

PROGRESS

Many software vendors have come upon difficult times in the last year or so. There are a lot of reasons which are bantered around: the demise of the dot-com boom, the declining economy, 9/11, and others. Of the many factors, two in particular seem to be imposing themselves on the Progress marketplace.

The first of these is/was Y2K. The huge amount of accelerated capital investments made in anticipation of this “problem” meant that most companies did not want or need new software and hardware for some time after the fact. This has continued to have a suppressing effect on expenditures in the IT world. Not as many companies are buying new software, and it certainly appears that there are not as many initiatives (at least domestically) in terms of customizations, new projects and upgrades to existing packages.

The next factor was the “irrational exuberance”, the underlying greed of the dot-com bubble. For stocks to be valued at wildy exaggerated P/E ratios, for companies with no profit to acquire huge, otherwise stable businesses didn’t make sense then and doesn’t now. Whatever the cause, the effect of the demise of dot-com-dom has been manifold: a lot of otherwise readily available venture capital has dried up, a lot of people with computer experience competing for a very few jobs, a reticence to spend, and thus a general depression of the marketplace.

How does all of this affect PSC? The business has been surprisingly stable through it all. That is one measure.

But what about the signs and symptoms? Were we guilty of our own hubris? Have we felt that we would survive in spite of it all? There have been too many indications that PSC has ignored a lot of handwriting on the wall.

What does the current business scene tell us?

For years, there had been a healthy market for consulting, both within PSC and for independents. There are very few solitary consultants any more. Most have migrated to permanent positions. Progress’ own consulting staff is a shadow of what it was. This, when PSC, like many companies, touted consulting as an insulator from declining product revenues.

Less consulting implies fewer customization or new-initiative projects. It implies people are turning to other solutions for their problems. Is this a result of the heavy pre-Y2K investment, or is it a larger trend? Some of both, but it is our opinion that primarily, we are witnessing a shift within the Progress community which PSC fails to recognize. We hope that they will continue to observe these not-too-subtle demographic changes and strive accordingly to cultivate a more loyal base as well as new customers.



Accessing File Summary Information Using COM By Jared Middleton

If you use any version of Microsoft Windows, you have probably seen File Summary Information at least once. When you right-click a file in Explorer and choose **Properties** (or Alt-double-click the file icon), you will see a properties dialog with a **General** tab, and maybe a **Security** tab if using NTFS. On some files, the **Properties** dialog also shows **Summary**, **Statistics**, **Contents** and **Custom** tabs. These tabs contain many additional useful attributes such as: Title, Subject, Author, Company, Category, Keywords, Comments, and much more.

Have you ever wondered what caused the extra tabs to appear? This article will explain where these screens come from and how you can access their attributes from the comfort of the Progress 4GL. Plus, for those of you who have upgraded to Windows XP, I have a tip showing how you can make many of these attributes display in Windows Explorer's standard detail view.

Due to the ubiquity of Microsoft Windows and Office, some people assume that the summary info exists for all file types. However, only specific files called OLE compound documents have the ability to store additional information about a document in persistent property sets, collectively called Document Summary Properties. These property sets are managed by COM/OLE so that third-party clients can read this information without the aid of the main application responsible for the file.

Document properties are not basic file system properties. They cannot be accessed using the Progress FILE-INFO handle or Window's own File System Object API. They are a collection of special properties attached only to OLE compound documents (particularly those used by Office applications).

There are two different ways to access document properties from a Progress application. You can use the Automation server method or you can call an ActiveX DLL that implements the underlying interfaces.

MS Office Automation servers

The code in Figure A demonstrates the first method of accessing the document properties of Word, Excel and PowerPoint files using COM calls to the Automation server of each document's native application.

```

/*****
  Procedure:   doc-prop.p
  Description: Demo program to access
  Document Summary Properties using MS Office
  Automation servers.
  Written by:  Jared Middleton   (March 26,
  2002)
  *****/

DEF VAR chApp          AS COM-HANDLE NO-UNDO.
DEF VAR chDocument     AS COM-HANDLE NO-UNDO.
DEF VAR chProperties    AS COM-HANDLE NO-UNDO.
DEF VAR cFile          AS CHARACTER  NO-UNDO.
DEF VAR lOpen         AS LOGICAL    NO-UNDO.
DEF VAR iCnt          AS INTEGER    NO-UNDO.

SYSTEM-DIALOG GET-FILE cFile
  FILTERS
    'Word Files (*.doc)'      '*.doc',
    'PowerPoint Files (*.ppt)' '*.ppt',
    'Excel Files (*.xls)'     '*.xls',
    'All Files (*.*)'        '*.*'
  UPDATE lOpen.
IF NOT lOpen THEN RETURN.

/* Examine file extension to determine file
association */
CASE SUBSTR(cFile,LENGTH(cFile) - 3,4):
  WHEN ".doc" THEN DO:
    CREATE "Word.Application" chApp NO-
    ERROR.
    IF VALID-HANDLE(chApp) THEN
      ASSIGN chDocument =
chApp:Documents:OPEN(cFile) NO-ERROR.
    END.
  WHEN ".ppt" THEN DO:
    CREATE "PowerPoint.Application" chApp
    NO-ERROR.
    IF VALID-HANDLE(chApp) THEN
      ASSIGN chDocument =
chApp:Presentations:OPEN(cFile,,FALSE) NO-
    ERROR.
    END.
  WHEN ".xls" THEN DO:
    CREATE "Excel.Application" chApp NO-
    ERROR.
    IF VALID-HANDLE(chApp) THEN
      ASSIGN chDocument =
chApp:Workbooks:OPEN(cFile) NO-ERROR.
    END.
  OTHERWISE
    MESSAGE "No Automation server defined
for this file extension"
    VIEW-AS ALERT-BOX.
END CASE.

IF VALID-HANDLE(chDocument) THEN DO:
  ASSIGN chProperties =
chDocument:BuiltInDocumentProperties.

DO ON ENDKEY UNDO, LEAVE:

```

```

        DISPLAY chProperties:ITEM("Title"):NAME
+ ":" FORMAT "x(27)"
        chProperties:ITEM("Title"):VALUE
FORMAT "x(47)"
        WITH FRAME frmDispX NO-ERROR.
    DO iCnt = 1 TO 30 WITH FRAME frmDisp:
        DISPLAY chProperties:ITEM(iCnt):NAME +
":" FORMAT "x(27)"
        chProperties:ITEM(iCnt):VALUE
FORMAT "x(47)" NO-ERROR.
    END.
    /* chProperties:ITEM("Comments") = "New
Comment". */
    END.
    chDocument:CLOSE(0) NO-ERROR. /* -1 =
save, -2 = prompt */
    chApp:QUIT().
END.

RELEASE OBJECT chProperties NO-ERROR.
RELEASE OBJECT chDocument NO-ERROR.
RELEASE OBJECT chApp NO-ERROR.

```

Figure A

As you can see, the Automation server demo procedure is fairly generic. It uses the same com-handles regardless of the file type selected. The only difference is in how the document object is instantiated.

To access Document Properties, you first need to create an appropriate application and document instance and then access the BuiltInDocumentProperties property.

Use the **Item** method to return a single member of the **DocumentProperty** object by specifying either the name of the property or the index number.

Figure B shows the available built-in document property names and corresponding index number.

Container applications don't necessarily define a value for every built-in document property. If a given application doesn't define a value for one of the built-in document properties, returning the **Value** property for that document property causes an error.

The Automation server method has some other disadvantages, primarily in terms of performance and



JARGON™

- Develop powerful Wireless applications in our new V3 Jargon Writer!
- Run wireless apps either online or offline when out of coverage area
- Create "bolt-on" solutions to your Progress app in days, not months.
- Integrate barcode scanners, printers and other wireless peripherals
- Deploy with AppServer or WebSpeed. Save money with AppServer.
- Enjoy easy, powerful wireless deployment for under \$150 per handheld.
- Leverage your 4GL experience - no need to master Java, C++, or HTML.
- Download a FREE Evaluation Copy or contact us for a personal online demo

JARGON SOFTWARE Inc
708 North First Street, Suite 432
Minneapolis, Minnesota 55401
612 338 1175 | fax 612 338 2974
www.jargonsoft.com (On-line demo!)
info@jargonsoft.com

4

use of system resources. Each Automation server must load its objects and methods into memory so that these are ready to be called.

To show memory usage, I ran the demo program using a blank document of each supported type and then examined the NT Task List. The approximate results were: Word = 10MB, Excel = 6MB, PowerPoint = 6MB. That's a lot of overhead just to access some document summary properties!

Warning: *“Microsoft does not currently recommend, and does not support, Automation of Microsoft Office applications from any unattended, non-interactive client application or component, because Office may exhibit unstable behavior and/or deadlock behavior when run in this environment.”*

1 Title	16 Number of Characters
2 Subject	17 Security
3 Author	18 Category
4 Keywords	19 Format
5 Comments	20 Manager
6 Template	21 Company
7 Last Author	22 Number of Bytes
8 Revision Number	23 Number of Lines
9 Application Name	24 Number of Paragraphs
10 Last Print Date	25 Number of Slides
11 Creation Date	26 Number of Notes
12 Last Save Time	27 Number of Hidden Slides
13 Total Editing Time	28 Number of Multimedia Clips
14 Number of Pages	29 Hyperlink Base
15 Number of Words	30 Number of Characters (with spaces)

Figure B

OLE File Property Reader

Document properties can be viewed and changed without using native Automation servers. Microsoft Developer Support provides a free component, which anyone can download and use.

