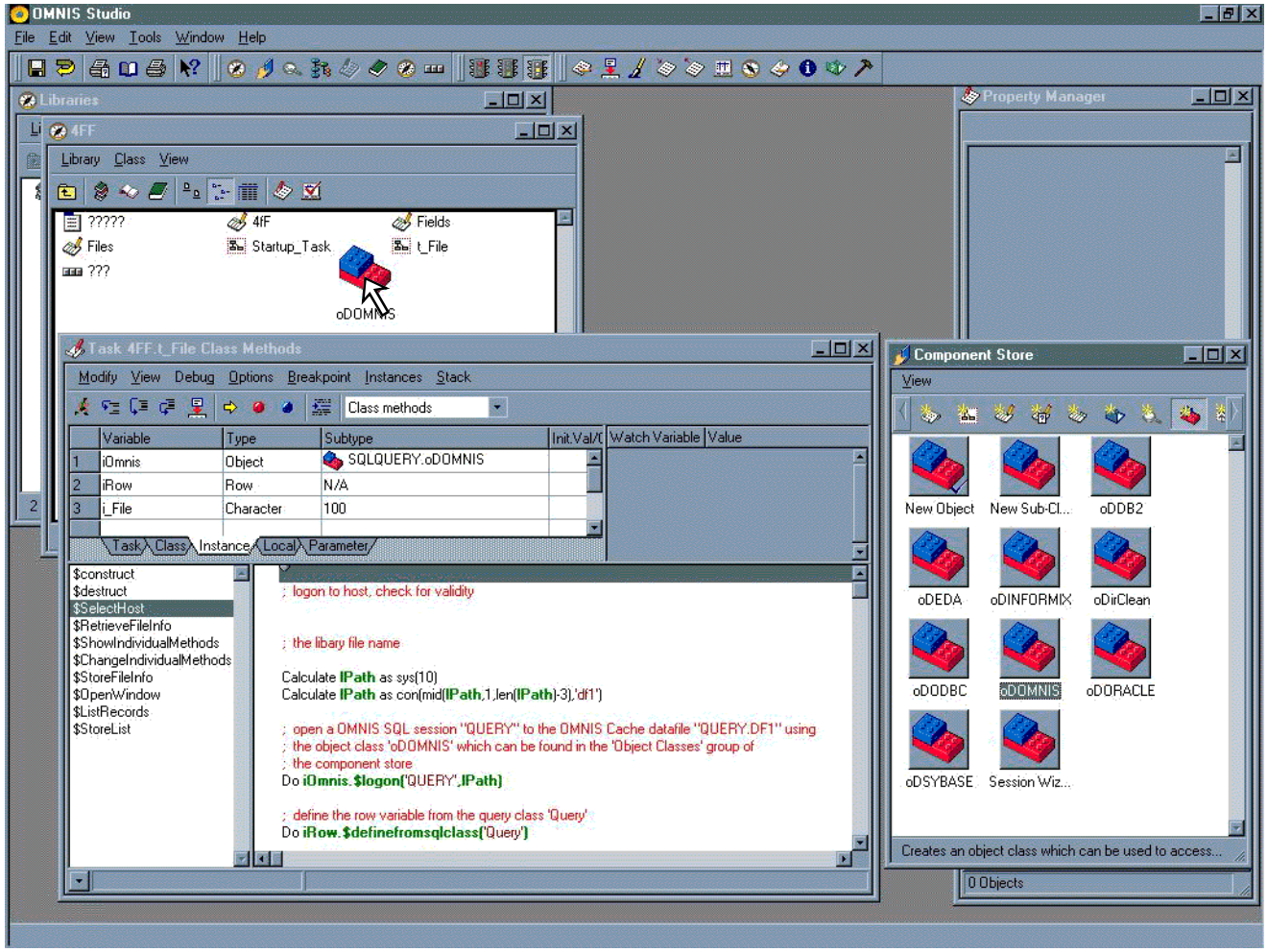

Abbildung 53 Host-Anbindung in Omnis

Eingedenk des Beispiels, aus dem wir die Vorlage herhaben, irritiert Sie da nichts? Schauen Sie sich doch einmal dieses Objekt an! Klicken Sie es an - dann arbeitet Omnis kurz und Sie bekommen automatisch das SUPERCLASS-Objekt mit Namen **oDAM** als Beigabe. Das geht bei Ihnen nicht? Dann bitte löschen Sie das Objekt und holen sich es einfach nochmal vom Component Store, klicken Sie es dann an oder betätigen die Rechte Maus, dann erzeugt Omnis Ihnen die SUPERCLASS. Nun, da Sie in die Library **4fF** das Objekt **dOMNIS** einfügten, können Sie in der Variablenübersicht für die Variable **iOmnis** dieses Objekt auch auswählen.

Doch wie testen? Die ganzen Instanzen aktivieren für einen einzigen, kleinen Test? Natürlich nicht - kopieren Sie einfach die Methode bis zur Logon-Zeile in eine neue Prozedur in der Startup-Task mit Namen Test, jedoch erst, nachdem Sie bitte den Variablentyp auf das Objekt in unserer eigenen Library umgeändert haben, sonst müssen Sie ihn zweimal ändern, da wir ja unser eigenes Objekt testen wollen und nicht das in der Beispiel-Library. Tauschen Sie zuletzt noch die Zeichen ‚Query' gegen ‚**4fF**' aus, bevor Sie die Befehle wieder mit dem DEBUGGER stepweise ausführen.

Naja, geschieht nichts - dummerweise haben wir mit der bereits aktiven Instanz unserer **Startup_Task** Probleme, doch erfinderisch wie wir nun mal sind, ändern wir einfach den Typus der Objektvariablen **iOmnis** in **lokal** um. Geht im Moment nicht über die einfache Drag&Drop-Funktion, mit der das Feld **iOmnis** aus der Variablenübersicht einfach dadurch geändert werden kann, dass seine Zeilennummer auf den „Tab-Button", also die Auswahl-

möglichkeit in „Reiter“-Form mit der Aufschrift „Local“ gezogen wird? Omnis ist sehr streng mit einer aktiven Instanz ist, doch umgehen Sie das bitte durch einfaches Einfügen einer neuen lokalen Variablen mit denselbem Namen und Eigenschaften.

