Main Menu | Log In | Subscribe | Links | FAQ | Advertising

Reborn Free CLARION online

published by CoveComm Inc.

Search

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Holiday Special
25% Discount
on
APM & AFE

Clarion MAGAZINE

December 2000 Index

Completely Dynamic Report Orders and Breaks Part 2
Steve Parker delves into dynamic group breaks in the report engine, and lays out the theory and the practice. Part 1 of 2.
(Dec 22,2000)

December 2000 News
Clarion news, notes, and happenings from around the globe.
(Dec 22,2000)

Finding Lost Files: A Redirection Class
In a perfect world, reading image files from disk wouldn't be a problem. Just store the filename in a database, display the file as needed in an image control, and be done with it. However, in a networked environment, several nagging issues pop up that can cause significant difficulty. Here's a redirection class that lets your application search for files on the paths you specify.
(Dec 22,2000)

Clarion Essentials CBT From SoftVelocity
Tom Hebenstreit reviews SoftVelocity's Clarion Essentials Computer-Based Training CD. As a step up from the original Clarion Foundations CBT, which concentrated mainly on learning the basics of the Clarion language and IDE, the Essentials course tackles more advanced concepts and features of the Clarion IDE and language.
(Dec 22,2000)

Freebie: Data Is Executed Policy - A Case Study
The paper, Data Is Executed Policy, presents a Clarion-centric approach, illustrated through a case-study, to requirements analysis and design through production system implementation. While the approach set forth in this paper is "normal fare" for Clarion application builders, it may be quite new, novel, and striking in its aggressive schedule and accomplishments. Thus, the paper is recommended as material to be given clients in support of a systems development proposal.
(Dec 22,2000)

Clarion MAGAZINE

Making Sense of ABC's ErrorClass

by Russ Eggen

Part 3 of 3

In Parts 1 and 2 of this series I looked at how `ErrorClass` works and how you can customize it to your own needs. But all of that has been hand code. Wouldn't it be easier to customize ErrorClass with a template?

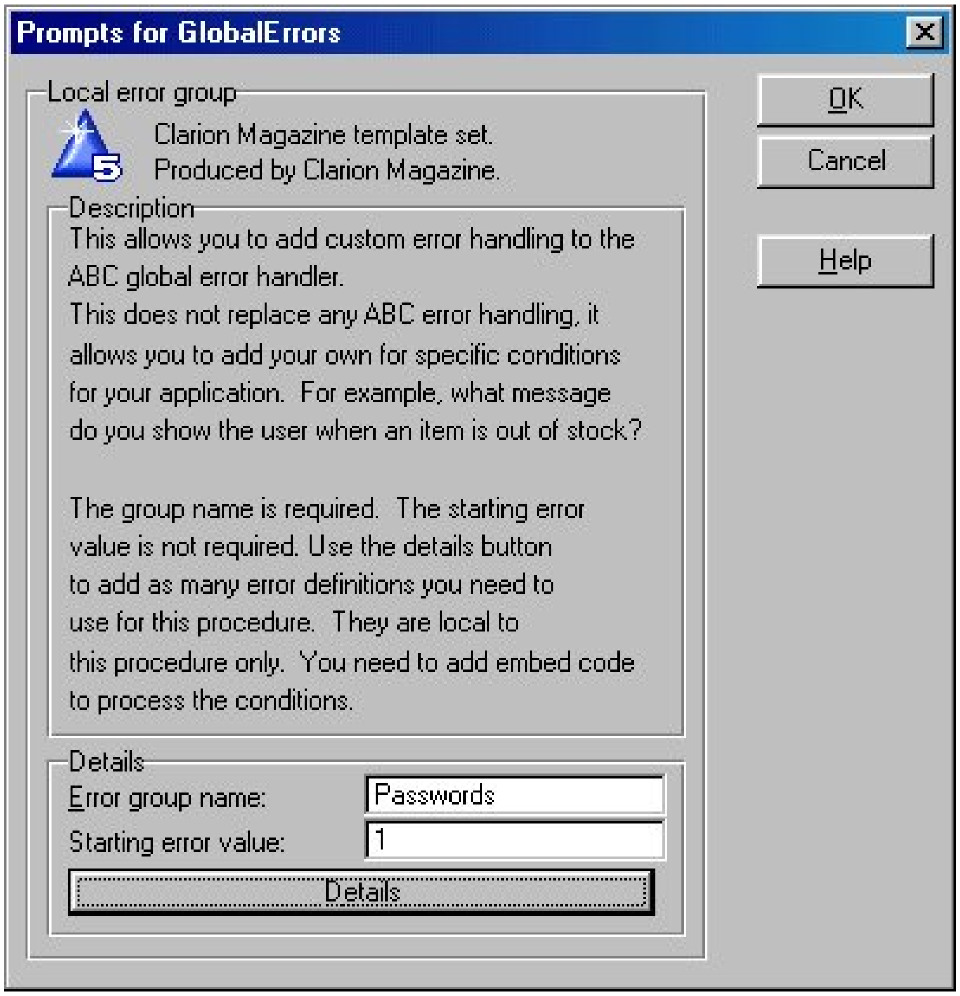
How about a template?

It's easy to make a template to add your own error messages. The template does not write the logic to handle the errors, only to make `ErrorClass` aware of the error numbers you wish to add. Since there is a lot of tedious source coding in the examples I have mentioned earlier, it would make sense to convert this to a template.

Also, to get things off to a good start, this template is "driven" by a generic template that you can use to add other templates that Clarion Magazine may publish from time to time. I like this method as it is immune to new template sets and changes SoftVelocity may release. In other words, no changes to the ABC templates are required. Since this template is out of the scope of this article, I invite you to inspect the source. There are plenty of comments that explain what is going on. It is also small enough for even beginner template coders to digest easily.

The following are screen shots from this template. The first thing you must do is define your error group, as shown in Figure 1.

Figure 1. Defining the error group



Prompts for GlobalErrors

Local error group

Clarion Magazine template set.
Produced by Clarion Magazine.

Description

This allows you to add custom error handling to the ABC global error handler.
This does not replace any ABC error handling, it allows you to add your own for specific conditions for your application. For example, what message do you show the user when an item is out of stock?

The group name is required. The starting error value is not required. Use the details button to add as many error definitions you need to use for this procedure. They are local to this procedure only. You need to add embed code to process the conditions.

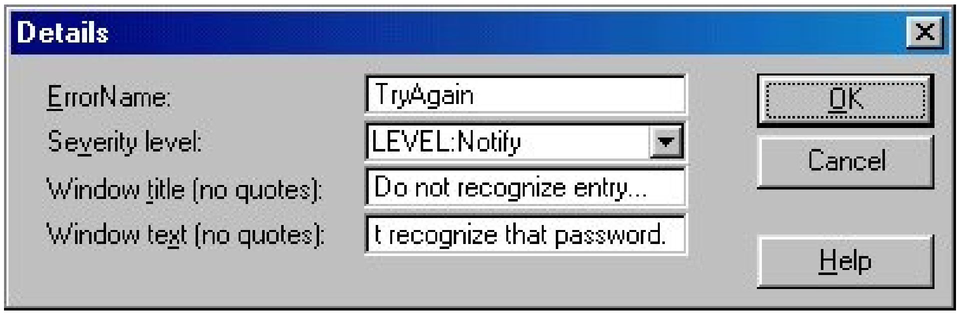
Details

Error group name: Passwords

Starting error value: 1

Filling in the group name and starting error number enables the Details button. Pressing this opens the details dialog. Press the Insert button to add the four elements for each message, as shown in Figure 2.

Figure 2. Defining an error.



Details

ErrorName: TryAgain

Severity level: LEVEL:Notify

Window title (no quotes): Do not recognize entry...

Window text (no quotes): t recognize that password.

All that is left is to inspect the code this template generates:

```
!!Define your custom error numbers
         ITEMIZE(1)       !Use entered seed number
TryAgain   EQUATE         !Declare custom error numbers
Hacker     EQUATE         !Declare custom error numbers
         END

!!Define your error group
Passwords    GROUP    !Your error handler
  USHORT(2)           !No of msgs to use
  USHORT(TryAgain)   !Install the error number
  BYTE(LEVEL:Notify) !severity level of this error
  PSTRING('Do not recognize entry...') !Window title
  PSTRING('The system did not recognize that password.')
  USHORT(Hacker)     !Install the error number
  BYTE(LEVEL:Fatal)  !severity level of this error
  PSTRING('Unauthorized Access!')  !Window title
  PSTRING('Hacker detected! Closing program.')
END
```

Look familiar? It should. The logic and the code to process these errors are the same. This example application and template are available at the end of this article. The behavior of this application should be no different than the first example. What is different is that the code you would otherwise have to write by hand is done with a template.

All you need to do is add the logic for when the errors are tested.

Global overrides

If you don't like ABC's error messages, you may add your own. In the previous example I showed you how to do this locally. What if you wish to do this globally? Follow ABC's lead and use your own TRN file. The actions to add the messages in your TRN file are the same as declaring a `GROUP`, as discussed previously. The only additional step is to include your error TRN file. `After Global Includes` is a good spot.

You can add your custom messages in the `Program Setup` embed and remove them in `Program End` embed, using the `AddErrors` and `RemoveErrors` methods as before. Since the error handler is the first thing setup and last object destroyed, these embeds will suffice.

Expansion macros

A very nice feature of `ErrorClass` is expandable macros. This means that you can get very specific with the messages, while coding these messages in a general sense. For example, the following error message is not very helpful:

"There is an error with this file!"

Unless you have an application that uses only one file, which is not likely, troubleshooting an error like this is difficult. What would help is the same message constructed like this:

```
There is an error %ErrorText in the file %File.
```

OK, this is better - you have data to work with. There is one nice bonus with the `%ErrorText` macro. The `%ErrorText` macro uses `%FileError(%FileErrorCode) -` the more specific backend server error information - when it is available, otherwise it uses `%Error(%ErrorCode)`. I'm sure that you understood that instantly. If not, here is the complete list of macros:

```
%File          ErrorClass.FileName property
%Field         ErrorClass.FieldName property
%Message       ErrorClass.MessageText property
%Error         Value returned by ERROR()
%ErrorCode     Value returned by ERRORCODE()
```

```
%FileError     Value returned by FILEERROR()
%FileErrorCode Value returned by FILEERRORCODE()
%ErrorText     %Error(%ErrorCode) ↵
                  or %FileError(%FileErrorCode)
%Previous      Text from prior defined error ↵
                  with the same id
%Procedure     Current procedure name
%Category      The category of the error
```

Inspect this portion of `ErrorClass` definition:

```
SaveError          CSTRING(255),PRIVATE    ! Clarion error
                                           !  message
SaveErrorCode      LONG,PRIVATE            ! Clarion error
                                           !  code
SaveFileError      CSTRING(255),PRIVATE    ! File system's
                                           !  error message
SaveFileErrorCode CSTRING(255),PRIVATE    ! File system's
                                           !  error code
```

These properties are private. So, you can’t access them in the application. But the macros can! This means that if you wish to know the exact error condition, use the macros in your messages.

For example, you could have code like this in a procedure:

```
GlobalErrors.SetField(‘Quantity’)
GlobalErrors.ThrowMessage(Msg:BadQty, |
  ‘Please use a lower value.’)
```

The `SetField` method places the name of the field in the `%Field` macro.

This could show a message like, "Not enough product on stock as requested in the Quantity entry. Please us a lower value." It would depend on your wording on the `Msg:BadQty` text. To produce the above message, the `Group` definition could look like this:

```
USHORT(Msg:BadQty)
BYTE(LEVEL:Notify)
PSTRING(‘Insufficient quantity’)
PSTRING(‘Not enough product on stock as ↵
 requested in the %Field entry. ‘)
```

History and logging

Now I want to talk about how the lumber industry got its start. Not really, just testing to see if you are still with me. Glad to see that you are.

A new feature added to `ErrorClass` is the ability to save error messages to a log. The log is stored in an ASCII file named ABCError.log. To turn this feature on, set the `LogErrors` property to `True`. If you wish to see what is written to this file, set the `Silent` property to `False`, which means the error is displayed on the screen and then written to a log file. If set to True, then only the error entry is written to the log, but not displayed on the screen.

This is very useful for debugging or troubleshooting purposes. As part of a configuration or option setting in your application, this could be turned on by demand at runtime.

This is where the `Category` property comes into play. You do this with the `SetCategory` method. Generally speaking, this method call could be placed after your code for `AddErrors`. However, you may use this method anywhere where it is appropriate to do so. Remember that I mentioned this near the start of Part 1 of this article. You can categorize your options in addition to the default ABC category. If you do not set the `Category`, ABC is the default. Again, this helps localize where the problem could be coming from.

Summary

In this article I've attempted to open up the mysterious "gray box" in the ABC `ErrorClass`. I also explored the several ways one could use it, by providing three different applications and a template. I showed that you could easily change the behavior of the methods by not changing the class definitions, but simply overriding the default behavior.

You can change the content of the messages, you may add your own. I described how to use the class for support and debug purposes by setting the value of one property.

I also exposed some new behaviors of `ErrorClass` so that you can log errors and have the option of showing these errors or simply logging them.

I hope that you got the idea that you may add your own messages that are tailored made for that special application of yours. I also hope you got the idea that you do not have to edit `ErrorClass` to gain access to the actual error and what you must do to get these conditions.

I am very much aware that some readers will insist they do. If you really wish to test for the actual error with Clarion code, then you may do so. But if you use any of the ABC methods that are somewhere in the derivation chain of `ErrorClass`, you are really doubling your work.

As with any articles I write, I try to convince the reader to be lazy and let any working code that is written on your behalf to do its "thing". All I ask is that you try these concepts. See what happens.

As always, I do like all feedback (even if hostile fire).

Enjoy!

Download the example application (requires C5.5)

Russ Eggen has been using Clarion since 1986. Before joining Topspeed as a consultant in 1996, he was an independent contractor. Currently, Russ is an instructor at SoftVelocity and is writing the curriculum for the classes. His main goal in life is to get a Clarion program to star in a Tom Clancy movie where the program helps the hero save the world.

Reborn Free CLARION online

published by CoveComm Inc.

Search

Clarion MAGAZINE

Feature Interview: Bob Healy

by Dave Harms

Bob Healy is the founder of Matrix Information Systems of Chesapeake, Virginia. Matrix develops and sells several software packages, written in Clarion, including Navigator 5 Constituent Management System for non-profit organizations, and C.E.U. Professional for continuing education units tracking. Bob spoke with Dave Harms, Clarion Magazine's editor.

How did you get started with Clarion?

I was in sales and marketing for a chemical company. When PCs first came out, I went out and got the first XT that hit the street. I went back to school and took all the courses. I started doing some small stuff in dBAse, tracking numbers in VisiCalc. I tried all the xBASE variants, even did Borland's Reflex, which was a decent product in its time. Then one day I found this product on the shelf, Clarion Personal Developer. I opened it up, started playing with it, it's like wow, I've arrived. And 24 hours later I had [the professional version]. It was 2107, or 2108.

I started writing custom applications for people, and it got so I was doing better at that than I was working for the chemical company. So on January 1, 1990, I started my own company. I've been doing custom software development and contracting ever since.

How do you market your products?

The marketing we've been doing on our own software has been word of mouth, and some emailings and mailing lists. We're finally getting to the point where we're using some commercial marketing.

Tell me about the non-profit product. Is it a standalone package?

The non-profit actually front-ends the customer's accounting system.

And it creates batches for import?

Right. We did that for a bunch of legal reasons - the liabilities in writing an accounting system from scratch are too great. They can go with our front-end system, which manages all the contributions, solicitations, and whatever, and they don't have to change their accounting system. I can convince you to change your operating system, your word processing program, anything. If I even hint about changing the accounting system...

What accounting packages do you support?

Cougar Mountain is a very good one that we've done a lot with. There's some other ones out there. Accounting for non-profit organizations is different than general accounting; it's called fund accounting. It's a whole different world – much more regulated. It's nothing for people to pay $25-30,000 for an accounting package. The level of detail is much more intense than in a standard accounting package. I have one customer with a chart of accounts of 36,000 line items.

If they can spend that kind of money on accounting it must be a good market to be in.

They're open to custom solutions, and a lot of them do have a lot of money. There's one big company that sells software for non-profits, and their base package is $20,000.

