Deploy VFP and SQL Server Data on Your Handheld

GEN 301
Whil Hentzen

If your customers haven't asked yet, they're going to: "We'd like to use our handhelds as data lookup or capture devices, and synchronize that data with our desktop and server applications." This session will show you how to move VFP and SQL Server data to and from your handheld using VFP as a middle tier, and how to create simple applications on your handheld for use with that data.

I'm a pretty impatient type of individual. One of my favorite lines comes from one of the Addams Family movies, where Wednesday asks for the salt. Morticia reprimands her, asking, "What's the magic word?" And Wednesday, in a dead monotone, replies, "NOW!!!!"

Similarly, I want access to my data all the time. At least the data that I use frequently. And that would be my calendar, my address book, and my To Do list. Unfortunately, the apps that come with PDAs that are supposed to synchronize with your PC are usually dumbed-down versions of the PC tool. For example, the "To Do" list manager for the Palm Pilot that syncs with Outlook's To Do list manager has about a third of the functionality. Considering that Outlook's To Do list manager is fairly lame to begin with, the Palm version is useless for anyone who has more than three things to do on a specific day.

Thus began the quest to "roll my own."

Why a Palm Pilot?

Before I get started, I suppose I should explain why I'm using a Palm Pilot instead of a WinCE machine. Well, originally, I had a WinCE machine. It was a Phillips Nino, and at first glance, looked to be quite the killer PDA. But one of the reasons I bought it was because WinCE supported VB, and I figured that I could write a little VB app for my Nino that would transfer data back and forth with my desktop.

Well, it turns out that there are different versions of WinCE, an important fact that is pointed out in the small print on page 47 of the manual. The hand-held version of WinCE, that runs on PDAs that have a full keyboard, like the Jornada, will use a subset of VB. But the palm-sized version of WinCE that runs on machines like the Phillips Nino and the Casseipia, must be programmed using C.

Yeah, right.

Combine that with the fact that WinCE just wasn't that stable – the Nino would lock up on me about once every two weeks, and the sync mechanism was a bear to install and keep running – and that WinCE machines chew up batteries faster than VB developers press Ctrl-C, well, the Nino is now resting peacefully alongside my Kodak DC-50 digital camera and my AT&T wireless headset in the box of "PC Toys That Didn't Work Too Well."

The Palm Pilot (I have a V) is a wonderful tool. Plenty of memory, half the size of a deck of playing cards; the V needs to be recharged about once a month. And the sync mechanism with Outlook and Day-Timer has worked flawless for half a year now.

Databases for the Palm

I've been investigating various tools for creating Palm PDA databases that purport the capability of syncing with files on a PC. First, I'll mention three inexpensive, shareware tools and provide a brief overview of each. Next I'll describe two professional tools, and how I chose one over the other. Finally, I'll describe, step by step, how to move data from a Fox application to your Palm PDA, and back.

First, I should explain that these tools use the term "database" in a rather "loose" fashion. They are flat file managers – allowing you to define a few columns in a single table, and perhaps a couple of indexes. In each case, they use the term "database" when you and I would use the term "table", and for consistencies sake, I'll repeat that error for the rest of this article. The sophisticated ones allow you to define a type of "lookup", either with hard-coded values, or a reference to a separate lookup file. But E.F. Codd and C.J.

Date aren't going to be awed by any display of relational technology in these tools yet. So don't get your hopes up yet, OK?

All three of these tools reside on your Palm PDA. They each allow you to define and use one or more databases on your PDA in a proprietary file format. Two of these tools also have a related tool that you run on your PC (Windows, Mac, Linux, and yes, even DOS) that converts the proprietary file – after you've sync'd it from your PDA to your PC - to a CSV (comma-separated values) formatted file.

SimpleDB

In some advertising agencies, it's common practice to develop a "throw-away" campaign that is the first to be presented to a picky client. The theory is that the client will automatically not like the first presentation, and so the agency presents work that is purposefully less than their best.

Same thing here. Unless your database needs are really simple, you'd be well-advised to skip by this one right away. I just presented it so that you'd appreciate the others more. On the other hand, if you're looking for a simple database application for a "friend" of yours – a less-than computer literate spouse, parent, child, or business associate, SimpleDB may be just right.

It allows the user to create databases with a customizable heading and up to five fields per database. You can also add free-form notes to each record. It doesn't, yet, allow sorting or searching, or cut and paste between items, and it doesn't have a facility for transferring data between your PDA and your PC.

Simple DB is also currently in beta, but has been for quite a while, so it's reasonably stable, and the README file – the only documentation that comes with the program - has a list of known limitations and bugs.

ThinkDB

ThinkDB has considerably more functionality than SimpleDB, and even has a couple of features that my third selection, JFilePro, does not.

It allows you to create up to 100 simultaneous databases (remember, that means "tables" to you and me) on a device (your PDA), and a database can contain up to 65,000 records. Each database can have up to 36 user-defined fields, and you can identify a database as belonging to a category. This last feature allows you to view a list of only select databases – such as business, personal, athletic training, or however else you might want to group them.

ThinkDB has a large number of data types, including text, integer, long, float, list (a lookup field), date, time, checkbox, expression (similar to a calculated field in a Visual FoxPro report), primary key and external join (similar to a foreign key, in that it points to a primary key in another ThinkDB database.)

ThinkDB has a mini-Form Designer where you define your record entry form. Since it's very possible that you won't be able to display all 36 fields in a ThinkDB database on one "screen" on your PDA, you can define "tabs" in the Form Designer, and display each fields on a specific tab. You can create a dozen different "lists" (ThinkDB's name for what we think of as a Visual FoxPro local view). These lists each have their own set of columns, sort orders, and default filter.

You also have the ability to enter multi-line data, find on either one or multiple fields, and work with a "home page" that lists all of the ThinkDB databases on your PDA. The ThinkDB preferences page allows you choose eight different settings, including how the main screen displays data, whether to confirm before delete, whether to keep the current database sorted while in use, and whether to open the last database used the next time ThinkDB is started.

ThinkDB comes with a Desktop Companion that opens existing databases for viewing and editing, import and export with Access, and do a number of functions along the lines of VFP's MODIFY STRUCTURE facility.

ThinkDB comes with a good-sized on-line help file that has clear descriptions of how to operate the program. The only thing I didn't really like is the wording regarding their registration key setup: "An incorrect Palm username will create an invalid registration key. A new registration key may be issued only if the correct name is similar to the one provided previously, and the process can take several weeks/months."

JFilePro

Probably the granddaddy of Palm database tools, JFilePro has a predecessor, JFile, that is in widespread use. JFilePro was released late in 1999, and its files aren't directly compatible with its ancestor (they can be converted with a utility that is provided with JFilePro.)

JFilePro allows up to 60 databases on a single PDA, 20 character field names, 50 fields per database, and 4000 characters per field. Naturally, these are maximums - the capacity of your Palm PDA may impose stricter limits.

You get four functions, or what they call "views." These include the Main view – a list of all databases currently installed, New/Modify Structure view, the Database view – essentially a Browse, and the Record view, which allows display and editing of a single record.

JFilePro provides most of the same capabilities as ThinkDB, including multiple field types, a robust preferences selection, and standard add/edit/delete functionality. It does not have the same type of "lookup" feature, nor does it allow you to relate multiple databases using keys like ThinkDB does. However, what it lacks in "relationality" it more than makes up in sorting, searching, and filtering.

You can view column totals, control column widths, filter on one or multiple field, via a filter string using a "contains in", "starts with", "not in" or "in range" operator.

The converter application that comes with JFilePro runs on Windows 9x and NT, and allows you to convert a PDB (Palm database) file to CSV format and back. This feature is pretty powerful, using an IFO definition file that allows you to define and alter how data is moved between PDB and CSV formats.

JFilePro comes with two versions of its help file – in both HTML (not .CHM) format and an RTF document that makes good use of Word's outline formatting so you can navigate through it quickly.

Shareware Conclusion

As with many Internet technologies, PDA databases are a story of "two steps forward, five steps back" when it comes to functionality. The price you pay for additional deployment capabilities – being able to take your data with your in you shirt pocket and have it be available instantly (as opposed to a four minute boot up sequence) – is limited functionality – primitive relational ability and small capacities. Let's look at some alternatives.

Alternatives to Shareware

As I've mentioned, there are basically three routes you can take when developing applications for your Palm, because, when you want to transfer data between your desktop PC and a handheld, that's really what you're doing – developing an application. The first route is to go quick and dirty with one of the tools I've just described, and, in some cases, that's plenty good. If you're simply moving personal data from your PC to your handheld and back, a shareware tool that costs $19 or so is probably the best choice, and I covered three of varying sophistication in January.

The third route is to go hard core – writing in C++ (or, believe it or not, Java). But, as Markus Egger says, "Life is too short to write in C++." Seriously, if you're going to become a full time Palm developer, this is the way to go – there are several development environments, including CodeWarrior for C and the KVM for Java (it's a virtual machine for the Palm, and it's the next step up from Java, thus, the "K" moniker), that are good.

But what about the rest of us? Those of us who need a more robust environment than a flat file with a five or 36 field limitation? This middle ground is addressed by several development environments, including Pendragon Forms (originally known as Pilot Forms, produced by Pendragon Software) and Satellite Forms by PUMA Technologies. Pendragon has a number of limitations that, while much more programmable than the three shareware products I just discussed, just didn't do much for me

Satellite Forms, on the other hand, is a pretty darn nice development tool for the Palm OS, and it may well be the only thing you ever need to deploy robust handheld applications on your Palm and keep them synced with a desktop or server-based application back at the home office. If you noticed how I slid the word "server-based" into the previous sentence, you might be wondering, "Just exactly what did he mean by that? Could he have meant…" and, yes, I did. Not only can you set up your Satellite Forms app to sync back with your desktop, you can even create a Satellite Forms application that will synchronize directly with a server application like Oracle. Yes, "WOW!"