Dsofile.dll is an ActiveX component for programmers to use in order to read and modify both the standard and custom properties from any OLE Structured Storage file. This includes (but is not limited to) Word documents, Excel workbooks, and PowerPoint presentations. Due to its size and speed, the DLL can be much more efficient than trying to Automate Office applications to read document properties.

You can download the DLL from Microsoft directly (see the link for Q224351 under References), but be aware that it only comes bundled as part of a packaged VB

demo, in the form of a self-extracting EXE (MS web page provides install instructions) – not exactly straightforward.

You can also download the DLL directly from Progressions FTP site and register it yourself (**recommended**). The DLL can be saved anywhere you like, such as C:\Windows, C:\WINNT, C:\WINNT\system32, or you can create a new folder to store it.

To use the Property Reader, you need to register Dsofile.dll on your system. Assuming you have the appropriate permissions, you can use the **Run** dialog box on the **Start** menu, as follows:

To register the DLL (modify the path as required):

```
regsvr32 c:\winnt\dsofile.dll
```

To unregister it:

```
regsvr32 /u c:\winnt\dsofile.dll
```

The sample code in Figure C demonstrates the second method of accessing document properties using the ActiveX DLL.

```
/*  
*****  
Procedure: dsofile.p  
Description: Demo program to access  
Document Summary Properties using ActiveX  
OLE File Property Reader DSOFIELDLL.  
Written by: Jared Middleton (March 26,  
2002)  
*****  
*/  
  
DEF VAR chDSOFile AS COM-HANDLE NO-UNDO.  
DEF VAR chProperties AS COM-HANDLE NO-UNDO.  
DEF VAR cFile AS CHARACTER NO-UNDO.  
DEF VAR lOpen AS LOGICAL NO-UNDO.  
DEF VAR iCnt AS INTEGER NO-  
UNDO.  
  
SYSTEM-DIALOG GET-FILE cFile  
FILTERS  
'Word Files (*.doc)' '*.doc',  
'PowerPoint Files (*.ppt)' '*.ppt',  
'Excel Files (*.xls)' '*.xls',  
'All Files (*.*)' '*.*'   
UPDATE lOpen.  
IF NOT lOpen THEN RETURN.  
  
CREATE "DSoleFile.PropertyReader" chDSOFile  
NO-ERROR.  
IF VALID-HANDLE(chDSOFile) THEN DO:  
ASSIGN chProperties =  
chDSOFile:GetDocumentProperties(cFile).
```

```

IF VALID-HANDLE(chProperties) THEN DO:
  IF chProperties:IsReadOnly THEN
    MESSAGE "Cannot make changes to"
cFile +
    ". Make sure you have modify
permissions for the file."
    VIEW-AS ALERT-BOX.
  ELSE DO ON ENDKEY UNDO, LEAVE:
    DISPLAY "Name:" AT 1
chProperties:Name FORMAT "x(47)" AT 30
    "Title:" AT 1
chProperties:Title FORMAT "x(47)" AT 30
    "Author:" AT 1
chProperties:Author FORMAT "x(47)" AT 30
    "Company:" AT 1
chProperties:Company FORMAT "x(47)" AT 30
    "Comments:" AT 1
chProperties:Comments FORMAT "x(47)" AT 30
    WITH NO-LABELS FRAME frmDisp.
/* chProperties:Comments = "New Com-
ment". */
  END.
END.
END.

RELEASE OBJECT chProperties NO-ERROR.
RELEASE OBJECT chDSOFile NO-ERROR.

```

Figure C

Dsofile.dll has only one creatable object, DSOleFile.PropertyReader. This object provides access to document properties through a single method, GetDocumentProperties, which takes the name of an OLE Structured Storage file and returns a DocumentProperties object that you can then Automate to read or modify property values.

The DocumentProperties object provides access to the most common summary information used by Office applications (Figure D). When the object is created, the specified file is open for exclusive access. If the file is in use by another application when you try to open it, you get a trappable error telling you that the file is locked.

AppName	character	read
Author	character	read/write
ByteCount	integer	read
Category	character	read/write
CharacterCount	integer	read
CharacterCountWithSpaces	integer	read
CLSID	character	read
Comments	character	read/write
Company	character	read/write
CustomProperties	com-handle	read
DateCreated	date	read

DateLastPrinter	date	read
DateLastSaved	date	read
HasMacros	logical	read
HiddenSlides	integer	read
Icon	com-handle	read
IsReadOnly	logical	read
Keywords	character	read/write
LastEditedBy	character	read/write
LineCount	integer	read
Location	character	read
Manager	character	read/write
MultimediaClips	integer	read
Name	character	read
PageCount	integer	read
ParagraphCount	integer	read
PresentationFormat	character	read
PresentationNotes	integer	read
ProgID	character	read
RevisionNumber	character	read
SlideCount	integer	read
Subject	character	read/write
Template	character	read
Thumbnail	com-handle	read
Title	character	read/write
TotalEditTime	integer	read
Version	character	read
WordCount	integer	read

Figure D

Custom Properties

In addition to the “Built-in” properties, there are “Custom” property sets defined. Using the **Custom** tab of the file **Properties** dialog, the user can view custom properties already created for the document, or create new ones. Custom properties can also be created programmatically.

For example, Company XYZ might have a custom property, DeptHomePage, which lists the Intranet home page of the department that created the document. Authors put this property in their document, then display it in a footer so whoever reads the document can go directly to the Intranet home page of that department.

If there is demand, I may discuss Custom Properties in a follow-up article.

TIP for Windows ME, 2000, XP users

Many of the summary properties can be viewed in Explorer and in folder windows without your having to open the files or their Properties dialog boxes. To arrange to get all the details, select View, Choose Details (in Windows XP) or View, Choose Columns (in Windows

Me and 2000). Check the items appropriate for the files you keep in the current folder, and click OK.

For customizing on the fly, right-click any of the column headings above the file list and then choose a category you want to display or hide. Not all categories appear in this context menu, but if you don't see one that you want, select More... at the bottom of the menu to open the Choose Details dialog box.

References

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/office97/html/ORKhT/023.asp>

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;q224351>

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dn_voices_askgui/html/askgui04102001.asp

Jared Middleton

Jared is currently the Senior Business Systems Analyst for Allied Telesyn, an international network equipment provider. He started working with Progress V5 in 1989 and has since been designing business applications and creating custom software development tools for employers in the public and private sectors. Jared is an active member of the Washington Progress Users Group (WaPUG) and the Progress E-Mail Group (PEG). On occasion, Jared can also be found consulting and training for TechSmiths, Inc. He can be reached at jared_middleton@alliedtelesyn.com

Enhancing the ADM2 'CurrentChanged' Processing By John D. Johnson

It has been said, "everything has its price." One of the significant improvements incumbent in the move from ADM1 to ADM2 is the ability to deploy SmartObject applications in an n-tier environment. However, the price of this improvement has been a significant increase in the difficulty associated with enhancing the response to the "CurrentChanged" condition. This article will provide one approach to overcoming this problem, while also illustrating the preferred methodology for implementing custom enhancements to ADM2 code.

A Little History

Those readers who are unfamiliar with ADM1 may not understand the "CurrentChanged" condition, so I'll provide a brief explanation. Because the ADM operates within an optimistic locking approach – that is, the query fetches the data without locking the database records, it is possible for two users to update the same record at the same time. To maintain data integrity under these conditions, it is necessary to compare the current data with the original when the record is fetched for update purposes. In ADM1, the "Update Object" – normally the SmartViewer – was connected to the database and actually controlled the transaction. It contained a procedure called adm-current-changed that performed this process. It did a BUFFER-COMPARE and the default error response was to notify the user that someone else had modified the record, cancel the update process, and display the new "starting" data so that the user could once again input his changes. It was, therefore, relatively easy to override the default response by writing a local-current-changed procedure. But the adm-current-changed procedure disappeared when ADM2 was released. Why?

In ADM2 the viewers and browsers are no longer connected to the database and no longer control the database update process. These duties are now the responsibility of the SmartDataObject – which is designed to be split across the AppServer boundary. Data extracted from the database is collected in a RowObject Temp-Table within the Server-side SDO component and communicated to the Client-side component by passing this Temp-Table across the boundary. User-requested changes are collected in the RowObjUpd Temp-Table on the client and passed to the server by passing this Temp-

Table. The RowObjUpd Temp-Table is also used to pass “post-update” data back to the client. Consequently, the buffer compare process is performed on the Server – away from the User Interface – and errors are returned via messaging. To make matters worse, the doReturnUpdate procedure that contains the code we need to modify is contained in the data.i method library which gets compiled directly into each SDO, so we cannot simply write a “local” version to enhance its behavior. But we still have options.

“Modifying data.i”

Since the doReturnUpdate procedure is compiled into the SDO from the data.i method library, the standard ADM code actually represents the “local” version. Therefore, the only way to modify the behavior is to modify the code that is compiled into the SDO. There are several ways to do this. I discussed these alternatives in my “ADM Tips and Tricks” article in the 36th issue of Progressions, but I’ll briefly reiterate here.

The easiest way to do this is to open the data.i file located in the DLC/src/adm2 directory, make your changes, and save the file back to its original directory – NOT A RECOMMENDED PRACTICE! The installation of a new release of Progress will wipe out your changes.

Okay, let’s modify the data.i method library and save it in a local directory. New releases won’t cause the loss of your changes, but you will not be able to take advantage of improvements in other code within the method library – ALSO NOT RECOMMENDED.