What's your pricing like?

$2500. And that's everything. There's a huge realm of non-profit organizations that can't afford a $20,000 product. We're trying to hit the smaller to medium-sized organization. It's a pretty comprehensive package – it does everything from marketing, solicitation management, mailing lists, tracks and collects demographic information from contributors. We try to stress it's a marketing tool. We have some complex reporting, and crosstab spreadsheets.

What third party products do you use?

Steve Stockstill's Report Wizard, which I think is the greatest thing since the flush toilet, the spreadsheet wizard and his crosstab wizard. I use Mike Hanson's QBE, because you can do multiple tables. I use Lee White's RPM of course, and CPCS.

What version of Clarion?

Clarion 5B.

How do you handle documentation and training?

One of the things I try to do, and this comes from my sales background, is to do things in the trade terms of the people who will be using the product. That's probably 50% of it right there. I try to make it easy to use. If somebody in the non-profit community is working with our product and they don't understand it, they have a problem. As far as documentation, I use Help & Manual, which as far as I'm concerned is *the* best product around. And I've tested every one out there. It does a very good job of help files, it's very easy and simple to use, and when you're done creating your help files it will generate for you a complete compiled manual with index and table of contents, page numbers, the whole bit. It does a tremendous job, and it's under $200. It's the best thing on the market.

Are you creating HTML help?

I'm doing standard help, but Help & Manual will do HTML help automatically. They have a fully functional trial version. The only thing is on the unregistered version it gives you a little line on the bottom of each window saying it was made with an unregistered copy. My partner, who is by no means a programmer, did most of the documentation, and had absolutely no problem at all. He's a marketing guy. He was development director for a large non-profit, so he was sort of my backbone core of information. He can go and talk to people in the non-profit community and speak their language.

It seems to be a common element of successful vertical market packages. It's important to have someone who knows the market well.

It really is. The other thing that was interesting to me, is, I have a hard time writing documentation. As you know when you write a program and have a lot of time in it, you're intimate with the code. It's difficult, and very anticlimactic, when you're done, to sit down and write documentation from the other side of the desk, as if you've never seen it before. So having someone doing the documentation from the perspective of being the user was very beneficial.

Do you provide on-site training?

We offer training but nobody has ever actually taken us up on it. One thing that we've tried to stress is play around with it for a day - you should be able to figure out everything in here. There are only a couple of areas that require any level of sophistication at all. And even that you can play around with and pretty much figure out. I try to make the program very clean and attractive and easy to use, and have it flow well. And that eliminates a lot of [tech support]. If the program looks jumbled, people have a hard time using it.

I try to keep field sizes the same. I don't like stair-step fields. I use left margin justified prompts. My labeling is pretty clean, pretty understandable. I also try to put enough stuff into the setup, user definable things, to give them some flexibility in some of the definitions of fields. They can define their own field values and things.

A lot of it's back end simplicity too. It's a report-intensive environment. They want to generate a lot of reports. Whether it be Report Writer or a tool like Report Wizard, field naming conventions have to be something they understand. I don't get phone calls, "What's in field B3?" Keeping your field prompts simple and understandable. A lot of programs, you look at the prompts, and you wonder what's that for?

So you let the users design a lot of reports?

Here's a lesson I learned early on. DBAs cringe, but I don't do a normalized database. The first version of my non-profit program was absolutely perfect, pristine, normalized database. I never got off the phone, because it's a report environment. "I need to do a report where I have the client name [one file] and their address [different file] and the contributions [different file] and the persons that they memorialized or recognized [different file]." And I had to explain normalized database and linking files on the phone.

Now I give them a couple of dozen standard reports, give them Report Wizard, and basically say go pick the fields you want, the fields you want to total on, and the way you want to sort it, and that's great! I'm not dealing in a DBA environment. A lot depends on the market you're selling to. If it's going into an organization that does have DBAs and people who write reports, that's a different environment.

But you have to do a lot more work on the back end to maintain your relational integrity.

Not to maintain it, to develop it. It takes more developing on the front end to make it easier for the end user, because you are trying to make it as simplistic as possible. For the average end user Report Wizard meets 99% of all the report requirements I ever get.

It’s saved me countless hours. If I could have only one third party product, that would be the one. Great product.

What are your future development plans? Are you headed to the web?

Future plans include porting most of the products over to Clarion 5.5 and adding the improvements that customers request and taking advantage of some of the new features in the product. I am hoping to give the products more of an Outlook type of interface and look and feel.

I have had some requests for porting one of our products to the web and I hope to have something on this by the first of the year. That will be new C5.5 development, but it is for our C.E.U. (Continuing Education Unit) Management System.. As for the non-profit software I doubt their will be a web interface. As the program maintains accounting information, most of them do not want outside access in any way.

Main Menu | Log In | Subscribe | Links | FAQ | Advertising

Search

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion MAGAZINE

The Clarion Challenge Results!

by Dave Harms

Making Sense of ABC's ErrorClass - Part 3
(Dec 5,2000)

Feature Interview: Bob Healy
(Dec 5,2000)

The Clarion Macro Challenge Results
(Dec 5,2000)

December 2000 News
(Dec 5,2000)

Two weeks ago I described a macro that wraps a variable in a `CLIP()` function, and pointed out that it's tricky writing one macro that can handle a variable in the middle of a line as well as at the end of the line. In the middle of the line you can use `[CTRL+RIGHT]` to find the end of the variable; if the variable is at the end of the line then that keystroke takes you to the next line, so you probably want to use `[END]` instead.

I asked readers to solve this problem, and received several replies in short order. Clearly there are a few developers who are used to thinking outside the box.

As Vince Sorenson pointed out, a solution is easy once you get past the unspoken assumption that the macro has to start with the cursor at the beginning of the variable. Place the cursor at the end of the variable, and use the following macro:

```
) [CTRL+LEFT] CLIP(
```

Slick! Gordon Smith takes a slightly different tack and writes macros to work on selected text. Highlight the variable, then run this macro:

```
[CTRL-X] CLIP( [CTRL+V] )
```

Selecting the variable has several additional benefits. You can use it to `CLIP()` an entire statement, because you've copied the value to the clipboard it's at hand if you need to search for the next instance of the variable or statement, something also pointed out by Steven Hill. Very nice!

Jim Gambon and Rick Martin both added one more twist to the copying to clipboard trick. They chose to edit c55edt.ini and set the keystroke for MarkWord to some value such as CtrlW. The macro then looks like this:

```
[Ctrl+W][Ctrl+X] CLIP( [Ctrl+V])
```

Now why didn't I think of that?

Clarion MAGAZINE

Converting PowerBrowse from Legacy to ABC

by Alan Telford

Do you have an application that you would like to convert to ABC but haven't? Would you like some guidelines for when to convert and what to convert? Do you have existing legacy templates with no ABC equivalent e.g. PowerBrowse? All of the above? Well keep reading - this article highlights some of the problems in converting large applications over to ABC, gives some homegrown Laws of conversion, and then solves the legacy PowerBrowse problem.

But first, some history. I may not be an expert at conversion but I certainly do have experience. I first started programming 6.5 years ago. My first job was to write a conversion program to carry data over from a legacy program to the TopSpeed file format (six weeks). My second job was to convert the legacy program into Clarion for Dos 3.0 (one to two years)

My third job was to take the recently converted Clarion for Dos program and convert it (again!) into Clarion for Windows (I first started converting with CW1.5 but didn't actually release it until CW2003)

My current job is to maintain the existing application (now in C55rc2), while releasing add-on functionality every three to four months.

On my Todo list: Convert from Legacy to ABC; Convert from TopSpeed into SQL; Convert from procedural code to object-oriented code; *Convert, convert, convert* …

I've wanted to convert to ABC for nearly 12-18 months now, but I haven't. Why not? The cost! Who's going to pay for it? The user doesn't pay for conversions – after all, a conversion should affect the internal structure and program organization (which the user doesn't see) while hopefully not affecting the windows and reports (which the user does see).

The next big problem is time, or lack of it. In my case the software package I've written is used by McDonald's Family Restaurants to manage their back office functions from payroll, inventory control, cash management and staff rostering through to basic accounting. It's rather large and consists of over 20 DLLs in main stream use, and another 10-20 utility DLLs or EXEs for occasional use. Like many multi-DLL applications there is one common DLL which contains all file declarations and global data. If I converted to ABC I would have to first convert the files DLL which would instantly stop all the other DLLs from working. So the conversion is all or nothing. And in my estimation it would take two to four months to convert my application. I don't have time to stop the clock and convert them all, while still releasing new functionality every three to four months.

If I do convert I will of course encounter the next problem. BUGS, BUGS, BUGS… A once stable application that has been tampered with will inevitably have some new bugs. Conversions are definitely risky, and the biggest risk is the users would see this as a backward step instead of a forwards step, and decide to take their business elsewhere. What is a programmer to do?

Enter my laws of conversion.

Law 1: Don't Convert

If you can get away without doing a conversion, then don't convert. Anytime you change the code, you decrease the stability of the code, and increase the possibility of bugs.

If it still works, and the user doesn't want it updated, then *leave it alone*. Converting costs time and money so don't do it.

Sometimes unfortunately, you can't leave the application alone:

- Your last legacy programmer is about to leave, and there will be no one left to maintain the legacy application
- You have a lot of changes to make or new features to add to an application, and you wish to write these using ABC
- You've run out of work?
- You have spare time on your hands?
- You're starting to feel the pressure as SoftVelocity says that C55 is the last release that will support the Legacy templates.

In these cases then I advise …

Law 2: Do The Bare Minimum

If you have to convert, then do as little as possible. The less you change, the less you will have to fix later. As I read my version notes, I find that some of my procedures haven't been changed in three to four years, and are unlikely to change in the next three to four years either. Guess what? The code in these procedures is working fine. It's running like clockwork.

Other procedures seem to need changing every few months, and you guessed it – that is where most of the bugs occur.

So do the smallest amount of change possible required to convert, and you will hopefully minimize the number of new bugs that you introduce.

But what if the bare minimum doesn't cut it? What if a particular browse requires some major new functionality? This is where the old 80-20 rule comes in. 80% of the time you can get by with the minimum. The other 20% of the time you have to apply the next law.

Law 3: Don't Convert – Rewrite

Laws 1 and 2 are designed to save you time which you can then spend on Law 3.

When I look at my original Clarion for Dos code I wrote when I was first learning, I want to curl up and die. Looking at code I wrote two years ago is not so serious but usually embarrassing. Even in code I wrote six months ago I can usually see room for improvement. I don't know about you but there's very little code I'm 100% satisfied with. If I have to convert some code, and converting involves risk, then I might as well use the opportunity to redesign and rewrite the code so I'll be happy with it (for at least another six months anyway).

This option is particularly good if you're adding new features which you can charge the customer for, because then they end up paying for the conversion.

The above three laws are fairly general, and should serve you well for many types of conversions. For the specific Legacy to ABC conversion Simon Brewer wrote some great articles in July and August about how to start. I very much enjoyed his articles, but was left wondering what to do about the dreaded …

PowerBrowse

I recently read a newsgroup comment: "Free of PowerBrowse at last – Yeah". Nice for some I suppose. Why all this strong antipathy towards a third party template?

- it's legacy,
- it's been abandoned (no ongoing development),
- it still has bugs (fortunately none that bother my users too much)
- there's no ABC conversion program provided

With Law 2 in mind – do as little as possible – I set to thinking how I could convert the 162 PowerBrowse legacy procedures in my application into ABC format with as little work, and as much safety, as possible.

Wouldn't it be nice if there was an ABC PowerBrowse, with the same embed points? Then I could just export from legacy, tweak the TXA and import into ABC. But how do I do this? Enter the plan.

1. Take a PowerBrowse procedure and copy the code into a SOURCE procedure.
2. Compile the procedure to see what needs changing
3. Make the changes
4. Test to see if it works
5. Study the templates to see how to automate the changes
6. Modify the templates
7. Test again

Armed with the plan, I started my PowerBrowse conversion.

1. Take a PowerBrowse procedure and copy the code into a SOURCE procedure.

First I created a basic PowerBrowse procedure with single sort order, locator, and inline entry but no embedded source.

Then I created an ABC app, using the same dictionary, and added a source procedure. Then, using cut and paste, I copied all the data declarations from my PowerBrowse into the data embed, and all the program code into the code embed.

My PowerBrowse now existed (without any templates) in an ABC app. This is a bit like the traditional one-way templates that other languages provide. Once my PowerBrowse is in source code I have to maintain it via source, and can't use the template power any more.

2. Compile the procedure to see what needs changing

Then I took the brave step and compiled: There were only 14 errors, which fell into four groups.

- Opening and closing the file
- Saving and restoring the window position
- Standard error messages
- Relational update/delete of file (for inline entry)

I began to feel hopeful – this could work?!

3. Make the changes

The next step was to solve each of the 14 errors by replacing the problem legacy code with new ABC code which did the same job.

Opening and closing of the file (five errors)

Legacy file opening code:

```
IF <FileName>::Used = 0
  CheckOpen(<FileName>,1)
END
BIND(M40:RECORD)
<FileName>::Used += 1
```

Legacy file closing code:

```
<FileName>::Used -= 1
IF <FileName>::Used = 0 THEN CLOSE(<FileName>).
```

This translates into the following ABC code:

```
Relate:<FileName>.Open
Relate:<FileName>.Close
```

After making the above changes to the source procedure the results were: five errors down, and nine to go.

Saving and restoring the window position (two errors)

Legacy code:

```
INIRestoreWindow('BrowseName','IniFile.ini')
INISaveWindow('BrowseName','IniFile.ini')
```

A quick look at a standard ABC browse reveals the code required:

```
INIMgr.Fetch('BrowseName',QuickWindow)
INIMgr.Update('BrowseName',QuickWindow)
```

Standard error messages (four errors)

Legacy code:

```
Button# = StandardWarning(Warn:UpdateError)
Button# = StandardWarning(Warn:SaveOnCancel)
Confirm# = StandardWarning(Warn:StandardDelete)
CASE StandardWarning(Warn:DeleteError)
```

A normal legacy browse wouldn't have these messages at all – in fact the job would already be finished. These occur because PowerBrowse has edit-in-place functionality just like the ABC browses do, except about a hundred times easier to use (well, that's a *slight* exaggeration)

ABC has a GlobalErrors class to handle these sort of error messages. By comparing the legacy error messages with those in "ABERROR.TRN" I came up with the following replacements:

```
Button# = GlobalErrors.Message(Msg:RetrySave,Button:Yes|
  +Button:No+Button:Cancel,Button:Cancel)
```

```
Button# = GlobalErrors.Message(Msg:SaveRecord,Button:Yes|
  +Button:No+Button:Cancel,Button:Cancel)
```

The `Warn:StandardDelete` and `Warn:DeleteError` are handled directly by calling the ABC relational delete method and didn't need replacements.

Keep in mind Law 2: I'm not trying to improve the PowerBrowse code at all, simply get it working under ABC as quickly as possible.

Relational update/delete of file (three errors)

Legacy code:

```
RISnap:<FileName>
Error# = RIUpdate:<FileName>()
IF RIDelete:<FileName>()
```

The first line is used by Legacy templates to setup for cascade updates. This is handled automatically by ABC templates therefore a replacement is not needed. The other two lines translate to:

```
Error# = Relate:<FileName>.Update()
IF Relate:<FileName>.delete(1) = LEVEL:Benign
```

The relational delete in ABC is a great improvement, because the one call handles prompting the user to confirm the delete, retrying the delete if unsuccessful, and informing the user of any failure. So one line in ABC replaced 19 lines in legacy. Hmmm – maybe this ABC isn't so bad after all.

After I made the above changes to the source procedure, the ABC application compiled successfully.

4. Test to see if it works

With much anticipation I ran the application and tested the functionality.

Browse worked fine. Locators worked fine. Inline entry worked fine.

YES! YES! YES! IT WORKED!

With two hours work (reading those ABC manuals) my PowerBrowse was in an ABC application, but stored in a SOURCE template. Remembering Law 2 (Do the bare minimum) this would be a good-enough solution for 80% of the PowerBrowse procedures I have. They don't need to be changed. But I like a bit more elegance so I also completed the following steps:

5. Study the templates to see how to automate the changes
6. Modify the templates
7. Test again

Now I have a set of equivalent templates which allow PowerBrowse procedures to run under ABC as easily as under Legacy. Click the link at the end of the file to download an install program which installs both the Legacy and ABC PowerBrowse templates with libraries using C55 Gold.

