ist und die Anwender denoch dezidiert ein anderes zur Verarbeitung auswählen? Wir lassen sie es natürlich tun! Sie werden schon wissen, was sie wollen, sie sehen schließlich alles deutlich vor sich. Um diese Verhaltensweise unseres Programmes zu erreichen, müssen wir nichts anderes tun, als die Ergebnisliste verschwinden lassen, denn dann liest die Methode **$InitPage2** brav einfach den aktuellen Wert ein und gut. Wird die Liste **i_LST_Show** dagegen nur sachte angetippt zur Umsortierung, ohne gezielt einen neuen Datensatz auszuwählen, geschieht nichts weiter, der mitten in der Verarbeitung stehende Satz bleibt also weiterhin verwendbar.

Unsere Feldmethode **i_LST_Show** muss deshalb noch ein wenig geändert werden:

w_4fF,
i_LST_Show

```
On evClick

    Calculate #F as $cinst.$objs.i_LST_Result.$visible.$assign(kFalse)

    Calculate #F as $cinst.$objs.Pane.$currenttab.$assign(2)
```

Dieser Fall ist damit vom Tisch.

3.4 Reaktionsmöglichkeiten auf äußere Einflüsse

Der zweite, ähnlich gelagerte Fall war eine Aktion unserer Anwender außerhalb unseres Fensters. Um solche Fälle abfangen zu können, muss die Standard-Methode **$events** für das gesamte Window aktiviert werden. Sie ist dazu da, solche feldübergreifenden Aktionen wie das Schließen des Fensters zu überprüfen oder auf Vergrößerungen oder Minimierungen zu reagieren. Oder eben darauf, dass das Fenster wieder in den Vordergrund gerückt wird. Im Moment genügt jedoch der Event **ToTop** vollständig.

Diese neue Methode **$event** besteht also augenblicklich nur aus folgenden Befehlen:

w_4fF,
$event

```
On evToTop

    Do $cinst.$ToTop()
```

Da der **ToTop**-Event auch beim erstmaligen Öffnen des Fensters gesendet wird, kann der Aufruf der **$ToTop** sogar aus der **$construct** entfernt werden. Was geschieht nun aber bei dem „Neuaktivieren“ unseres Fensters? Bisher wurde einfach geprüft, ob eine korrekte Veränderung durchgeführt wurde mithilfe der Variablen **i_Changed**, dann wurde unsere aktuelle Datei noch gegen den Background-Wert des Dateinamens im Navigatorwindow verglichen und gegebenenfalls darauf reagiert. Dieses Verhalten passt aber nicht ganz zu dem, was wir gerade eben beim Wechsel der Seite im eigenen Fenster getan haben. Dort haben wir einen fliegenden Wechsel nur zugelassen, wenn entweder keine Verarbeitung offenstand oder eben, wenn er ausdrücklich gefordert wurde.

Sollten unsere Anwender also außerhalb unseres Fensters gearbeitet haben, obwohl sie hier bei uns noch offene Fragen haben, dann sollten wir ganz ähnlich vorgehen und ihre aktuelle Arbeit im Vordergrund lassen. Das bedeutet erst einmal, dass wir über **i_Changed** prüfen, ob Werte korrekt verändert wurden und in diesem Fall alles beim Alten lassen, nicht wie bisher unsere Benutzer auffordern, die Änderungen zu bestätigen oder zu verwerfen. Den zweiten Fall laufender Verarbeitung haben wir aber gerade eben erst programmiert, erkennbar an der sichtbaren Suchliste für Eigenschaften. Klar, wir könnten auch hier alles beim Alten lassen. Doch was, wenn sich unsere Anwender die Mühe machten, ihre Eigenschaften auf anderem Weg zu suchen? Sie müssen ja nichts weiter tun, als es dann dem Navigator-Fenster mitteilen, an dessen Vorgaben möchten wir uns ja so gut als möglich ausrichten. Also werden wir folgendes tun: wenn das Navigator-Fenster eine Eigenschaft aufzeigt, die nicht mit unserer eigenen alten übereinstimmt und wir haben gleichzeitig diesen Schwebezustand der sichtbaren Suchliste, dann werden wir den Navigator-Wert übernehmen, ansonsten tun wir nichts. Wir haben nur das kleine Problem, dass unsere Suchliste auch die Suche von Dateien erlauben sollte, erinnern Sie sich? In diesem

Fall werden die Anwender wohl etwas dagegen haben, wenn wir wild die Eigenschaft überschreiben - doch wenn sie gerade dabei sind, Eigenschaften zu suchen, mag es ihnen sogar als Hilfe erscheinen, wenn wir bereitwillig ihre ferne Auswahl akzeptieren. Um diese beiden Fälle zu unterscheiden, lassen wir uns einfach den Namen des hinter der Suchliste stehenden SCHEMAS sagen, schließlich stammt sie aus der Datei-Methode **$ListRecords**. Oooops, das geht ja nicht, wir überschrieben schließen wegen der Anzeige die erste Spalte mit **i_Text**, damit akzeptiert Omnis die Listendefinition leider nicht mehr als von einem SCHEMA stammend und gibt keinen vernünftigen Wert mehr in **$sqlclassname** zurück. Aber die Spaltenanzahl beider in Frage kommenden Dateien unterscheidet sich deutlich, das heißt wir können über diesen Wert ausreichend genau differenzieren. Information liegt in der Unterscheidbarkeit und manchmal genügt auch eine eigentlich uninteressante Angabe, um besser vorwärtszukommen - das ist so ungefähr die umgangssprachliche Lehre des Bertrand'schen Schachtelparadoxons. Es ist das Paradox der Information, schließlich zeigt jedes Paradox in der Mathematik einen Mangel auf, der im alten Trott kaum zu beseitigen ist.