The recommended approach is as follows:

1. Create a new method library – preferably using the Structured Include template – and copy the complete doReturnUpdate procedure from data.i to our new method library.
2. Make the desired code modifications to our new copy.
3. Save the new method library into our local src/adm2 directory as a different name. My example calls it mydata.i.
4. Open the SDO and open its Procedure Settings Dialog.
5. Click on the Method Libraries button and add mydata.i to the list of included method libraries for the SDO.

If we attempted to compile the SDO at this point, we



Code analysis tools and services

- **Proparse: the 4GL code parser**
- **AutoDox generates HTML docs for your code**
- **Prolint: open-source 4GL lint by Jurjen Dijkstra**
- **Services: custom tools development and parser assisted code analysis**

**Speed up your code understanding and analysis.
Increase your code consistency and reliability.
Find bugs before they get out the door.**

www.joanju.com

Progressions

Cross Country Skiing:	John Campbell
Birding:	Connie Campbell
Observing:	Harriet Coates
Horseback Riding:	Michael Bartlett

Progressions is published bi-monthly, which means 6 times a year, not 24.

Now that we are an electronic newsletter, we are offering the same price for everyone, no matter where you live! The annual subscription price is \$60.00 USD.

We accept Visa, Mastercard, and/or American Express (please include your expiration date and whether or not you have enough of a credit line for us to head for Katmandu or simply the border). All checks must be drawn on a US bank; checks and money orders must be in US dollars. In order to protect our contributors, the information in this periodical is copyrighted 1992-2002 White Star Software, Inc., and/or the author. All rights reserved worldwide and to the end of the cosmos.

Progressions

**PO Box 250 Carbondale, CO 81623
970.963.3545 Voice 970.963.3548 Fax**

would encounter a significant error. We would be attempting to compile multiple copies of the `doReturnUpdate` procedure – Progress’ version and our own – into the SDO. NO CAN DO! So we need to find a way to skip the original version of the procedure during the compile process.

To accomplish this, all we need to do is to create another include – I called mine `myexclude.i` – and add it to the the SDO’s list of method libraries. This include contains only a single line of code. (Figure A)

Figure B shows the required order for all of the method libraries. Placing `myexclude.i` first means that, by the time the standard `doReturnUpdate` procedure – in `data.i` – is encountered by the compiler, the “exclude” preprocessor has been defined. Since the standard code is wrapped in a test for the existence of this preprocessor, the standard copy of the procedure will be skipped. Our enhanced copy will be compiled because it is not wrapped in a preprocessor test, but the `mydata.i` method library must come last because the `doReturnUpdate` procedure references elements that are defined in `data.i`.

Coding the Enhanced Processing

The default response to the “CurrentChanged” condition in ADM2 is quite similar to that exhibited in ADM1. For each record updated by the user, the updated record and a copy of the original data are passed to the server side in the `RowObjUpd Temp-Table`. The server-based component of the SDO fetches the record from the database and compares it to the data originally sent to the user. If the two copies of the data fail to match – indicating that someone else has also updated this record, the original copy in the `RowObjUpd Temp-Table` is replaced by the more current copy, the `addMessage` procedure is run to store the appropriate error condition in the “message queue”, and the `RowObjUpd Temp-Table` is passed back to the client-side component of the SDO for processing by the `doReturnUpdate` procedure.

If an error message has been stored, the `doReturnUpdate` procedure moves the most current data from the `RowObjUpd Temp-Table` to the `RowObject Temp-Table` so that the user will be able to see the changes made. But in the process, the changes the current user has made are discarded.

Following the execution of `doReturnUpdate`, the stored error message is displayed to the user, the current data is displayed, and the user must re-enter his changes. For

the purposes of this article, we will enhance this response in the following manner:

1. We will continue to notify the user and halt the update whenever a “CurrentChanged” condition arises, but will vary the error message based upon the severity of the condition.
2. We will use the `ChangedFields` field of the `RowObjUpd` record to determine whether both users have attempted to update the same field(s). For those fields where no conflict exists, the current users changes will be preserved.
3. We will display the “merged” data to the user and require the user to confirm the Save by initiating the Save process again.

Figure C shows the pertinent portion of the code in the enhanced `doReturnUpd` procedure. The modifications are shown in **BOLD** type. The block in which all of our modifications are found is headed by the statement:

```
FOR EACH RowObjUpd WHERE
    RowObjUpd.RowMod = "U":
```

Each record found represents current data returned from the server. The `FIND` of the matching `RowObject` record provides currency on the record that represents the data input by the current user and the `FIND` of the `bRowObjUpd` record with the same `RowNum` but a blank `RowMod` value provides us with the data as it was before either user input changes. Thus, we now have all versions of the data available.

At this point, the default behavior would simply copy the current data from the `RowObjUpd` record to the matching `RowObject` record, so here is where we place our important modifications.

Capturing the values of all the `BUFFER HANDLES` allows us to dynamically address specific fields within these buffers – giving us the ability to build a new version of the data on a field-by-field basis. While the record represented by the `hROU` handle contains the other user’s updates, the record represented by the `hRO` handle contains the current user’s changes. We will attempt to merge these two sets of changes into a single record by moving “uncontested” changes from the current user’s record into the other user’s copy returned from the server, but we first need to determine that each field change is “uncontested”. That is, the field wasn’t changed by both users.

The fields changed by the current user are identified in the ChangedFields field of the hROU record. Therefore, all we need to do to accomplish this is to parse the ChangedFields list, acquire the handle of the identified field in each of the hROU and hAltRow buffers, and compare the current value of the field to the original value. This tells us whether or not the field was changed by the other user.

Here, we perform a fetchMessages() function to clear the default error messages from the message log so that we can replace them with our own message. If the field has been changed by another user, we do nothing further at this point – meaning that the current user’s change to that field will be discarded. If no other user has changed the value of this field, we use the ASSIGN statement to move the current user’s change from the hRO buffer-field to the hRO buffer-field and add the field name to the list held by the cExcept variable.

Once all of the changes have been merged into the RowObjUpd record, we allow the default action to BUFFER-COPY this record to the RowObject Temp-Table. We also run the addMessage procedure to add our error message to the message log. Our last task here is to capture a new copy of the data just returned from the server in the RowObjUpd Temp-Table as our “before-image”. Progress’s code uses the BUFFER-COPY statement, which requires a static exception list, but we must exclude our current user’s changes that we’ve just merged into the record. So we will replace that statement with the BUFFER-COPY() function, which allows for a dynamic exception list – supplied here by the value of our cExcept variable.

By the time this procedure ends, we have the following situation within the client portion of the SDO:

1. The RowObject Temp-Table contains the merged result of changes by all users.
2. The RowObjUpd Temp-Table contains a copy of the data as it supposedly exists within the database. The fact that it’s RowMod value is blank flags this as the “before-image” record.
3. The message log contains an error message that will explain to the user that his update was not completed successfully and why.

Our changes to the doReturnUpdate procedure are complete, but further modifications are needed elsewhere to implement the desired UI activity. The Save process initiated in the UI has not been successfully completed, so the fields are still enabled in the update object. Consequently, even though the new data is now available in the RowObject Temp-Table, it is not displayed in the UI. To overcome this problem, we will create a custom version of the updateRecord procedure to enhance the default processing. Since we wish this to affect all update objects, we will implement the override by placing it in a custom super procedure associated with our update object class. That is, we will activate the viewercustom.p super procedure.

Figure D lists the custom override of the updateRecord procedure. We allow the standard ADM version(s) to run first by issuing the RUN SUPER statement. The “CurrentChanged” condition will result in the “ADM-ERROR” value being returned from the standard ADM processing. In this case, our enhancement executes the cancelRecord procedure. This results in termination of the update mode – thereby allowing the data being displayed to be refreshed with our merged information.

But if we stopped here – with the input fields disabled, we would be misleading the user into thinking that what he is seeing is a representation of the data that is currently in the database. In reality, it represents our best guess at what data SHOULD be placed into the database – and we have warned the user that he should review this data and re-Save it (See Figure E). To ensure a user-friendly process, we need to place the screen back into the update (enabled) state after refreshing the display. This is accomplished by running the updateMode procedure and passing it the “UpdateBegin” parameter value.

Thus, we have enhanced the response to the “CurrentChanged” condition and provided the user with a friendlier process. The techniques and code discussed here are by no means the only workable approach, but are intended to provide a starting point from which each developer can let his or her imagination be the guide. Happy coding.