My new ABC PowerBrowse still has a lot of legacy file handling code, but its all Clarion code which compiles and works. That's good enough for me.

> **Note:** The original germ of this idea came to me as I looked at the UnivReport template from Larry Teames of CPCS reporting fame. When Larry upgraded from Legacy to ABC he used Law 2. He didn't rewrite the whole templates using an OOP class, but simply made his templates coexist with the ABC templates, and call ABC calls when needed. My first reaction when discovering this was "That's a bit cheeky!" I felt a bit cheated. Then I noticed one immediate benefit. There were no conversion problems with UnivReport procedures. Exactly the same embed points exist under ABC as under Legacy. There was no new OOP reporting system to learn – my legacy system worked fine under ABC. I figured if Larry could do it for his templates, then I could do it for PowerBrowse.

Some intrepid users might consider doing the same for the Clarion Legacy templates. Wouldn't that make an interesting proposition: an ABC compatible version of Legacy templates which allowed us to use our existing Legacy procedures inside new ABC applications. This might be pushing it, but it's certainly an interesting idea! I've wanted to use some of the new ABC file handing code in my legacy procedures for a long time. This would enable me to do it.

> *Editor's note: Both Larry and Alan's approaches are similar to the OOP design pattern known as "Adapter." This is a class that allows two classes with incompatible interfaces to communicate. The analogy isn't perfect since in this case templates are used and there is procedural legacy code on one side of the equation, but the principle is well-established in software design.*

Back to PowerBrowse: how do I actually manage the conversion now?

Converting via TXA

Now that the ABC PowerBrowse template support is there it's a straightforward job to convert from Legacy to ABC.

1. Use File|Selective Export to export all PowerBrowse procedures in an application to a TXA file.
   Note – just choose PowerBrowse procedures, no Forms or Windows or any other template types.
2. Load the TXA into the editor.
3. Do the following global search and replaces

Old	New
FROM ToolCraft PowerBrowse	FROM TCraftAbc PowerBrowse
NAME ToolCraft	NAME TCraftAbc
NAME Clarion CloseButton	NAME TCraftAbc CloseButton

Save the modified TXA

4. Create a new ABC application using the same dictionary, or use an existing ABC application (as long as the same dictionary is used)
5. Add the PowerBrowseLibrary global extension template to the ABC application.
6. Import the modified TXA into your ABC application.
7. Compile

Obviously I've simplified the process as this ignores any embedded code. If this code is incompatible with ABC then you will find out about it. The big difference is that now you don't have to worry about the template-generated PowerBrowse code.

Summary

Keep in mind my three laws of conversion:

Law 1: Don't (if you can get away without converting then don't convert)

Law 2: Do The Bare Minimum (minimize risk of new bugs by changing as little as possible)

Law 3: Don't Convert – Rewrite (if you need to make major changes then rewrite the procedure using your greater experience and insight you've gained over the previous few months).

I don't recommend that you create new ABC browses in PowerBrowse format. This is simply a method for carrying the existing PowerBrowses into ABC. There are still a few bugs in PowerBrowse and nobody is fixing them, so don't make the problem any bigger than it already is. Remember that this approach is just a great application of Law 2: Do the Bare Minimum.

Now you have the luxury of converting ABC PowerBrowses to ABC Browses as and when you need to.

Download the C5 templates

Download the C5.5 templates

For the most recent version see Alan Telford's home page at:

http://homepages.paradise.net.nz/alantelf/

Alan Telford has been programming in Clarion since 1994. He is the Chief Software Developer at Maxtel Software Ltd, a New Zealand software company specializing in writing back office computer solutions for McDonald's Family Restaurants and other *similar markets.*

Main Menu | Log In | Subscribe | Links | FAQ | Advertising

Reborn Free CLARION online

published by CoveComm Inc.

Search

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion MAGAZINE

Completely Dynamic Report Orders and Breaks

Converting PowerBrowse from Legacy to ABC
(Dec 12,2000)

Completely Dynamic Report Orders and Breaks Part 1
(Dec 12,2000)

December 2000 News
(Dec 12,2000)

Part 1 of 2

By Steve Parker

Leaving aside the Report Formatter, a Clarion report, at its core, is really very simple. A report is a `Print()` in a loop (see "Conditional Sort Orders and Page Breaks in Reports: Part 1"). From the perspective of the templates: a Clarion report is a Process with a `Print()`statement and access to the Report Formatter.

The specific method that distinguishes a Process template procedure from a Report template procedure is `TakeRecord`.

In a Process, `ProcessManager.TakeRecord` (Parent Call) - instantiated as `ThisProcess.TakeRecord` - is where the record is retrieved and read. After the Parent Call, code can be executed to manipulate the data just read to accomplish the Process' specific purpose. Standard stuff.

In a Report, `TakeRecord` - instantiated as `ThisReport.TakeRecord` - performs the same function, providing the same embeds. But, and this is where it differs from a simple Process, `TakeRecord` (Parent Call) is followed by the `Print()` statement:

Figure 1. TakeRecord/`Print()` embeds

```
ThisReport.TakeRecord PROCEDURE

ReturnValue          BYTE,AUTO

! Start of "Process Method Data Section"
! [Priority 3500]

SkipDetails BYTE
! [Priority 8500]

! End of "Process Method Data Section"
  CODE
  ! Start of "Process Method Executable Code Section"
  ! [Priority 500]


  ! [Priority 4500]

  ! Parent Call
  ReturnValue = PARENT.TakeRecord()
  ! [Priority 5001]
        !After TakeRecord
  PRINT(RPT:detail)
  ! [Priority 8000]
        !After Print()
  ! End of "Process Method Executable Code Section"
  RETURN ReturnValue
```

The `Print()` statement "prints a report structure to the Windows default printer or the destination specified by the user in the Windows Print... dialog." "Report structures" are page headers, details, page footers and page forms. Note that it is no longer possible to use the `Print` statement to print a string directly to the printer, as could be done in CDD.

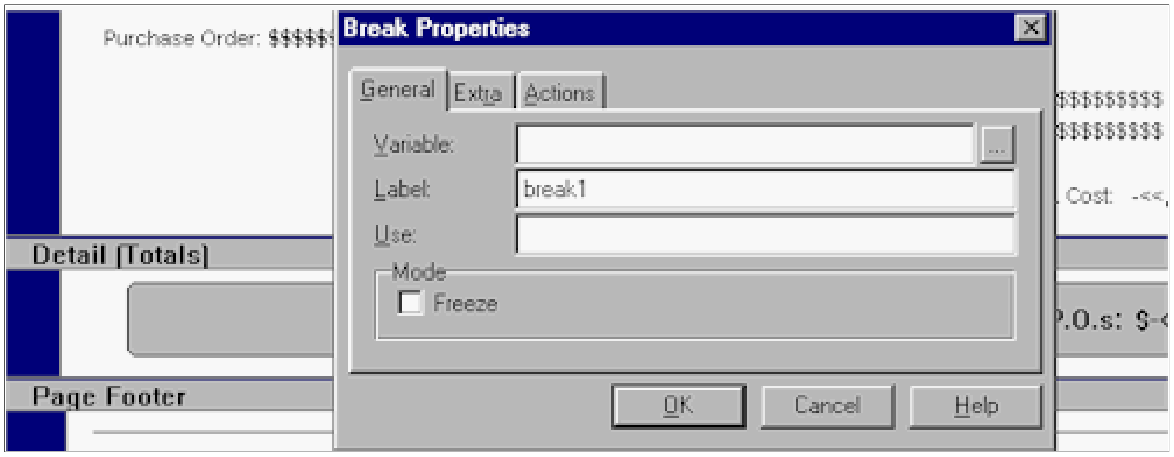
On reflection, this simple fact of `Print()`-in-a-loop conveys enormous flexibility to the developer. And, this flexibility does not entail using exotic code or techniques. (Of course, if you're a CPCS user, you needn't bother with anything that follows. Larry Teames had done all the hard work for you.)

Subtotals, For Example

Probably the three most common customizations to reports are grand totals (covered in "Conditional Sort Orders and Page Breaks in Reports: Part 2"), handling of no-records (see "Template Writing Made Easier: The Template Wizatron") and subtotals.

To implement subtotals, select Bands | Surrounding Break and click on the detail band around which the break is to be placed. Then, select the variable on which to break.

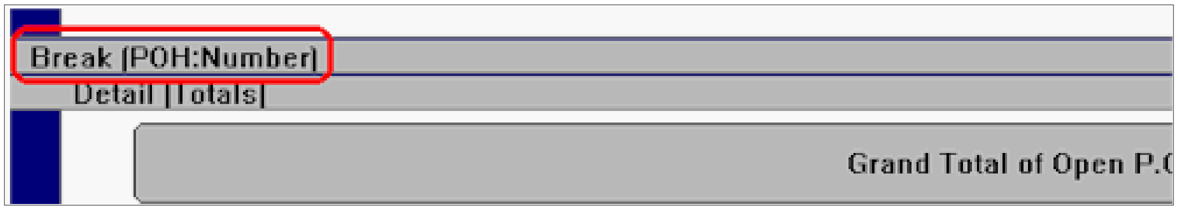
Figure 2. Placing a group break in the Report Formatter



Variable:
Label: break1
Use:
Mode
Freeze

This places a break on the band. Nothing else, just a `Break` which triggers the associated logic in the report engine.

Figure 3. Break in a report



This generates the following declaration in the report structure:

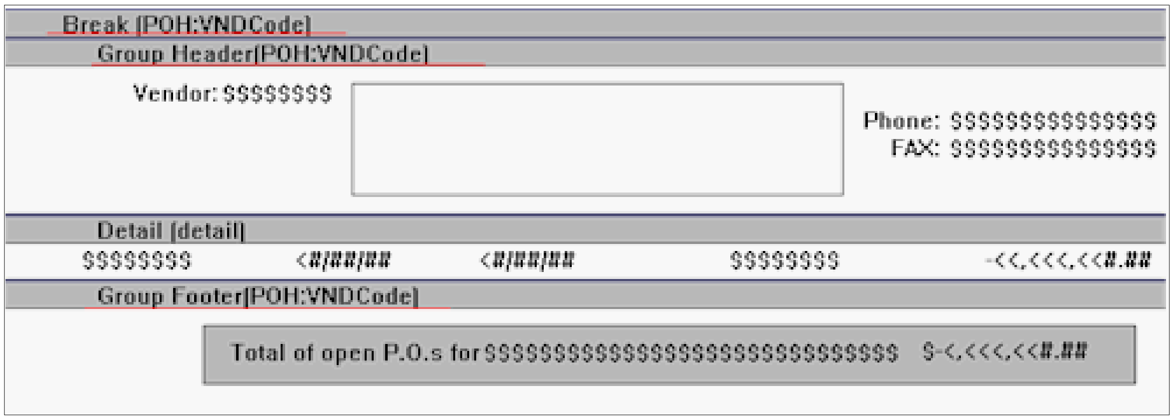
```
vendor BREAK(POH:VNDCode),USE(?vendor)
```

If you want to place fields in a Group Header, such as vendor name and address for example, click Bands | Group Header and click on the break band. This will populate a Group Header into which fields can be populated:

```
vendor BREAK(POH:VNDCode),USE(?vendor)
HEADER,AT(0,0),FONT(,10,,,CHARSET:ANSI)
STRING('Vendor:'),AT(781,73),USE(?String9),TRN,#ORIG(?String9)
STRING(@s8),AT(1313,73),USE(POH:VNDCode),TRN,#ORIG(?String10)
TEXT,AT(2094,94,2958,688),USE(Address),#ORIG(Address)
```

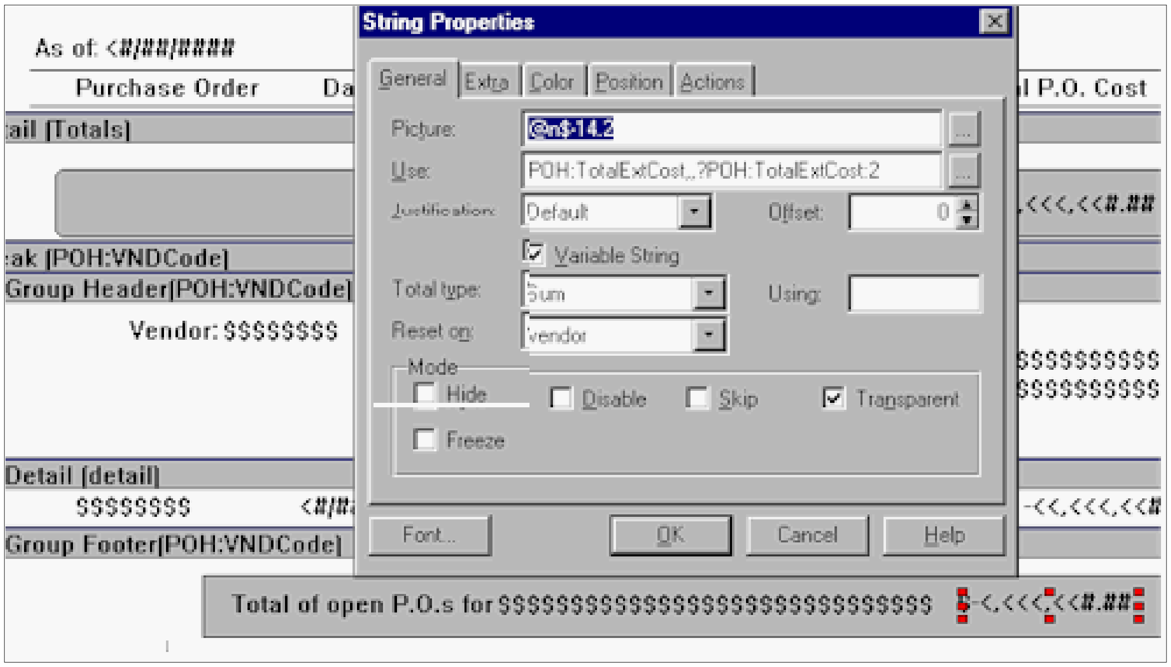
Selecting Bands | Group Footer and clicking on the break band will populate a corresponding Group Footer.

Figure 4. Report Formatter with Header and Footer



The Group Footer is the appropriate place to populate total fields holding subtotal fields.

Figure 5. Subtotal field in Group Footer



Picture: @n$14.2
Use: POH:TotalExtCost,,?POH:TotalExtCost:2
Justification: Default
Offset: 0
[✓] Variable String
Total type: Sum
Using:
Reset on: vendor
Mode: [ ] Hide [ ] Disable [ ] Skip [✓] Transparent [ ] Freeze

Subtotals are typically implemented based on a key or, for sort orders needed only in reports run periodically, an index (which, of course, ought to be rebuilt just before running the report).

"Typically," reports requiring subtotals are based on a key but this is *not* a requirement of the Report Formatter. When creating the breaks necessary to implement subtotals, I am not constrained by either the key declared in the file schematic or, in fact, any other key for the base file. In fact, I could select a field from a child file or from any file in the schematic. I can, should I wish, even use a local variable for a break.

Obviously, then, very great care must be taken in setting up breaks. If the order in which records are read does not group all values in the desired field together, the same field may appear multiple times in the report. Or, when values *are* grouped together, it is possible for needed details to be hidden.

Suppose I am creating a report on outstanding purchase orders by vendor from my POHeader file. In this case, I would break on the vendor field. But if the file is being processed in PO number order, I could easily end up with:

```
Vendor: ABC Distributors
        PO Number: 2363 $ 345.56
Vendor: Midnight Sales
        PO Number 6793  $9874.48
        PO Number 6874  $ 566.11
Vendor: ABC Distributors
        PO Number: 7643 $   1.23
```

instead of the expected:

```
Vendor: ABC Distributors
        PO Number: 2363 $ 345.56
        PO Number: 7643 $   1.23
Vendor: Midnight Sales
```

```
        PO Number 6793  $9874.48
        PO Number 6874  $ 566.11
```

On the other hand, if I am processing the `PODetail` file in vendor order, breaking on vendor, I may end up with all the purchase orders for a given vendor lumped together, not broken down by PO Number. This report would require nested breaks on vendor and PO number.