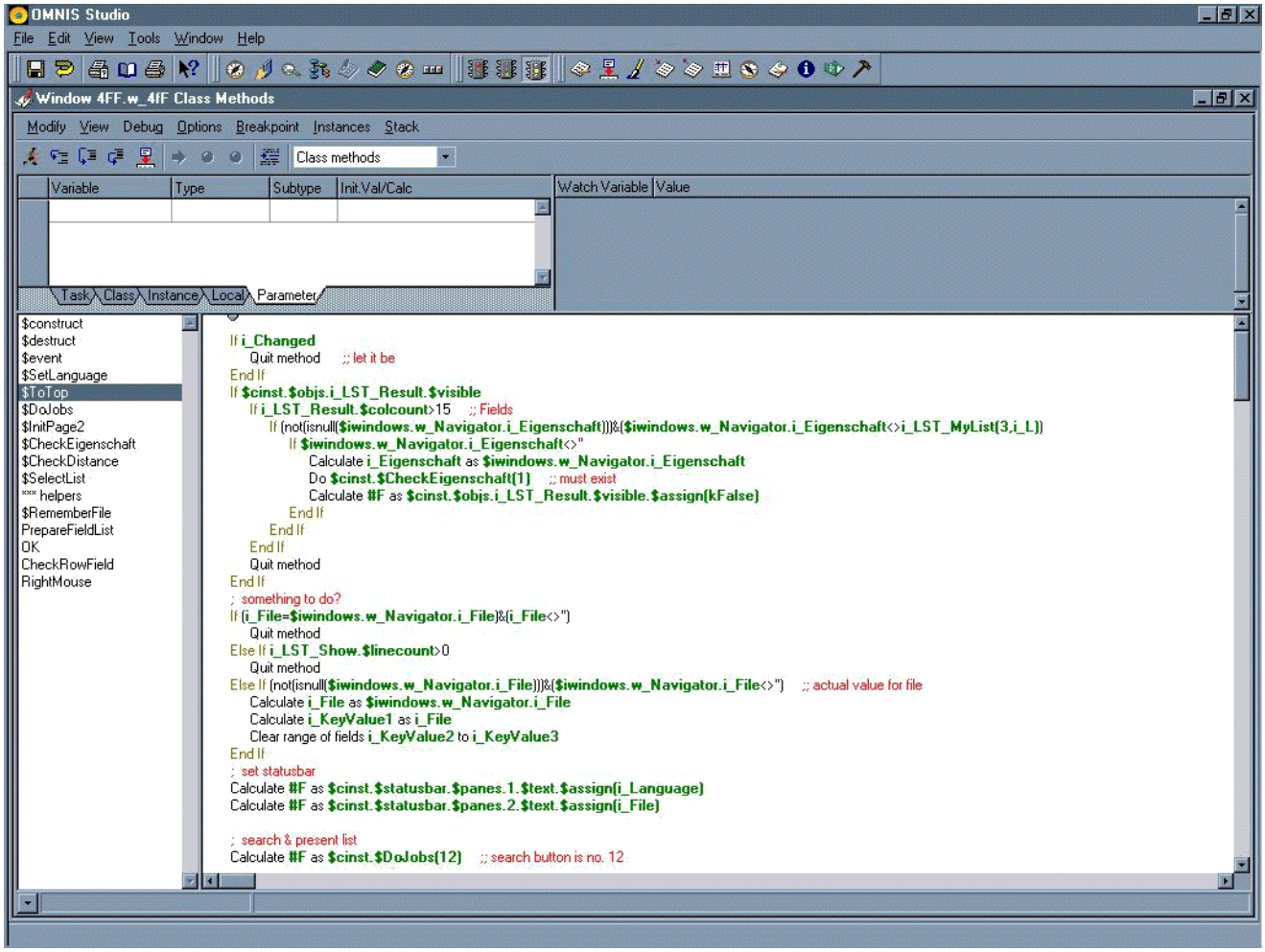

Abbildung 163 Angepaßte Methode $ToTop

Die Abfrage, ob die Navigator-Eigenschaft denn tatsächlich mit einem Wert versehen ist

w_4fF, $ToTop

```
If $iwindows.w_Navigator.i_Eigenschaft<>''
```

kann natürlich mit einer Und-Verknüpfung **&** an die direkt darüberstehende Abfrage angekoppelt werden, doch

in diesem Falle wäre die ganze Konstruktion zu lange für die Abbildung geworden, man hätte sie nicht mehr sehen können.

Nun kommen wir also an den letzten Fall - die Auswahl eines bestimmten Wertes von der Suchliste durch unsere Anwender. Diese Auswahl wird bei der Feldmethode mit dem Event **evClick** ermöglicht, dafür muss das Feldobjekt **i_LST_Result** jedoch mit „enabled“ = **kTrue** bestimmt sein, entgegen meiner bisherigen Vorgabe. Die verschiedenen Listenarten reagieren eben auch verschieden und alle Varianten kann ich mir kaum merken.

Wählen unsere Benutzer eine der angebotenen Eigenschaften aus, ist eigentlich alles geritzt. Wir müssen nur die daraus abgeleiteten Werte mit Hilfe der individuellen Datei-Methoden ermitteln lassen, dann sind wir fertig. Doch bei einer neuen Eigenschaft haben wir mehr zu tun, bevor wir die abgeleiteten Größen errechnen können. Nicht nur, dass wir uns merken müssen, wenn wir eine neue Eigenschaft vor uns haben. Sie muss nämlich die im Ernstfall, wenn die Arbeit also wirklich in die Datenbank gerettet werden soll, dann auch in der **Fields**-Datei angelegt werden. Wir müssen darüberhinaus auch die Werte dieser neuen Eigenschaft erst einmal halbwegs korrekt festlegen. Wir brauchen also irgendwo eine Stelle, die uns die allerersten Angaben zur Verfügung stellt - hatten wir schon einmal, erinnern Sie sich? In den individuellen Methoden der **Fields**-Datei haben wir eine Prozedur **$NewRecord** dafür vorgesehen.