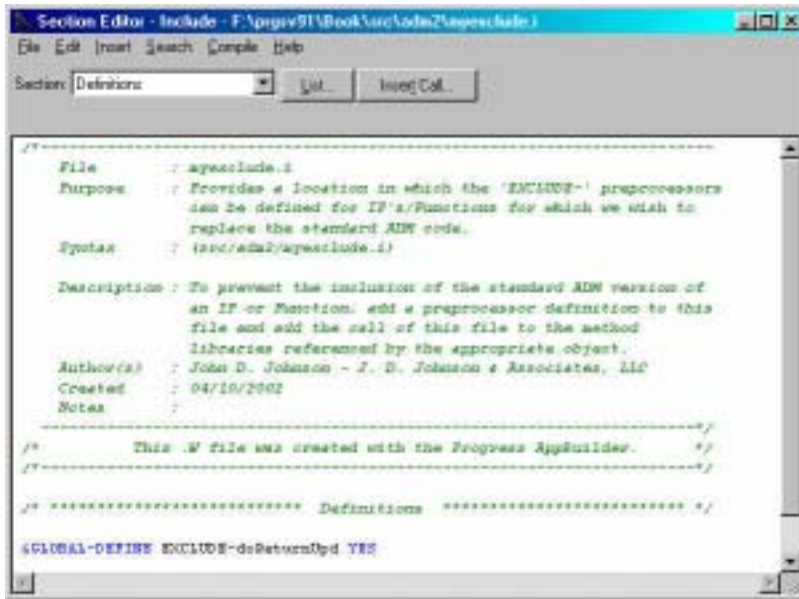


Figure A

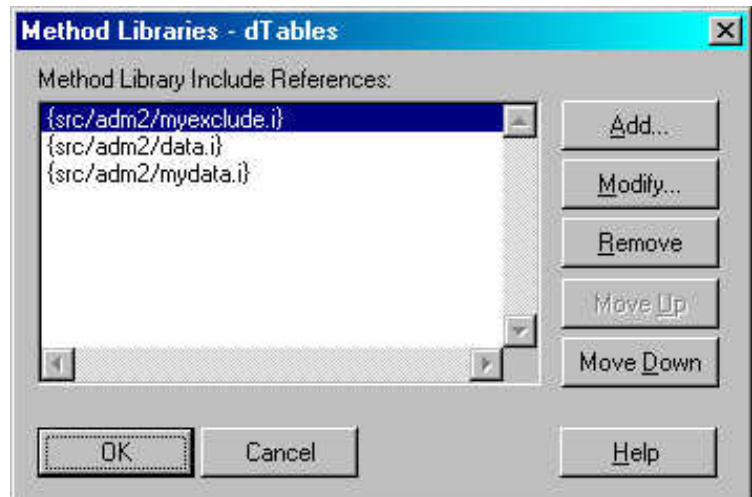


Figure B

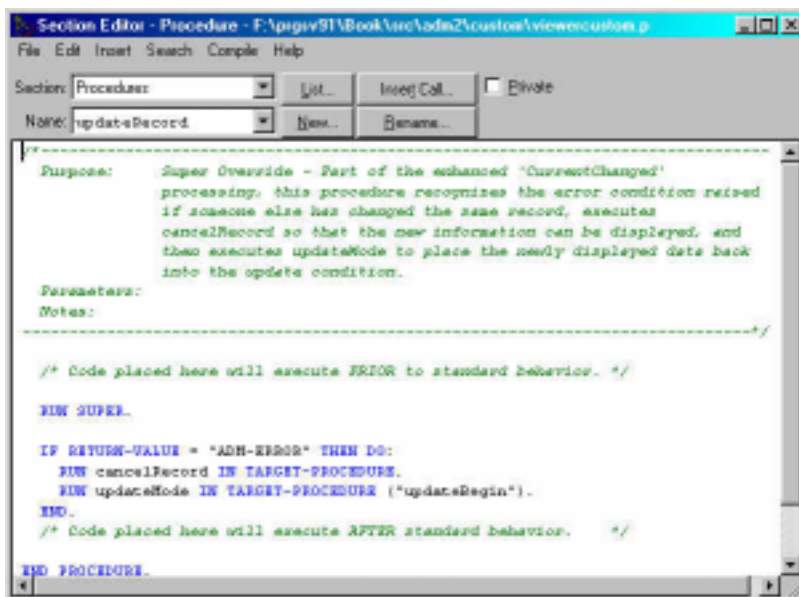


Figure D

```

FOR EACH RowObjUpd WHERE RowObjUpd.RowMod = "U":U:

    /* Refreshing the update record if required (changed data was reset on server).*/
    IF INDEX(pcUndoIds,STRING(RowObjUpd.RowNum) + CHR(3) + "ADM-FIELDS-CHANGED":U) <> 0
THEN
DO:
    FIND RowObject WHERE RowObject.RowNum = RowObjUpd.RowNum.
    /* The following line was moved here from farther down */
    FIND bRowObjUpd WHERE bRowObjUpd.RowNum = RowObjUpd.RowNum
        AND bRowObjUpd.RowMod = "" :U.
    ASSIGN hRO = BUFFER RowObject:HANDLE
        hROU = BUFFER RowObjUpd:HANDLE
        hAltRow = BUFFER bRowObjUpd:HANDLE
        cExcept = "RowMod,ChangedFields".

    DO iCntr = 1 TO NUM-ENTRIES(RowObjUpd.ChangedFields):
        ASSIGN cField = ENTRY(iCntr, RowObjUpd.ChangedFields)
            hROField = hROU:BUFFER-FIELD(cField)
            hROUField = hAltRow:BUFFER-FIELD(cField).

        cMsg = fetchMessages().
        IF hROField:BUFFER-VALUE NE hROUField:BUFFER-VALUE THEN
            cMsg = "You and another user both changed the " + cField + " field."
                + CHR(10) + "Your other changes have been preserved, but" + CHR(10)
                + "you must verify the data and attempt the Save again.".
        ELSE
            ASSIGN hROUField = hRO:BUFFER-FIELD(cField)
                hROField:BUFFER-VALUE = hROUField:BUFFER-VALUE
                cExcept = cExcept + "," + cField
                cMsg = "You and another user both changed the same record."
                    + CHR(10) + "Your other changes have been merged with those"
                    + CHR(10) + "of the other user, but you must verify the data"
                    + CHR(10) + "and attempt the Save again.".
        END.
    RUN addMessage (cMsg, "", "").
    /* Copy the refreshed db field values to row object. */
    BUFFER-COPY RowObjUpd EXCEPT RowMod TO RowObject.
    hAltRow:BUFFER-COPY(hROU, cExcept, "").

```

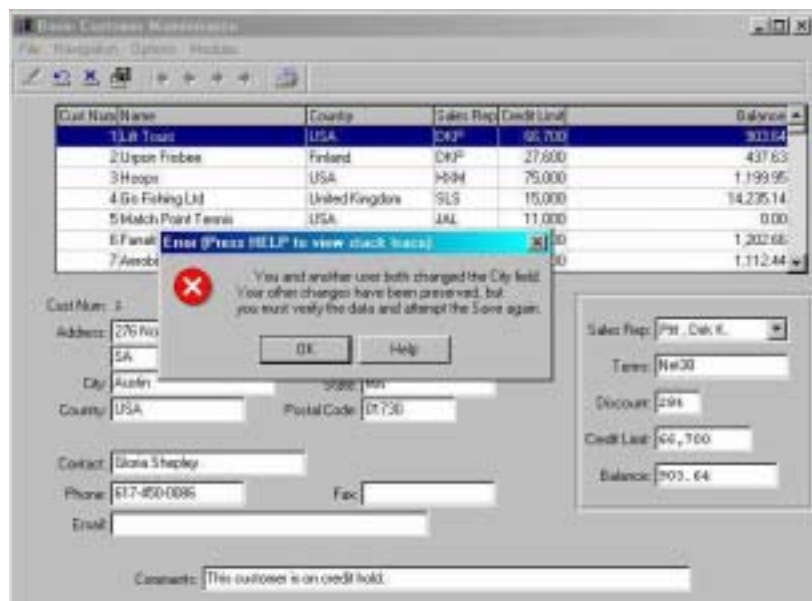
Figure C

John D. Johnson

John is the author of the **Roadmap To The SmartObjects** book and spends much of his time roaming the World doing such things as Progress training, code tuning, replication, and project management. But he's never too busy for new clients.

J. D. Johnson & Associates
Voice: 972-733-1422
Fax: 972-733-1580
jdj@jdjohnson.com

Figure E



Relevant Relations by Jop Kluis

Every now and then I get in circumstances which force me to do some reverse-engineering. This is always the consequence of not having any decent documentation. The process itself is often tedious work, which requires some precision. I once had a group of trainees produce through reverse engineering a data model from a Progress database. When we compared their work with the original data model used to create the database we found that nobody came up with a 100% fit. And for each difference they had a good story. This has to do with working on assumptions, the lack of knowledge of most of the available tools and the fact that a relational database is nothing more than a large container holding vast amounts of data without storing relevant information about the relations between data within the database. In this article I will try to explain which difficulties you can encounter as you use tools to extract a data model from an existing database.

Just the other day I got to talking to a company which was completely lost in the jungle of relations in its database(s). This company used some Progress based packaged software. They had a great need for more and better reports. Their software supplier offered them an additional 'module' containing over 150 ready to use extra reports. Not only were over 120 of these reports not relevant and the remaining 30 reports only 'close but not a 100% fit', but this additional 'module' did not come cheap. Being true and loyal Progress users they always read everything that is published on Progress. So they also read some of my BI stuff. I was called in to help them out using BI tools. After some explaining they understood they would have to 'build' multi-dimensional data 'boxes'. These 'boxes' –Cognos calls them Cubes and CorVu calls them Dynamarts – are the bases for all the reports and analyses. These boxes not only contain the necessary data in a multi-dimensional format but also enable you to present an easy to understand data model for the end user while the real data structure in the database itself is truly incomprehensible. So the trick is to build some queries which 'translate' the data with its complex structure to a much easier model. In order to do so it is very – if not extremely handy – to have a good data model available. Unfortunately, the supplier would not provide such documentation. So we had to apply some reverse engineering.

Progress is a relational database per definition. Not an SQL-database, but still a relational database. Now, we all know that relational databases are the most successful databases. As hierarchical-and network-databases proved to be only successful in some particular circumstances, the relational database moved in and captured the market. As we speak almost all the databases in the world are relational databases. However, successful does not mean "most technically sound concept". All together there is a long list of shortcomings of the relational database. There is one in particular that comes to mind in this case. *The relations between the entities in the database are not stored in the database!* The mechanism is simple. I've got an order containing a customer-number. When I find the customer identified by the same customer-number I have a match and I know who the customer for this order is. Unfortunately, the "When I find..." Could easily be "If I find..." because the relational database will not guard the relation between this customer and order for me. As a matter of fact: the database is not even aware that there is a relation between customers and orders. If I want to make sure that for each order there is a customer available I will have to code this myself.