While keys and indices are typically used in creating subtotals, the greatest care must be taken in selecting both the key and the break field(s). It’s a simple point, but it is imperative that I ensure that the processing order somehow corresponds to the desired output.

Now suppose that the report is sorted not on a key but on a runtime variable (as in the People example which ships with Clarion 5 and 5.5) or suppose the end user wants to be allowed to select the fields on which to provide subtotals.

If the report is sorted on runtime variables or a runtime selection, it is not possible to set breaks at design time. This is because breaks set at design time are *set* to a variable and the order of the breaks is also set. If sorting on runtime variables, the order may change and some of the breaks may not be used. Fixed bands, in a fixed order, simply won’t accommodate what I need to do. Similarly, if the user is allowed to elect which subtotals to print and which not to print, fixed breaks are meaningless.

In other words, in these not exactly unheard of scenarios, I cannot use the Report Formatter to create either breaks or subtotal bands.

But, if a Report is just a `Print()`in a loop, making a runtime calculation of when, where and how to break this is not all that difficult (at least, conceptually).

The Theory

In order to produce dynamic group breaks, I need to know what fields are being used as break fields, when the value in any of those fields changes and (a minor point) which group breaks are desired by the user.

Which fields are breaks? By and large, this is not an issue because, in some manner, I will control which fields are offered as breaks and, therefore, which are selected. If I know which fields are available and how they are made available, I can easily set flags telling me which were actually used.

Similarly, which breaks are requested by the user is something I will also control. On first glance, I would guess that I can use a series of flags so that I know which particular breaks are requested.

This leaves knowing when a field value changes.

And this is not all that difficult. If I were in a hand-coded loop, I would do something like:

```
Loop Until Access:POHeader.Next()
  If POH:Number <> SaveNumber
    !need break
  End
  SaveNumber = POH:Number
End
```

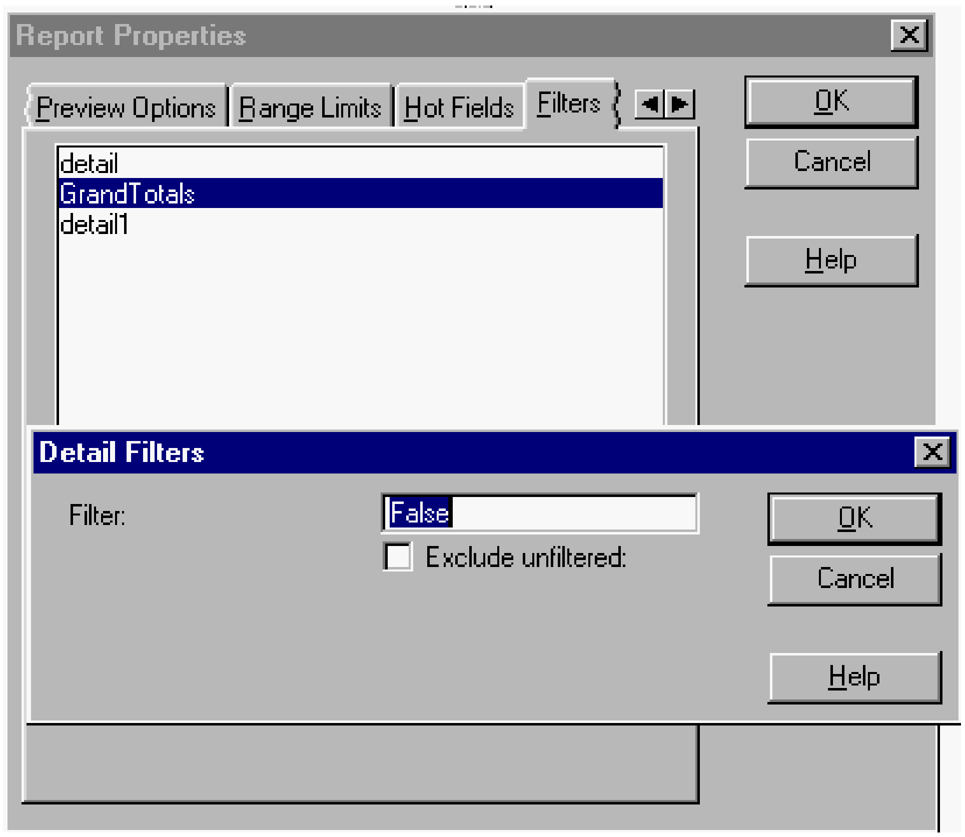
It is my preference, in code like this, to accumulate totals myself. So, this code would fill out like this:

```
Loop Until Access:POHeader.Next()
  If POH:Number <> SaveNumber
    !need break
    SubTotal = 0                    !re-initialize subtotal
  End
  SaveNumber = POH:Number           !save current number
  SubTotal += POH:Amount            !accumulate total
End
```

The only issue is how to produce a break band. But, this turns out to be easy also.

If I create a band, a standard detail band, I can populate the `SubTotal` field in it. I can set it up so that it doesn't print automatically by setting its filter to "False."

Figure 6. Preventing a band from printing normally



Then, because this band *is* a valid report structure, I can use the `Print()` statement to make it print when I want it to:

```
Loop Until Access:POHeader.Next()
  If POH:Number <> SaveNumber
    Print(RPT:Totals)
    SubTotal = 0
  End
  SaveNumber = POH:Number
  SubTotal += POH:Amount
End
```

Now, because the Report template provides the `Loop` for me, I don't need my own loop structure. I simply need to embed

```
If POH:Number <> SaveNumber
```

```
  Print(RPT:Totals)
  SubTotal = 0
End
SaveNumber = POH:Number
SubTotal += POH:Amount
```

in `TakeRecord`, after Parent Call (priority 5001).

This works because `POH:Number` is from the current record but `SaveNumber` is from the previous record (notice that `SaveNumber` isn’t updated until after this check). If a new PO has been read, the `If POH:Number <> SaveNumber` check is valid.

If the number has changed, the band with the subtotal will print before anything is done with the current record (like printing it, which occurs *after* this embed).

There are two problems with this approach.

First, the first record read will always satisfy the `POH:Number <> SaveNumber` condition and, therefore, trigger the printing of the subtotal band. To correct this I declare a variable, `FirstLoop`, a byte, with an initial value of 1 and add this code:

```
If FirstLoop
  SaveNumber = POH:Number
  FirstLoop = False
End
```

before the code shown above. This ensures that `SaveNumber` is not empty at the first `SaveNumber <> POH:Number` check.

Second, after the last record is read, the `POH:Number <> SaveNumber` condition will *not* be checked. Therefore, the final subtotal band(s) will not be printed.

Remedying this is also straightforward.

As discussed in "Conditional Sort Orders and Page Breaks in Reports: Part 1", `ThisWindow.AskPreview` is called after all records have been read and processed but before the report is closed. So, in `ThisWindow.AskPreview`, before Parent Call, I place this code:

```
If ~Aborted
  Print(RPT:Totals)
End
```

If more than one field can be used for a break, I just repeat the code for each field, making the necessary changes. And, if the user is allowed to select which breaks to print, then I wrap it in a condition:

```
If FirstLoop
  SaveNumber = POH:Number
  SaveVendor = POH:Vendor
  FirstLoop = False
End
If BreakOnNumber
  If POH:Number <> SaveNumber
    Print(RPT:Totals)
    SubTotal = 0
  End
  SaveNumber = POH:Number
  SubTotal += POH:Amount
End
If BreakOnVendor
  If POH:Vendor <> SaveVendor
    Print(RPT:VenTotal)
    VenSubTotal = 0
  End
  SaveVendor = POH:Vendor
  VenSubTotal += POH:Amount
End
```

Summary

Dynamic breaks are straightforward, in theory, at least: create a detail band for each possible subtotal and prevent it from printing. If the subtotal is wanted, check whether the value has changed. If so, issue a `Print()` statement for the appropriate band and re-initialize totals.

Does the theory work? Next time, I'll walk you through a real report with user defined breaks.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reborn Free CLARION online

published by CoveComm Inc.

Search

Clarion MAGAZINE

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion Magazine's Holiday Schedule

This week's double issue wraps up the 2000 publishing year, and the Clarion Magazine office is now closed for the holiday. I wish all of you the best of the Christmas season, and I look forward to seeing you in 2001!

As most of you know, Clarion Magazine publishes four times per month, and in months with five Tuesdays the magazine takes a week off. January is one of those months, and so the next issue of Clarion Magazine will be out Tuesday, January 9th, 2001.

While everyone's away celebrating Christmas, however, the ClarionMag elves will be hard at work, implementing a new web server and magazine delivery software. You can see a sneak preview at www2.clarionmag.com.

The new site doesn't look much different from the current site, and in fact is missing some links and graphics. But under the hood a lot has already changed. The current web site mainly serves static HTML; much of the non-article information on the new site is generated from a MySQL database. This makes many tasks much easier, including formatting and publishing articles, creating article cross-references, and managing links to other sites.

As well, the new site (when fully operational) will sport a number of new features, including (but not limited to):

- improved authentication: At present, when you attempt to access a back issue you haven't subscribed to, you get a login screen. The new site informs you that you are a valid subscriber but you don't have access to that page.
- better ad management for third party advertisers
- frequent user surveys
- reader feedback pages
- an option to update your personal information, including online password changes

The new site is being developed using a combination of Clarion and Java technology.

Servlets, JavaServer Pages, custom tags, and JavaBeans handle the delivery of magazine content to readers, making heavy use of data from a MySQL database, while a Clarion application manages that database. If you're interested in this technology, keep an eye out for my upcoming book on JSPs, Servlets, and MySQL (title not finalized), to be published by Hungry Minds (formerly IDG) in March, 2000.

Dave Harms

Publisher

Interfacing Satellite Forms Applications and Clarion for Windows

by Randy Rogers

Satellite Forms is a rapid application development environment for 3Com Palm devices that is similar in many ways to Clarion for Windows. This article highlights the Satellite Forms development process and the steps necessary to successfully interface to a Clarion for Windows application using the Satellite Forms Hotsync Control.

I first used Satellite Forms several years ago while working on a project that required a questionnaire to be implemented on a hand held device. We currently use a Palm interface in two of our municipal accounting applications; building inspections and meter readings.

Those applications have requirements that may not be familiar to a lot of Clarion developers, so I began by looking through the sample applications that ship with Clarion for Windows. I wanted to find an application that had some component that might be enhanced by having a Palm device interface. After a bit of searching, I decided on the autolog.app located in the c:\c55\examples\autolog directory. One feature of this application allows the user to keep a history of trip logs for various vehicles. I thought that this would be the perfect place for a Palm interface. A salesperson could keep the trip logs on a Palm device and later upload those logs to the PC application.

Any good project begins with a specification, so here are the requirements I have defined for this project:

- The Palm application interface has to be similar to the UpdateTripLog form procedure in the application.
- The Palm application needs to be able to store multiple trip logs for any of the vehicles in the existing database.
- The Palm application has to be easy to use. Two buttons, upload and download, will be added to the AutoLog application toolbar to control the database synchronization process.

I will use Puma Technologies Satellite Forms Enterprise Edition Version 4.0.0 (Build 715.1) EE and SoftVelocity Clarion for Windows Version 5.5.

So, where to begin. Actually, the process is relatively simple and consists of the following basic steps:

- build the Satellite Forms application
- import the Satellite Forms database files into the Clarion dictionary
- implement the Satellite Forms ActiveX Control

- and finally, write the Clarion synchronization code for the buttons.

Building the Satellite Forms Application

The Satellite Forms IDE is very similar to many application development environments. There is a menu and tool bar across the top, a left panel that provides a tree view of the Application Contents, and a right panel that is used to visually define the Palm screen form layouts and also to enter scripts. Satellite Forms supports most of the controls you would expect: button, entry, string, list, drop list, radio, check, memo, bitmap image, etc. Satellite Forms scripting language is similar to Visual Basic though not as robust. Scripts can be associated with form events or certain control events.

One of the biggest hurdles in Palm development is getting your application to talk to the Palm device. Doing that directly involves writing a conduit, which isn't particularly easy. It's much simpler to create a Palm application that uses the Palm desktop to synchronize to a database on the PC, and then have your Clarion application use that same database.

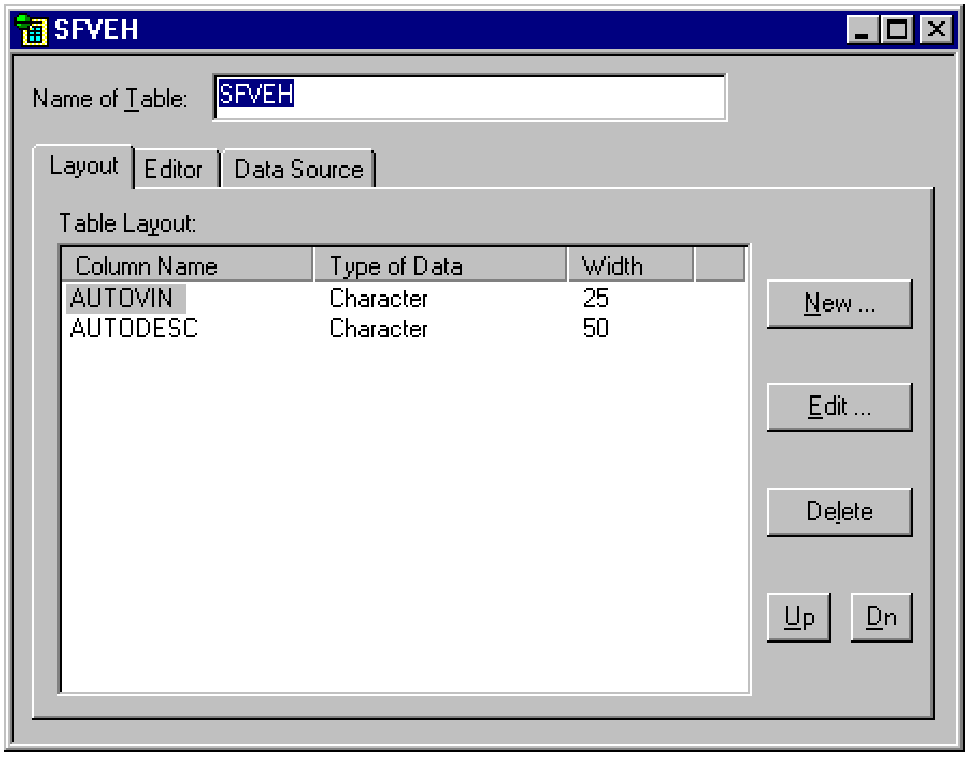
Satellite Forms supports two database formats: Microsoft Access and DbaseV. Since Clarion readily supports the DbaseIV format (a subset of DbaseV), I will use that format for the example database.

To begin, open the Satellite Forms App Designer and select File/New from the menu to create a new application. In the left hand Contents of Application panel, double click on Properties. The application properties dialog will appear. Enter `TripLog` as the name of the application and select DbaseV as the Desktop database; leave the other fields as defaulted for now; press OK. Select File|Save As from the menu and save your application as `TripLog.sfa`.

Creating the Database

The next step is to create the SatelliteForms database. The application will need two files, one to hold vehicle information for lookups and the other to hold the trip log details. Name the vehicle table `SFVEH` and the triplog table `SFTRI`. Add the vehicle table first by right clicking on Tables and selecting Insert Table from the popup menu. Change the Name of Table to `SFVEH`, change the first column name to `AUTOVIN`, data type to character and width to 25. Press the new button to add a second column, `AUTODESC`, with data type character and width of 50. The `AUTOVIN` will hold the corresponding field from the vehicles file from AutoLog.dct. The `AUTODESC` field will contain a description of the vehicle containing color, year, make, and model. When you are finished, the definition should look like Figure 1.