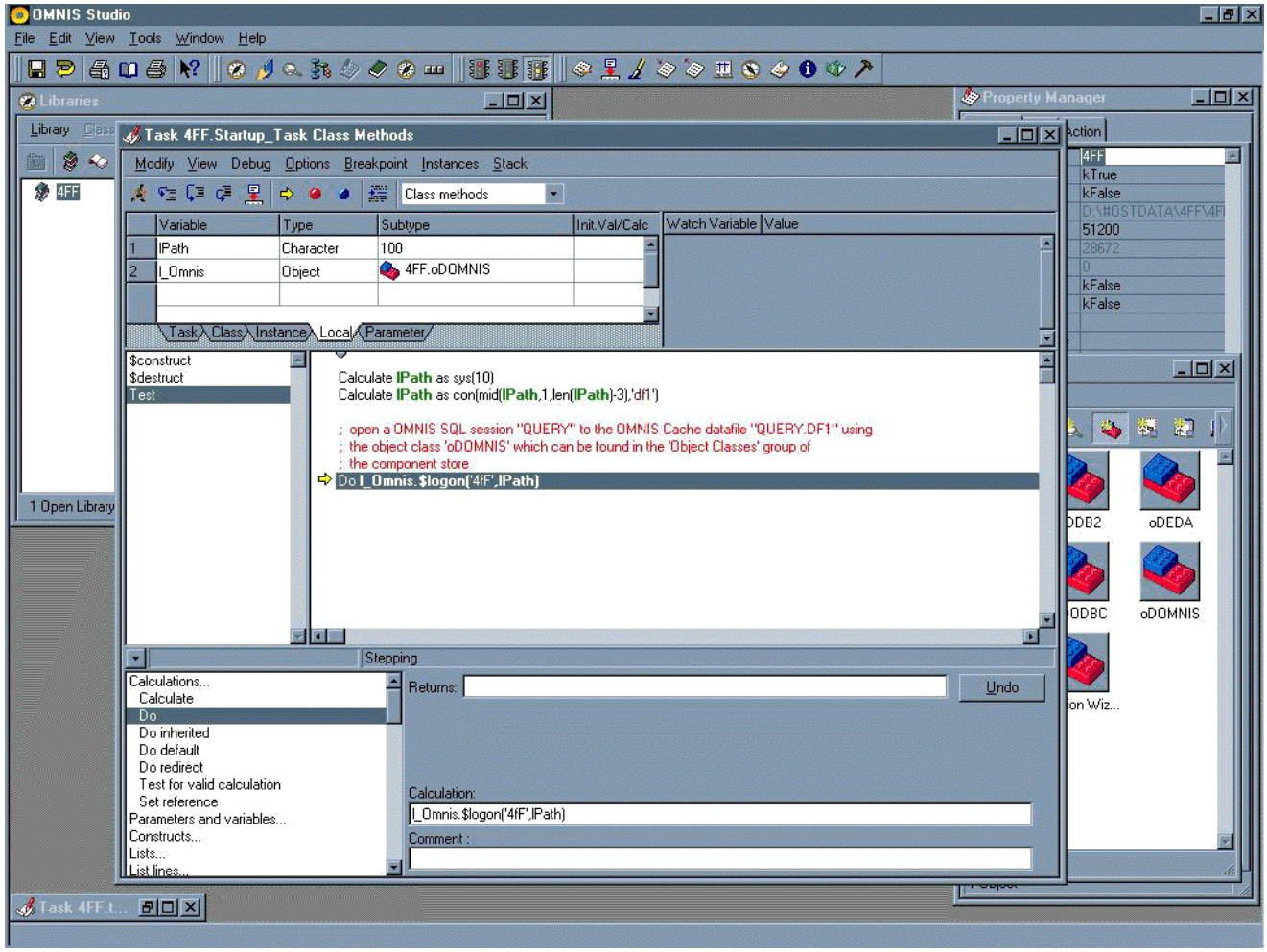

Abbildung 54 Testfall Logon

Nun probieren Sie es wieder - was geschieht? Direkt nach dem Befehl zur Durchführung des **Logon** befinden Sie sich tatsächlich in dem Objekt, denn der Variablentypus **OBJEKT** benötigt keine eigenständige Initialisierung. Jedesmal, wenn Sie irgendwo eine Variable mit diesem Typus verwenden, die noch nicht instanziert ist, führt dies Omnis automatisch für Sie durch.

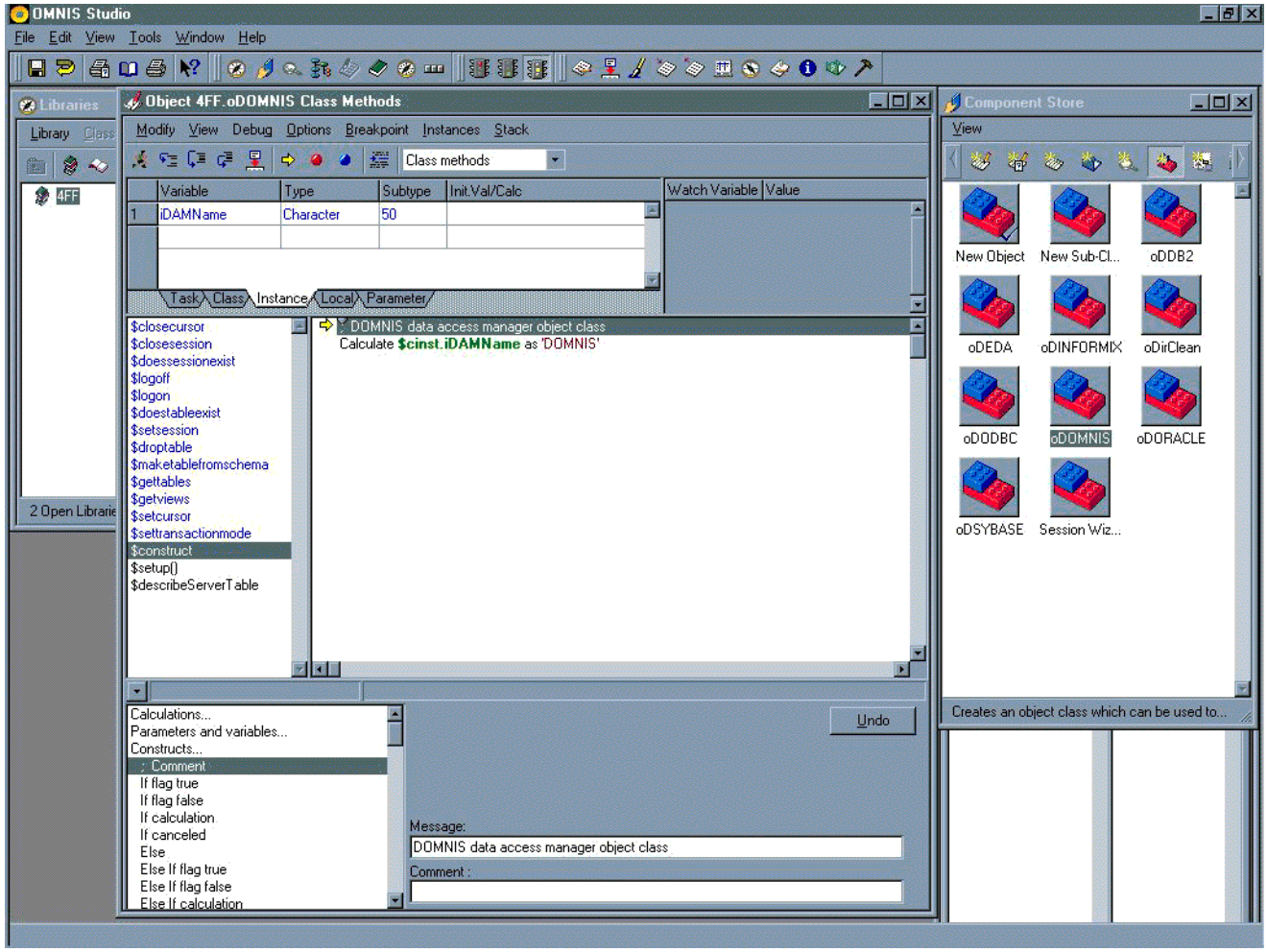

Abbildung 55 Instanzierung von Variablen des Typus Objekt

Ok - Sie sehen hier die ganzen Methoden der **SUPERCLASS** (in Blau) und nach der Instanzierung des „vererbten„ Objektes führt Sie Omnis brav in die gewünschte Methode der **SUPERCLASS** ein.