An Overview of Satellite Forms

Satellite Forms (SatForms) is a development tool that allows you to create an application that runs on your Palm – with forms, controls, tables, and such. However, it's much more than just that. The SF development package also includes a couple of tools – an ActiveX control and a DLL – that you use in your VFP (or VB, or Access) application to make your application "HotSync-aware."

This is a critical concept, so I'll explain in more detail. The big win with handheld applications is being able to (1) download some stuff from your PC (or some data source – perhaps a company LAN application, a server-based application, like Oracle, or even a web-based application) to your handheld, (2) manipulate that data on your handheld – modifying existing records or adding new and deleting existing records, and (3) taking your handheld back to the office and updating the PC database with that information from your handheld. And, most of all, you want all this to happen automatically when you put your Palm into its cradle and hit the magic Sync button. You don't want to have to write a program and run it while your Palm is cradled in order to manually fetch and dump data.

Thus, SatForms consists of four pieces. The first is an IDE that runs on your PC, as shown in Figure 1.
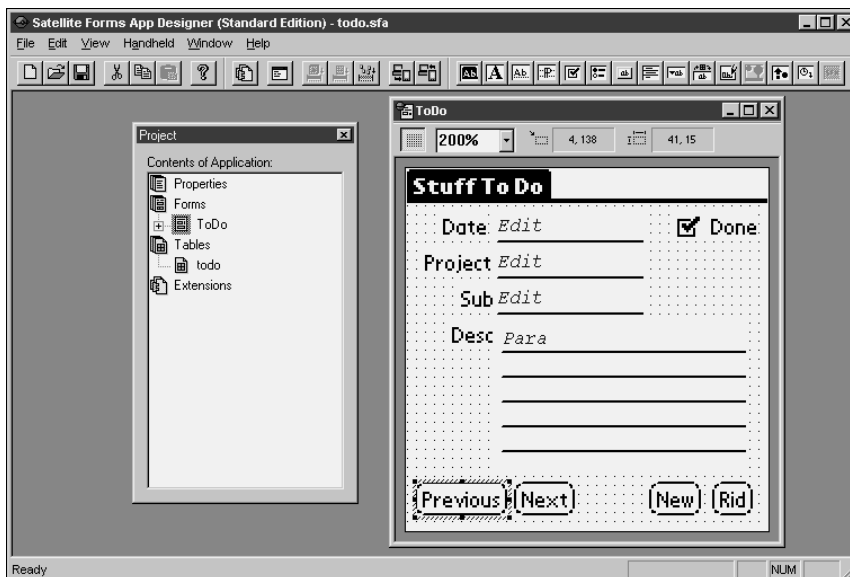

Figure 1. The Satellite Forms IDE.

Sure, it looks a bit simplistic, but, hey, think about what you're developing – a monochrome application running on a 33 MHz processor with a meg of RAM and a 160x160 pixel square display. You don't need a lot of bells and whistles.

The Palm Philosophy

It's probably a good idea to point this out now. The biggest difficulty I've had (other than deciphering the documentation) is getting used to the mindset of developing for the Palm OS. The whole design concept of the Palm was – and still is – centered around a minimalist philosophy. The Palm is not a replacement for your PC, and they have steadfastly resisted the inclination to keep adding stuff to it.

Thus, when you're developing for the Palm, keep this in mind: What is the absolute minimum you need to have on the Palm? What can you dispense with? Get rid of the "nice to have" mentality – cuz you probably don't have room for it. It's like traveling across country in an MG vs. a Cherokee.

Given this frame of mind, don't think that you can't develop a rich interface and rewarding usability. It's just not going to be on the same scale as the application you're developing for PCs.

The IDE does more than allow you to create forms, though. It's also the mechanism you use to "compile" your application and download the application you developed and the tables that are used by your application to your Palm.

The second piece of SatForms is an SDK that you load on your Palm in order to "host" SatForms applications. (You can host multiple SatForms applications on the same Palm, as you probably expected.)

Thus, when you want to run a SatForms application on your Palm, you tap the SF icon, and then select which SF application you want from a list.

The third piece of SatForms is an RDK (Redistribution Development Kit) that you can use if you don't want all of your SatForms apps hosted in the same container. This RDK allows you to create Palm icons for each app and install each of them separately. It's more work than using the SDK, but for some folks, it's more appropriate.

The last piece is an ActiveX control (and a matching DLL for ActiveX-impaired folks). This is the cool part. In order to have VFP exchange data with your handheld automatically, you'll have to have a VFP application running on your desktop before you initialize the HotSync process on your Palm cradle. You drop this ActiveX control on a form that's part of your application, and it automatically detects the execution of a HotSync process.

You use methods of this ActiveX control to perform actions like copying tables between your handheld and your desktop. You can even automatically update the application on the handheld during a HotSync.

It's Not Quite That Easy

The documentation that comes with SatForms is pretty good for what it covers – I'd give it a 9 ½ for most chapters. The one place it falls down is explaining how to use the ActiveX control, which, in my opinion, is the most important piece of the whole thing. It took all of an hour and a half to learn how to write a simple one page form, create a table, connect the form and the table, download the app and the table to my Palm, and run it successfully on the Palm.

On the other hand, it took a couple of weeks to get the ActiveX control to work. It's not that hard – it's just that the doc, the FAQ, the online forums, the archives and the other support mechanisms are, er, bad. The doc is sparse, the relevant files on PUMA website are simplistic to the point of being wrong, and PUMA doesn't seem to have a significant presence on their own message board. There's an OK tutorial that explains how to do this with Access 97, but it misses a couple of key points, and explanation of the Access code is awful.

Once I puzzled it all out, though, it's pretty straightforward, and I've enjoyed working with the product a lot.

What you want in your handheld application

Before I start slinging code ("Oh no, he's not going to talk about *design* first, is he?"), it's a good idea to reflect on what you want in this "VFP Data on your Palm" arena. Each implementation is done differently and makes assumptions. Coming into this new, it's good to state what those assumptions are. First of all, you probably have an application running on your desktop or server. This application is the primary repository for the data that you want to carry around (and manipulate) on your Palm.

Second, at some point, you're going to want to download some (but not all) of this data to your handheld. And you're going to want to do this by popping your handheld into its cradle and pushing a button. You don't want to have to execute menu options or click command buttons on software that's running on your desktop, or tap software icons on your handheld.

Next, you're going to take your handheld out of its cradle and work on it for a while. It seems that construction sites, oilfields, and subway trains seem to be the most popular sites, if you just look at the ads in the popular press, but, truthfully, you could work on it, yes, anywhere, except perhaps the shower.

The type of work you'd do "in the field" includes three types – changing existing records, adding new records, and perhaps deleting existing records – that would require synchronization with your desktop app at some later point.

Finally, you're going to traipse back into the office, pop your handheld into its cradle, push the sync button, and have your handheld's changes moved back to your desktop app.

Don't forget that the data on the desktop app might have been changed in the interim – your administrative assistant (why would you have one in this day and age? I dunno, but it could happen) might have updated your calendar, or the marketing weenies might have updated the master copy of the product database that you're carrying around in your pocket.

Satellite Forms Components

SatForms comes with four components, and I just covered those briefly. The key things you have to remember is that there are two essential pieces – the SatForms IDE, with which you build the app that runs on your handheld, and the SatForms ActiveX control, which you place on your desktop app, and that automatically detects HotSyncs and can run code you write to customize the syncing of data.

Overview of development process

Suppose a copy of SatForms just showed up on your desk one day. What would you do to get your application's data onto your handheld?

First, you'd install the SatForms package on your PC. During the installation process, you'll need to have your cradle connected to your PC, your handheld in your cradle and your synchronization software running. Then all you need to do is follow the instructions. During the installation, the SatForms SDK gets installed on your Palm, and the SatForms ActiveX control gets installed and registered on your PC. Takes less than five minutes.

Next, you'll develop the application that you want to run on your Palm, using the SatForms IDE. This usually takes more than five minutes, but might not take that much longer. It took me perhaps twenty minutes to create a single table, a two page form whose controls mapped to fields in that table, and download it to my Palm V.

The third step is to, if this hasn't already been done, develop the application that's going to house your data on your PC. Many times this will already have done, but it will now need one more module – the one that contains the ActiveX control that waits for a HotSync to occur.

In other words, you need to have an application running on your PC at all times – or at least all the time that a HotSync could occur – so that when you press the Sync button on the cradle, your PC can detect this and handle the data transfer for your app.

This may seem strange at first, but when you think about it, you've already got one of these running – the Palm HotSync Manager that's setting down thar in your System Tray. It's kind of like having an NT service on your machine, ready, lying in wait for a HotSync to occur.

Thus, what you'll often do is either add a module to your existing app or create a separate, stand-alone app that has access to your desktop application's data. You'll place the SatForms ActiveX control on a form in this module/stand-alone application, and add code to selected methods in the ActiveX control to control how you want data synchronized. The amount of code you add depends on how sophisticated your synchronization requirements are. In either case, this module/stand-alone application will be running all the time.

The last two steps are to deploy your apps on both pieces of hardware. On your PC, you have multiple choices. You could go whole hog and create a separate installation package that installs your PC app as a separate program, or just run your VFP app from within the interactive development environment. You could even configure your application to run as an NT service. On the handheld, it's much easier. One of the menu choices in the SatForms IDE to download your app and tables to your handheld. Pretty painless.