Figure 1. Creating the Satellite Forms database



SFVEH

Name of Table: SFVEH

Layout | Editor | Data Source

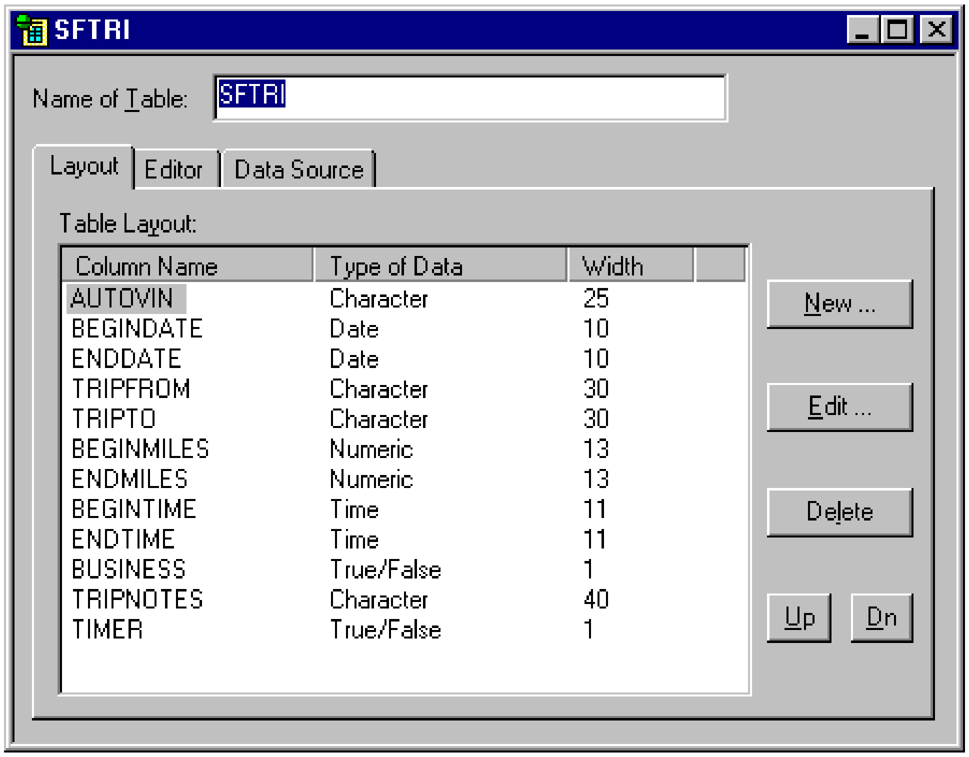
Table Layout:

Column Name	Type of Data	Width
AUTOVIN	Character	25
AUTODESC	Character	50

This file will be used for a drop down vehicle selection list on the Trip Log form and should be recreated whenever the user selects the download button from the AutoLog application toolbar. Since this file is for lookup purposes only it will not be processed by the upload button.

Next, create the `SFTRI` table. Right click on Tables and select Insert Table from the popup menu. Change the Name of Table to `SFTRI` and enter the following column definitions, as in Figure 2.

Figure 2. Creating the `SFTRI` table



SFTRI

Name of Table: SFTRI

Layout | Editor | Data Source

Table Layout:

Column Name	Type of Data	Width
AUTOVIN	Character	25
BEGINDATE	Date	10
ENDDATE	Date	10
TRIPFROM	Character	30
TRIPTO	Character	30
BEGINMILES	Numeric	13
ENDMILES	Numeric	13
BEGINTIME	Time	11
ENDTIME	Time	11
BUSINESS	True/False	1
TRIPNOTES	Character	40
TIMER	True/False	1

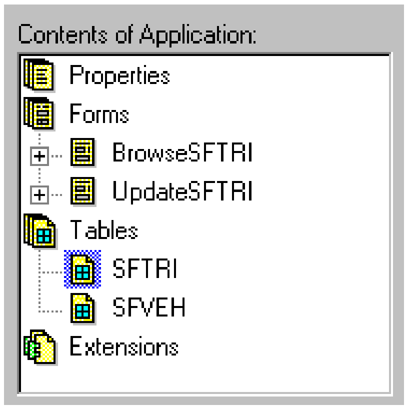
Enter a dummy record for each table by clicking on the editor tab then save your application. The Satellite Forms App Designer will prompt you to create the database files; name them sfveh.dbf and sftri.dbf respectively.

Ok, now you are ready to start developing the Palm application screens. Since the Palm interface needs to be similar to the Clarion application interface, start with a browse listing the Trip Logs currently stored on the Palm device. The list will contain vehicle description, trip date and destination and will have standard update buttons. Right click on Form 1 and select Properties from the popup menu. Change the Name of Form to `BrowseSFTRI`, leave the number of pages set to 1, enter `SFTRI` in the Linked Table field, and leave the User Permissions as defaulted. Press the OK button.

There is one more form which you can define now; this is a bit out of sequence but will save some jumping around later and simplify the description of the forms creation process. Right click on Forms in the Application Contents tree and select Insert Form from the popup menu to create a new form. Right click on the new form in the Application Contents tree and change the Name of Form to `UpdateSFTRI`, set the number of pages to 3 (more on this later), enter `SFTRI` in the Linked Table field, and leave the User Permissions as defaulted. Press the OK button.

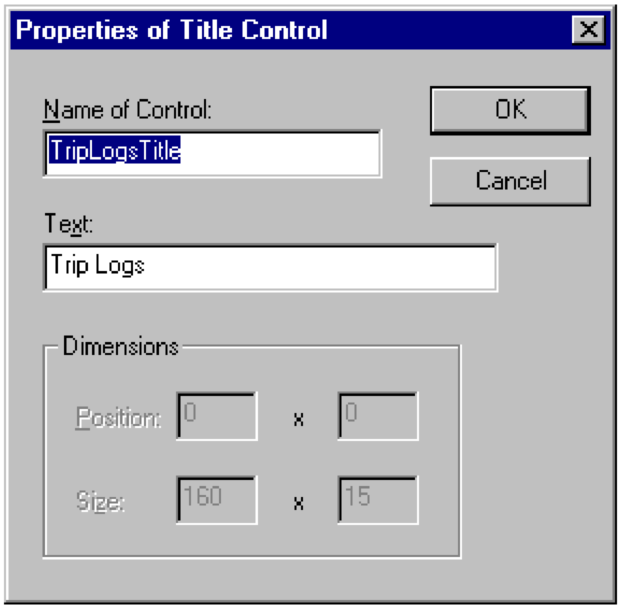
At this point you should have two forms: `BrowseSFTRI` and `UpdateSFTRI`. There should be no controls on the forms. You should also have two tables defined, `SFVEH` and `SFTRI`, and empty database files sfveh.dbf and sftri.dbf should exist in the working directory.

Figure 3. The application contents



The next step is to populate the controls on the `BrowseSFTRI` form. Double click on `BrowseSFTRI` in the Application Contents tree and populate a title control onto the window, then right click on the control and set its properties as follows:

Figure 4. Populating a title control


Properties of Title Control

Name of Control: TripLogsTitle

Text: Trip Logs

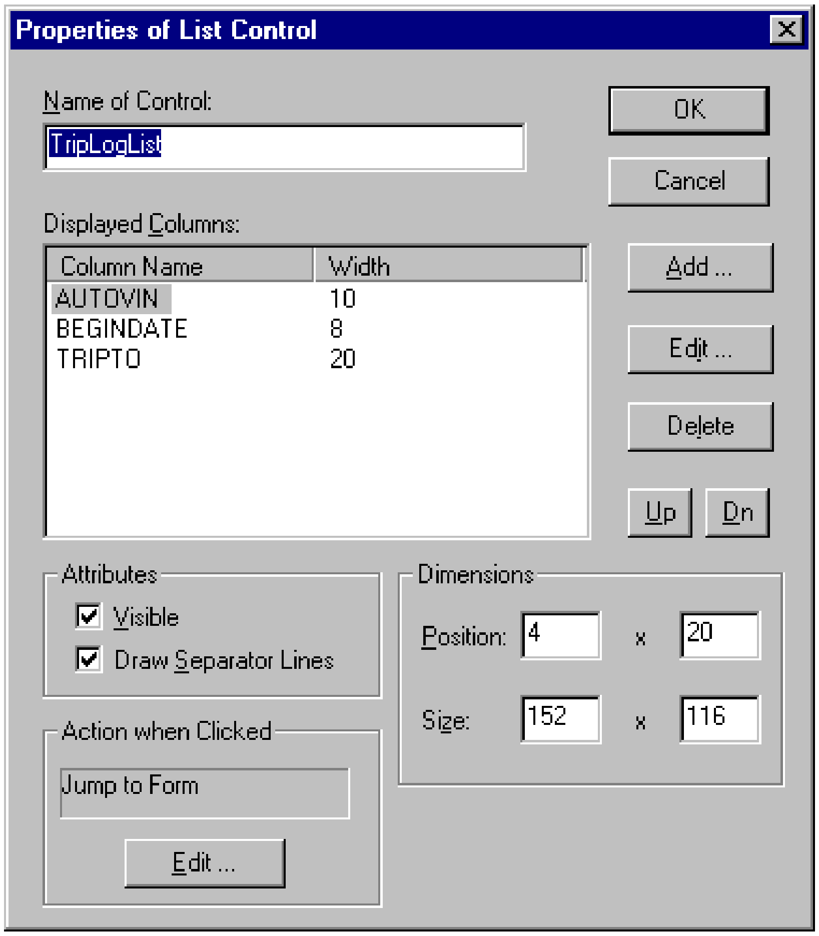
Dimensions

Position: 0 x 0

Size: 160 x 15

Next populate a list box control with the following properties.

Figure 5. Populating a list box control



Name of Control: TripLogList

Displayed Columns:

Column Name	Width
AUTOVIN	10
BEGINDATE	8
TRIPTO	20

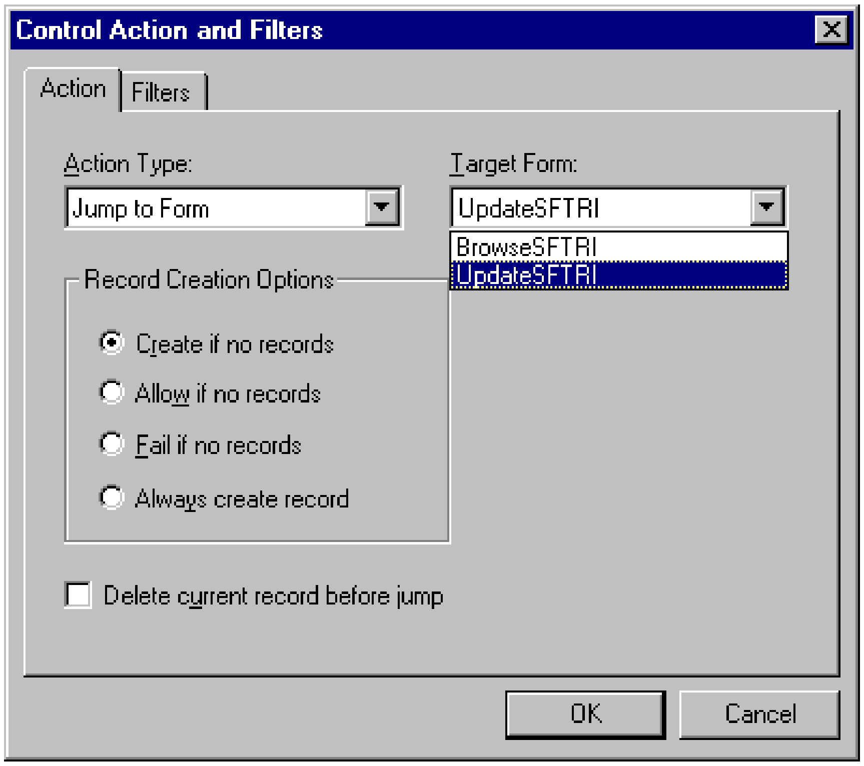
Attributes
- [✓] Visible
- [✓] Draw Separator Lines

Action when Clicked: Jump to Form

Dimensions
- Position: 4 x 20
- Size: 152 x 116

If you are following along, you will notice that when using the Jump to Form Click Action you must select the target form from a list of existing forms. Because you have previously created the `UpdateSFTRI` form, it appears in the pick list.

Figure 6. Setting the target form



Action Type: Jump to Form

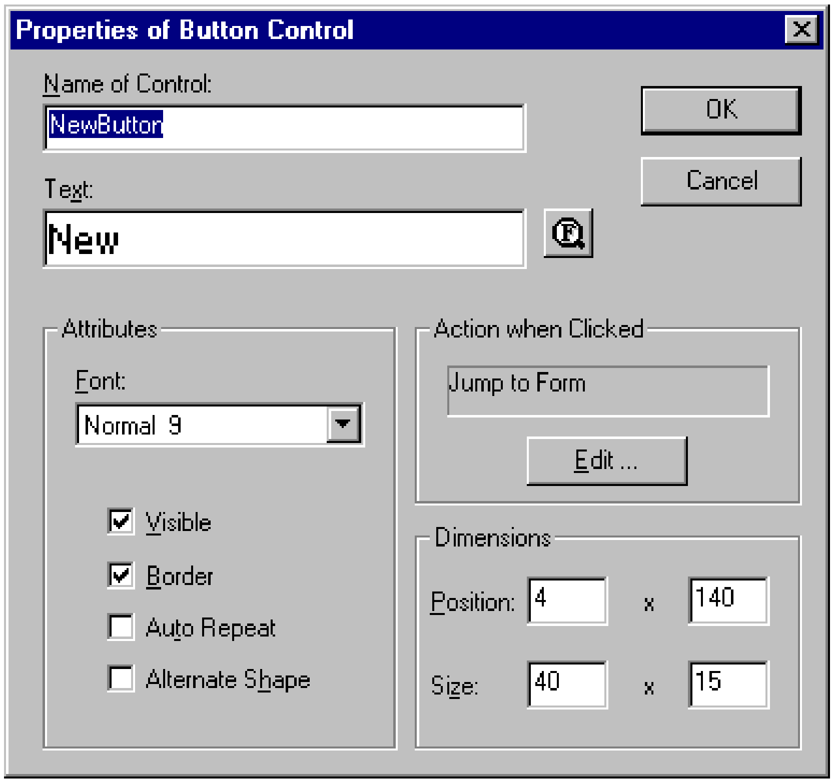
Target Form: UpdateSFTRI

Record Creation Options
- (●) Create if no records
- ( ) Allow if no records
- ( ) Fail if no records
- ( ) Always create record

[ ] Delete current record before jump

Finally add the three update buttons.