As a result of this a database schema is nothing more than a bunch of entities with no relations between them. As we have to create these 'links' between data in different tables ourselves it is essential to have a good data model. Reverse-engineering a data model from a Progress database is simple. For each file where not file-name begins "_" provides me with a list of all the entities. For the relations between these entities Progress even provides a simple tool. Unfortunately, such tools will come up with a lot of relations that are not relevant or worse relations I do not want in there. This gets even worse if you apply the rules for foreign-keys --from an article I wrote about in Progressions #48.

An example:

Here we have a simple situation. We are registering in our database companies, for each company in which regions they are active, for each region on which locations they have stores and for each location which cash registers they have. This can basically be implemented in two ways.

```
a:
company (company-id, company-name, ...)
region (company-id, region-id, region-desc,
...)
```



WHITE STAR BOOKS AND PUBLICATIONS



MAKING GOOD PROGRESS: An absolutely comprehensive book on Progress. Almost 1,000 pages and 35 chapters on everything you need to know about the application development process --from design to tuning.

\$130 (Priority ship included)

\$144 USD Canada (air ship)

\$160 USD International (air ship)

DISASTER PREVENTION: A booklet by Adam Backman on what you need to know to guarantee your system's integrity. Insights on RAID, replication, on-line back ups and much more. There's a lot more to backing up your system than just putting a tape in the drive.

\$24 US

\$27 USD Canada/International

HIGH PERFORMANCE REPORTS: This guide will lead you through the steps of analyzing user requests and designing reports that will be efficient and maintainable; how to analyze and rework slow reports to speed them up.

\$23 US

\$26 USD Canada/International

CODE SMARTER, PROGRAMMING FOR THE FUTURE: All of the features for "Future Proof" programming and how to use them: the preprocessor, internal procedures, dynamic functions, publish/subscribe, super procedures, dynamic buffers, widgets and browsers . . .

\$26 US

\$29 USD Canada/International

WHAT DO I DO NOW? An introduction to the principles of event-driven programming and the many new features and concepts of V7/V8, this booklet will get you up to speed with the preprocessor, schema and UI triggers, internal procedures, the query command and new issues in scoping and transactions.

\$25 US

\$28 USD Canada/International

PROGRESSIONS: NOW IN ELECTRONIC FORMAT!!! A technical journal by Progress programmers, Progressions covers a broad range of subjects for all levels of expertise. It features articles (with code) on such topics as report tricks, database design, undocumented commands, editorials, product reviews, as well as articles on Versions through 8.2 for both GUI and character mode. 1 year subscription is 6 issues. Back issues are available: US. \$7.00 each Canada/International US \$8.00.

PRICING FOR ALL: US, CANADA, INTERNATIONAL

\$60.00 USD/one year

VERSION 8/9 FOR \$89: Can't find Version 8/9 information? It's right at your fingertips! We have 11 back issues of Progressions—our V8/V9 Package. Each issue contains one or more articles on useful Version 8/9 topics: insights, tips, the editor, dynamic menus, launching windows apps, triggers, benchmarks, strip menus, application security, designing a GUI interface, internal procedures and much more. Progressions V8/V9 Package -11 issues

US \$89 USD

\$99 USD Canada/International

We accept Visa, MasterCard, AMEX (Include Expiration Date) fax credit card orders 970.963.3548 or email to info@wss.com. We also accept company checks drawn on a US Bank. Sorry, no purchase orders. Shipping and Handling included.

Checks must be in U.S. Dollars payable to White Star Software. CA residents add 7.75%

White Star Software

970.963.3545

info@wss.com

www.wss.com

970.963.3548 FAX

```
location (company-id, region-id, location-id, location-desc, ....)
cash register(company-id, region-id, location-id, register-id, ....)
```

b:

```
company (company-id, company-name, ....)
region (region-id, company-id, region-desc, ....)
location (location-id, region-id, location-desc, ....)
cash register(register-id, location-id, register-desc, ....)
```

In this example in situation **b** each register-id is unique for the whole system. Therefore a simple reference to location will suffice, because each location-id is unique for the whole system, etc.

A tool scanning for relations in the database would find in situation **b**:

- relation between *company* and *region* through company-id
- relation between *region* and *location* through region-id
- relation between *location* and *cash register* through location-id

No problem. Each of these relations is a relevant one. The disadvantage of this implementation is the amount of finds you will need. Suppose I've just found a cash register record and I want to know to which company this register belongs. I'll have to find the related location first, then the related region and finally the related company. This is not a problem in situation **a**. Here I can do a straight find from cash register to company. Nicely done. However, if we were to use the same scan for situation **a** as we used for situation **b** we would get this result:

- relation between *company* and *region* through company-id
- relation between *company* and *location* through company-id
- relation between *company* and *cash register* through company-id
- relation between *region* and *location* through company-id and region-id
- relation between *region* and *cash register*

through company-id and region-id

- relation between *location* and *cash register* through company-id, region-id and location-id.

Just four simple entities and already six relations. And these are even existing relations.

So if a scan for relations in a relational database comes up with a bunch of relations you will have to validate these relations. Here a simple True or False will not do. In our example the relation between *company* and *cash register* is an existing one, but not as relevant as the relation between *company* and *region*. So you will have

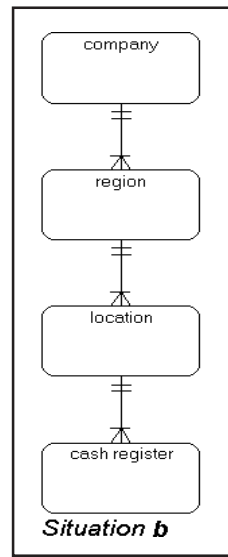
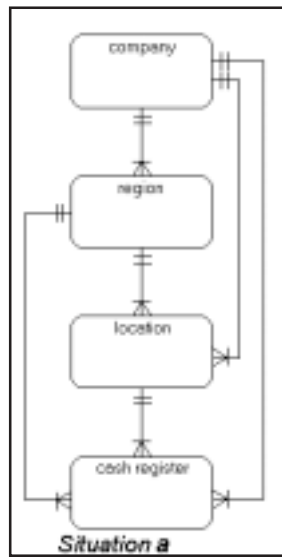
to qualify them in at least one of three categories: non-existing, existing but not relevant and existing and relevant.

Now what's this talk about non-existing relations? Do they exist? Yes, non-existing relations can and do exist. Confused? Let me try to explain.

In the article in Progressions # 48 I argued that it would be perfectly all right to have a reference to an attribute in another table without this attribute having to be (part of) the key, or even alternative key. Now, if the person

who built your relations scanner was smart enough to be aware of this, or stupid enough to just scan for similar attributes your scan could find a lot of non-existing relations.

Just imagine a simple system to register customers and suppliers. Nothing fancy. Now, a scanner could come up with multiple relations between the customer table and the supplier table based on the fact that both tables have a country-id attribute, a currency-id attribute and a language-id attribute. Imagine all our US-based customers are related to all our US-based suppliers because they happen to be in the US. Both our US-based customers and our US-based suppliers have a relation with all our Hong Kong-based customers because they all speak English and relations with the Russian mob because they all prefer dollars. I don't think so! Non-existing relations. Make sure you take them out of the model before presenting this to your users.



Just a final warning. The scanner which could find all these wonderful non-existing relations in the database could easily have missed a few existing and relevant ones as well. Remember that this scanner will have to work with attribute names. So, if for some reason the developers worked with country-id in the customer table, but with country_id in the country table (please note the subtle difference between a hyphen and an underscore) the scanner will miss it all together.

Ok, every now and then you will have to apply some reverse engineering. This simply can't be helped. However, please do not rely solely on tools to do the work for you. These tools can at the same time be too smart and too stupid to do it right.

Jop Kluis

Jop has been around in the Progress world for over 10 years now. Over the years he has published a wide variety of articles on software development. As a designer and project manager, database design has always had his warm attention. "The database is often the heart of our systems. If we do not have the time to design these properly, where are we going to find the time to correct this later?"

Jop is Project manager with Propago and boardmember of the PUG Netherlands. He welcomes any thoughts about database design. E-mail: j.kluis@propago.nl



 <p style="text-align: center;">Visual Printing and Enhanced Reports</p> <ul style="list-style-type: none"> ● uses Windows printer drivers ● data processing with 4GL ● incl. tool for graphic design ● no runtime licence cost ● supports bmp/jpg images ● embedding rtf-texts (font,...) ● generates pdf-files (email) ● supports WebSpeed /-Client 	 <p style="text-align: center;">CASE-Extension for the Data Dictionary</p> <ul style="list-style-type: none"> ● view Progress-DB structures ● create/update DBs directly ● reengineer Progress-DBs ● read/write Progress df-files ● compare/maintain versions ● supports areas ● autogenerates references ● print resizable ER-Diagrams ● copy/paste tables or fields 	 <p style="text-align: center;">Project management, compiling, analyzing</p> <ul style="list-style-type: none"> ● compiles project file lists ● Includes compiler server ● also compiles in char-mode ● uses different Progress vers. ● compiles for different OS ● contains xref-analyze frontend ● shows db structure & content ● keeps track of project errors 	 <p style="text-align: center;">logfile analyzing, system checking</p> <ul style="list-style-type: none"> ● usable with Windows or UNIX ● check activity of NS/DB/WS/httpd ● analyze logfiles of NS/DB/WS ● check drive space, space in DB ● execute self defined scripts ● analyze self defined logfiles ● get notified by e-mail, http ● or screen output
---	--	--	---



IAP GmbH • Markenstrasse 8 • D-22767 Hamburg • Germany
Tel. +49 40 30 89 03 - 0 • Fax +49 40 30 89 03 - 10
email: info@tools4progress.com



T4P
TOOLS FOR PROGRESS

Information and free testversions at www.tools4progress.com

They're Not Toys: Using XML And Sockets To Do Real Work by Steven Lichtenberg

Two of the most powerful features of the newest versions of Progress are the ability to directly manipulate sockets and the built-in XML parser. These tools allow users to manipulate a wide variety of environments and truly open the world of application-to-application development as well as business-to-business processes.