Overview of day-to-day use

OK, now that you've got your PC application running and your SatForms application installed on your handheld, it's time to use it. I'm going to assume that you've got some data in your PC application to start with. If you don't, you can skip a couple of steps.

Put your handheld in its cradle, and hit the Sync button. As the sync process runs, your PC app will detect the HotSync, and run the code in the appropriate methods. The data on your PC will be moved down to your handheld as you defined in the code you wrote in the methods of the SatForms ActiveX control. Take your handheld out, and work with it as desired. At the end of the day (or whatever), put your handheld back in its cradle again. Make sure that the PC application is still running, and hit Sync on the cradle.

Details about the SatForms Data Structures

There's actually an intermediate step in this process that I've kept hidden from view to this point. You might be wondering how the data in the tables in a VFP database get moved down to the Palm – surely the SatForms app on you Palm doesn't use DBFs as the data storage mechanism, does it?

In a word, no. Satellite Forms stores data in a proprietary .PDB file format. On of the chores of the SatForms ActiveX control is to convert the data on the PC to the PDB file format. Of course, to do so, it

would seem that the data on the PC would need to be in some sort of standard or recognizable format. Funny enough, Puma Technologies decided upon the dBASE 5 .DBF file format as their standard file format for the PC. Thus, the ActiveX control looks for data in a dBASE 5 .DBF, and converts that data to the PDB format used on the handheld.

This means that, unless you're using dBASE 5 DBFs in your Visual FoxPro app (hee hee), you'll need to move data from your application's data store to dBASE 5 DBFs. The code you write in the SatForms ActiveX control's methods are, in large part, responsible for this chore. Obviously, it's not a big task – not nearly as big as if you were using, say, Access or Paradox, each of which have their own particular "challenges" when it comes to working with DBFs.

So here's what happens underneath the hood. When the SatForms ActiveX control in your PC application detects a HotSync, it will call a method that creates a set of intermediate dBASE 5 DBFs, and then converts data from the application's data store to these DBFs. (Note that I said "create" – if they were already there, they get completely overwritten. More on this in a second.) You write the code in this method. Then the ActiveX control will call a second method that automatically converts those dBASE 5 DBFs to PDB format and transfers the data to the handheld.

On the return trip, when the handheld data is being moved back to the PC, the opposite happens. The ActiveX control detects a HotSync, and calls a method that grabs the PDB data from the handheld, and creates the appropriate dBASE 5 DBFs again. Then the ActiveX control calls a second method (that you put code in) that converts those dBASE 5 DBFs to the data format used by your application.

What this means, of course, is that you can keep your application's data in any format that your app can read – free tables, VFP database tables, or a remote data source that your application accesses through remote views or SPT. And, as I mentioned briefly in last month's article, you don't even need to do this on your own desktop – if you want to spend the money, you can get an "Enterprise version" of SatForms that syncs with a server like Oracle instead of a desktop.

OK, back to this business of the dBASE 5 DBFs getting overwritten. The fundamental problem with synchronizing data between two sources that data could be changed on both sides, and you have to figure out which data is newer, and resolve conflicts when the same data on both sides has changed. The SatForms ActiveX control doesn't do anything about trying to resolve these conflicts – it just assumes the data in the dBASE 5 DBFs is "the data, all the data, and nothing but the data." There is no conflict resolution performed between the DBF and the PDB on the handheld.

Thus, on the front end, it is your responsibility to make sure that the DBF contains exactly the data you want it to contain because it's all going down to the handheld. Then, it's your responsibility to grab the entire DBF and handle any possible conflict between it and the data on your desktop. You do this by writing code in the two methods I mentioned earlier that move data to and from the DBF and your desktop.

But, hey, you're a database developer – so that's the fun part, right?

The SatForms IDE
The Satellite Forms development package is used to create applications that run on your Palm handheld. Let's dig into this rich and flexible environment.

The Satellite Forms IDE, shown in Figure 2, is used for three purposes – to create tables that your Satellite Forms application will access on your PC, to create the actual application (you know, a UI and some business logic) that resides on your handheld, and a mechanism to deploy that application and the related tables to your handheld. There are actually additional functions, like the ability to manage extensions (the Satellite Forms version of ActiveX controls), but we won't get into those here.
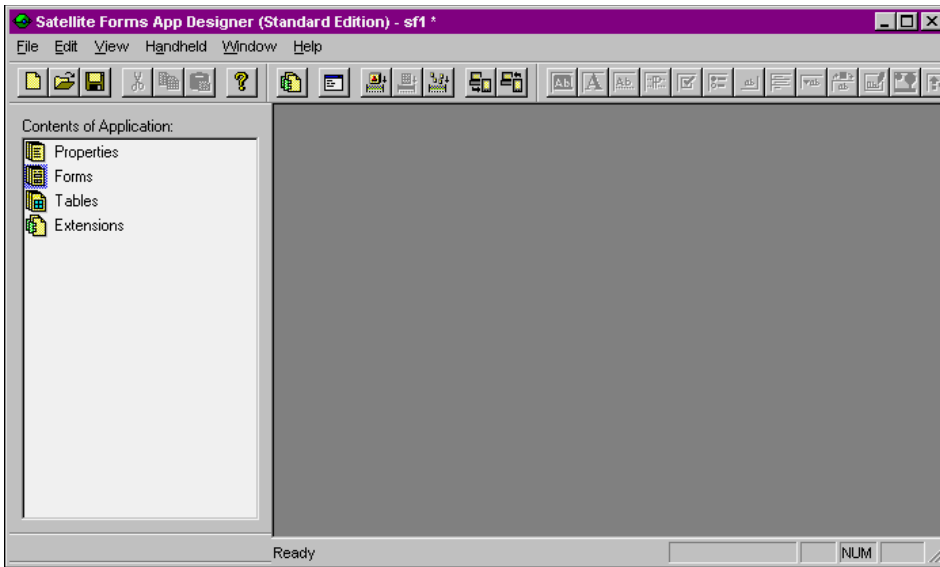
Figure 2. The Satellite Forms IDE before adding tables or forms.

Before you begin, you'll want to set a couple of application-wide properties. Right-click on the Properties node in the Contents of Application tree view and select "Properties" from the resulting context menu, or just double-click on the Properties node. The Application Properties screen will be displayed as shown in Figure 3.
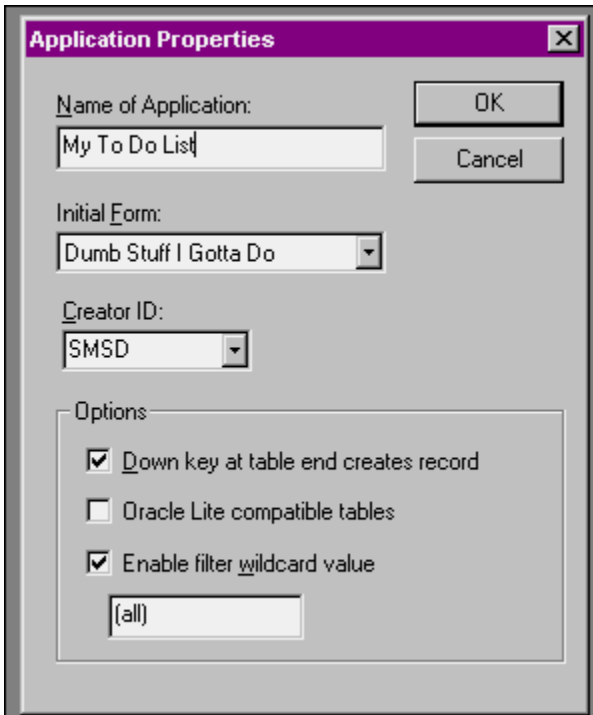

Figure 3. Setting Application-wide properties.

The one property you'll really want to set is the "Name of Application" property – if you don't, your application will be named "Untitled", and that's what will show up on your handheld. You don't have to

specify an Initial Form if you only have one form in your app; otherwise, come back to this dialog once you're done with creating the initial form.

The other properties can wait till later.

The SatForms Table Designer

The Table Designer in Satellite Forms is extremely easy to work with, nearly to the point of being trivial. Firing up the tool and playing around for a few minutes is all you need to completely cover its capabilities.

As you may remember, SatForms relies on an intermediate data store on your PC when moving data back and forth between your handheld and your desktop. You use the Table Designer to create this intermediate data store. When you deploy the SatForms application to your handheld, this intermediate data store will be converted to a proprietary ".PDB" field that is loaded onto your handheld.

By the way, this intermediate data store that resides on your PC takes the form of a dBASE V .DBF table – pretty darn convenient, eh?

In order to create a table, right click on the Tables node in the Contents of Application tree view in the left side of the SatForms IDE. You'll have three choices in the context menu: Insert Table, Import Table, and then either Show or Hide Tables, depending on whether the Tables node is expanded to show the tables below the node.