Sie ist bisher jedoch auf den Fall ausgelegt, dass bereits ein Datenbanksatz mit der geforderten Eigenschaft vorhanden ist. Mit einem neuen Parameter, der unseren aktuellen **4fF**-Datensatz enthält, können wir diese Methode mit ihren Prüfungen für uns verwenden. Direkt hinter die bisherige Ermittlung aller **4fF**-Datensätze mit dieser Eigenschaft

Individuelle Methoden, $NewRecord

```
Calculate l_LST as l_Ref_4fF.$ListRecords(l_Eigenschaft)
```

fügen wir nun den neuen, zweiten Parameter **p_Row_4fF** vom Typ **Row** ein.

```
If p_Row_4fF.$linecount=1

    Calculate #F as l_LST.$merge(p_Row_4fF)

End If
```

Der Rest funktioniert wie gehabt und kann für unsere Zwecke verwendet werden, wenn wir mit **$StoreFileInfo** unsere Arbeit auch sichern.

Die Feldmethode der **i_LST_Result** ist jedoch später für noch mehr verantwortlich als nur für Eigenschaften, deshalb tut sie nichts weiter, als beim Empfang eines Mausklicks eine entsprechende Klassenmethode aufzurufen, die den umständlichen Teil dann erledigt:

w_4fF, i_LST_Result

```
On evClick    ;; Event Parameters - pRow ( Itemreference )

    Do $cinst.$SelectList
```

Diese Methode soll zuerst also die beiden Fälle unterscheiden, die sie zu bearbeiten hat. Das tut sie ganz genauso wie die **$ToTop**, wählt anhand der Spaltenzahl aus. Als nächstes muss sie die drei verschiedenen Auswahlmöglichkeiten berücksichtigen: 1) die regulären Eigenschaften, wobei der Sonderfall auftreten kann, dass die Anwender exakt dieselbe Eigenschaft wählen, die sie bereits verwenden, 2) die versehentliche Auswahl der Leerzeile oder gar keine Auswahl und 3) die Auswahl der Zeile mit dem Befehl, eine neue Eigenschaft zu berücksichtigen.

Die erste Alternative ist einfach. Wir nehmen die Eigenschaft und lassen die abhängigen Werte über die **$Check-Eigenschaft** ermitteln, falls es nicht die aktuelle Eigenschaft ist. Im Falle fehlerhafter Auswahlen wie der Leerzeile oder einem Klick auf die Liste ohne Auswahl einer Zeile setzen wir die Eigenschaft einfach auf den alten Wert

zurück und bei einer neuen Eigenschaft holen wir uns die **Fields**-Daten über die individuellen Methoden der **Fields**-Prozedur **$NewRecord**. Die dafür neu eingeführte Instanzvariable **i_NewEigenschaft** ist vom Typ **BOOLEAN** mit dem Anfangswert **kFalse**. Da wir sie hier setzen, müssen wir auch daran denken, dass wir sie wieder zurücksetzen. Dies könnte bei der Verarbeitung des Abbruches in der **$DoJobs**, **Case**-Zweig 42, erfolgen, doch noch allgemeiner lässt es sich in der Initialisierungsroutine für unsere Verarbeitungsseite **$InitPage2** unterbringen, natürlich nach der ersten Abfrage, ob überhaupt etwas getan werden soll - also nach Abfrage auf die Sichtbarkeit der Suchliste **i_LST_Result**. Dort fügen wir jetzt auch das Zurücksetzen der Kennzeichnung ein, ob wir eine korrekte Datensatzänderung durchgeführt haben. Damit sieht der vollständige Anfang der **$InitPage2** nun wie folgt aus:

w_4fF,
$InitPage2

```
If $cinst.$objs.i_LST_Result.$visible
    Quit method
End If
Calculate i_NewEigenschaft as kFalse
Calculate i_Changed as kFalse
```

Der neue Datensatz darf jedoch nicht auf Existenz der Eigenschaft geprüft werden, deshalb wird er mit dem Parameter „1“ für die **$CheckEigenschaft** versorgt.

Die individuelle Methode **$DoEigenschaft** ergänzte ich übrigens noch um:

individuelle 4fF,
$DoEigenschaft

```
Calculate p_Show(3,p_L) as i_InstanceEigen
```

Damit ist auch die Anzeige vollständig auf die neue Eigenschaftswahl umgestellt.

Die neue Methode **$SelectList** sieht dann so aus:

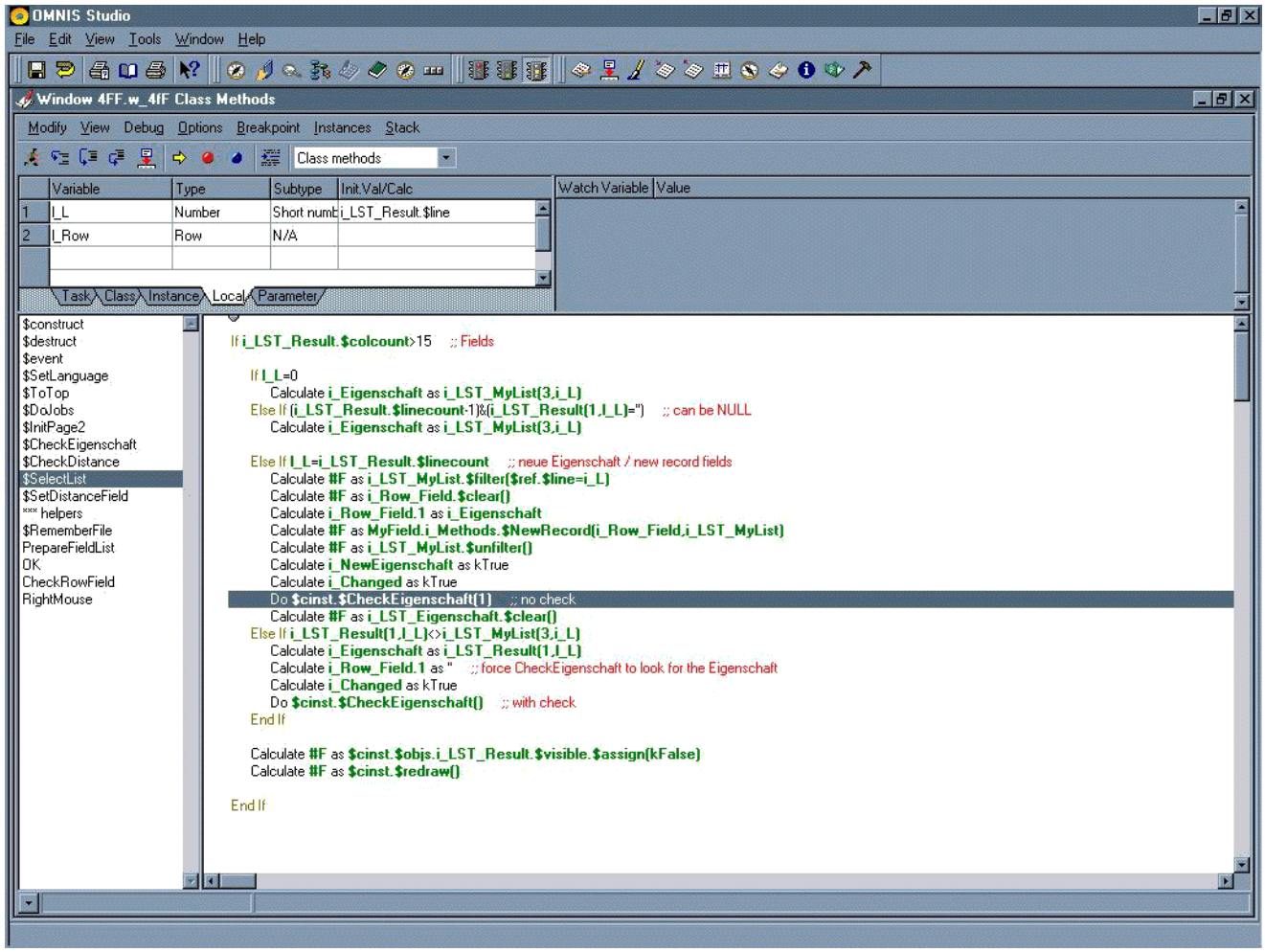

Abbildung 164 Listenverarbeitung $SelectList

Beim Testen mit den neuen Eigenschaften hat mich darüberhinaus die Leerzeile über der einzigen Zeile „neue Eigenschaft“ genervt, ich entfernte sie deshalb durch Ergänzung der Abfrage in der Methode **CheckRowField**:

w_4fF, CheckRowField

```
If (i_LST_Result.$linecount<19)&(i_LST_Result.$linecount<>0)
```

Damit sollte dieses Eingabefeld **i_Eigenschaft** abgehandelt sein. Das nächste zu bearbeitende Feld ist nun **i_Value**, das die Anzahl möglicher Werte des betrachteten Datenbankfeldes angibt. Für dieses Feld gibt es keine Prüfungen, außer dass es nicht Null werden darf und ganzzahlig sein muss, es hat aber sehr wohl abhängige Größen, ganz wie die Eigenschaft. Weil es sogar diesselben Größen sind, kann ich deshalb die **$DoEigenschaft** verwenden, wenn ich daran denke, dass diese Prozedur sich ihren aktuellen Values-Wert aus der übergebenen Datei-Liste **p_Result(5,p_L)** holt.

Die Feldmethode von **i_Value** wird dann bequemerweise von **i_Eigenschaft** kopiert, jedoch ohne die Behandlung der Rechten Maus.

w_4fF, i_Value

```
On evAfter    ;; Event Parameters - pClickedField, pClikkedWindow, pMenuLine, pCommandNumber, pRow

    If #MODIFIED
```

```
Do $cinst.$DoJobs(,l_Button)    ;; 2. parm
End If
```

Wegen der **Tooltip**-Steuerung müssen wir auch an nichts weiter denken, außer dass die Methode **$DoJobs** einen neuen **Case**-Zweig benötigt, bei mir mit dem Wert 18.