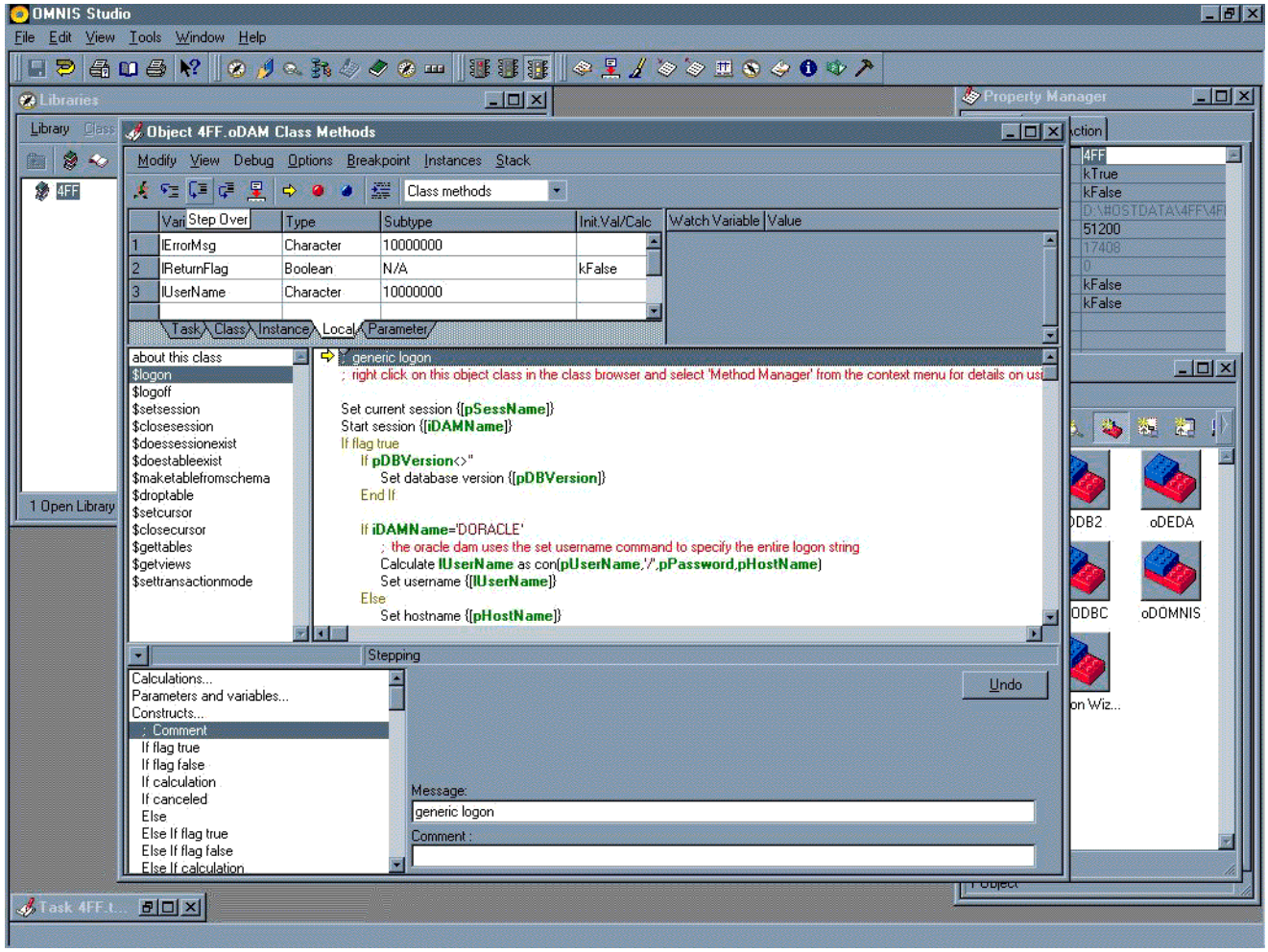

Abbildung 56 Aufruf einer Superclass-Methode

Und was geschieht als Nächstes? Omnis beschwert sich wieder über unseren Namen! **4fF** mag ja für eine Methode nicht übel sein, für die Bequemlichkeit in Omnis sollten Sie jedoch einen führenden Buchstaben für Ihre Namen wählen. Also sollen wir wieder von vorne anfangen?

Aber wozu denn? Erinnern Sie sich nicht mehr an die Rechte Maus für Variable? Fügen Sie doch einfach ein führendes „s_“ ein!

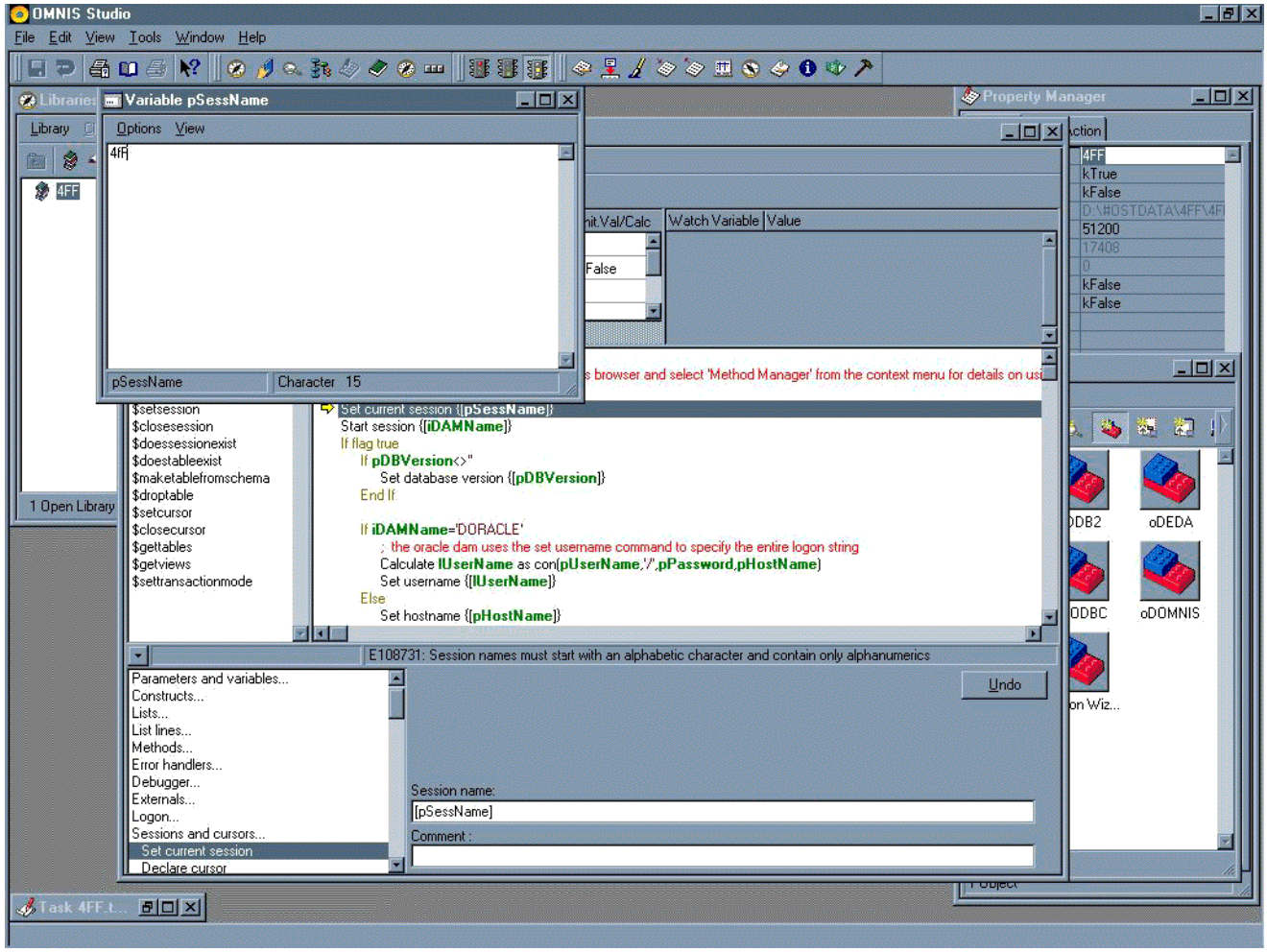

Abbildung 57 Eingriff in laufende Verarbeitung

Jetzt ist Omnis zufrieden - und Sie werden durchgelassen. Die ganze Prozedur scheint nunmehr erfolgreich zu sein, vergessen Sie aber bloß nicht, dass Sie den Session-Namen dann auch tatsächlich im Programm-Code ändern.

Und es hat tatsächlich problemlos geklappt! Prüfen Sie es mit dem **SQL Object Browser**, Sie finden dort Ihre Session wieder. Doppelklicken Sie die Session, so öffnet sich Ihnen die Übersicht aller Objekte dieser Session - freilich gibt es nur ein einziges. Wenn es Sie interessiert, öffnen Sie den **Data File Browser** zwecks näherer Informationen, er zeigt Ihnen, dass **New File** aus der **Component Library** stammt, wenn Sie sich über Doppelklick zum Posten „Data slots“ und von dort aus zur Einzelauflistung der Dateien durchhangeln. Lassen Sie sich nun aber über die Rechte Maus mit der Auswahl „Insert Data“ das interaktive SQL von Omnis zur Verfügung stellen, so sehen Sie, dass dieses File nicht nur keine Felder enthält, es ist auch nicht auf dem Server bekannt. Dies zeigt Ihnen die Fehlermeldung in der Statusanzeige, am Fuße des SQL-Fensters.