Beyond some simple demos though, many developers have never seen just how useful these tools can be. In truth, when you think of all the current buzzwords, these are the tools that you will use to implement them.

A Word About Sockets

First, what are sockets and why do we need them? A socket is a connection to a procedure that is listening to a specific TCP/IP port. TCP/IP ports are given arbitrary numeric designations that allow administrators to understand what is running where on their system. The more well known ports are by definition and convention given common numbers so all developers will be writing to a standard environment. Common ports are 80 for web browser requests, 25 for SMTP mail requests (sending mail), 110 for POP3 requests (receiving mail) and many others. The "standard" ports are defined in a file called "services". On Unix systems, this file is in the /etc directory. On Windows systems it is located in c:\winnt\system32\drivers\etc. The services file can, and most likely will, have additional ports defined in addition to the "standards". In order to connect to a Progress database in a client/server environment, you need to designate a TCP/IP port for the connection.

In the Unix world the procedures that listen on these ports are called "daemons". On Windows, the equivalent processes are called "services". There are similarities and differences to these background routines but for the purposes of this discussion, they are roughly equivalent. The background process sits and waits for an outside process to request information then sends the information along the previously defined pipe to the recipient.

Using the Socket

Once the socket is defined, we have to be able to communicate through it. This is accomplished via the routine sockets.p, which will create the connection and return the handle to that connection. Sockets.p is run persistently and allows for opening and closing any socket connection as required.

```

/*****
* Program: sockets.p
* Author : Steven Lichtenberg
* Date   : 01/25/2001
* Purpose: General sockets APIs to be run
*         persistently to create and delete sockets
* Tables Used: none.
* Modified:
*****/

PROCEDURE GetSocketHandle :
  DEF OUTPUT PARAMETER op_hMySuperProcedure
  AS HANDLE NO-UNDO.
  ASSIGN op_hMySuperProcedure =
    THIS-PROCEDURE:HANDLE.
END PROCEDURE.

SUBSCRIBE TO "GetSocketHandle":U ANYWHERE.

function opensock returns handle
  (input purl as char,
   input pport as char):

  define variable hopener as handle no-
  undo.
  define variable ret      as log      no-
  undo.
  define variable connstr as char      no-
  undo.

  create socket hopener.
  connstr = "-H " + purl + " -S " +
  pport.
  ret = hopener:connect(connstr).
  if ret then return hopener.
  else return ?.
  delete object hopener no-error.
end function. /* opensock */

function closesock returns logical(
  input hclose as handle):
  return hclose:disconnect().
  delete object hclose no-error.
end. /* function closesock */

```

Opensock is the routine to establish the connection and closesock is the routine to close the connection. The subscribe option at the top of the routine is a mechanism to communicate to other routines whether the connection is available or if the SUPER needs to be instantiated.

Using the Established Connection

Now that the connection is established, we need to get it to do something useful for us. In this example, we will be creating an XML document and sending it to the VSIFax server. As part of the VSIFax suite, an XML server is included. While this is designed to be a manual transfer of XML documents, a direct socket connection is not only possible but works quite well.

Details on the XML-F specification are available from the VSIFax web page <http://www.esker.com>

These specifications detail the DTD for sending and receiving fax information to the XML-F portion of the VSIFax server. The `sendvsixml.p` routine will build the XML document based on that specification.

Documents sent to VSIFax need to be either text documents which can be added as a text node directly to the XML document or an image file. Image files are typically of type `.tiff` or `.pcl`. These image files will need to be base64 encoded in order to attach them as a text node on the document. A simple routine for encoding base64 documents is `base64encode.p` which is available from The FreeFramework Project at <http://www.freeframework.org>. This routine is the basis for creating a text node from a binary file.

```

/*****
* Program: sendvsixml.p.
* Author : Steven Lichtenberg
* Date   : 01/02/2002
* Purpose: Background routine to generate
an XML fax request based on a particular
document number and update the status
record to show that it has been sent
* Tables Used: None
* Modified:
*****/

function opensock returns handle
  (input purl as char,
   input pport as char) in super.
function closesock returns logical
  (input hclose as handle) in super.

function new-child-node returns handle
  (input documenthandle as handle,
   input parenthandle as handle,
   input nodename as char,
   input nodevalue as char) forward.
function new-node returns handle
  (input documenthandle as handle,
   input parenthandle as handle,
   input nodename as char) forward.
def input parameter hostid as char no-undo.
/* URL to vsifax server */
def input parameter hostport as char no-
undo. /* generally 2996 but is modifiable
*/
def input parameter faxno as char no-undo.
/* telephone number for fax */
def input parameter faxto as char no-undo.
/* who to send to */
def input parameter infile as char no-undo.
/* name of file to send */
def input parameter refno as char no-undo.

```

```

/* reference number back to database */
def input parameter usrid as char no-undo.
/* name of the user sending the fax */
def output parameter osuccess as logical
init yes no-undo.

def var hsock      as handle no-undo.
def var hdefs      as handle no-undo.
def var hfunc      as handle no-undo.
def var hxdoc      as handle no-undo.
def var icnt       as int    no-undo.
def var hrec       as handle no-undo.
def var mdocbuf    as memptr no-undo.
def var callno     as char   no-undo.
def var result     as char   no-undo.
def var reason     as char   no-undo.

```

```

PUBLISH "GetSocketHandle":U (OUTPUT hfunc).
IF NOT VALID-HANDLE(hfunc) THEN
do:
  run sockets.p persistent set hfunc.
  session:add-super-procedure(hfunc).
end. /* if not valid-handle(hfunc) */

/* open the socket and pass each XML docu-
ment to the xml server */

```

First we run `mkxdoc` to create the XML document. This is a fax-submit document that will then be understood by the VSIFax server. `mkxdoc` consists of a series of calls to the functions `new-node` and `new-child-node`. These functions come directly from the XML toolkit by Geoff Crawford and were published in previous editions of *Progressions*.

```

run mkxdoc(input faxno,
           input infile,
           input refno,
           input usrid,
           input faxto,
           output mdocbuf).

/***** this next part seems silly
and inefficient here but the VSifax XML
daemon issues a server disconnect after
each response so we need to reconnect for
each iteration of the temp table */

hsock = opensock(hostid,hostport).
if not hsock:connected() then do:
  osuccess = no.
  return "Can not open VSIFAX Socket".
end. /* if not hsock:connected() */

```

The `write(command)` below sends the fax-submit document we just created across the socket to the fax server. We then wait for a submit-response document to be sent.

```

hsock:write(mdocbuf,1,get-size(mdocbuf)).

```

Since we are immediately looking for a response and this routine will run in batch mode, we do not use a read-response procedure or a wait-for. Just run the read routine directly and grab the information coming back. This avoids the issues of blocking for input when there is no terminal defined for the process. This can be accomplished using a PROCESS-EVENTS command as well but this way is simpler when there is always supposed to be a response.

Mkstatus obtains the status document and calls mkxmlstat which parses it up from the XML.

```
run mkstatus(input refno,
             input hsock,
             output result,
             output reason) no-error.
if error-status:get-message(1) <> "" then
do:
  osuccess = no.
  return " Error in status update " +
    error-status:get-message(1).
end. /* if error-status:error */
```

Now close the socket and report any errors in processing.

```
if hsock:connected() then closesock(hsock)
no-error.

if reason <> "normal" then do:
  osuccess = no.
  return " End of processing error - "
    + result + reason.
end. /* if reason <> "normal" then */
osuccess = yes.
Return.
```