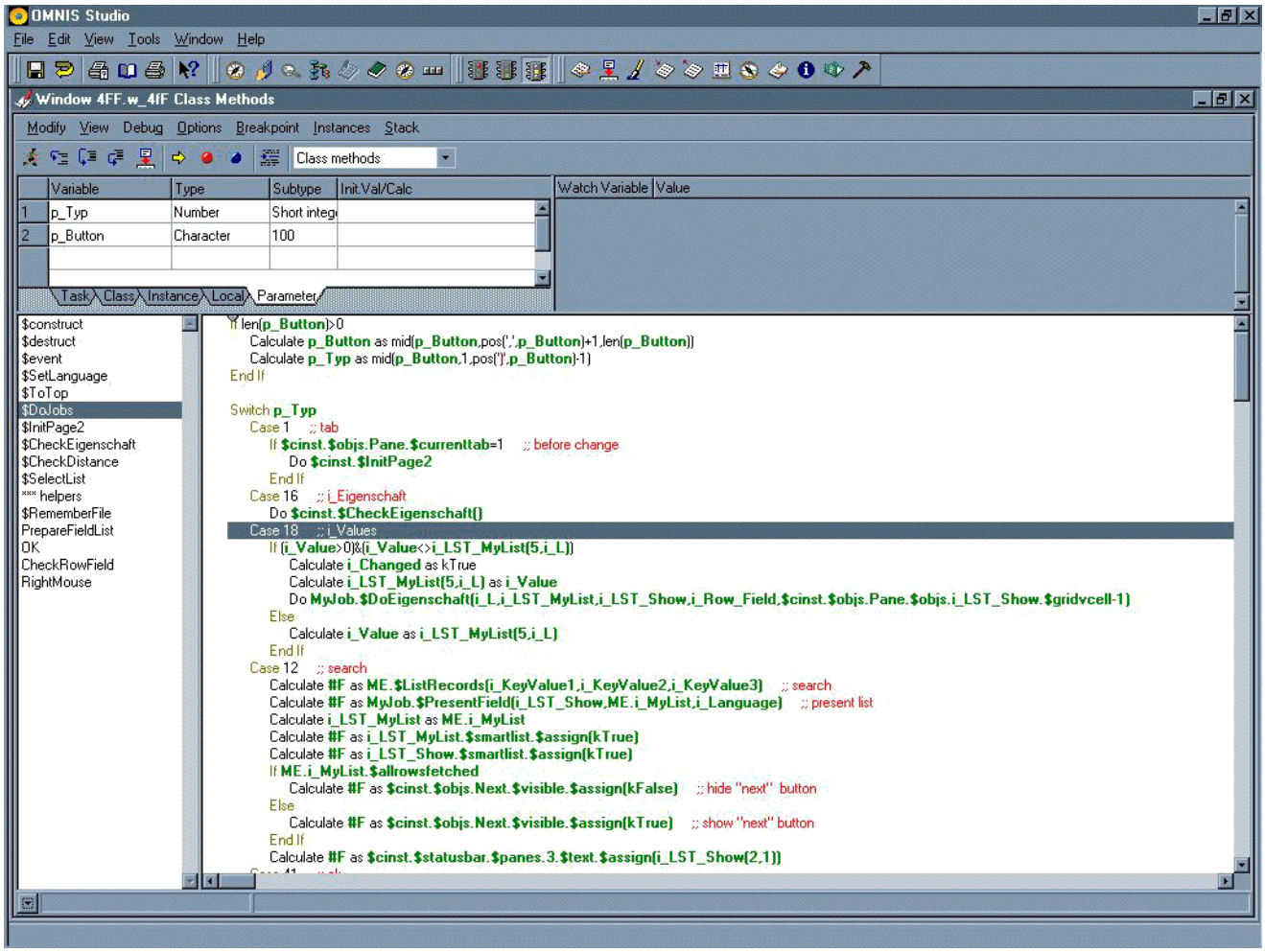

Abbildung 165 Neuer Case-Zweig für die Verarbeitung von i_Values in $DoJobs

Weiter im Text. Die nächste Eingabemöglichkeit ist die Angabe, ob unser Feld redundant ist, was uns dadurch gesagt wird, dass wir den Namen des Dateifeldes erfahren, das den Identifikator des Komponentenobjektes beinhaltet. Oder anders ausgedrückt: redundante Felder sind solche, die eigentlich nicht nötig sind, weil sie Attribute von Objekten beschreiben, die in einer eigens für diese Objekte vorgesehenen Datei schon längst niedergeschrieben sind. Einen Verweis auf dieses Objekt muss es alleine schon deshalb in unserer aktuellen Datei geben, weil unser Feld sonst nicht redundant wäre - so ist die Definition von Redundanz. Für unsere Zwecke hätte natürlich auch einfach ein „boole'sches" Feld mit Ja/Nein-Werten genügt. Doch wir sind neugierig und wollten selbstverständlich wissen, welches andere Feld unser aktuell bearbeitetes Feld denn so abwertet, dass es nur aus Bequemlichkeit oder Transzendenz überhaupt in der hiesigen Datei mitgeschleift wird.

Da diese Kenntnis jedoch nicht von zentraler Bedeutung für uns ist, lassen wir den Fall von gekoppelten Schlüsselfelder für unser Komponentenobjekt unberücksichtigt. Sollten Sie eine Datei haben, die solche kombinierten

Schlüssel verwendet, so suchen Sie sich den aussagekräftigsten aus - falls es wichtig genug für Sie ist, ergänzen Sie doch die Applikation! Sie haben sicher schon festgestellt, wie ungeheuer einfach Omnis Ihnen alles macht.

Für die Angaben, die wir für das Feld „Redundanz“ entgegennehmen können, ist das Einzige, was wir überprüfen können, ob das ausgesuchte Feld selbst redundant ist - dann ist die Auswahl unserer Anwender wohl nicht ganz so glücklich gewesen. Was dann tun? Den armen Benutzern auf die Finger schlagen mit einer Fehlermeldung „Du Idiot, was tust du denn da!!“ oder einfach den Identifikator übernehmen, der in dem ausgewählten, leider redundanten Feld als eigener Leithund bestimmt ist? Wir tun Letzteres, schließlich ist der Hauptzweck hier nur zu erfahren, ob das Feld redundant ist oder nicht.

Für diese **POPUP**-Liste habe ich bereits einen **Tooltip** bestimmt, ihre Feldmethode **$event** sieht also wie gehabt aus:

w_4fF,
i_LST_Redundance

```
On evClick    ;; Event Parameters - pRow ( Itemreference )

    Do $cinst.$DoJobs(,l_Button)    ;; 2. parm
```

Der neue **Case**-Zweig in der **$DoJobs** hat die Nummer 20.