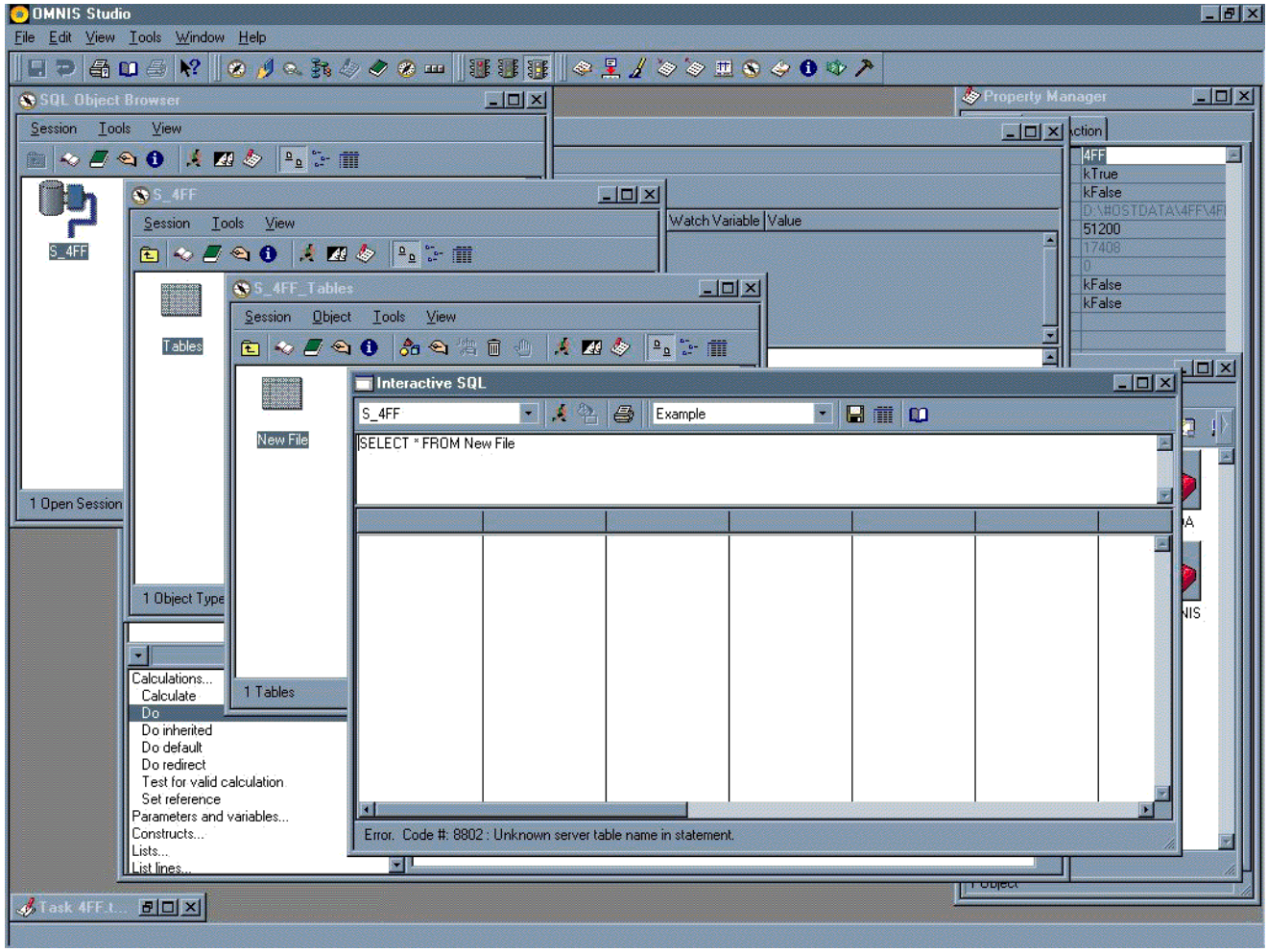

Abbildung 58 SQL Object Browser

Ok - das ist ja auch weiter kein Problem, denn als SQL-Kenner wissen sowohl Sie als auch ich, dass wir unsere Files erst auf dem Host bekanntmachen müssen. Aber es ist schön zu sehen, wie man dann später über das allbekannte und allseits beliebte SQL an sie herankommt!

Der Test verlief erfolgreich - weiter geht's im echten Code in der Dateitask **t_File**. Und wie geht es weiter? Die Namen sollten an die bisherige Konvention angepasst werden, zwischen den verschiedenen Syntax-Elementen einen Unterstrich einzufügen. Für den Namen der Session, die wir bisher fehlerhaft als **4fF** ausgewählt haben, schlage ich vor, die Art des SQL-Hosts zu verwenden, der dann in späteren exklusiveren Versionen unserer Applikation durch gespeicherte Daten für jede Datei auf einen anderen Host verweisen kann, wenn wir in der Datei-Task die entsprechenden Objekte mitaufnehmen. Diese Möglichkeit programmieren wir selbstverständlich noch nicht, doch wir bedenken sie schon durch Benutzung von Variablen. Der Wert dieser Variablen wird im Augenblick noch „hardcoded" vorgegeben, doch ich hoffe, dass Sie sich entsprechend schlecht fühlen: das sind schließlich festverdrahtete Programmelemente - und als solche immer verdächtig! Nur die Niedlichkeit der Applikation und ihr Zweck, zuerst der Demonstration von Studio zu dienen, erlauben diesen Faux-Pas.

Die Kommentare des Beispiels sind nicht für unsere Zwecke geeignet, also müssen wir sie ändern, sodass sie für Sie aussagekräftig werden, diese Kommentare haben schließlich eine dokumentierende Funktion. Wie oft ist es Ihnen nicht schon vorgekommen, dass Sie sich durch eine fremde Anwendung hindurcharbeiten mussten und

sich über fehlende oder noch schlimmer, sogar falsche Kommentare ärgerten? Ich benutze dazu auch gerne die Kommentarzeile des eigentlichen Befehls, wie steht's da mit Ihnen? Beim Austausch des Session-Namens, der den festverdrahteten String durch eine Variable ersetzt, denken Sie doch auch noch an Drag&Drop aus der Variablenübersicht? Ist zu praktisch, um es zu vergessen!

Bevor wir das Ganze jetzt gleich durchspielen, noch einen Hinweis auf die Variable des Typus **ROW**. Das sind Listen mit nur einer Zeile - und mit allen SQL-Fähigkeiten, die Omnis via **TABLES** kennt. Sie stellen die Repräsentanten der Datensätze dar, die vom SQL-Server eingelesen und Ihrer Applikation angeboten werden, um verändert und zurückgespeichert zu werden. Sowohl die **$select**- als auch die **$fetch**-Methode können über dieses Objekt sehr bequem angewendet werden.

Ooooops - ein Cursor? Sie fragen nach dem Cursor-Handling? Klar, Sie haben völlig recht! Eine SQL-Session wird auch in Studio mit einer Positionierungsmethode namens „Cursor" gehandhabt, ist sogar sehr praktisch als Wegweiser. Doch dann brauchen wir die Eröffnung der Session gar nicht mehr in der Datei-Task selbst - dort genügt es nun, einfach den richtigen Wegweiser zu benutzen!

Also verlagern wir die Eröffnung der Session in die **Startup_Task** - und müssen in der spezifischen Verarbeitung einer Datei, also der Instanz der Datei-Task, nur noch den Namen des Cursors kennen. Das spart uns die Methode **$SelectHost**, sehr schön! Je kleiner die Anwendung ist, umso weniger Fehler kann sie machen. Aus Bequemlichkeitsgründen schlage ich vor, Host-Name, Cursor-Name und Name der Instanzenvariable des Session-Objektes in der **Startup_Task** identisch zu wählen. Übrigens - wenn Sie genau wissen wollen, wo und was diese so oft zitierten „sys()"-Werte bedeuten, schauen Sie einfach in der hilfreichen Hilfe nach, dort werden alle angebotenen Systemwerte aufgelistet und genau erklärt. Sie sehen dann auch, dass „sys(10)" der genaue Pfad der Omnis-Anwendung ist, bei der wir unser Dateisystem untergebracht haben. Machen Sie sich ruhig eine Hilfe-**BOOKMARK**, die Systemwerte könnten Ihnen immer wieder ganz nützlich sein.