Figures 7-10. Adding the update buttons



Properties of Button Control

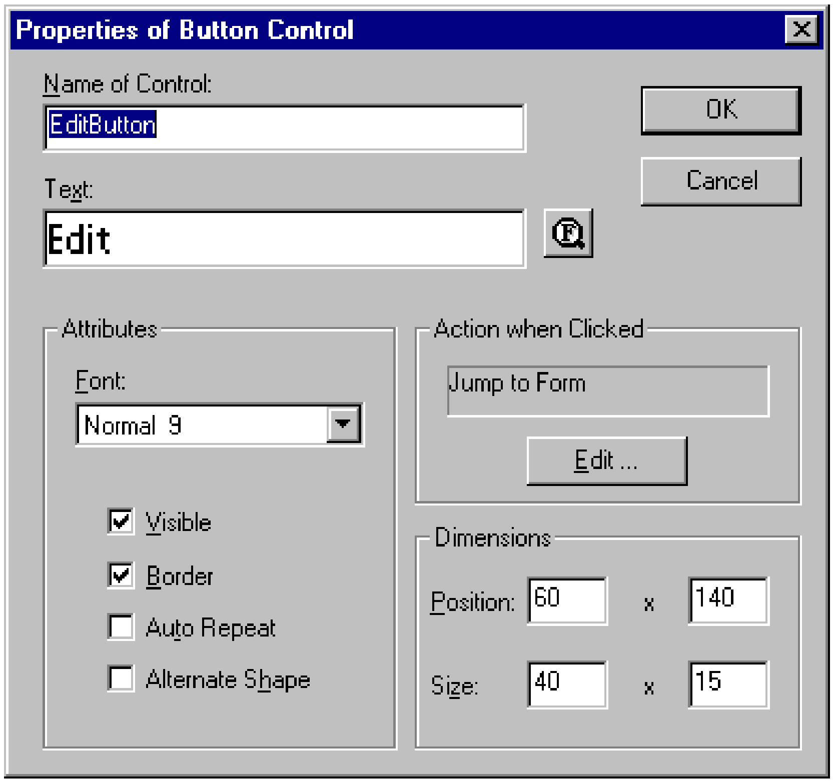
Name of Control: NewButton

Text: New

Attributes
Font: Normal 9
☑ Visible
☑ Border
☐ Auto Repeat
☐ Alternate Shape

Action when Clicked
Jump to Form

Dimensions
Position: 4 x 140
Size: 40 x 15



Properties of Button Control

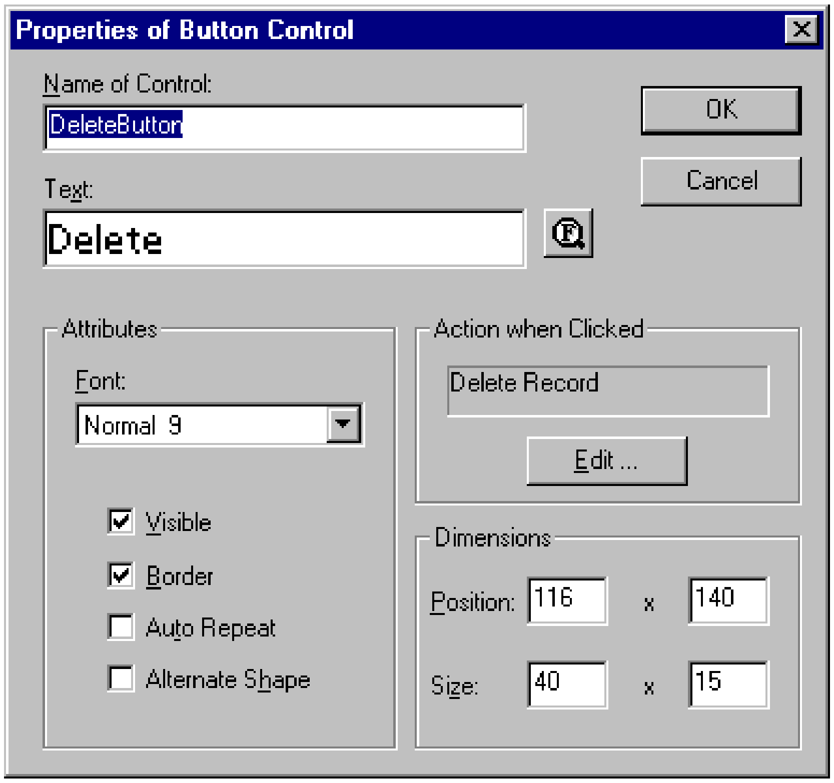
Name of Control: EditButton

Text: Edit

Attributes
Font: Normal 9
☑ Visible
☑ Border
☐ Auto Repeat
☐ Alternate Shape

Action when Clicked
Jump to Form

Dimensions
Position: 60 x 140
Size: 40 x 15



Properties of Button Control

Name of Control: DeleteButton

Text: Delete

Attributes

Font: Normal 9

[✓] Visible

[✓] Border

[ ] Auto Repeat

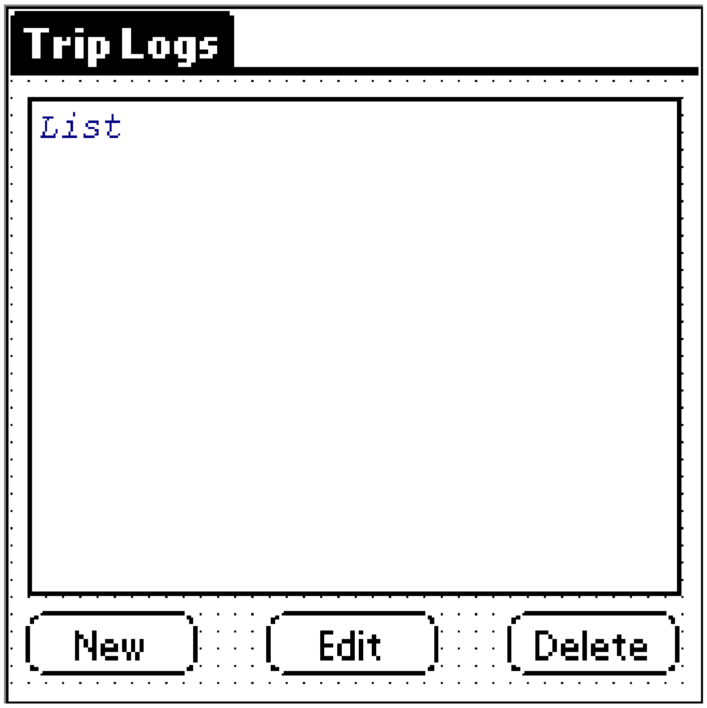
[ ] Alternate Shape

Action when Clicked: Delete Record

Dimensions

Position: 116 x 140

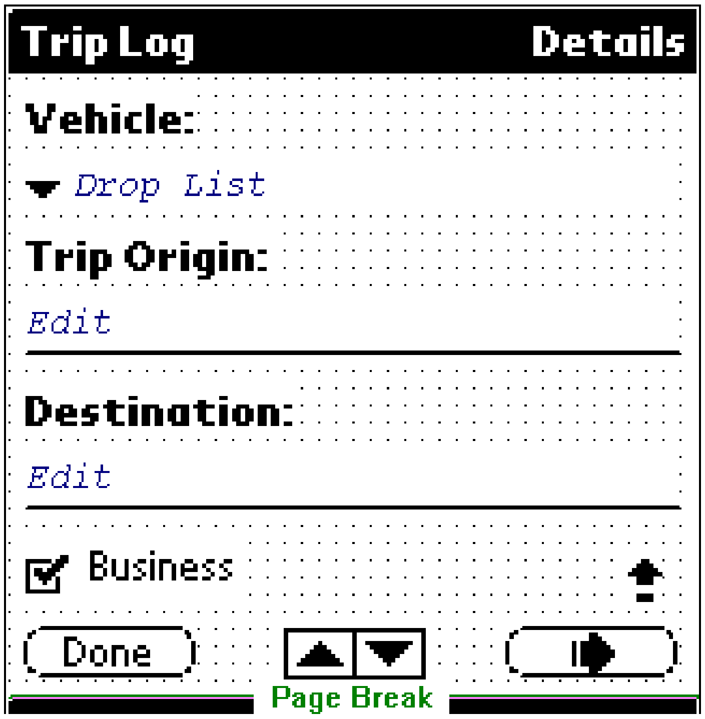
Size: 40 x 15



You should be getting the feel of how easy it is to work in the Application Designer. It has a few quirks but, once you adjust, application development is really quite rapid. All that remains is one more form, some scripts, and the ActiveX for the interface with Clarion.

One of the biggest adjustments you will have to make is getting used to working with the Palm's small screen area. My first attempt was to try to squeeze everything onto one page. There really isn't much room on that little screen, which is why I've created three pages for this form (declared earlier in form properties). Satellite Forms makes it easy to navigate between the different pages of an update form. I will take each page in turn and point out anything special. The details can all be found in the attached satellite forms application.

Figure 11. Page one of the update form



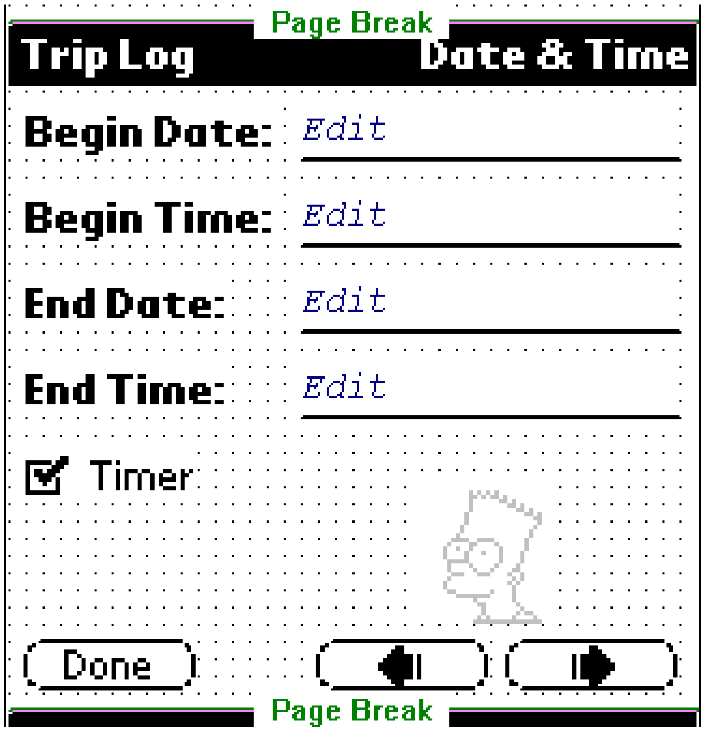
Each page gets its own Title Control (page heading). To get the appearance I wanted, I added extra spaces to the text to cause the black background to stretch all the way across the top of the form. All the prompt strings are done in bold face.

The vehicle is selected from a dropdown list. This control works well as it presents a list of choices for the user but can actually retrieve a different value. That is, you can display the vehicle make and model and retrieve the `AUTOVIN` field from the lookup table.

Trip Origin and Destination are free form edit fields.

The upward pointing arrow is the graffiti shift indicator. The button with the right arrow goes to the next page. The up and down triangles navigate between trip log records, which lets the user edit different records without having to return to the browse. The Done button returns to the `BrowseSFTRI` form.

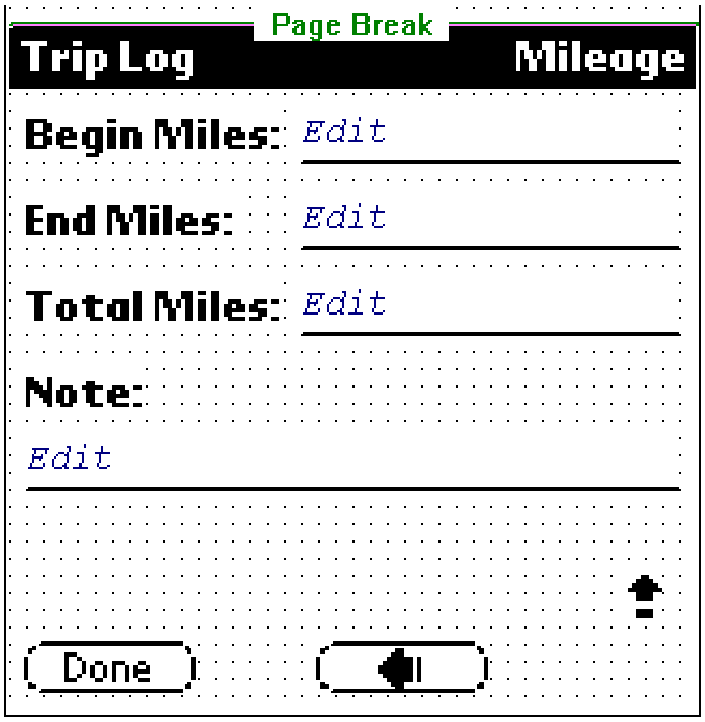
Figure 12. Page two of the update form



The Date and Time fields use Satellite Forms AutoKeyboard feature which allows the runtime engine to present a calendar for the user to select a date or a numeric keyboard for time entry.

I decided to add a timer feature so a person using the Palm application could have the end date and time constantly updated. This wasn’t really required but was easy enough to implement using a script in the Form's OnTimer event. Then I animated some of my favorite cartoon characters…

Figure 13. Page three of the update form



Total Miles is calculated in the Form's `OnChange` event. Validation is performed on the Begin and End Miles whenever the page is validated.

Scripts are page specific so you need to check the page number in your scripts when working with multipage forms otherwise you will receive errors. I found certain scripting functions did not work as described in the documentation and was not able to obtain support online. I never did call the tech support line. It sure makes me appreciate what a tremendous benefit we have with the Softvelocity and comp.lang.clarion newsgroups.

If you normally synchronize desktop applications with your Palm device then you should change your default hotsync settings while in development mode. You will find yourself doing a lot of hotsync operations during the testing and debugging of your Palm application. I normally have my HotSync action settings set to synchronize my Outlook Contacts and Calendar but changed them to do nothing during the Palm development cycle.

You should now have a working copy of the TripLog Palm Application loaded on your Palm device. Satellite Forms is an easy to use programming environment for developing database applications for the Palm device. It is extensible through third party tools or you can write your own extensions in C using CodeWarrior. The new Enterprise Edition version 4.0 has a free runtime, and an evaluation version is available for download from their website. An in-depth description of Palm OS Programming With Satellite Forms can be found at http://www.wirelessdevnet.com/training/Palm/Palm_sf.html.

One annoying feature of the Satellite Forms Application Designer is the fact that it stores absolute pathnames for the databases used at design time. For this reason you must install the Satellite Forms Application files supplied with this article into the C:\Program Files\Satellite Forms EE\Projects\TripLog directory or you will receive errors when you try to save the application.

Importing The Database

Importing a Satellite Forms database into Clarion is a snap. Just open the AutoLog.dct file in the Clarion dictionary editor and use the File/Import File option and select the Dbase IV file driver to import the sfveh.dbf and sftri.dbf files. Enable file creation for both files, and clear the full pathname. The assumption here is that the Satellite Forms database files will exist in the program working directory and have the default names. Press Close and save the changes to the AutoLog.dct file.

Implement the Satellite Forms ActiveX Control

In deriving this example, I wanted to localize, as much as possible, the HotSync interface. I decided to create a window procedure, `winSatForms`, that would take a string caption and a `HotSyncQueue` as parameters. The `HotSyncQueue` would consist of a command code and a filename. At this point there are just two possible commands: `CopyTableToPalmPilot`, and `GetTableFromPalmPilot`, and two files that could be transferred. Place the following code in the indicated global embed point:

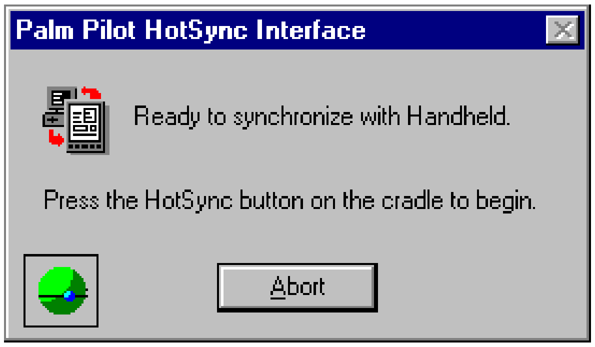
```
Global After Global Includes", Priority 4000
Status_HotSyncStart                EQUATE(1)
Status_HotSyncEnd                  EQUATE(2)
Status_HotSyncCommandComplete      EQUATE(3)

                      ITEMIZE(1),PRE(HsCmd)
CopyTableToPalmPilot    EQUATE
GetTableFromPalmPilot   EQUATE
                      END

HSQUEUETYPE           QUEUE,TYPE
iHsCmd                  UNSIGNED
szParam                 CSTRING(256)
                      END
```

The purpose of the `winSatForms` procedure is to provide a common user interface, process the commands in the passed queue, and return a boolean value to indicate success (true) or failure (false). So, if you are following along, add a new window procedure named `winSatForms`.

Figure 14. A sync procedure window

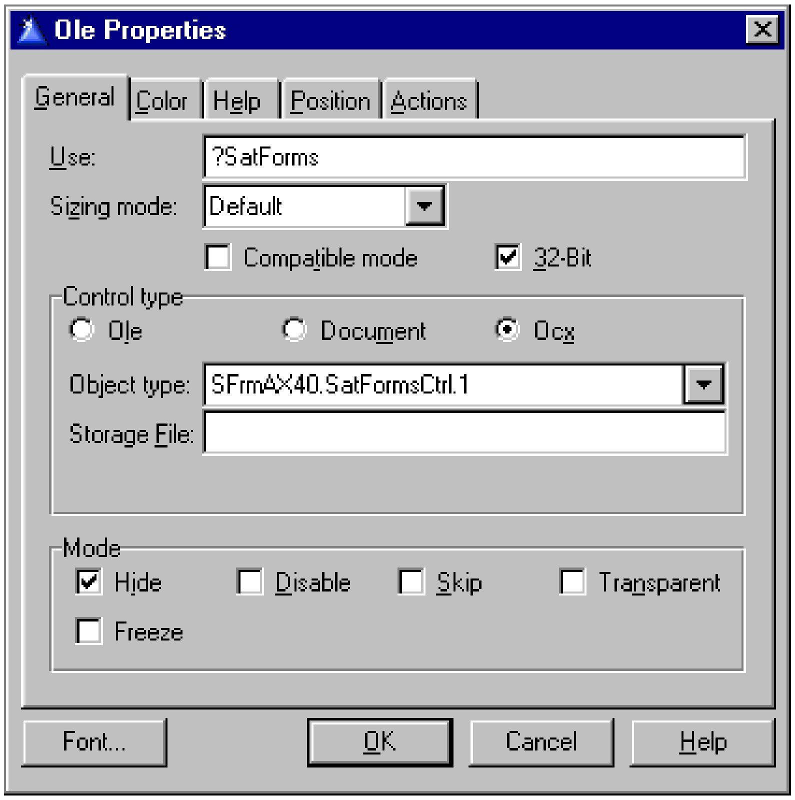


Ready to synchronize with Handheld.