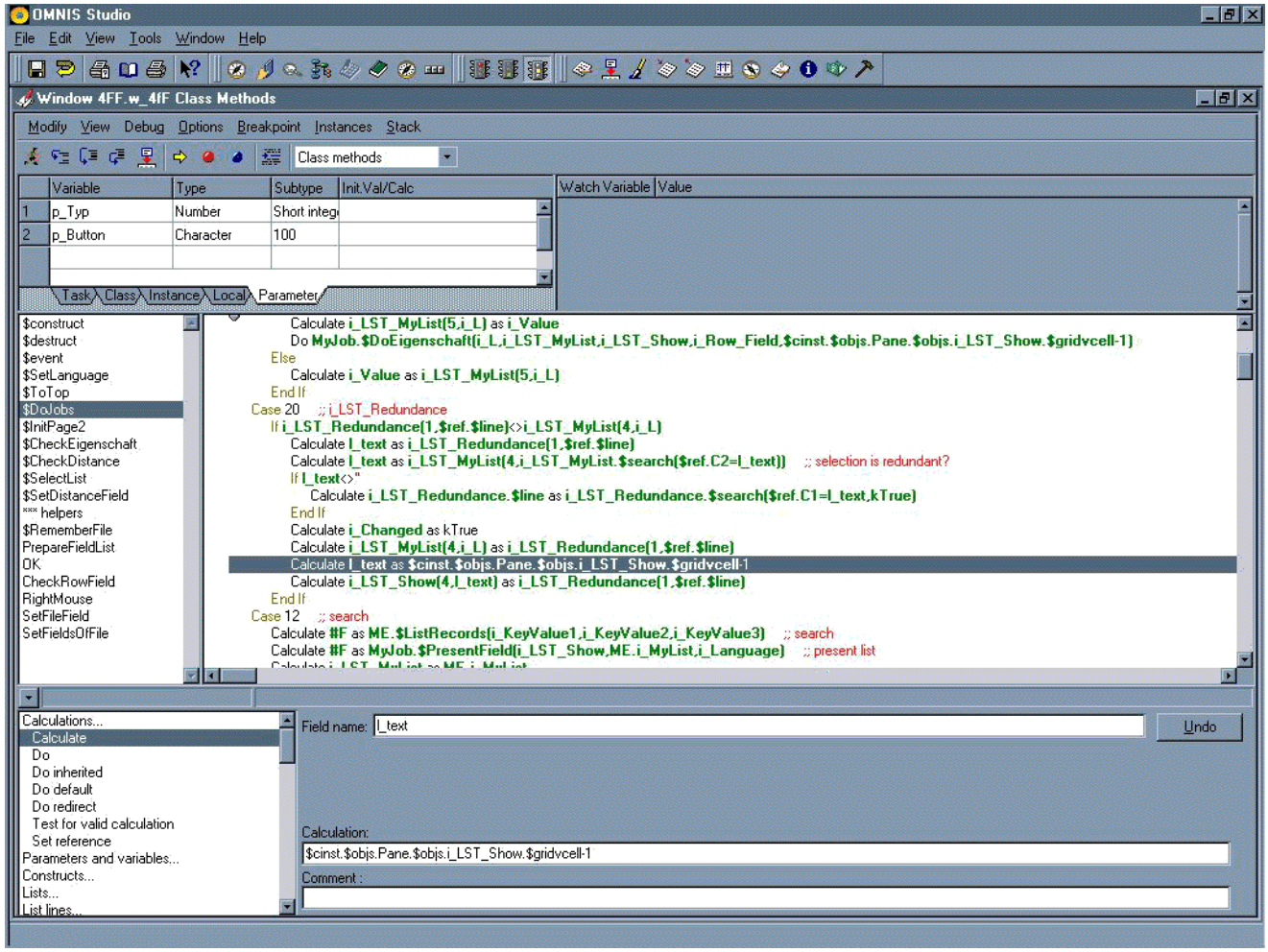

Abbildung 166 Neuer Case-Zweig für die Verarbeitung von i_LST_Redundance in $DoJobs

Fällt Ihnen auf, wie problemlos Omnis meine Faulheit kompensiert? Ich hatte zur Ermittlung des Feldnamens,

der möglicherweise als Redundanzverweis in meinem ausgewählten Feld hinterlegt war, eine lokale Variable **l_text** eingeführt, die ich skrupellos nun auch für die Bestimmung der numerischen Zeilennummer verwendete, die meine Anwender in ihrer Anzeigeliste **i_LST_Show** ausgesucht hatten. Klar, ist nur primitivste Konvertierung, doch praktisch ist es schon!

Beim Testen fiel mir übrigens auf, dass die Liste **i_LST_Redundance** nicht korrekt versorgt wird in der **$InitPage2**. Das ließ sich einfach dadurch beheben, dass das Statement

w_4fF, $InitPage2

```
Calculate i_LST_Redundance.$line as i_LST_Redundance.$search($ref.C1=i_LST_MyList(4,i_L))
;; actual line
```

unbedingt ausgeführt wird, denn bisher wurde es nur innerhalb des Abfrageblocks **If i_File<>i_Row_Info.1** berücksichtigt.