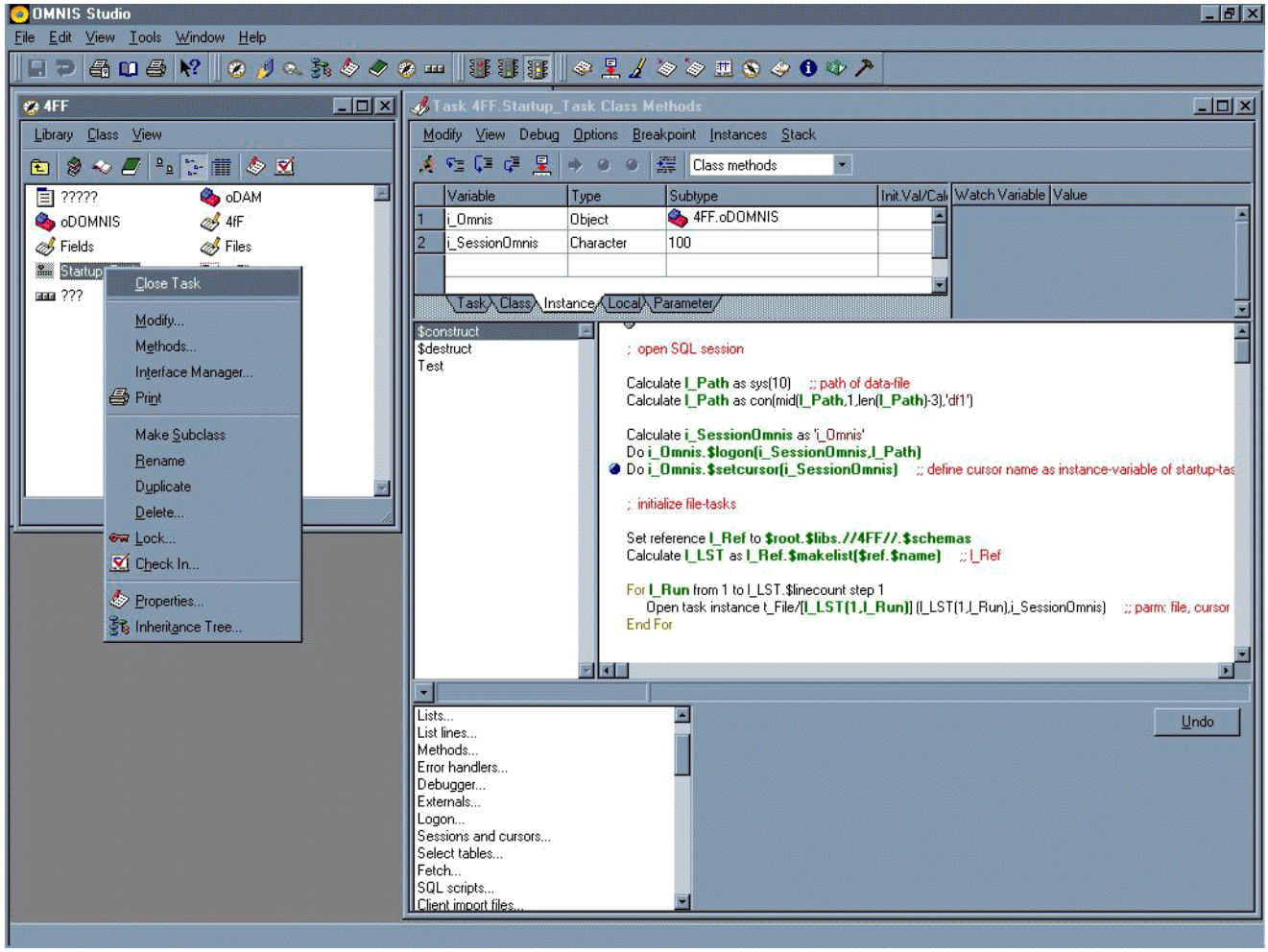

Abbildung 59 Eröffnen der SQL-Session

Um diese Befehle durchzuführen, genügt der einfache Stepper des Debuggers nicht - Sie können zwar immer noch zeilenweise durch die Methode der **Startup_Task** durchlaufen, Sie können auch über das Menü **INSTANCES** die richtige Instanz **4fF** aktivieren, um alle Instanzvariablen zu überprüfen - aber Sie schaffen es nicht einfach so, in die Dynamik einer Instanz integriert zu werden. Und genau das müssten wir ja nun, nicht wahr, wenn wir Befehle ausführen wollen? Doch der Prozess, den diese Instanz nun darstellt, ist bereits abgearbeitet - und der Faden nicht mehr aufzufinden. Eine Möglichkeit wäre, testweise einen neuen Prozess zu beginnen, der einen Teilbereich dieser Instanz verwendet in seinem eigenen Durchlauf, also von außen eine der Instanzmethoden aufzurufen, doch das ist an dieser Stelle nicht wirklich der Mühe wert! Schließen Sie also die ganze Task - am besten über den **BROWSER** und die Rechte-Maus-Funktionalität für das Task-Objekt an dieser Stelle, denn über diese Funktionalität können Sie ganz bequem die Task auch wieder starten. Vergessen Sie jedoch nicht, zuvor einen **BREAKPOINT** zu setzen! Ein einfacher **BREAKPOINT**, üblicherweise mit blauem Punkt vor dem betreffenden Statement erkenntlich, genügt vollauf - Sie setzen ihn am bequemsten mit der Buttonleiste der Methodendarstellungen der **Startup_Task**.

Omnis führt Sie dann bereitwillig durch die ganzen Objekte und erfüllt dabei noch alle Befehle. Die Session können Sie danach wieder mit dem **SQL OBJECT BROWSERS** ansehen, nachdem Sie mit dem Menü **STACK** Ihre Methode nach einem empörten Aufschrei von Studio wieder abgebrochen haben, das brav auch die folgende

Schleife durchlaufen wollte - denn die Instanzen für unsere Datei-Task sind natürlich längst aktiviert, oder haben Sie sie bereits wieder geschlossen?

Gehen wir also zurück zu unserer Methodendarstellung der **t_File**-Task. Löschen Sie dort bitte die Methode **$SelectHost** mit der Rechten Maus und vernichten Sie auch noch die restlichen Spuren - die Instanzenvariablen, die in dieser Methode definiert wurden. Wie? Wählen Sie einfach den Typ „Instance" der Variablenübersicht aus, klicken die Überschrift der Variablenliste an, also entweder „Variable" oder „Type" oder „Subtype", und benutzen Sie die Rechte Maus. An dieser Stelle wird Ihnen nämlich angeboten, überflüssige Variable zu entfernen. Und Sie möchten doch auch keinen unnützen Ballast mitführen? Das ist übrigens nicht nur Spielerei - Sie haben bestimmt auch noch die Instanzenvariable für das Session-Objekt in der Datei-Task, ich jedenfalls habe diese noch vorgefunden. Da ich gerne für gleiche Aufgaben gleiche Namen benutze, und Omnis alleine bei der Benutzung des Namens die Instanz eines Objektes aktiviert, hätte ich dann zwei Sessions zu demselben Host. Das ist zwar sicher an sich keine üble Sache, doch da ich diese Vorgehensweise nicht geplant habe, wäre das nichts weiter als ein Fehler in meiner Programmierung - die sind zwar nie zu vermeiden, man muss aber nicht vorsätzlich hineinstolpern.