If you click on Insert Table, the Table Designer dialog will appear, as shown in Figure 4. The first thing to do, obviously, is name the table. The value you enter in the Name of Table text box will be used both internally by the Satellite Forms engine as well as for the name of the file on disk. Note that you can later change this value – but doing so only changes the value used internally by SatForms – the name of the file on disk stays the same.
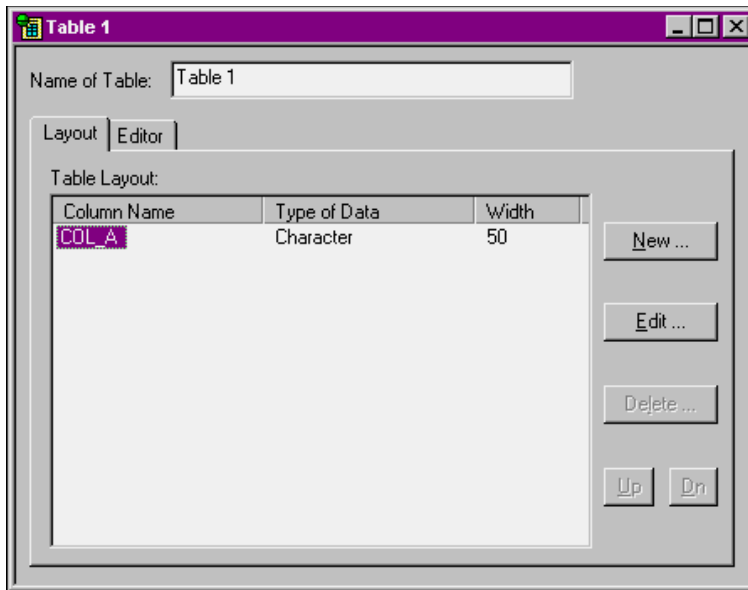

Figure 4. Creating a new table with the SatForms Table Designer.

As you see in Figure 5, there are two tabs in the designer. The Layout tab is used to create and edit the table structure – much like MODIFY STRUCTURE does in Visual FoxPro. You click the New button to create a new field, the Edit button to modify the highlighted field, and the Delete button to remove a row. Up and Dn ("Down") are used to rearrange the order of the fields, which, as all good theorists know, is relationally meaningless.
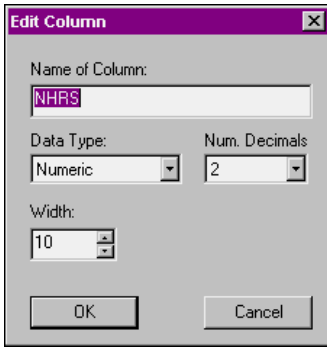
Figure 5. Use the Edit Column dialog to change attributes of a field definition.

When you click the New or Edit buttons, you get the Edit Column dialog as shown in Figure 4. You can enter the name of the column, select the data type (available choices are Character, Numeric, True/False, Date, Time, Ink, and Time Stamp – and I'll cover "Ink" and "Time Stamp" in future columns), and then choose the width and number of decimals when appropriate. For example, True/False fields are automatically defined as having a width of 1, and you don't get to change it, just like in VFP.

     You may be cringing at having to call up a separate dialog to edit a field's definition, but there is a reason. Back in Figure 3, you'll notice that there is no "OK" button in the Table Designer dialog. Yes, closing the dialog saves your changes – thus, instead of confirming your changes at the table level, you do so with each field.

     The real excitement starts when you click on the Editor tab of the Table Designer, as shown in Figure 6. Yes, it's a real, honest-to-goodness Browse window for you to enter and edit data!
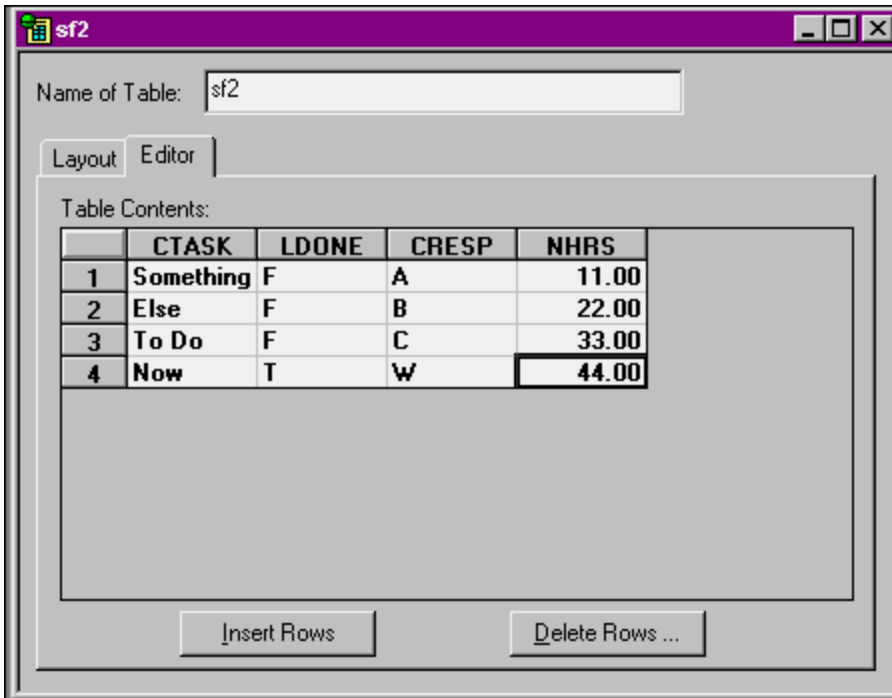
Figure 6. You can enter data into a Satellite Forms table via the Editor tab of the Table Designer.

You insert and delete rows via the appropriate command buttons in the bottom of the dialog, and type in data in the appropriate rows and columns as you desire. As soon as you type data in a field, the information is committed to the disk – I've found that you don't even have to leave the field (much less the row) in

order to save your changes. This is okay, really, because the Editor tab is supposed to be used for entering sample data for testing – it's not for your users, nor even for you to put large amounts of data into the table.

That's about all there is to creating a table. In this example, I've created a table for a simple To Do list – with fields for the name of the Task, who is Responsible for the task, how long it should take, and a logical flag for whether the task is done or not.

Now let's build a data entry form for this table that we can use on our handheld!

The SatForms Forms Designer
Just like with a table, you can create a new form in the SatForms IDE by right-clicking on the Forms node and selecting the Insert Form menu option in the resulting context menu. Doing so will display an empty form like shown in Figure 7.
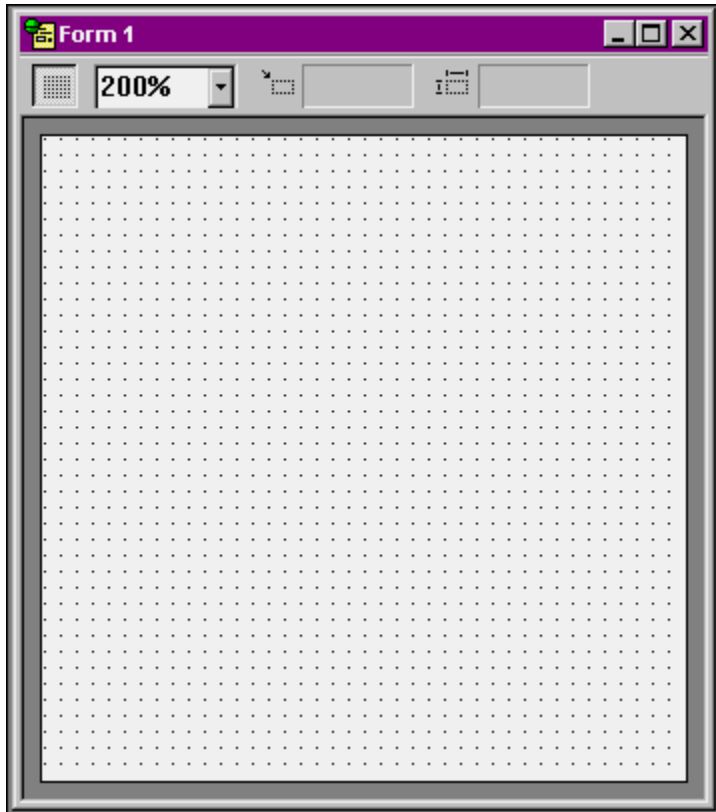
Figure 7. An empty form in the SatForms Form Designer.

The first thing you'll notice is the size of the empty form – it's 160 pixels square. You can't resize this form – by definition, all SatForms forms are this size because it's the size of the Palm "desktop." You don't have to use the entire form, of course, but you can't "tile" windows like you can in GUIs like Windows or the Mac OS.

Manipulating a Form
There's a bunch of stuff you can do with a form even before you start to put controls on it. First of all, there's the toolbar in the Form Designer window. The left most button allows you to turn the 'grid' (the dots on the empty form) on or off. The combo box (that shows 200% in Figure 5) allows you to change the magnification of the form displayed in the IDE. If you have a control selected, as in Figure 7, the next two read-only text boxes show the position of the upper left corner of the selected control, and the width and height, respectively, of the selected control.

Right-clicking in the empty form displays a context menu with two available choices – Form Properties and Scripts. We'll tackle the Scripts option later – it allows you to attach code to various form-level events. Selecting the Form Properties menu option opens the Form Properties dialog as shown in Figure 8.
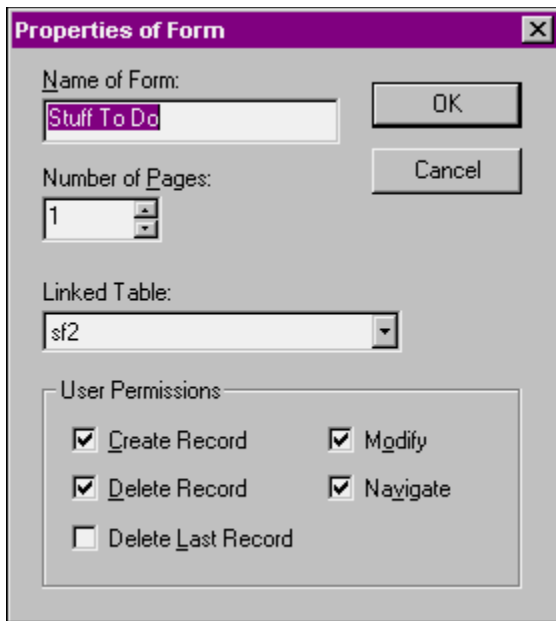

Figure 8. The Satellite Forms Form Properties dialog.