Press the HotSync button on the cradle to begin.

This is what the window I created looks like. The control in the bottom left is the satellite forms ActiveX control. I populated it onto the window using the OLE control template with the following properties:

Figure 15-16. Populating the OLE control



Use: ?SatForms

Sizing mode: Default

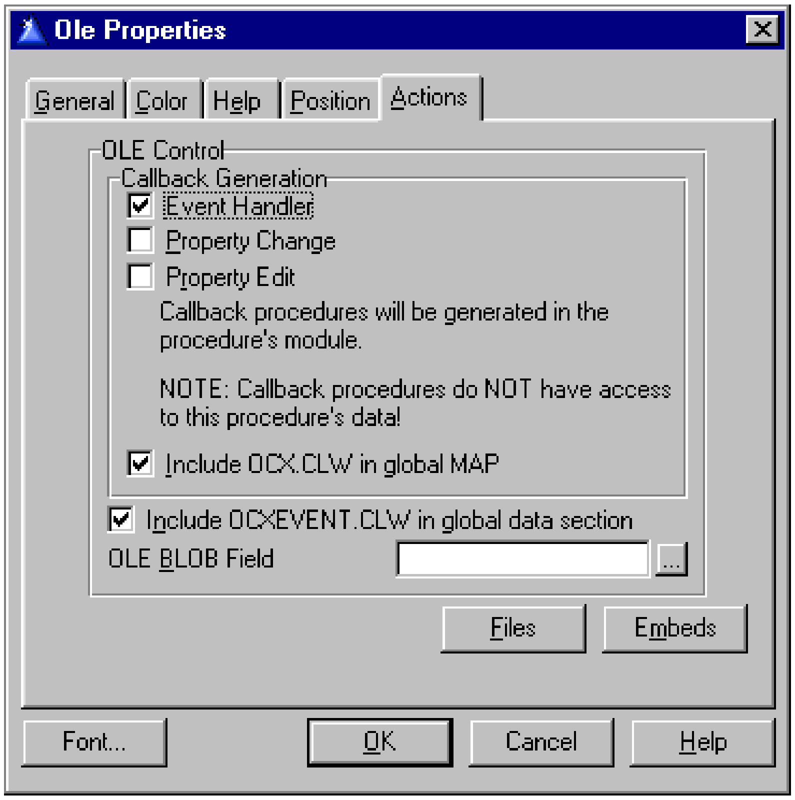
[ ] Compatible mode [✓] 32-Bit

Control type: ( ) Ole ( ) Document [✓ Ocx]

Object type: SFrmAX40.SatFormsCtrl.1

Storage File:

Mode: [✓] Hide [ ] Disable [ ] Skip [ ] Transparent [ ] Freeze



OLE Control
Callback Generation
☑ Event Handler
☐ Property Change
☐ Property Edit
Callback procedures will be generated in the procedure's module.

NOTE: Callback procedures do NOT have access to this procedure's data!

☑ Include OCX.CLW in global MAP
☑ Include OCXEVENT.CLW in global data section
OLE BLOB Field

Here's a quick explanation of how the ActiveX control works. Once the ActiveX control is enabled, it waits until the user presses the hotsync button on the cradle. When that happens, the ActiveX control posts an event to the event callback procedure (which is automatically created by the OLE control template). At this point the Clarion code needs to issue commands to the control to indicate the desired action(s). These requests are stored by the control and don't actually get executed until the event callback procedure returns.

If the event handler is to gain access to the queue that was passed to the `winSatForms` procedure, it will need a module level queue reference variable. The `winSatForms` procedure will initialize the reference variable which the event callback procedure will use to cycle through the queue records. After retrieval, each queue record is copied to a local group so the callback can access the member elements and issue the appropriate commands to the control. There will also be a module level variable to store control status information.

Place the following code in the indicated embed points:

"Module Data Section", Priority 4000

```
iSatFormsStatus UNSIGNED
Qref            &QUEUE,AUTO
HotSyncEvents   ITEMIZE(EVENT:USER),PRE(EVENT)
HotSyncStart            EQUATE
HotSyncEnd              EQUATE
HotSyncCommandComplete  EQUATE
                END
```

"This Window Init", Priority 9001:

```
Window{PROP:Text} = szCaption   !Set window caption
Qref &= Q                       !Initialize Queue reference variable
?SatForms{'Enabled'} = TRUE     !Activate the HotSync Control
```

"This Window TakeWindowEvent", Priority 2800

```
  OF EVENT:HotSyncStart
     Window{PROP:Hide} = TRUE
  OF EVENT:HotSyncCommandComplete
  OF EVENT:HotSyncEnd
```

```
    POST(EVENT:CloseWindow)
```

"Control Events, ?AbortButton Accepted", Priority 5000

```
  iSatFormsStatus = FALSE
  POST(Event:CloseWindow)
```

"OLE Event Handler, Data Section", Priority 5000:

```
  Q GROUP(HSQUEUETYPE),PRE()
  END
  CreatorID       CSTRING(16)
  SDDI_PluginName CSTRING(16)
```

"OLE Event Handler, Code Section", Priority 5000:

```
  CASE OCXGETPARAM(ref,1)
  OF Status_HotSyncStart
     CreatorID = 'SMSF'
     SDDI_PluginName = 'Sddi_PalmDB.dll'
     LOOP I# = 1 TO RECORDS(Qref)
       GET(Qref,I#)
       Q = Qref
       EXECUTE(Q.iHsCmd)
         OLEControlFEQ{'CopyTableToPalmPilot(' & Q.szParam & ',' & |
                       CreatorID & ',' & SDDI_PluginName & ',0,1,0)'}
         OLEControlFEQ{'GetTableFromPalmPilot(' & Q.szParam & ',' & |
                       CreatorID & ',' & SDDI_PluginName & ',0,1,0)'}
       END
     END
     POST(EVENT:HotSyncStart)
  OF Status_HotSyncCommandComplete
     iSatFormsStatus = OCXGETPARAM(ref,2)
     POST(EVENT:HotSyncCommandComplete)
  OF Status_HotSyncEnd
     iSatFormsStatus = OCXGETPARAM(ref,2)
     POST(EVENT:HotSyncEnd)
  END
```

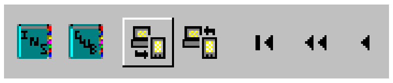
Satellite Forms has changed its format significantly with version 4.0 which can cause some difficulty with the HotSync ActiveX events, particularly if you are converting from an application that used an earlier version of the control. `GetTableFromPalmPilot` and `CopyTableToPalmPilot` now take six parameters (version 3.5 only required the Filename parameter). A few comments on the parameters:

- `Filename`: Full path to file and file name must be capitalized.
- `CreatorID`: The `CreatorID` specified in the Satellite Forms App Designer Project properties. Use "SMSF" when using the SDK and your application specific ID when using the RDK.
- `SDDI_Plugin_Name`: This is the name of the SDDI dll "Sddi_PalmDB.dll"
- `CreateFlag`: Reserved for future use, use 0
- `VersionMajor` and `VersionMinor`: This must match the parameters specified in the Satellite Forms App Designer Project properties.

Synchronization Buttons

The final step is to add the Toolbar buttons to the application toolbar.

Figure 17. The toolbar buttons



I added two buttons to the Main procedure toolbar. I placed them to the left of the browse toolbar buttons. The left button is the "Download to Handheld" button; the right button is the "Upload from Handheld" button.

The "Download to Handheld" process needs to create a sfveh.dbf file based on data stored in the vehicle file. It also needs to create an empty sftri.dbf file and copy both files to the handheld.

Here is the code I placed in the `?DownloadButton Accepted Event, Priority 5000` embed:

```
CREATE(SFTRI)
CREATE(SFVEH)
Access:SFVEH.Open
Access:SFVEH.UseFile
Access:Vehicles.Open
Access:Vehicles.UseFile
SET(VEH:AutoVINKey)
LOOP UNTIL Access:Vehicles.Next()
  SFV:AUTOVIN = VEH:AutoVIN
  SFV:AUTODESC = FORMAT(VEH:Year,@N4) & ' ' & |
                 CLIP(VEH:Color)      & ' ' & |
                 CLIP(VEH:Make)       & ' ' & |
                 CLIP(VEH:Model)
  Access:SFVEH.Insert()
END
Access:Vehicles.Close
Access:SFVEH.Close
FREE(Q)
Q.iHsCmd = HsCmd:CopyTableToPalmPilot
Q.szParam = PATH() & '\SFVEH.DBF'
ADD(Q)
Q.iHsCmd = HsCmd:CopyTableToPalmPilot
Q.szParam = PATH() & '\SFTRI.DBF'
ADD(Q)
IF winSatForms('Download tables to Handheld',Q)
   !Success
ELSE
   !Failed
END
```

The "Upload from Handheld" process needs to parse data out of the sftri.dbf file and add the appropriate records to the `TripLog` file. It does not need to process the sfveh.dbf file.

Here is the code I placed in the `?UploadButton Accepted Event, Priority 5000` embed:

```
FREE(Q)
Q.iHsCmd = HsCmd:GetTableFromPalmPilot
Q.szParam = PATH() & '\SFTRI.DBF'
ADD(Q)
IF winSatForms('Upload tables from Handheld',Q)
   !Success
   Access:TripLog.Open
   Access:TripLog.UseFile
   Access:SFTRI.Open
   Access:SFTRI.UseFile
   SET(SFTRI)
   LOOP UNTIL Access:SFTRI.Next()
     TRI:AutoVIN      = SFT:AUTOVIN
     TRI:BeginDate    = SFT:BEGINDATE
     TRI:EndDate      = SFT:ENDDATE
     TRI:TripFrom     = SFT:TRIPFROM
     TRI:TripTo       = SFT:TRIPTO
     TRI:BeginMiles   = SFT:BEGINMILES
     TRI:EndMiles     = SFT:ENDMILES
     TRI:BeginTime    = DEFORMAT(SFT:BEGINTIME,@T4)
     TRI:EndTime      = DEFORMAT(SFT:ENDTIME,@T4)
     TRI:BusinessTrip = CHOOSE(SFT:BUSINESS='F',0,1)
     TRI:TripNotes    = SFT:TRIPNOTES
     Access:TripLog.Insert()
   END
   Access:SFTRI.Close
   Access:TripLog.Close
ELSE
   !Failed
END
```

Save, Compile, and Run the Autolog application. Press the DownloadToHandheld toolbar button then press the hotsync button on your Palm cradle. If all goes well your application should load the sfveh.dbf and empty sftri.dbf files onto the handheld. Use the TripLog application on the Palm device to enter a couple of TripLogs. Place the Palm device back in the cradle then press the "Upload from Handheld" button on the AutoLog application toolbar. Press the Hotsync button on the cradle and the database will be updated with the trip logs you entered on the handheld.

That about wraps it up for this article. I hope you have found it informative and helpful. If you have any questions or comments, please feel free to contact me.

Download the TripLog application

Download AutoLog application

Randy Rogers is a data processing professional with over 35 years of experience in a wide variety of industries including accounting, municipal government, insurance, printing, and pharmacoeconomics. He is the president of Keystone Computer Resources and creator of NetTools, Queue Edit-in-Place and Screen Capture Tools for Clarion application developers. Randy has a degree in Mathematics from Florida State University and has taught programming at the community college level.

Search

Clarion MAGAZINE

Completely Dynamic Report Orders and Breaks

Part 2

by Steve Parker

In the previous episode I presented a strategy for setting report breaks when I do not know the break fields at design time. In the case described, I do not know the fields or even whether breaks are desired until runtime.

The strategy described was based on the realization that I cannot use standard break fields as provided by the report formatter. Instead of break fields and group bands, I must use standard detail bands and prevent them from printing until I want them to. See Figure 1, below.

One detail is created for each possible break field. During each iteration of the report's loop, the current value of the field is checked against its previous value. If changed, the corresponding band is printed, totals are re-initialized and (re-)accumulated. If the break value has not changed, totals are simply accumulated.

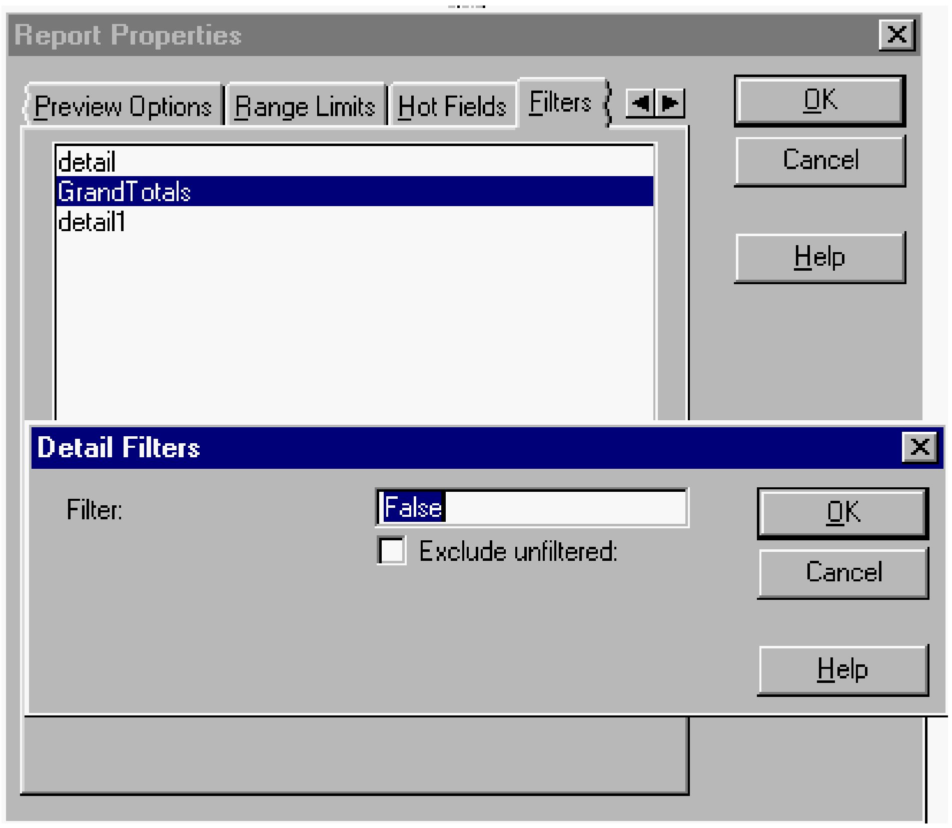
Sample code, placed in `TakeRecord` (Priority 5001) might look like:

```
If FirstLoop                        !if first record
  SaveNumber = POH:Number           !prime comparison fields
  SaveVendor = POH:Vendor
  FirstLoop = False                 !don't do this again
End
If BreakOnNumber                    !if user selected this break
  If POH:Number <> SaveNumber       !and it changed
    Print(RPT:Totals)               !print the band
    SubTotal = 0                    !re-initialize totals
  End
  SaveNumber = POH:Number           !save current value
  SubTotal += POH:Amount            !accumulate totals
End
If BreakOnVendor
```

```
  If POH:Vendor <> SaveVendor
    Print(RPT:VenTotal)
    VenSubTotal = 0
  End
  SaveVendor = POH:Vendor
  VenSubTotal += POH:Amount
End
```

In this week's episode, I'm going to walk through a real report that bases its sort order and breaks on values not known until runtime.