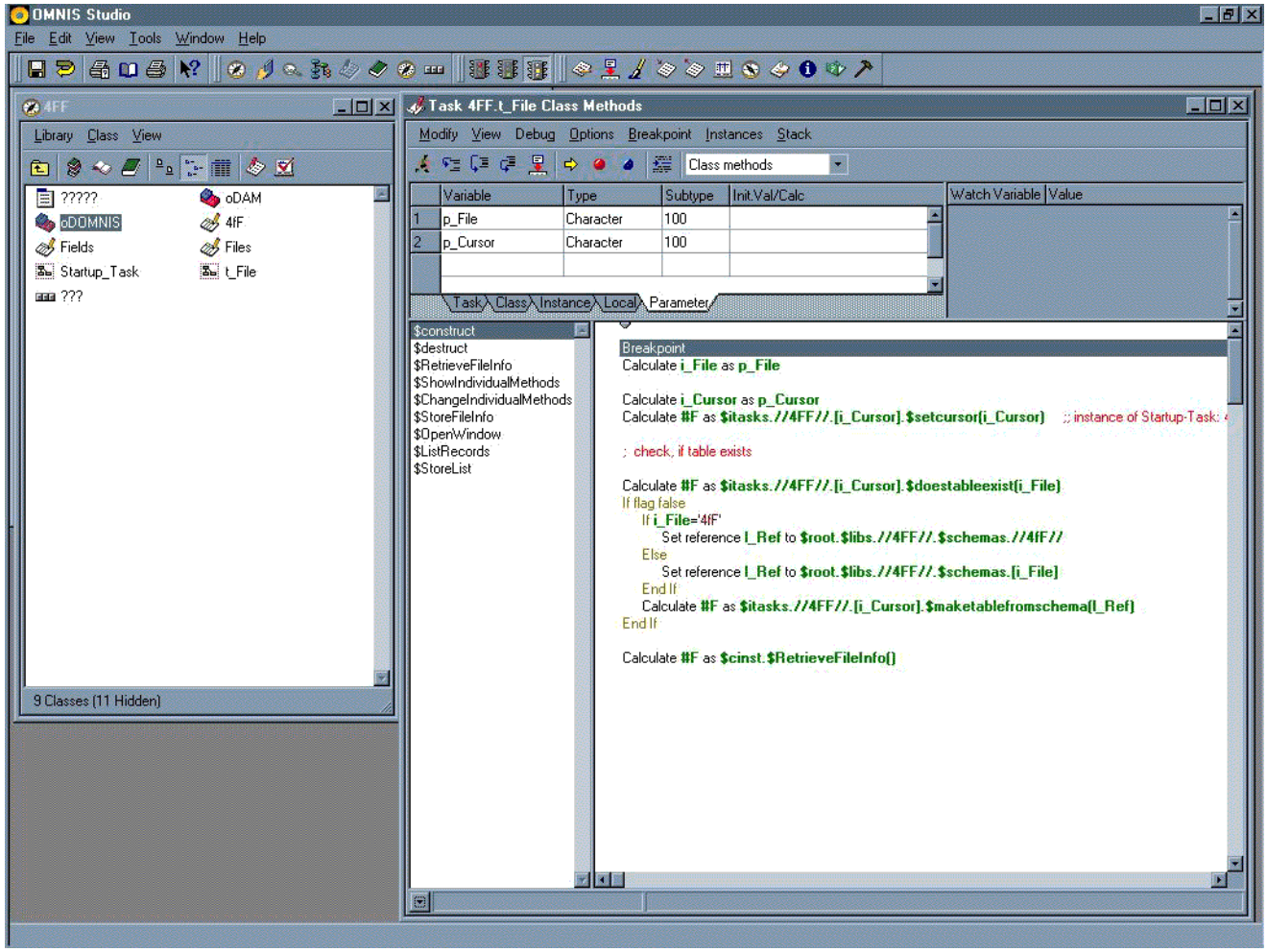

Abbildung 60 Erstellen der SQL-Dateien aus Schemas

Haben Sie eigentlich auch ein bisschen herumgespielt? Mit dem INTERFACE MANAGER auf das Objekt **oDOMNIS**, um dessen Angebot an Methoden zu überprüfen und sich die Prüfung **$doestableexist** und den

Befehl **$maketablefromschema** bequem per Drag&Drop auf die Befehlszeile zu holen, mit dem NOTATION INSPECTOR, um sich die ganzen Pfadangaben für die Referenzierung zu besorgen? Haben Sie auch ein bisschen hier und da durch die Befehle gehangelt, um so direkt beim Laufen zu sehen, wie sie arbeiten?

Dann wissen Sie sicher, warum es eine blöde Idee war, einen SCHEMA-Namen mit einer Zahl anfangen zu lassen! Jetzt muss jeder Zugriff über diese Fehlersyntax-Behandlung erfolgen - nun ja, denken Sie beim nächsten Mal dran: keine Objektnamen ohne alphanumerische Anfänge, auch wenn es Ihnen Studio natürlich erlaubt!

Ist Ihnen übrigens der Unterschied zwischen dem Zugang der Notation im NOTATION INSPECTOR für den Befehl **$maketablefromschema** und für die **Startup_Task 4FF** aufgefallen? **$maketablefromschema** greift auf die Klassenbeschreibung zu, erwartet von Ihnen eine Referenz auf die Klassenbeschreibung eines SCHEMAS, also müssen Sie unter dem Oberbegriff **$libs** im NOTATION INSPECTOR nachsehen, doch für die Benutzung des Session-Objekts ist die Instanz verantwortlich - deshalb müssen Sie Ihre Notation unter **$itasks**, also der Auflistung der aktuell realisierten Task-Instanzen, nachschlagen.

Auch in der Notation ist natürlich die Indirektion erlaubt!

Nun, haben Sie herumgespielt? Dann haben Sie sicher auch einige Instanzen offen und Omnis klopft Ihnen bei den Tasks ständig auf die Finger, dass diese Instanz bereits existiert ist, oder vervielfältigt Ihnen die Objektinstanzen wie Sandkuchen spielender Kinder an einem schönen Badestrand.

Da ist es wohl am besten, wenn Sie einfach die ganze Anwendung schließen und ganz von vorne beginnen? Bloß - haben wir ja noch richtige Baustellen vor uns und würden die doch zu gerne erst testen! Kein Problem - statt eines BREAKPOINTS aus der Buttonleiste plazieren Sie einfach den Befehl, dass Omnis hier auf Sie warten muss: **Breakpoint**. So können Sie sich selbst garantieren, dass Sie niemals eine Baustelle unfertig vergessen können! Ist doch praktisch - wer will schließlich unreife Anwendungen programmieren?

Zum Schließen der gesamten Applikation verwenden Sie einfach wieder den BROWSER - zum Öffnen über das Menü „Library“ des BROWSERS bietet Ihnen Omnis wie die meisten anderen Programme praktischerweisee Ihre letzten Auswahlen an, sodass Sie sie bequem wieder starten können.

Brav hält es die Verarbeitung dann auch am befohlenen Ort an - und Sie können steppenderweise jeden Atemzug sehen, den Studio für Sie durchführt. Das Ergebnis? Gehen Sie in den SQL OBJECT BROWSER - jetzt finden Sie alle Dateien und sehen auch die Feldauflistung per SQL. Einfach so!