You can name the form – this name is not displayed to the user, but rather is used by the developer – for example, it's what is displayed under the Forms node in the Contents of Application tree view.

SatForms doesn't support a 'tabbed dialog' interface, but their substitute is a multiple page form. If you've used Microsoft Access, you're probably familiar with the concept of "multi-page forms." Think of Leslie Nielsen in The Naked Gun, when he pulls out his police ID and an accordion of credit cards unfolds from his wallet. SatForms multiple page form works sort of like that – you can specify events or write code that moves from page to page, much like you move from tab to tab in a page frame in VFP. You can specify how many pages your form has in this dialog.

The Linked Table combo box performs the same function as the VFP data environment, with the difference that you can only have one primary table linked to a form. (This isn't as big a restriction as it may seem at first. More in a future column.) The Linked Table combo box is populated with all of the tables in the Tables node in the Contents of Application tree view.

The last part of the Form properties dialog allows you to set what they call "User Permissions" – or, in other words, what functions a user is allowed to do. This functionality is extensible – these are defaults for the form. As you can see in Figure 6, you can allow the user to add, delete and modify records, navigate between records, and even delete (or not delete) the last record in the table.

Placing Controls on a Form

Placing controls on a form in SatForms works much like other visual tools you may be familiar with. You have a palette or toolbar of available controls and you drop those controls onto your form. With the SatForms IDE, all you have to do is click on the toolbar button and the corresponding control will be placed on the form.

Note that the control is always placed in the upper left corner of the form – you then need to drag the control to where you want it to reside.

The SatForms Controls Inventory

Many of the native SatForms controls are similar to VFP controls:

Title
Text
Edit
Paragraph
Check box
Radio button
Command button
List box
Droplist (combo box)
Lookup
Ink Control
Bitmap
Graffiti Shift Indicator
Auto Stamp
Custom Control

Most need no explanation but a few don't have direct counterparts to VFP or VB, or aren't immediately obvious.

The Title control is essentially a caption for the form. An example is shown in Figure 8, where its value is set to "Dumb Stuff I Gotta Do."

The Ink Control allows you to capture a signature or other freehand drawing on the Palm – you can think of it as a Palm-specific General control.

The Graffiti Shift Indicator control places an "indicator" graphic on the form that shows the shift status of the Graffiti handwriting recognizer. This graphic will show different symbols depending on whether Graffiti is in lowercase, shifted or caps-lock mode.

The Auto Stamp control automatically enters a date or time stamp in a table column.

The Custom Control allows you to specify a custom control you've built or acquired from another source, much like you'd specify an ActiveX control in Windows.

Controls Properties

You can set properties of every control by right clicking on a control of interest and selecting Control Properties from the context menu, as shown in Figure 9. As with other visual environments, some properties are shared by more than one control, while other properties are unique to a specific control.

All controls have Name of Control, Dimensions and Visible properties. Most controls have Data Source, Font, Read-Only and Don't Modify Table properties. Most controls also allow you to specify an Action When Clicked.
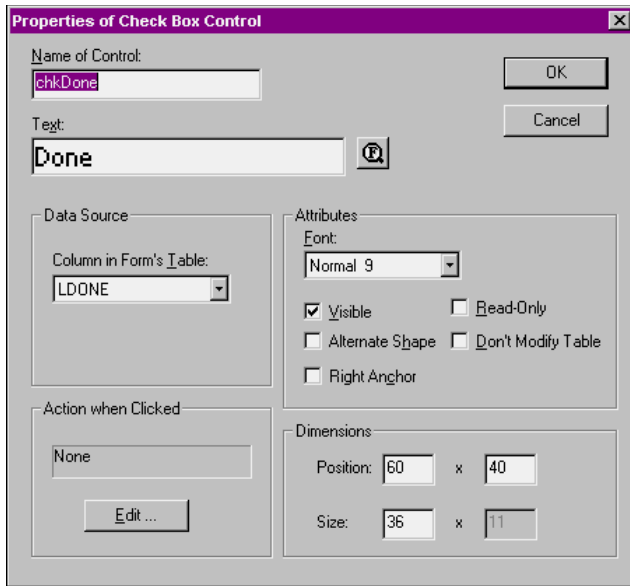
Figure 9. The properties dialog for the Check Box control

The combo box in the Data Source property is populated with the appropriate fields for the table that is linked to this form. For a check box control, only logical fields in the linked table are displayed in the combo box, for text box controls, only character and numeric fields are displayed in the combo.

Command buttons have special properties because they're never attached to data – they're always used to initiate an action. (Well, I suppose you could use one as a fancy label, and just change the caption, but that's not the intended purpose of the control.) See Figure 10 for the properties dialog of the Command Button control.
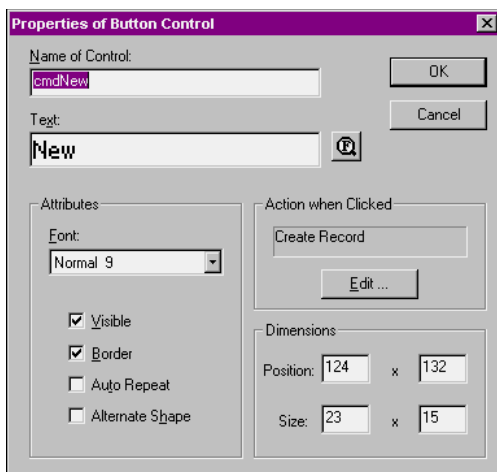
Figure 10. The Button control's property dialog.

As you can see, it has common properties like Name, Text (what you and I think of as "caption"), Font, and Dimensions. There are also some obvious button-specific properties like "Visible" and "Border". "Auto-Repeat" changes the nature of how the control fires its action on the handheld. Normally, a control fires at the end of a tap – when the stylus is lifted from the handheld screen. (That's our "mouse-up" event.) When AutoRepeat is checked, the control's action fires as soon as contact is made – and if you keep the stylus pressed against the screen (lightly, folks, lightly – you're not drilling for oil!), the action will be repeated at a periodic (but undocumented) interval.

The real magic is contained in the Action when Clicked property. Click on the Edit button, and the dialog shown in Figure 11 will appear.
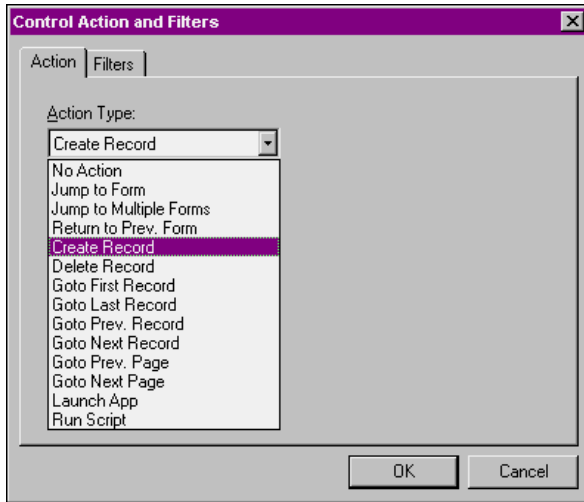

Figure 11. Selecting an Action to attach to a button.

As you can see, there are a wealth – or, as Kelly Bundy would say, a *veritable cornucopia* - of pre-defined actions, including record and form navigation, and record maintenance. Note the last two options in the combo –Run Script allows you to run code that you've written yourself in case one of these options isn't suitable – and Launch App allows you to run another SatForms application.

There's even more power under the hood, though. Selecting the Filters tab, and then clicking the Add button, as shown in Figure 12, allows you to create a specialized type of action that filters a table. This interface is a bit awkward at first – you can think of the Filters tab as being just another type of action – a "Filter Table" action in the Action combo box.

Thus, you could create a button named "Filter" that would set a filter on the table based on criteria you specify in the Create New Filter dialog. Note that you can use either a hard-coded expression or the current value of a control on the form.
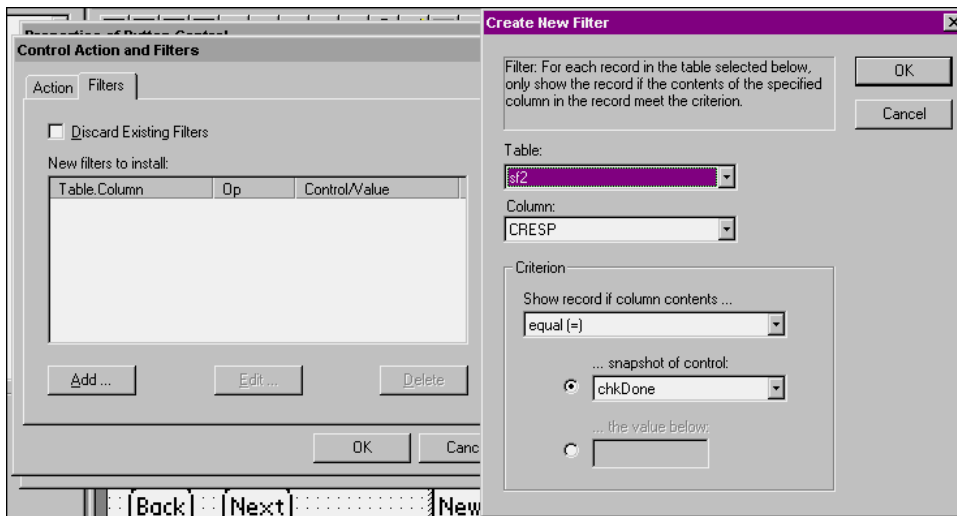

Figure 12. Adding a filter to a button.