Das nächste zu beackernde Feld ist die Dateiangabe für Vorläufer und Nachfolger. Sie hat wie die Eigenschaft zu funktionieren, außer dass es keine Möglichkeit für unsere Anwender gibt, eine neue Datei hier einfach zu beantragen. Zuerst wird also die Feldmethode hinter dem Fenster-Objekt **i_FileField** versorgt:

w_4fF, i_FileField

```
On evAfter    ;; Event Parameters - pClickedField, pClickedWindow, pMenuLine, pCommandNumber,
pRow
    If #MODIFIED
        Do $cinst.$DoJobs(,l_Button)    ;; 2. parm
    End If
```

Alles wie gehabt. Auch der neue **Case**-Zweig 21 in der **$DoJobs** kann kopiert werden, muss nur unwesentlich abgeändert werden, denn wegen der etwas aufwändigeren Verarbeitung verwende ich die dafür vorgesehene Fenster-Routine **$CheckDistance**.

w_4fF, $DoJobs

```
Case 21    ;; i_FileField
    Do $cinst.$CheckDistance()
```

Die Methode **$CheckDistance** sieht der **$CheckEigenschaft** recht ähnlich. Zuerst organisiert sie sich die Referenz auf die **Files**-Datei, dann erfolgt die Unterscheidung, ob der Feldwert nur bearbeitet oder auch geprüft werden soll, um zu erkennen, ob ein Datensatz nur eingelesen oder ein neuer Eingabewert auch geprüft werden soll. Da die gesamte Prozedur ein wenig üppig wird, werden einzelne Teile wieder ausgelagert.

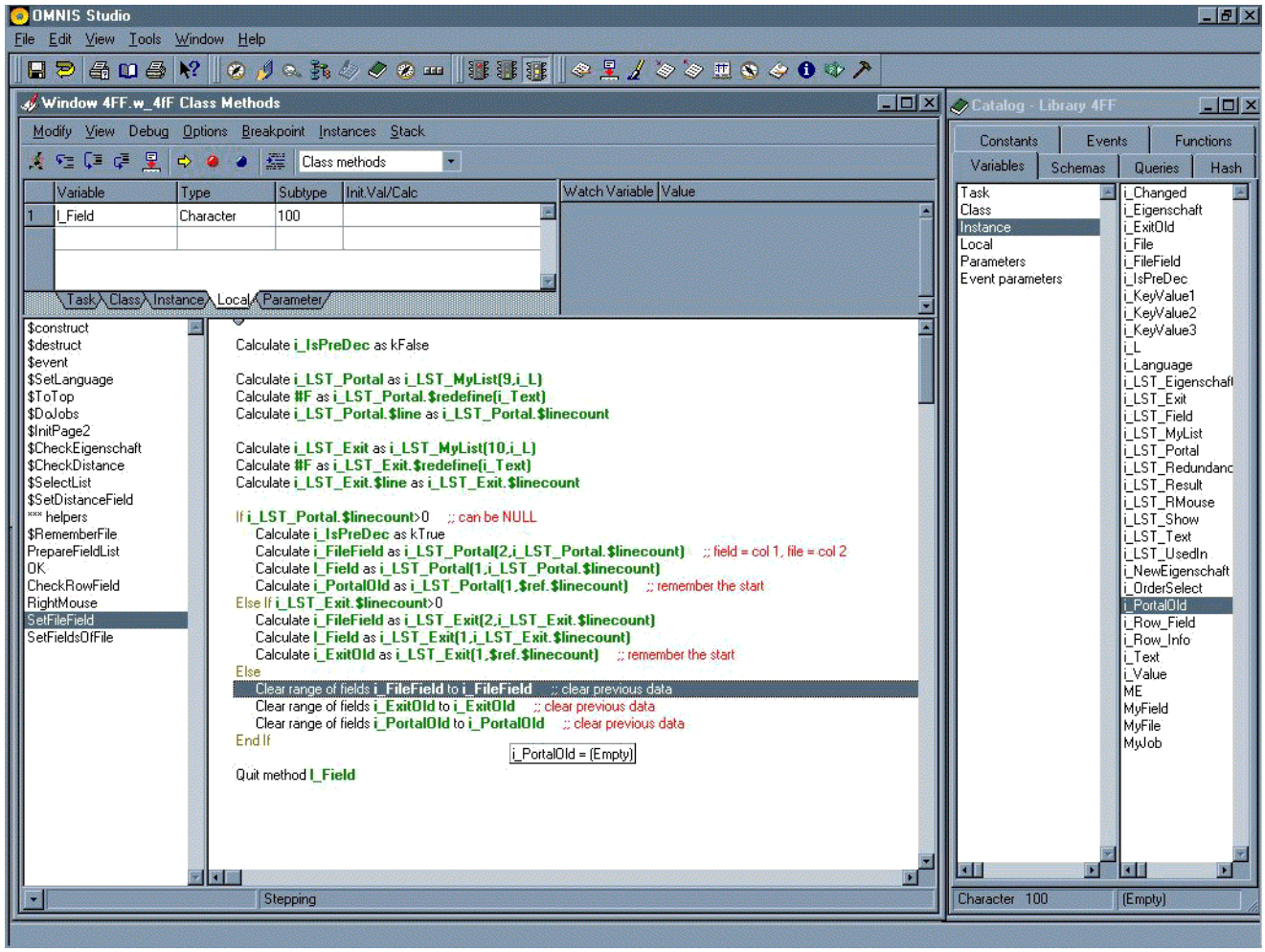

Abbildung 167 Einlesen des Datensatzes SetFileField

Bei einer neuen Eingabe braucht wie immer nur dann etwas getan werden, wenn sich der Wert überhaupt änderte. Ob sich ein Feldwert änderte, ist hier jedoch ein wenig umständlicher festzustellen als im Falle der Eigenschaft, denn der Feldwert hier ist Bestandteil einer Liste und zwar aus ihrer allerletzten Zeile. Welcher Liste, ist wiederum abhängig von unserem Markierungselement **i_IsPreDec**. Aus Übersichtlichkeitsgründen landet auch dies in einer eigenen internen Prozedur. Die Prüfung, dass die ausgesuchte Datei nicht gerade die eigene ist, unterlasse ich im Augenblick noch, da es mir das Testen vereinfacht, setze mir aber einen **BREAKPOINT** als Eselsohr, den ich jedoch für die Abbildung entfernte, damit sie noch vollständig auf dem Bild sichtbar bleibt.