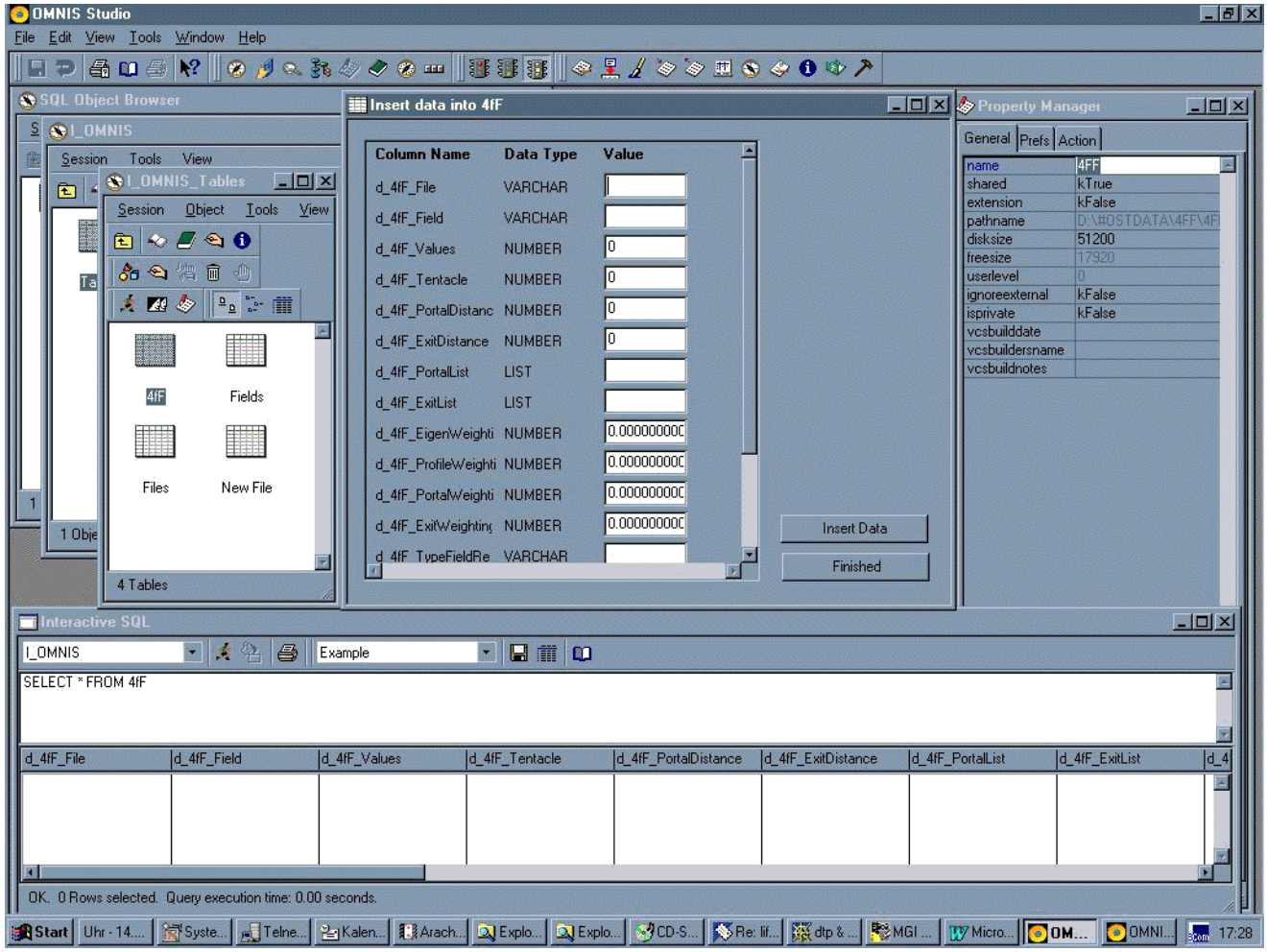

Abbildung 61 Erzeugte Dateien auf dem Host, interactive SQL-session und „insert data"-Auswahl über die Rechte-Maus-Funktionalität auf die Tables der SQL Object Browser-Darstellung

Also - das wäre erledigt! Am Host sind wir angeloggt, die Dateien existieren dort nun auch - beenden wir also rasch noch diesen ersten Schritt unserer Anwendung und ergänzen noch die Methode **$RetrieveFileInfo**, bevor wir an die nächste schöne Eigenschaft von Studio herangehen.

Zwei Dinge interessieren uns für die Datei-Task: der Name des Windows, das zur Datei gehört und die Methoden. Klar, wenn wir den Dateisatz einlesen, wozu eine Variable des Typs **ROW** ideal ist, können wir natürlich direkt auf diese beiden Felder der **ROW** zugreifen, doch aus Übersichtlichkeitsgründen empfehle ich, zwei Instanzvariablen mit den Werten zu bestücken: **i_Window** und **i_Methods**, letztere wiederum als Binärvariable und nicht als Objekt definiert. Warum ich das vorschlage? Erinnern Sie sich noch an die Vielfalt von erzeugten Objektinstanzen, die sogar bei einfacher Namensnennung erfolgte? Ein Binärfeld ist wie eine eingefrorene Instanz, wenn Sie ein Objekt darin speichern - es instanziert sich nicht erneut und es benimmt sich dennoch vollwertig.

Die Methode ist klein und übersichtlich - zuerst brauchen wir ein Objekt des Typus **ROW**, das uns die benötigten Zugriffswege zu unserer Datei **Files** verschafft, die wir hier unanständigerweise festverdrahtet in der Parameterübersicht vorgegeben haben. Dieses Objekt muss jedoch noch in Kenntnis gesetzt werden, welche Datei es zu betreuen hat - dies erfolgt durch das statement **$definefromsqlclass**.

Übrigens - um sie auszutesten, müssen Sie natürlich entweder alle Instanzen schließen - oder an die kleine Test-Routine in der **Startup_Task** denken, in der Sie dann einfach den Befehl **Calculate #F as $itasks.//4fF// .$RetrieveFileInfo** einsetzen und per Stepper ausführen müssen. Damit starten Sie einen neuen Prozess, der sich in die bestehende (eigene) Instanz einklinken und ihre Methoden benutzen kann.

#F ist übrigens ein generelles Flag für Omnis, wird in vielen Befehlen automatisch gesetzt und kann nur die Werte richtig und falsch bzw. 0 und 1 bzw. kTrue und kFalse enthalten.

Warum der Befehl, den Cursor zu setzen, in dieser Methode wiederholt wird? Nun, diese wenigen Befehle nicht direkt in die **$construct**-Prozedur unserer Datei-Task zu integrieren, hatte nur den einen Grund: es wird nötig sein, diese Verarbeitung auch noch dann zu verlangen und durchzuführen, wenn die Instanz schon längst aktiviert worden ist. Dann aber kann zwischen Instanzierung und Ihrem Aufruf folgendes geschehen sein: Sie benutzten den SQL-Browser oder eine andere Task musste auf einen anderen Host über ein anderes Session-Objekt zugreifen - dann sitzt Ihr „Wegweiser“ Cursor nicht mehr korrekt. Wenn aber der Wegweiser nicht mehr korrekt sitzt, laufen die SQL-Befehle in die Irre - um das zu vermeiden, setzen wir eben einfach unseren Cursor wieder, dazu hatten wir die Instanzenvariable **i_Cursor** ja gedacht.

Zurück zu unserer Programmierung. Als nächstes müssen wir die Suche für unsere eigene Datei, deren Task-Instanz wir gerade verarbeiten, vorbereiten - da Omnis uns viel Arbeit abnimmt, brauchen wir nur die Bedingung, die **WHERE**-Klausel, zu formulieren und stoßen dabei auf ein weiteres Syntax-Problem, Hochkommas. Denn Hochkomma zeigen Omnis an, dass nun Strings, also Buchstabenreihen, folgen sollen und kein Befehl oder Variable. Das ist normalerweise eine feine Sache - doch leider gehört mitten in unseren SQL-Befehl ja ebenfalls ein Hochkomma, da Werte für nichtnumerische Feld in SQL nun mal so auszusehen haben. Doch auch hier bietet Omnis Abhilfe - benutzen Sie einfach chr(39) bei der Stringverknüpfung **con()** - das ist im ASCII-Code nämlich genau das Hochkomma.