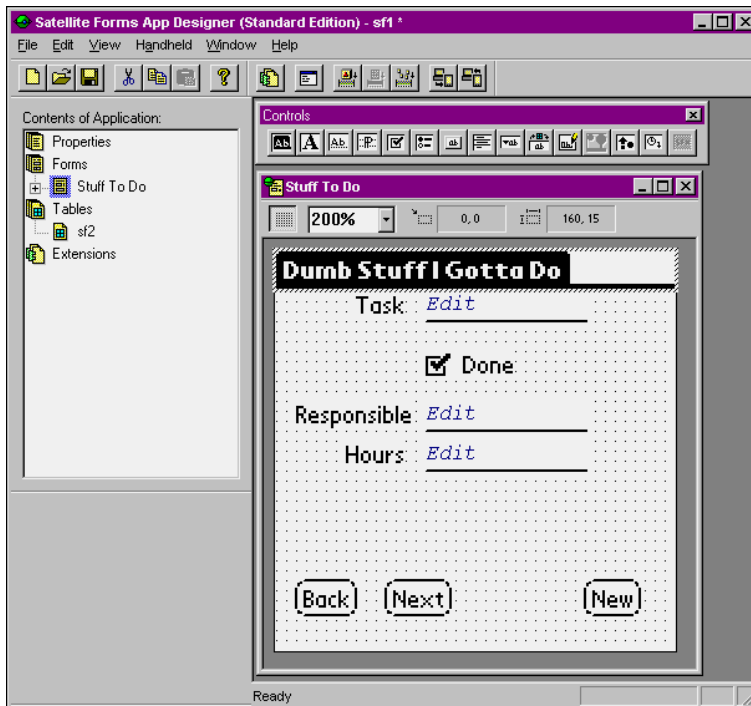The form for the table created above in Figure 4 might look like that shown in Figure 13.

Figure 13. A sample Satellite Forms application form.

Saving your Project

When you create your tables and forms, SatForms will try, like many other brain-dead tools, to save your work in a subdirectory under the directory where you installed Satellite Forms itself – yes, buried deep on drive C. Jeesh. Hasn't anyone at Puma ever had to reformat their C drive?

Obviously, don't accept their defaults – create your own "SatFormsProjects" directory on your development drive, and create separate directories for each SatForms project you create. Once you start saving components for a project there, SatForms will be good about saving subsequent components in the same directory as well.

When you're done with creating your tables and forms and stuff, you'll press Save (if you haven't done it already), and be greeted with a zillion (a zillion is a fraction of a bazillion) dialogs about naming your files. Well, OK, two dialogs.

The first one will ask you to save the SatForms application itself. The resulting file will have an "SFA" extension. You'll next be asked to save your table, and that file will have a "DBF" extension. However, if you look in your directory, you'll see lots more files than just those two. Let's take a look at what each of these are.

First, suppose that you named the project "SFDEMOAPP" and the table "SFDEMO1". You'll see the following files in your project directory:

sfdemo1.dbf
sfdemoapp.pda
sfdemoapp.sfa
sf-se_sfdemoapp.pdb
sft-se_sfdemo1.pdb

The DBF and SFA files are pretty easy to identify, since I just told you what they were. The SFDEMOAPP.PDA file is created by SatForms, and is the file that gets transferred to your handheld. In other words, it's the application that runs on your Palm. The SFT-SE_SFDEMO1.PDB file is the proprietary data file on the handheld where data is saved. I believe that the SF-SE_SFDEMOAPP.PDB file

is associated with the PDA file, in that it contains part of your application – not your data – but the documentation is less than clear about this point.

Deploying your SatForms Application on your Palm

Once you've finished your application, it's time to put it on your handheld and run it there.

First, put your handheld in the cradle, and make sure the connection to your PC is tight. Next, in SatForms, open your project. Select the Handheld|Download App and Tables menu option. The dialog shown in Figure 14 will appear on your PC.
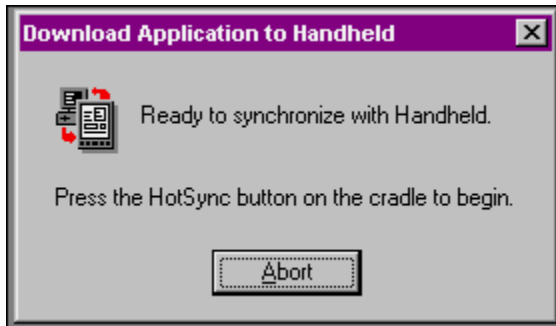

Figure 14. Press the Sync button your handheld cradle when ready to deploy your application and tables from the SatForms IDE to your handheld.

At this point, press the Sync button on the cradle. Both your PC and the handheld will display the usual messages about which files and applications are being synchronized – if you watch closely to the messages on your PC, you'll see Satellite Forms in the list.

Finally, the messages will finish. On your PC, you'll be returned to the Download Application to Handheld dialog in Figure 14. On your handheld, you'll probably see a message about 'Cleaning up' for a few seconds longer. Once that message goes away, the sync is done and you're ready to try out your new handheld application.

Take your handheld out of your cradle, turn it on (if it's not already on), and tap the SatForms icon on the main application screen. You'll see your sample app (along with others that you may have downloaded) in a list. Tap the name of your sample (hopefully it's not called "Untitled'), and the initial form will display. Tap the Next and Back buttons, change some data, add a new record – whatever you like. Open a different application on your Palm, turn your Palm off, and then turn it on and open your sample SatForms app again – the changes you made will still be there. That's all there is to it!

Transferring back to your PC

Finally, let's move those changes back to your PC.

Repeat the steps in the previous section, with the one exception of selecting the Handheld|Upload Tables menu option instead of "Download App and Tables." Once you're done synchronizing, open the table in the SatForms Table Designer, click on the Editor tab, and you'll see the changes you made on your handheld back on your PC. Don't believe me? Take your handheld out of it's cradle, wrap it in tin foil, and put it in a lead-shielded box in the basement – then look at the table on your PC again. Yes, you've really moved your data from the handheld back to your PC!

We now have a database application running on our handheld. Presumably we also have a VFP application running on our PC (or server). We now have two goals.

Goal #1 - Synchronization

What we would like – ideally – is to have an application running on our desktop (or server) that houses a superset of the data that we're accessing on our handheld.

For example, suppose your desktop has your entire To Do list dating back five years. You may just want the last two weeks of tasks as well as anything upcoming to be accessible on your handheld. Thus, you'll want to move a subset of your PC's data to that intermediate data store – the dBASE V DBF that the

SatForms Table Designer creates - that gets transferred to your handheld. In other words, you might have 10,000 records in your VFP database. You might want to copy 480 of those to the DBF that SatForms uses. When the HotSync occurs, your handheld now has those 480 records. (Remember that the old table that your handheld had is now gone – completely replaced by these 480 records.)

Then, after you've been out and about for a while, you return to the office. You've changed five of those 480 records, and added a few more. The handheld database has 483 records. When you HotSync, those handheld records are moved into the DBF.

And, just like the previous HotSync, the DBF gets completely wiped out and replaced. The DBF now has those 483 records that were on your handheld, complete with the changes you've made to five of the initial 480. You'll now need to synchronize those 483 records with your master VFP database of 10,000 records.

To make matters worse, somebody might have snuck into your office while you were gone and added 17 more things to your To Do list. (Who let your spouse in there, anyway???) During this same HotSync where you're grabbing the 483 records off of your handheld, you're going to want to send those 17 new records back to your handheld – via the DBF, of course.

Obviously, we can do this – we're database programmers. The point is that this isn't done for us automatically – we're going to have to write code to make it happen.

Goal #2 - Magic
You know what would be great? Let's suppose you wrote the VFP code that did all this synchronization between the intermediate DBF and your big VFP database – looking at time stamps, primary keys, whatever. Wouldn't it be great if, whenever you put your handheld in the cradle and hit the HotSync button, your VFP application detected this HotSync on the PC, and, after the HotSync was done moving the handheld data to or from the PC's DBF file, executed the appropriate synchronization methods that you wrote in VFP? In other words, you didn't have to do anything on your PC or in VFP – your VFP app just sat there, in the background, doing its thing for you?

That's our second goal – to make this happen automagically.

We're going to do these in reverse – first, I'll show you how to write a VFP app that uses the SatForms ActiveX control to automatically detect a HotSync, and then moves data back and forth from the DBF to a VFP data store. (Naturally, it doesn't have to be VFP. It could be SQL Server, or anything other type of data that VFP can talk to.)

Once that's done, you'll realize that the other half – the code that actually synchronizes the DBF and your VFP data – is just database programming – stuff you've been doing for twenty years – and we'll look at that next month.

Building a HotSync-Aware form in VFP
For my first trick, I'm going to create a VFP form that binds directly to that intermediate dBASE V DBF file. This will help you get used to how the ActiveX control works. Once that's running smoothly, I'll show you how to separate the intermediate DBF from the form's controls, and integrate your own data with the form and the DBF.

It's most handy if you've already got your SatForms application done, so I'm going to assume that you've followed along with last month's article and have the following files in your SFDEMO project directory:

sfdemo1.dbf
sfdemoapp.sfa

There are some other files in that directory, like the PDA and PDB files I covered last month, but they're irrelevant for our purposes right now.

In this same directory, you're going to create your VFP form. Create a form, named SFDEMO1.SCX. Open the data environment and select the SFDEMO1.DBF table. Add some text boxes, check boxes, and command buttons to it so that it looks like Figure 15. There's a copy of this form in this month's Subscriber Downloads, by the way. I've used all VFP base controls, so you just have to open up SFDEMO1.SCX.