Here is the mkxdoc internal procedure.

```
/* build the XML document *****/
procedure mkxdoc:
  def input parameter faxno as char no-
  undo.
  def input parameter infile as char no-
  undo.
  def input parameter refno as char no-
  undo.
  def input parameter usr as char no-undo.
  def input parameter faxto as char no-
  undo.
  def output param docbuf as memptr no-
  undo.
  def var faxnm as char no-undo.
  def var encodefil as memptr no-
  undo.
  def var filstring as char no-undo.

  def var hdoc as handle no-undo.
  def var hroot as handle no-undo.
```

```
  def var haccnt as handle no-undo.
  def var hnotify as handle no-undo.
  def var hsender as handle no-undo.
  def var hrcpt as handle no-undo.
  def var hsubj as handle no-undo.
  def var hcontent as handle no-undo.
  def var hbody as handle no-undo.
  def var hcommand as handle no-undo.
  def var hfax as handle no-undo.
  def var htext as handle no-undo.
  def var filcnt as int no-undo.
  def var icnt as int no-undo.
  def var isok as log no-undo.
  def var vcompany as char init
    "My company name." no-undo.
/* change company to whatever you need */
faxnm = "faxuser".

/* Create all of the xnodes and manage
their handles. */

  create x-document hdoc.
  create x-noderef hroot.
  create x-noderef haccnt.
  create x-noderef hnotify.
  create x-noderef hsender.
  create x-noderef hrcpt.
  create x-noderef hsubj.
  create x-noderef hcontent.
  create x-noderef hcommand.
  create x-noderef htext.
  create x-noderef hfax.
  create x-noderef hbody.
/* root node */
  hroot = new-node(hdoc,hdoc,"fax-sub-
mit").
/* set attributes of root element */
  hroot:set-
attribute("resolution","fine").
  hroot:set-attribute("coversheet","no").
/* cover pages are created as part of the
document in this case, change to yes if
VSifax created cover pages are necessary */
/* account node */
  haccnt = new-
node(hdoc,hroot,"account").
/* id = vsifax user ID child of
account node */
/* substitute the name of your fax user
as defined in VSIFAX config */
  new-child-node(hdoc,haccnt,"id",faxnm).
/* change to your vsifax account */
/* email address of sender */
  new-child-node(hdoc,haccnt,
"mail-address",usr + "mycompany.com").
/* change as required */
/* email notifications set to never to
allow for more control we will manage noti-
fications of failure so we can give the
user more information than is in the stan-
dard VSIFAX notifications */
```

```

hnotify = new-child-node(hdoc,
hroot,"email-notification",usr +
"@mycompany.com").
hnotify:set-attribute("when","never").
/* part of the process is to generate a
more meaningful email, so we do not want
VSIFAX to generate the failure notices */
/* sender information */
hsender = new-
node(hdoc,hroot,"sender").
/* personal name (name of the sender */
new-child-node(hdoc,hsender,
"personal-name",username(usr)).
/* company name for the sender */
new-child-node(hdoc,hsender,
"company-name",vcompany).
/* fax number of sender */
new-child-node(hdoc,hsender,
"fax-number","").
/* Voice number of sender */
new-child-node(hdoc,hsender,
"voice-number","").
/* recipient information node */
hrcpt = new-node
(hdoc,hroot,"recipient").
/* personal name */
new-child-node(hdoc,hrcpt,
"personal-name",faxto).
/* fax number attached to recipient */
hfax = new-node(hdoc,hrcpt,
"canonical-fax").
if length(faxno) > 7 then do:
/* area code */
new-child-node(hdoc,hfax,
"area-code",substr(faxno,1,3)).
/* phone number */
new-child-node(hdoc,hfax,
"local-number",substr(faxno,4)).
end. /* if length(faxno) > 7 then */
else do:
new-child-node(hdoc,hfax,
"local-number",faxno).
end. /* else */
hsubj = new-child-node(hdoc,hroot,
"subject",refno).
hcontent = new-node(hdoc,hroot,
"content").
/* if you are using plain text as the
body, uncomment these lines */
/* hbody = new-child-ode(hdoc,hcontent,
"body","this is a test of the fax via XML
system").hbody:set-attribute("filename",
"inline.txt"). */

```

Create a comma delimited list of files to send and encode them

```

do filcnt = 1 to num-entries(infile):
if entry(filcnt,infile) = "" then
leave.

```

```

/* doencode will base64 encode the docu-
ment to be faxed and will return a memptr.
The memptr will need to be parsed up into
chunks until 9.1d when you will be able to
directly assign a memptr to a text at-
tribute of an XML document. */

```

```

run doencode(input entry(filcnt,infile),
output encodefil),
hbody = new-node(hdoc,hcontent,
"body").
do icnt = 1 to get-size(encodefil) by
76:
/* each line of the base64 encoded docu-
ment is 76 chars long. Testing shows the
encoding gets garbled if you pass more than
that at a time to the Xdoc node */
if icnt > 1 then icnt = icnt + 1.
hdoc:create-node(htext,"","TEXT").
hbody:append-child(htext).
htext:node-value = get-string
(encodefil,icnt,76).
if icnt > 1 then icnt = icnt - 1.
end. /* do icnt = 1 to get-size */

/* set transfer encoding to base64 so that
documents can be sent directly. If text
files will be used as well then modify the
code to exlude the content-type and trans-
fer-encoding attributes. All document types
that can be used by vsifax are acceptable.
These include pcl, tiff, text etc. */
hbody:set-attribute(
"filename",entry(1,infile)).
hbody:set-attribute(
"content-transfer-encoding","base64").
hbody:set-attribute(
"content-type","application/octet-
stream").
end. /* do icnt = 1 to num-entries */

/* this is how we will identify the
information back to the database */
hcommand = new-child-node(
hdoc,hroot,"command-reference",refno).
hdoc:SAVE("file","test1.xml").

```

```

/* complete the document and save it to a
MEMPTR variable */
isok = hdoc:SAVE("memptr",docbuf) no-
error.

```

```

/* Clean up */
delete object hroot. delete object
hacnt.
delete object hnotify. delete object
hsender.
delete object hrcpt. delete object hsubj.
delete object hcontent.
delete object hcommand.
delete object htext. delete object

```

```
hfax.
  delete object hbody.  delete object
hdoc.
end procedure. /* mkxdoc */
```

Doencode will encode the file and using the file-info handle will load it into a memptr variable.

```
procedure doencode:
  def input parameter infile as char no-
  undo.
  def output param encodefil as memptr no-
  undo.

  def var encodedirectory as char no-undo.
  def var localfile      as char no-undo.
  assign
    encodedirectory = session:temp-direc-
  tory
    localfile = encodedirectory + "/" +
    infile + ".enc".
  run base64encode.p(input infile,
    input localfile).

  file-info:file-name = localfile.
  input from value(file-info:file-name)
    binary no-map no-convert.
  set-size(encodefil) = file-info:file-
  size
    + 1.
  import encodefil.
  input close.
  os-delete value(file-info:file-name).
  return.
end procedure. /* doencode */
```

Finally, we receive another XML document from the server. We then need to determine whether the fax was submitted properly or if there was an error in the submission that will force the document to be resent.

```
procedure getxmlstat:
  def input  param memstat as memptr no-
  undo.
  def input  param refno   as char no-undo.
  def output param result  as char no-undo.
  def output param reason  as char  no-
  undo.

  def var xdoc      as handle no-undo.
  def var hroot    as handle no-undo.
  def var hnode    as handle no-undo.
  def var hchild   as handle no-undo.
  def var htext    as handle no-undo.
  def var i        as int    no-undo.
  def var j        as int    no-undo.
  def var invno    as char  no-undo.
  def var regresult as char  no-undo.
  def var vfxid    as char  no-undo.
  def var isok     as log   no-undo.
```

```
  create x-document xdoc.
  create x-noderef hroot.
  create x-noderef hnode.
  create x-noderef htext.

  assign  xdoc:encoding = "iso8859-1"
         isok =
         xdoc:load("memptr",memstat,false) no-
  error.
         isok =
         xdoc:save("file","submitresp.xml").
         isok =
         xdoc:get-document-element(hroot) no-error.
  node-block:
  do i = 1 to hroot:num-children:
    isok = hroot:get-child(hnode,i) no-
  error.
    /* if not isok then return. */
    case hnode:name:
    when "request-results" then do:
      regresult = hnode:get-attribute(
        "status").
      if regresult = "Normal" then
        assign
          result = "Fax Accepted for Deliv-
  ery"
          reason = "Normal".
      else do: /* regresult failed */
        hnode:get-child(htext,1) no-
  error.
        assign
          result = "Status Request Failed -
  "
          reason = if htext:node-value
            <> "" then
              htext:node-value else
                "Can not determine the reason"
          no-error.
          return.
        end. /* if regresult <> "normal" */
      end. /* hnode:name = request-results
  */

    when "service-reference" then do:
      isok = hnode:get-child(htext,1) no-
  error.
      /* if not isok then next. */
      /* skip null values */
      vfxid = htext:node-value.
      end. /* if hnode:name = "full-status"
  */

    when "command-reference" then do:
      hnode:get-child(htext,1).
      invno = htext:node-value.
      end. /* when "application-reference"
  */

  end case. /* hnode:name */
end. /* do i = 1 to hroot:num-children
*/
```

The call to `updaterec.p` here is to update the database with the status of the fax submission. If this is not

necessary, it can be omitted.

```
/* for our purposes, we update the
database here modify/remove as required */
if invno = "" then invno = refno.
run updaterec.p(input "{&filem}",
input "{&filem}.callno = ~'" + refno +
"~'" and {&filem}.fxresult = ~'~'",
input "{&updflds}",
input result + "|" + reason + "|" +
vfxid).
/* end of database updates */
end procedure. /* getxmlstat */
```

```
procedure mkstatus:
def input param refno as char no-undo.
def input param hsock as handle no-undo.
def output param result as char no-undo.
def output param reason as char no-undo.

def var isok as log no-undo.
def var vlength as int no-undo.
def var i as int no-undo.
def var respbuf as memptr no-undo.

set-size(respbuf) = 2056.
i = 1.
do while hsock:get-bytes-available() <>
0:
if i mod 1000 = 0 then pause 1.
assign i = i + 1.
if i = 10000 then leave.
end.
vlength = hsock:get-bytes-available().
isok = hsock:read(respbuf,1,2056,1).
if isok then run getxmlstat(
input respbuf,
input refno,
output result,
output reason)
no-error.
if error-status:error then return
error-status:get-message(1).
closesock(hsock).
end procedure. /* mkstatus */
```

```
/****** Functions *****/
/* these functions came directly from
Geoff Crawfords XML toolkit and appeared in
Progressions Magazine. For information
contact Geoff at geoff@innov8cs.com */
function new-child-node returns handle(
input documenthandle as handle,
input parenthandle as handle,
input nodename as char,
input nodevalue as char):
def var newnodehandle as handle no-undo.
def var newtexthandle as handle no-undo.

create x-noderef newnodehandle.
create x-noderef newtexthandle.
```

```
documenthandle:create-node(
newnodehandle,nodename,"ELEMENT").
parenthandle:append-
child(newnodehandle).

documenthandle:create-node(
newtexthandle,"","TEXT").
newtexthandle:node-value = nodevalue.
newnodehandle:append-child(
newtexthandle).

return newnodehandle.
delete object newnodehandle.
delete object newtexthandle.
end function. /* new-child-node */
function new-node returns handle(
input documenthandle as handle,
input parenthandle as handle,
input nodename as char):

def var newnodehandle as handle no-
undo.
create x-noderef newnodehandle.

documenthandle:create-node(
newnodehandle,nodename,"ELEMENT").
parenthandle:append-
child(newnodehandle).
return newnodehandle.
delete object newnodehandle.
end function. /* new-node */
```

All these routines together provide a very powerful method for managing fax documents and show how useful both XML and sockets can be in real life applications.