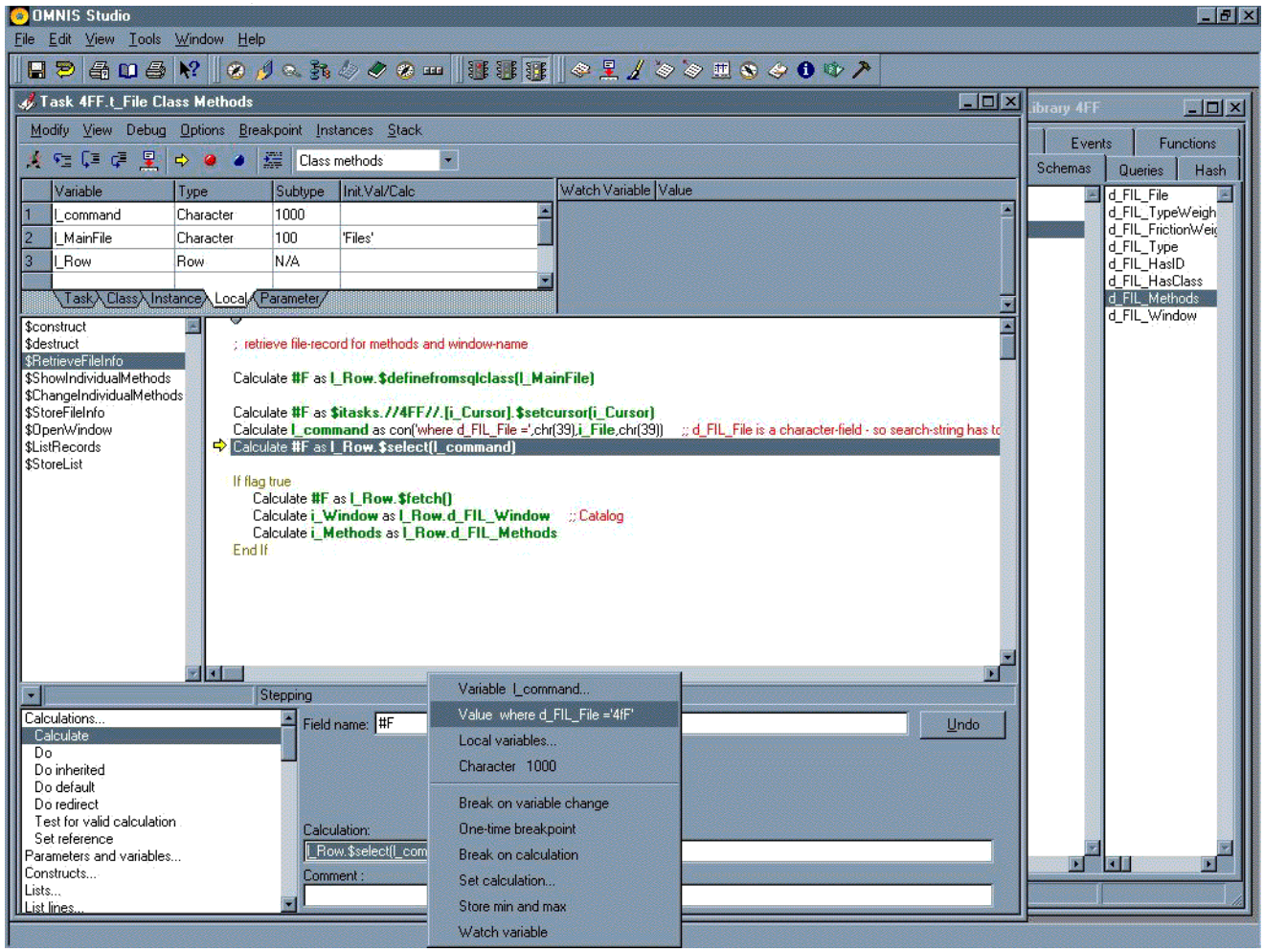

Abbildung 62 Zugriff auf einen Datensatz, Stringverarbeitung

Nach der erfolgreichen Suche müssen wir uns dann die Daten noch vom Host abholen. Wenn Sie diese Befehle mit Erfolg testen wollen, so gehen Sie einfach in den SQL Browser und wählen Sie für die Datei **Files** über die Rechte Maus die „insert data“-Option aus, um ein paar Sätze zum Auffinden zu erzeugen. In das erste Datenfeld geben Sie den Namen der Datei, also zum Beispiel **4fF** ein, dann geben Sie noch einen Testwert in das Feld für den Dialogfenster-Namen ein - und bestätigen es Omnis mit diesem Button „insert data“. Sie erhalten dann beim testweisen Durchführen unserer hübschen, kleinen Methode auch einen vernünftigen Wert für die Instanzenvariable **i_Windows**. Den Inhalt des gesamten Datensatzes können Sie sich über die lokale Variable **l_Row** ansehen - mit der Rechten Maus Funktion, wie gehabt auf den Variablen-Namen des auftauchenden Popup-Menüs klikken, dann öffnet Omnis Ihnen die ganze Feldanzeige.

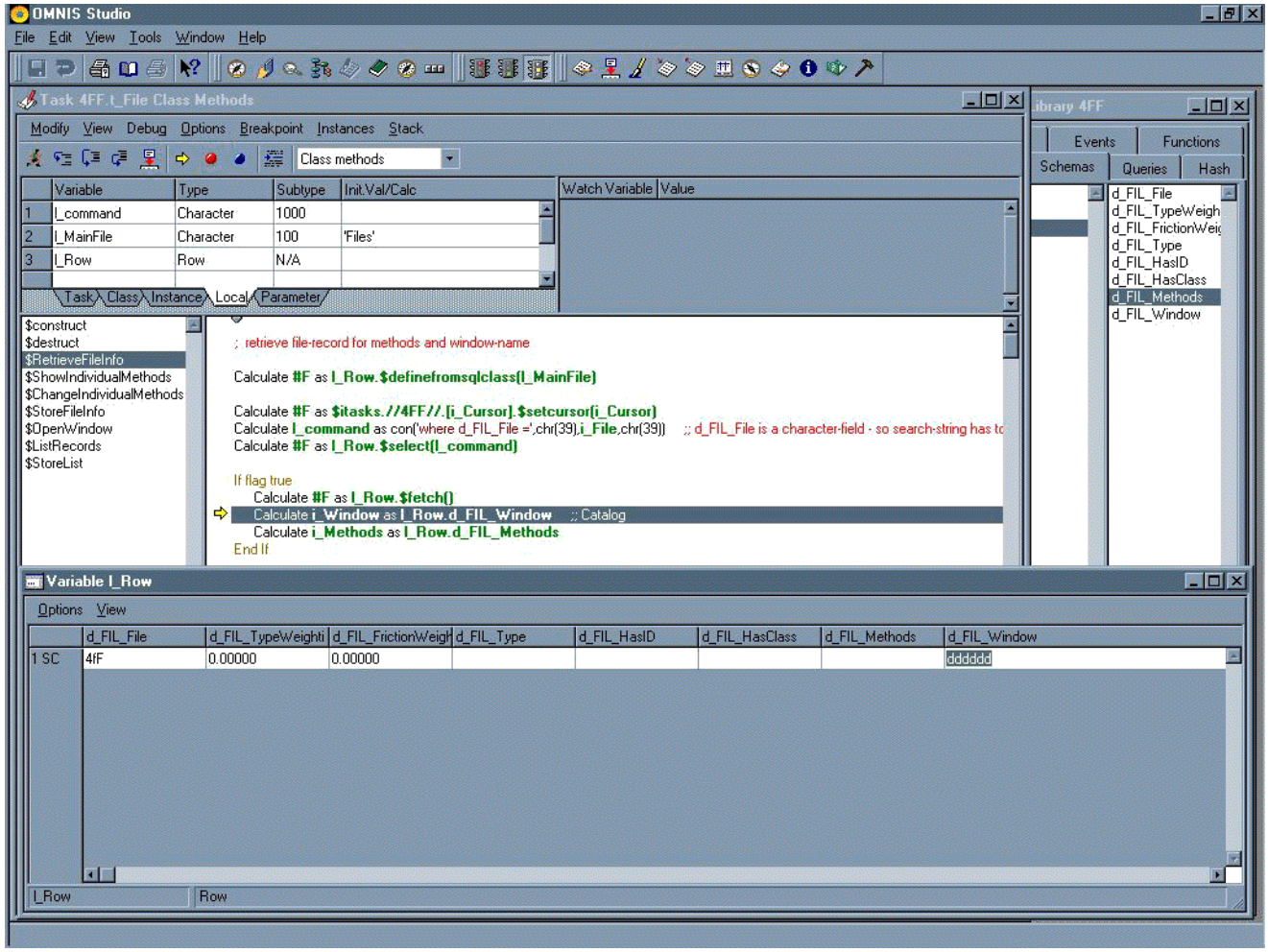

Abbildung 63 Zugriff auf einen Datensatz, Stringverarbeitung

Übrigens - die ganzen Dateifeld-Namen der SCHEMAS lassen sich bequem über den CATALOG (via Menü, Button oder Funktionstaste „F9“) auswählen und mit Drag&Drop auf die Befehlszeile ziehen, warum sich also Schreibarbeiten machen?

2.5 Methoden in Datenfelder

Das nächste Problem, das unsere Methoden in der Datei-Task anfordern, sind die individuellen Methoden pro Datei. In üblichen Applikationen werden dafür Programme oder Programmbestandteile benötigt, in Studio lassen sich diese Vorgänge jedoch genau dahin schaffen, wo sie hingehören - zu den Detailangaben über die Dateien selbst, denn die Individualität liegt nur in den Daten.

Sicher sparen wir natürlich nicht viel Programmobjekte bei unseren drei Dateien - denn ein Template-Programmobjekt benötigen wir sogar in Studio. Doch da die meisten Applikationen weitaus mehr Dateien versorgen als nur drei, können Sie vielleicht aus Ihrer allgemeinen EDV-Erfahrung abschätzen, wieviel kompakter eine solche Anwendung werden kann, die die individuellen Methoden aus ihrer Programmierung heraushält, ohne sie jedoch aufgeben zu müssen. Außerdem können Sie mit einer solchen Methode individuelle Programmierung per Datenimport in Ihre Anwendungen aufnehmen, ist in vielem praktischer als Installationen der üblichen Programmobjekte, egal wie bequem diese Installationen auch sein mögen.