Bind each of the controls to the appropriate field in SFDEMO1.DBF, using the control's ControlSource property. Here's the simple code in each of the command button's click events:

```
* Back
skip -1
thisform.refresh()

* Next
skip 1
thisform.refresh()

* List
on key labe ENTER keyboard "{CTRL-W}"
brow normal
on key label ENTER
thisform.refresh()

* New
local m.lcResp
m.lcResp = thisform.txtResp.value
insert into SFDEMO1 ;
   (cTask, cResp, dDue, lDone, nHrsEst, nHrsAct) ;
   values ;
   ("NewTask", m.lcResp, date(), .f., 0, 0)
thisform.refresh()
```

By the way, don't forget to set your tab order – one thing you don't have to do on a handheld!
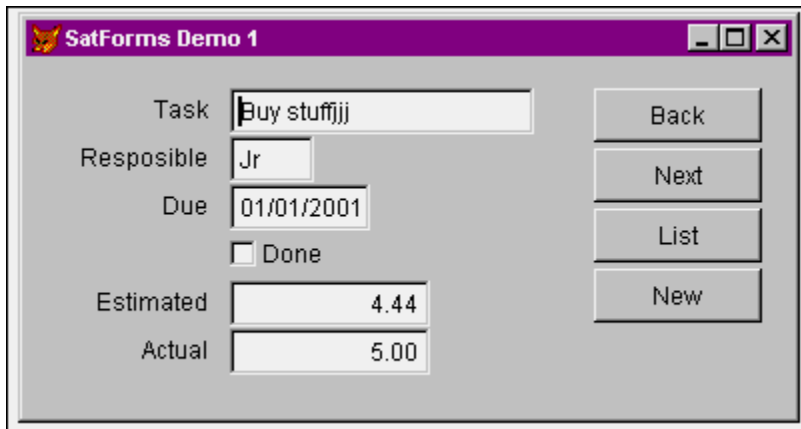
Figure 15. A simple data entry and navigation form in VFP bound to the SatForms intermediate DBF.

Run the form and make sure that you've got all the code you need in the buttons, that you've bound the controls properly, and so on.

Once you've made some changes, perhaps added a few records, close your form, and then do a HotSync with your handheld. If you have everything set up properly, you should have new data on your handheld.

Whoa! Where's the new data? It's NOT on your handheld, is it? Why the heck not? You just did a HotSync, right? Well, you have to tell the HotSync explicitly to download this DBF file. We need the SatForms ActiveX control to do so.

Adding the SatForms ActiveX control to your VFP form
Obviously, you'll need to drop the SatForms ActiveX control on your form. It should already be registered on your machine – that happened when you initially installed SatForms. Thus, the first thing you need to do (if you haven't gotten ahead of me) is register it with VFP.

Select the Tools|Options menu option, click on the Controls tab, select the ActiveX option button, and scroll down the list to the S's as shown in Figure 16.
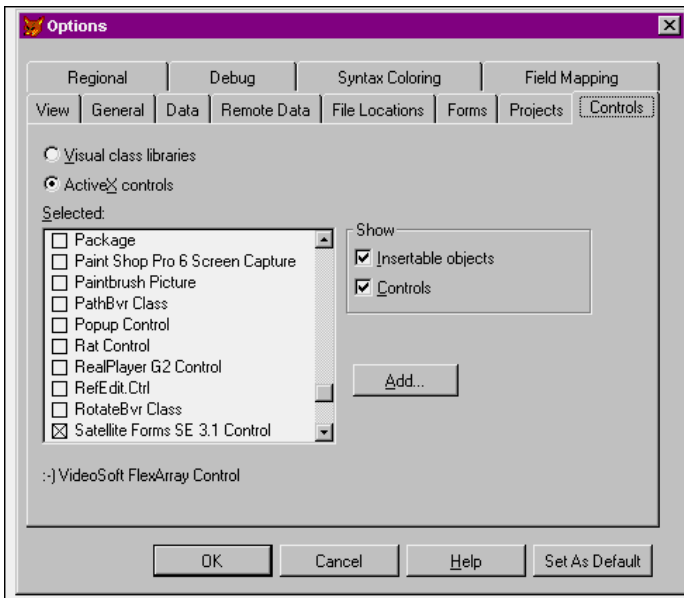
Figure 16. Registering the SatForms ActiveX control with Visual FoxPro.

Check the check box, click on Set As Default command button, and close the dialog.

Now you can access the control via the Form Controls toolbar. Open the Form Controls toolbar, click on the second button from the left (the one with the three books and the small arrow in the lower right corner), and select the ActiveX Controls menu option.

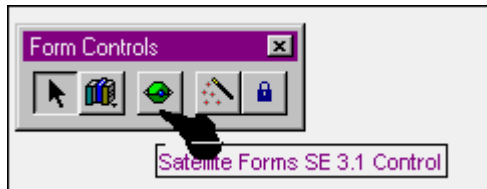The Form Controls toolbar will change to something like Figure 17.

Figure 18. The Form Controls toolbar, showing just ActiveX controls.

The middle button on the toolbar represents the SatForms ActiveX Control. Click on it and then click on your VFP form, just as you do when placing any other control on a form. This control won't ever actually be visible when the form is running, so put it somewhere way out of the way. (Milwaukee, perhaps?) See Figure 19.
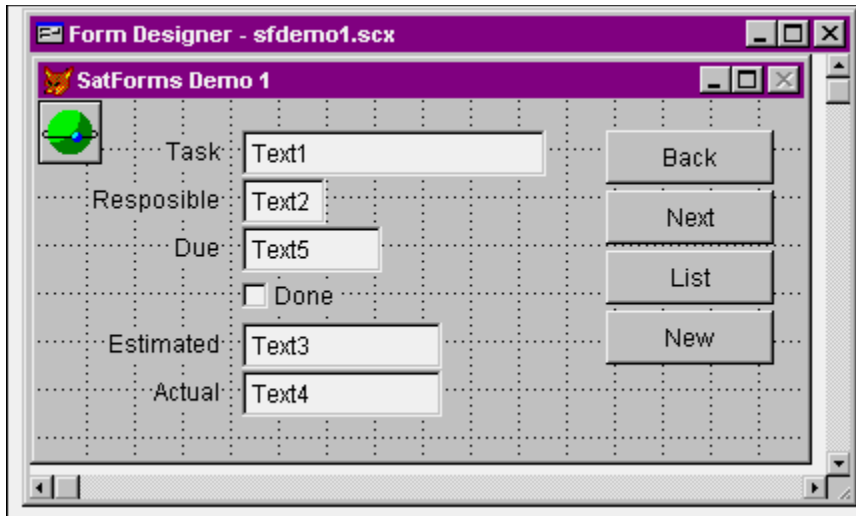
Figure 19. The VFP form with the SatForms ActiveX control.

The SatForms ActiveX control is, by default, named "olecontrol1" – why not change it to something more reasonable, like oleSatForm.

Implementing the SatForms ActiveX control

And now it's time to write some code. (You knew that was coming, didn't you?) The first thing to do is put the following line in the form's Init() method:

```
thisform.oleSatForm.Object.Enabled = .t.
```

For some reason, the SatForms ActiveX control is disabled when it's dropped on a form. You'll want to turn it on or nothing else will work.

Next, double-click on the ActiveX control itself, and locate the HotSyncStatus() method. The HotSyncStatus event gets fired many times during the HotSync process – the code that you put in this method will get fired in concert with the firing of the HotSyncStatus event. This is an important point – you don't call HotSyncStatus() in your code - you put code in this method that gets executed automatically.

In other words, your VFP form – with this ActiveX control on it – will be running on your PC. You'll put your handheld in the cradle, press the HotSync button, and the process will start. Part of the HotSync process runs on your PC, of course. The SatForms ActiveX control on your form detects that the HotSync process is running on your PC, and generates HotSync Status events. When the HotSyncStatus event is generated, it pass StatusCode and Param parameters, so your code in the HotSyncStatus() method needs to look at these parms and do the appropriate thing.

To begin with, we're only interested in the first parameter. So what are the possible values of this parm? The StatusCode parameter can take one of three values: 1, 2 or 3.

1   Start of HotSync
2   End of HotSync
3   HotSync is running

Here's some sample code that you can put in the HotSyncStatus method to see how these parms are passed to the HotSyncStatus() method:

```
* oleSatForm.HotSyncStatus()
do case
case StatusCode = 1
 * start
 debugo "code 1 - start"
case StatusCode = 2
```

```
 * start
 debugo "code 2 - end"
case StatusCode = 3
 * running
 debugo "code 3 - running"
othe
 * a mystery
 debugo "code " + tran(StatusCode) + " mystery code"
endcase
```

Once you've put this code in your form, save your work, and run the form. Open your Debug Output window as well, of course. You should be able to navigate through records, make changes, and so on, like usual. Then press the HotSync button on your Palm cradle. You'll get output in the Debug Output like that shown in Figure 20.
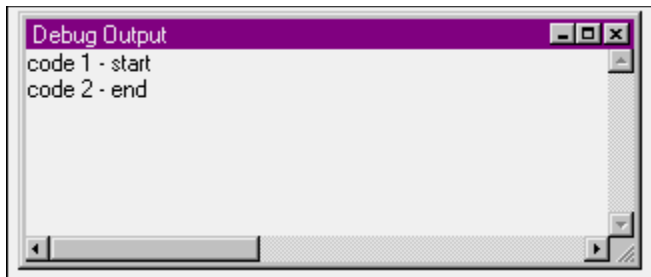

Figure 20. Events generated from the HotSyncStatus() event.