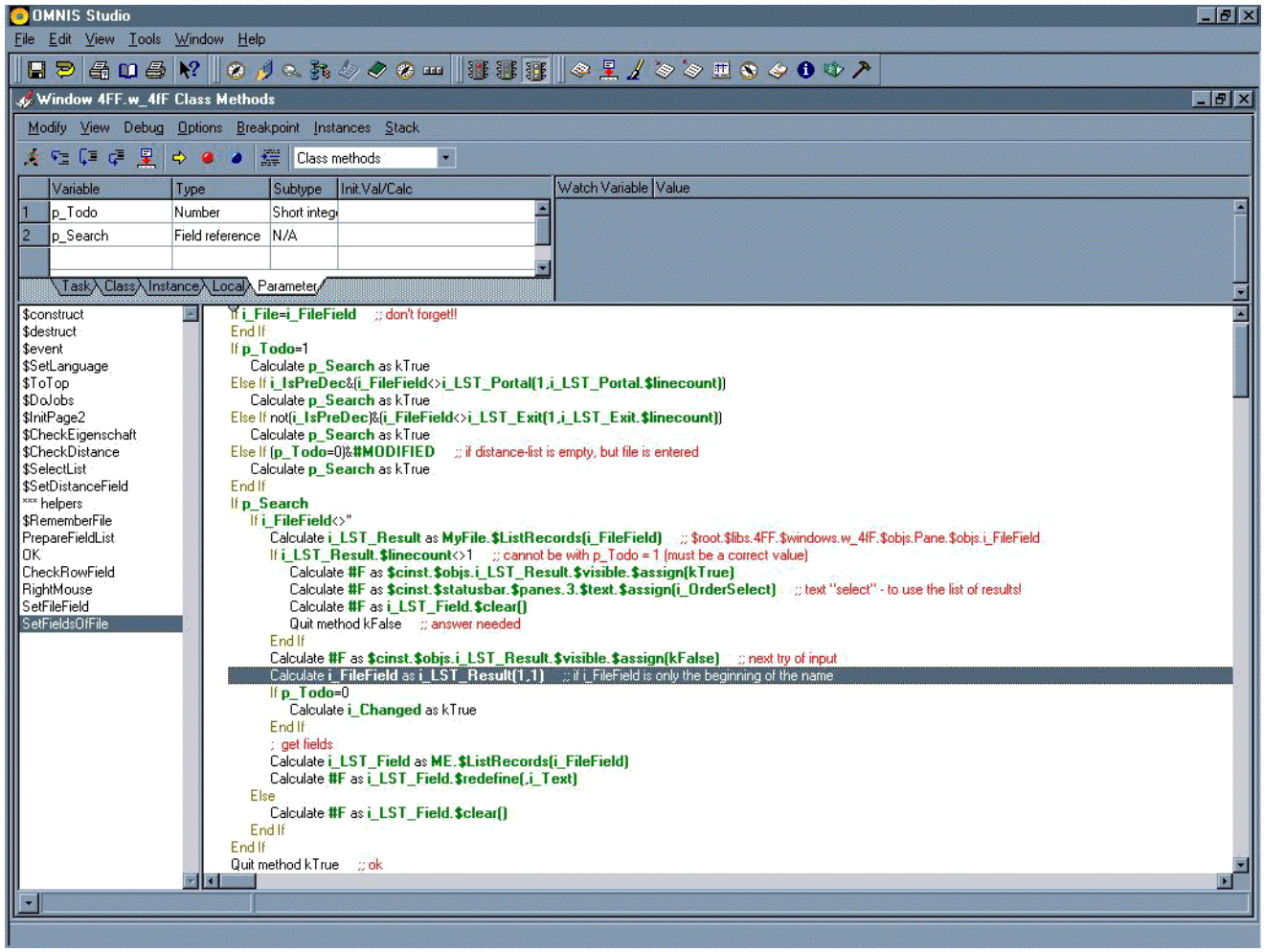

Abbildung 168 Einlesen des Feldliste der Datei SetFieldsOfFile

Die Behandlung der Liste gefundener Dateien in der Methode **$SelectList** sieht aus ähnlichen Gründen auch noch wie folgt aus:

w_4fF,
$SelectList

```
Else    ;; Files
        Breakpoint
End If
```

3.5 Feldänderungen im Fenster sind Dateiänderungen

Erinnern Sie sich noch, wie der Vorgänger bzw. Nachfolger die jeweilige Distanz-Liste erzeugte? Es war die individuelle Datei-Methode **$CheckDistance**, die sich den **4fF**-Datensatz des Vorgängers bzw. Nachfolgers ermittelte, dessen Distanz-Liste verwendete und einfach noch eine Zeile anhängte - den Vorgänger bzw. Nachfolger selbst. Die individuelle Datei-Methode **$CheckDistance** hat aber einen Nachteil - sie korrigiert automatisch die Liste **UsedInList**, was erst geschehen muss, wenn die Änderungen tatsächlich festgeschrieben werden. Das ist völlig ok, wenn diese Berechnungen für einen Datensatz durchgeführt werden, der gerade im Vorgang des Abspeicherns ist, doch nicht für unseren Fall. Wenn für mein aktuelles Feld ein anderes Dateifeld als Vorgänger bzw. Nachfolger bestimmt ist, muss die Berechnung der Distanz-Listen zwar erfolgen, es existiert aber noch gar kein