They are a wonderful addition to the language and to the tools available to the Progress programmer. With a little imagination, there is almost no limit to what these tools can allow you to accomplish.

Steve Lichtenberg

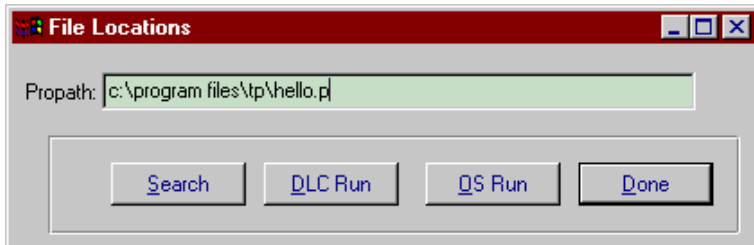
Steve started working in Progress with Version 6.2 in 1992. He has been working almost exclusively on Webspeed projects for the last 4 years and still is excited by the possibilities of web-based computing. After a stint in consulting, he is currently doing in-house Webspeed development at Safemasters, a major lock and safe service company. He can be reached at slichten@safemasters.com

Confused by Quotes? by John Campbell

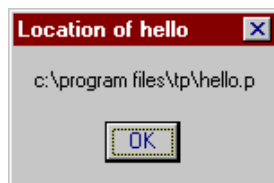
With the introduction of long file names and directory names, Microsoft finally was able to do something which Apple could do in 1978: namely allow for files longer than 8 characters, and with spaces in their names. The problem is, for us, it introduced more problems than it solved. Dealing with this issue can cause you immense headaches as a Progress programmer if you don't know what the problems – and the solutions – are.

We ran into the problem when writing a simple install routine. If you want to mix Progress r-code and operating system commands (such as `prodb` to create a new database), and you're working in the same directory structure, long file names with spaces can be a real headache.

First, if you have a directory name with a space in it, how does Progress react? How does the search command or the run command deal with this change? Let's take a look. We have constructed a program to demonstrate the options.



First of all, we'll put two files into a directory named `C:\program files\tp`. One is named `hello.p`, so we can run it from Progress, and the other is `hello.bat`, which we would like to run from the operating system. If we enter the file name into the field in the above screen, and do a



search on the value of the field, we get a valid result.

The code is simple, unadulterated Progress code, which has worked since the beginning of time:

```
assign SearchPath.
```

```
message search(SearchPath).
```

And likewise, if we run the value of the program we will also get a result. The content of `hello.p` is simple:

```
message "Found this file "  
string(time, "HH:MM:SS").
```

And the content of the DLC run button is also universal Progress code:

```
assign SearchPath.  
run value(SearchPath).
```

So far, we haven't learned anything new. But now comes the difficulty. What if we want to run an os-command statement such as an executable or a batch file? One of the things you learn very quickly is that since Microsoft used spaces to delimit commands from parameters, when they put spaces in their directory names, an operating system command gets confused by those spaces. Consequently, any DOS based command has to have quotes around it. For example, take a look at the following code for `hello.bat`:

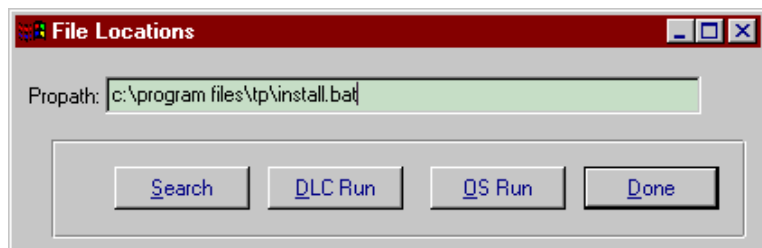
```
C:\>program files\tp\hello  
The name specified is not recognized as an  
internal or external command, oper-  
able program or batch file.  
  
C:\>"program files\tp\hello"  
  
C:\>echo hello, world  
hello world  
  
c:\>
```

First is the command with no quotes, and then the command with quotes.

This is a clue to what will happen when you start to invoke DOS commands from within Progress. Let's look at the Progress code to invoke a command.

```
assign SearchPath.  
os-command value(SearchPath).
```

Normally, you would invoke an os command in the silent mode, but then when it fails, you don't get to see the results. Not surprisingly, if you type in the command as shown, it does not execute.



OK, well, we know what to do: put quotes around the command: “C:\program files\tp\install.bat”. That’s simple enough. But consider this scenario: you don’t want the file name hard coded, you simply want to enter the value of the path name into propath, and then execute a mix of Progress programs and OS commands using this path. Which one do you use? The one without quotes, or the one with? Well, if you put quotes around a command or file name, and try to execute a Progress search statement, you will get a ? as the result. Try to run it, and it will fail. Is this Progress’ problem, or Microsoft’s? Progress has done the right work to mask the differences between directories with spaces and those without as far as Progress commands are concerned. But as soon as we insert quotes in the command, Progress will not decide to strip them out.

So we have a dilemma: do we put the entry with quotes, or not? Here are the rules:

1. If the value is a propath entry, then you cannot have quotes.
2. If the value is for the “search” or “run value” command, you cannot have quotes.
3. If the operating system directory has spaces in it, you must have quotes around the command.
4. If the OS directory does NOT have spaces in it, you cannot have quotes in the command.

So how do you handle this? It is particularly frustrating if you are allowing the user to enter a path, and will use it for multiple purposes. Do you instruct the user to put quotes, or not? The best thing to do is parse what they’ve entered, and use the result depending on the Progress command you’re invoking. Your logic will go something like this:

Is the entry to be used in propath or for “search” or “run value” commands? Check to see if it begins or ends with quotes, and strip them off.

Is the entry to be used in a “DOS” or “os-command”

invocation? Check to see if the entry has spaces in it. If it does, does it begin and end with a quote? If not, add them to the command before you execute it. Here goes:

1. Propath:

```
assign SearchPath
propath =
replace(searchpath, '""', '') + ", " + propath.
```

This is a nice use of the replace command. It says: take all instances of the double quote and replace it with nothing (not even a blank space). Note that this will not fail if the user did not put quotes in the SearchPath.

2. Search or run value

Here is a slightly different approach to the same kind of problem.

```
define variable WorkPath as char.
assign WorkPath =
replace(SearchPath:screen-value, '""', '').

message search(Workpath).
or
run value(WorkPath).
```

3. DOS or os-command

This gets a little trickier, since the path might contain quotes and no spaces, or no quotes and spaces.

```
define variable WorkPath as char.
WorkPath = SearchPath:screen-value.
/* is there a space and no quotes? Add them */
if index(WorkPath, " ") > 0
and (not ( (WorkPath begins '"')
and
(substring(workpath, length(workpath)) =
'"))
) )
then
WorkPath = '"' + WorkPath + '".

/* Are there quotes and no space? Strip
them */
else
if index(WorkPath, " ") = 0 and
index(WorkPath, '""') > 0
then
WorkPath = replace(WorkPath, '""', '').

os-command value(WorkPath).
```

There you have it. You may not have run into this problem yet, but if you do, it can drive you absolutely crazy unless you know these simple rules, in which case it’s a “piece of cake”.



Table of Contents

Every man is the son of his own works—

--Miguel de Cervantes

Accessing File Summary Information Using COM	Jared Middleton	2
Enhancing the ADM2 'CurrentChanged' Processing	John D. Johnson	6
Relevant Relations	Jop Kluis	12
They're Not Toys: Using XML And Sockets To Do Real Work	Steven Lichtenberg	16
Confused by Quotes?	John Campbell.....	22

GOT HEADACHES?

Do Any of these Describe You?

- ◆ You have a new Progress-based package you're implementing but want some customizations.
- ◆ The users of your existing application are asking for numerous modifications.
- ◆ You want to upgrade; you've made customizations; you don't want to spend a fortune "going back" to vanilla code but you don't want to lose what you've implemented.

Do You Need to:

- ◆ Add or remove some fields in an existing screen, change some labels or defaults, or even rewrite the screen entirely...
- ◆ Add your own tracking screens in the middle of the application flow...
- ◆ Put your own messages, validation or extended business rules into the application...
- ◆ Add fields or indexes to existing tables...
- ◆ Get better rollup information, and get it in real-time...

TailorPro is the solution to all these problems, and many more. It is a Progress-based application which allows you to insert your own application logic anywhere in your current programs. It enables you to add fields not tracked by your application, and perform data entry to them in the "middle" of existing screens. No source code or DB changes required!

TailorPro Saves You Money by:

- ◆ Cutting operating costs in your company by making your users more efficient.
- ◆ Dramatically reducing the implementation costs for custom application components.
- ◆ Eliminating the future costs of upgrading a customized application.

Look into the **TailorPro** features on our web demo: www.wss.com/tailorpro

White Star Software

voice: 970.963.3545

tailorpro@wss.com

fax: 970.963.3548