You'll notice that "code 3 – running" didn't display yet – I'll address that shortly. In the meantime, though, you can see that your VFP app is detecting the HotSync event initiated from the handheld cradle. Your VFP app is just sitting there in background, waiting to detect the HotSync.

Now it's time to do some database work.

Using the SatForms ActiveX control to move data

The SatForms ActiveX control has a couple more methods that are really interesting. They're called

CopyTableToPalmPilot(<tablename>)
GetTableFromPalmPilot(<tablename>)

and, as you might guess, they're used to move a table to or from the Palm. Note that you move data a table at a time, specifying the name of the table as an argument to the method.

In order to provide a more interesting programming experience, we're going to add the ability for the user to control which method will be used in our example. Put an option group, named opgDirection, on your form like shown in Figure 21.
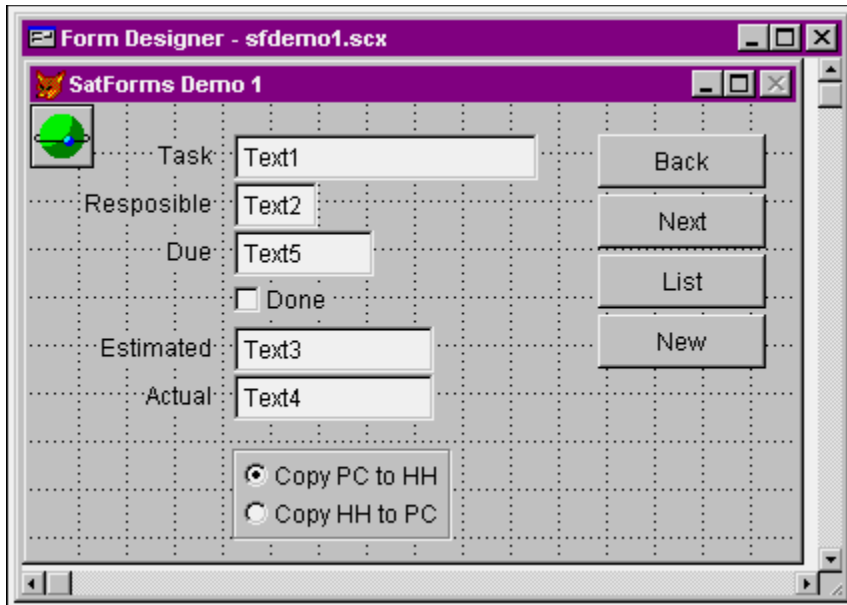
Figure 21. Adding a directional option group to your VFP form.

In your form's Init(), make sure that you initialize the value of the option group to a character string:

thisform.opgDirection.value = "Copy PC to HH"

Now we're going to add code to the HotSyncStatus() method that will look at the value of the option group and then call either the CopyTable or GetTable method as appropriate. The entire HotSyncStatus() method looks like this:

```
*** ActiveX Control Event ***
LPARAMETERS statuscode, param
do case
case StatusCode = 1
 * start
 debugo "code 1 - start"
 do case
 case thisform.opgDirection.value = "Copy PC to HH"
  debugo "code 1 - copying PC down to HH: SFDEMO1.DBF"
  This.CopyTableToPalmPilot(fullpath("sfdemo1.dbf"))
 case thisform.opgDirection.value = "Copy HH to PC"
  debugo "code 1 - copying HH up to PC: SFDemo1.DBF"
  this.GetTableFromPalmPilot(fullpath("sfdemo1.dbf"))
 othe
  messagebox("What is the value of opgDirection? " + tran(thisform.opgDirection.value))
 endcase
case StatusCode = 3
 * running
 debugo "code 3 - running. CmdType is: " + tran(this.CmdType)
case StatusCode = 2
 * end
 debugo "code 2 - end"
othe
 debugo "code ? - otherwise: " + tran(StatusCode)
```

endcase

Note that the DO CASE structure that contains the CopyTable and GetTable methods is contained in the Start (StatusCode = 1) code segement – not in the Running (StatusCode = 3) segment as you might expect. Why?

The Running code segment gets called each time a CopyTable or GetTable method completes. You can put your own code in this segment to do "post-transfer" work – such as your own code that synchronizes the intermediate DBF with your own VFP data.

To show you how this works, I added a couple of DEBUGOUT commands in the appropriate places. Let's see what happens when we run this.

First, run your VFP app, and enter "A" into a record in the VFP table. Then make sure that the "Copy PC to HH" option button is selected. Put your handheld in its cradle, press the HotSync button on the cradle, and wait for the sync messages to complete.

At this point, open up the matching application on your handheld, and look for the same record. You'll see the value "A". The Debug Output window on your PC should look like Figure 22.
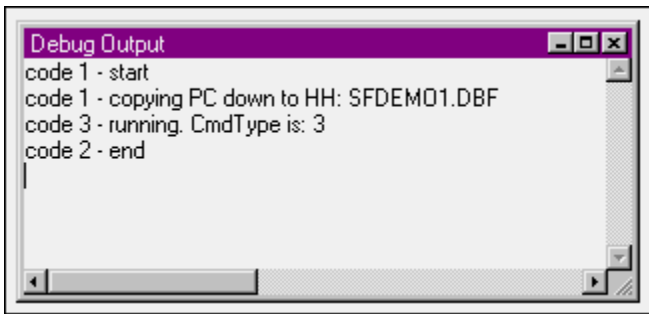

Figure 22. The Debug Output window after issuing a CopyTableToPalmPilot command.

Now, on your Palm, change "A" to "BBB". Put the handheld back in its cradle, click the "Copy HH to PC" option button in your VFP application on your PC, and then press the Sync button on your handheld's cradle. Wait for the sync messages to finish, and then find the record on your PC. You'll see "BBB" has taken the place of "A". Additionally, your Debug Output window should look like Figure 23.
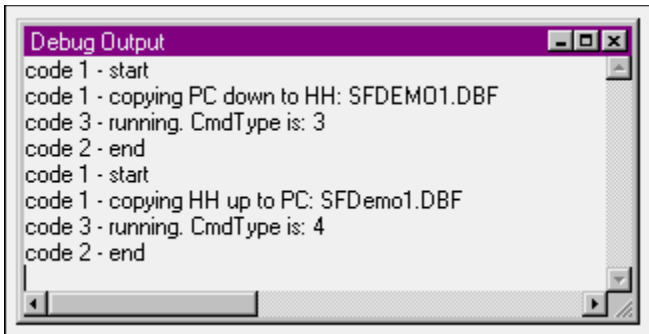

Figure 23. The Debug Output window after issuing a GetTableFromPalmPilot command.

You'll notice that the SatForm ActiveX control's CmdType value varies according to which method is being called:

* Property Nameof ActiveX Control: CmdType
* If CmdType is 1, then just finished CopyAppToPalmPilot
* If CmdType is 3, then just finished CopyTableToPalmPilot
* If CmdType is 4, then just finished GetTableFromPalmPilot

You can use this information to write reusable code that performs an appropriate function according to which type of data transfer was just performed.

Multiple tables

It'd be a pretty boring world if we just worked with single table applications all the time, wouldn't it? In this next example, we'll move more than one table back and forth, and watch how the process works. Because this is just an example, I didn't bother with hooking this second table into the application – the point is to show you how to transfer multiple tables.

First, you'll need a second table to work with. The easiest way is to open the SatForms IDE and create a second table. I created SFDemo2.DBF with a single dummy field, and added just one record to it.

Then, in the HotSyncStatus() method, I changed to code like so:

```
do case
 case thisform.opgDirection.value = "Copy PC to HH"
  debugo "code 1 - copying PC down to HH: SFDEMO1.DBF"
  This.CopyTableToPalmPilot(fullpath("sfdemo1.dbf"))
  debugo "code 1 - copying PC down to HH: SFDEMO2.DBF"
  This.CopyTableToPalmPilot(fullpath("sfdemo2.dbf"))
 case thisform.opgDirection.value = "Copy HH to PC"
  debugo "code 1 - copying HH up to PC: SFDemo1.DBF"
  this.GetTableFromPalmPilot(fullpath("sfdemo1.dbf"))
  debugo "code 1 - copying HH up to PC: SFDemo2.DBF"
  this.GetTableFromPalmPilot(fullpath("sfdemo2.dbf"))
 othe
  messagebox("What is the value of opgDirection? " + tran(thisform.opgDirection.value))
 endcase
```

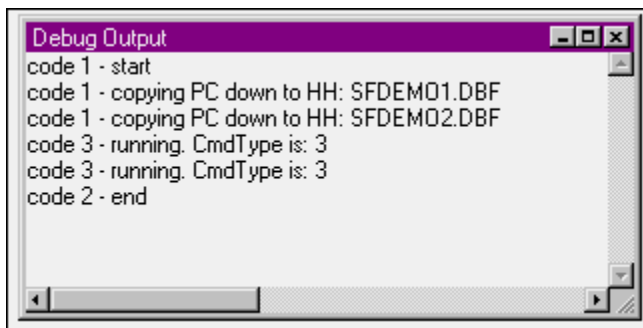When you run this, the results in the Debug Output look like Figure 24:

Figure 24. The Debug Output window after copying two tables.

Notice that the CopyTable method was executed twice before the code in the StatusCode = 3 segment was executed. This is intentional, so it means that if you are relying on running code in the Running segment before another Copy or Get command is executed, you'll need to employ some fancy footwork to make it happen.

Conclusion

So those are the basics to deploying your data on your handheld. Obviously, where the master data is coming from actually doesn't matter much – you could be accessing VFP DBFs directly, or using views that talk to a remote data source like SQL Server.

There's a lot more to this tool and these capabilities – multiple applications, custom controls, multiple users – but those topics are fodder for an advanced session.