Figure 1. Preventing a band from printing, using its filter



Filter: False

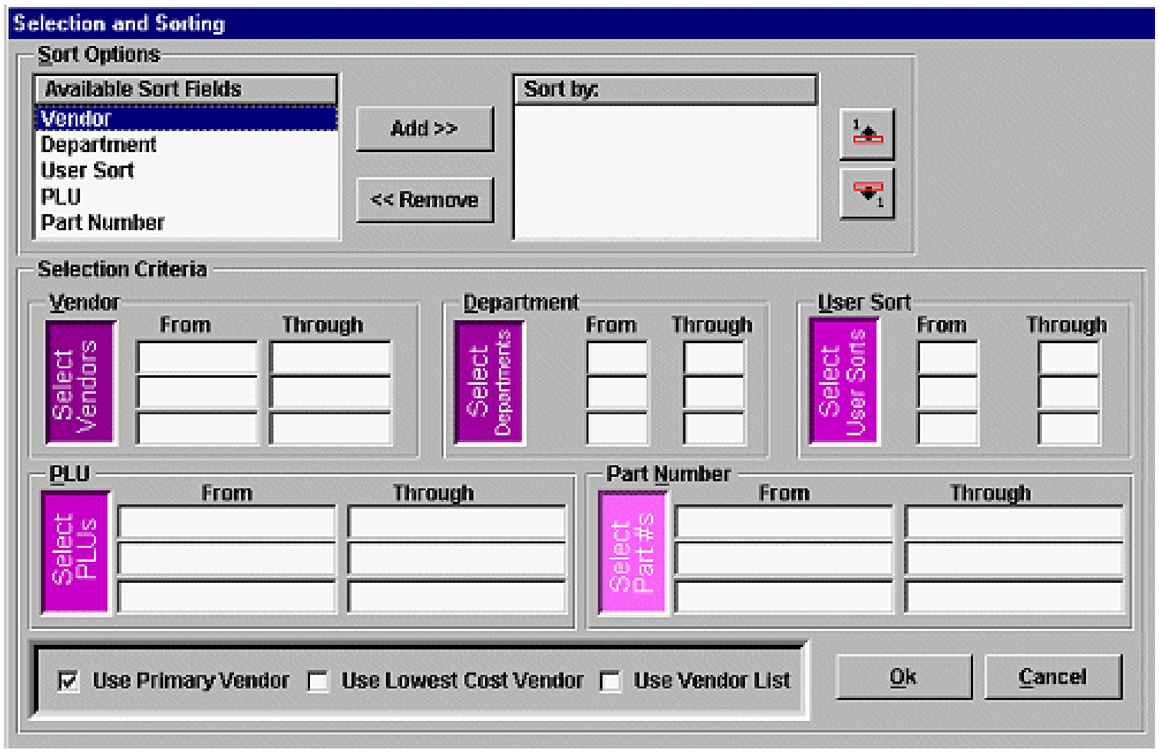
Exclude unfiltered: ☐

The Report

The report I need to create lists inventory items that have fallen below their order points.

I create the records with a Process template procedure. Users can set up both how they want the data sorted and how they want the data selected (how the Process is filtered) when creating the ordering list:

Figure 2. User criteria



(The Process actually creates a header file record and multiple, retailed, detail file records but that's an unnecessary complication at this point.)

The "Selection Criteria" group is where the user can select single values or ranges of values that are used to create the Process' filter. The group at the bottom left further refines that filter; that's interesting, but of no further relevance here.

The "Sort Options" group determines the sort order used when the items-to-order report is run. In this particular application, there are up to five parameters the user may select. The user may select any number of them, including none, in which case, I default to PLU (Price Lookup Unit or UPC code) for all further processing, though this feature could be implemented differently.

Once the user has selected which fields to use for sorting, they may also select the order (hierarchy) in the right hand list box (see "Three Ways to Present Many-To-Many Relations To The User: Part 2" by Tom Ruby).

The user selects which fields on which to sort and the order in which they are to be used in sorting. I store this information in a variable. I do not save the full contents of the right hand list but a single five character string.

The left hand list is created by me, with a fixed set of values, when the window is first opened and is composed of two fields: the description, which the user sees, and the code `PQ:Parm`), which the user does not see and which I will use now to store the selections. When this window is completed, I loop through the "Sort by" list (it "fronts" a queue, also of my creation). I save the code field for each selection:

```
Loop i# = 1 to Records(ParmList)
  Get(ParmList,i#)
  pSortParms = Clip(pSortParms) & PQ:Parm
End
```

In this application, I don't happen to care if no sorting was selected. But, if I do want to ensure that the user has selected a sort order, I can easily check `Records(ParmList)`:

```
If ~Records(ParmList)
  MESSAGE('You have not selected a sort order. This means '  &|
          'that your report will be a total mess, you yutz.', |
          'Hey Stupid!', ICON:Question)
  Select(?ParmList)
  Cycle
End
```

in the `TakeCompleted` method, before Parent Call. (Please note that this message would never be presented to a real user, it is strictly for fun amongst "us chickens.")

If I don't care whether the user selected an order but want at least one sort field, I can plug that too:

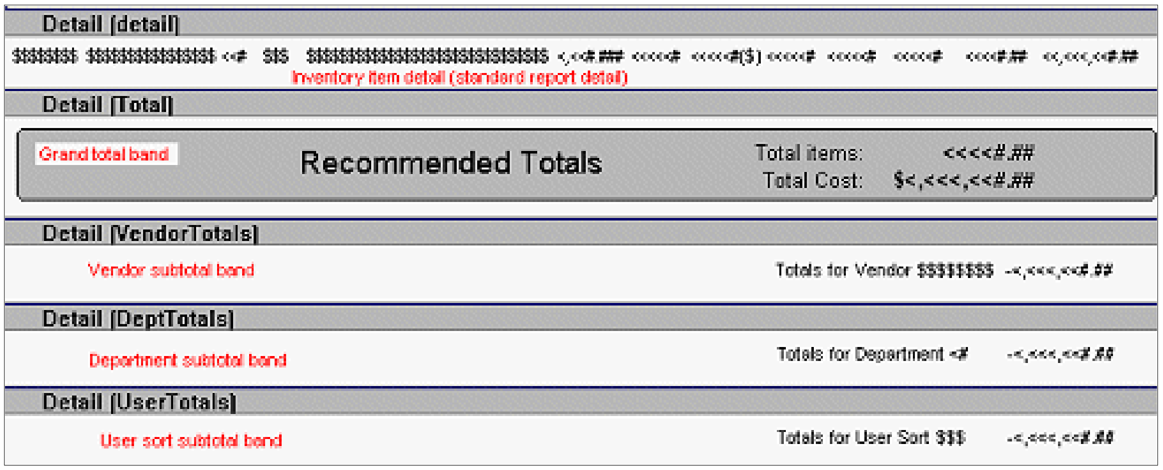
```
If ~Records(ParmList)
  !dummy message or no action
Else
  pSortParms = 'P'      !default to UPC code
End
```

All in all, this is quite flexible.

Of the five possible sorting selections in this window, only three are really relevant when I need to break and subtotal. Of the five possible values, PLU and Part Number are unique values (or, since multiple vendors could in theory duplicate part numbers, all but unique). This means that breaks or subtotals on these two fields need not be created because virtually every inventory item's PLU or Part Number would cause a break and a subtotal. A report that breaks on each item would be less than optimally informative.

If I only have potential breaks on Vendor, Department and User Sort, in the report formatter my report will look like Figure 3.

Figure 3. Report with all possible breaks



(I also include a grand totals band, in case that is required).

Setting the Sort Order

There are a number of ways to dynamically set the sort order for the report (see "Conditional Sort Orders and Page Breaks in Reports: Part II").

I can embed a `SetOrder` or `AppendOrder` method call before the report is opened (`OpenReport`, before Parent Call). This can be done in one of several ways:

```
ThisReport.SetOrder
ThisReport.AppendOrder
Self.Process.SetOrder
```

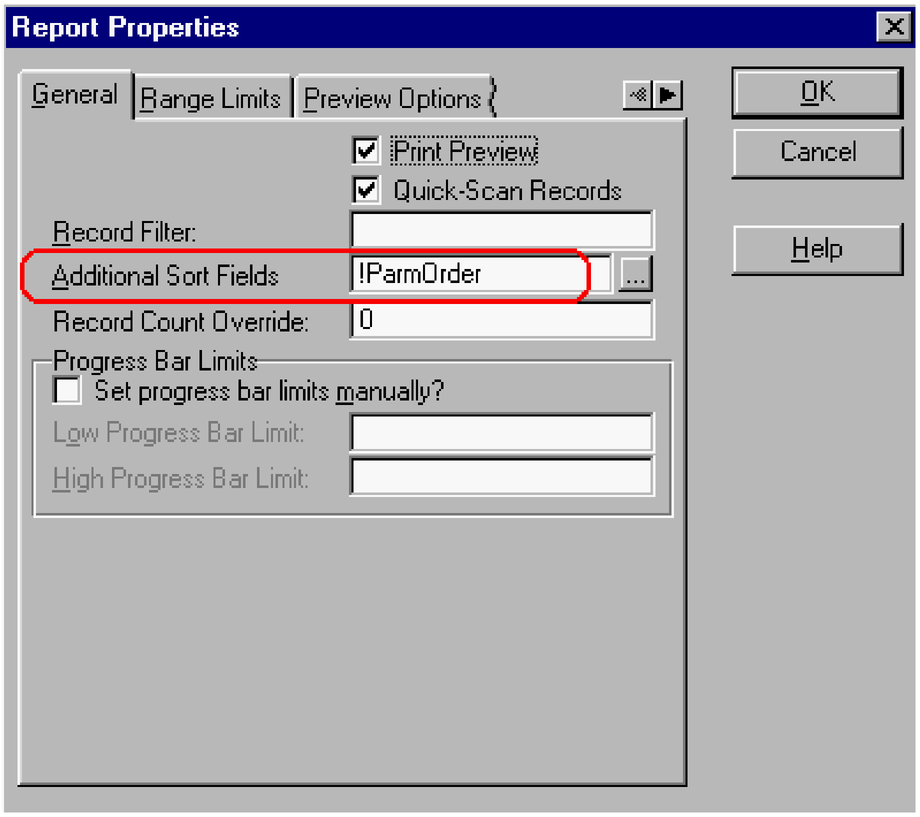
(also see the People example in the Examples subdirectory).

But I prefer to let the templates do this for me by declaring a local variable:

```
ParmOrder CSTRING(200)
```

and using it in the Additional Sort Fields prompt:

Figure 4. Let the templates handle it


Report Properties

General | Range Limits | Preview Options

[✓] Print Preview
[✓] Quick-Scan Records
Record Filter:
Additional Sort Fields: !ParmOrder
Record Count Override: 0
Progress Bar Limits
[ ] Set progress bar limits manually?
Low Progress Bar Limit:
High Progress Bar Limit:

This will cause the templates to generate

```
ThisReport.AppendOrder(ParmOrder)
```

in exactly the correct place in the final code.

The important thing is that both `SetOrder` and `AppendOrder` take an expression list as a required argument. This expression list is "A string constant, variable, EQUATE, or expression that contains an ORDER expression list." This list is a string (containing a list of fields) or just a list of fields. The on-line help provides the following examples:

```
MyView.AddSortOrder(ORD:ByCustomer)
MyView.AppendOrder('CUST:CustName')
```

This means that I need to take the five character string, `pSortParms` (which I saved to the header file as `ROH:Parameters`) and convert it back to a list of field labels.

Since I created the original list, I know which code goes with which field. So, this is not especially difficult.

I need to `Bind()` my local variable in order to use it in a filter. Then, I just need to loop through the characters in the file field, `ROH:Parameters`, and substitute the actual field labels from the detail file definition.

Late in the `INIT` method, I do the following:

```
!bind for use in filter
Bind('ParmOrder',ParmOrder)
ParmOrder = ''               !initialize
!construct index
Loop i# = 1 to Clip(Len(ROH:Parameters))
  !statement from acronym
  !read one character from field
  Case ROH:Parameters[i#]
  Of 'P'
    ParmOrder = Clip(ParmOrder) & ROD:PLU
  Of 'N'
    ParmOrder = Clip(ParmOrder) & ROD:PartNumber
  Of 'U'
    ParmOrder = Clip(ParmOrder) & ROD:UserSort
  Of 'V'
    ParmOrder = Clip(ParmOrder) & ROD:VNDCode
  Of 'D'
    ParmOrder = Clip(ParmOrder) & ROD:Department
  Else
    ParmOrder = ROD:PLU
  End
End
```

And my report will sort according to the user’s selections.

If I decided to call `SetOrder` or `AppendOrder` myself, before opening the report, I would still execute this code. Then,

```
ThisReport.SetOrder(ParmOrder)
ThisReport.AppendOrder(ParmOrder)
```

or

```
Self.Process.SetOrder(ParmOrder)
```

would still set the user’s selected order.

Print Subtotals?

The next requirement of this report is to allow users to make a runtime selection whether to print subtotals or not.

If the user selected PLU or Part Number as the first sorting criterion, as discussed above, subtotals and their attendant breaks make no sense. If User Sort, Vendor or Department is the first criterion, then they do make sense.

To discover what the user selected, I only need to check the first character of the file field storing the sort parameters. This field happens to be part of the header record (which also includes date, total items, total extended cost, Sys_ID and some other control information. If it corresponds to PLU or Part Number, I will not offer the user the option of subtotals. Otherwise, I do:

```
If Inlist(ROH:Parameters[1],'U','V','D')
  If Message('Print subtotals totals also?','Confirm',  |
     Icon:Question,Button:Yes+Button:No) = Button:Yes
    PrintTotals = 1
  Else
    PrintTotals = 0
  End
```

Notice that, in this case, the choice is all-or-none. If the user selects subtotal s/he gets any that may apply.

It would not be very difficult to use another procedure to set up additional parameters (say for individual subtotals) which could be returned in a string from a setup procedure and added to `ROH:Parameters`.

Printing The Subtotal Bands

The printing of the detail bands containing the subtotals depends on knowing when the file variables change.

To effect this, I need local variables, matching the variables the user selected. The idea is to save the values and compare the saved values to the next record when it is read but before it is printed.

However, the first record read will always trigger the printing by failing to match the previous record (there wasn't one, so the saved variable's value is blank or zero). So, in `TakeRecord`, after Parent Call (where the record is read), I use the following code:

```
If PrintTotals                    !If the user wants totals
  If FirstLoop                    !And this is the first record
    FirstLoop = 0                 !No longer first time
    SAV:Vendor = ROD:VNDCode      !Save vendor
    SAV:UserSort = ROD:UserSort !Save User Sort
    SAV:Department = ROD:Department     !Save Department
    TTL:Vendor = 0                !Initialize totals
    TTL:UserSort = 0
    TTL:Department = 0
  End
End
```

When the last record is read, the comparison will not take place (there's no "next" record to compare). So, I need to force the band printing from `AskPreview` (before Parent Call). This means that I need the check/print code in both embeds. Well, rather than write the code in both embeds, I move it to a routine.

Each variable which might be a break gets a routine similar to CheckVendor:

```
CheckVendor Routine
!print band if variable
 If (SAV:Vendor <> ROD:VNDCode) or ForcePrint = 1
   !changed or higher order one did
   ForcePrint = 1
   Print(RPT:VendorTotals)         !Print subtotals
   SAV:Vendor = ROD:VNDCode        !Store new check variable
   TTL:Vendor = ROD:ExtCost        !re-initialize totals
 Else
   TTL:Vendor += ROD:ExtCost       !accumulate
   ForcePrint = 0
 End
```

There is one routine for each variable, differing only in the variable labels of course.

The `ForcePrint` variable, not previously discussed, serves two purposes. First, it will make a band print even if the saved variable and the file variable are the same. This is necessary, as discussed, after the last record read. So, `ForcePrint = 1` in `AskPreview` produces the subtotal bands when the loop is finished reading the file.

One problem down; one to go.

A more important use for `ForcePrint` is to make details print when any higher (more accurately, "prior") level variable has changed. For example, if the user selected Vendor + Department and the Department changed, the Department subtotal will print and Vendor will not. That is as expected.

Now, suppose the Vendor changes. In this case, the Department band should print and re-initialize also. `ForcePrint` will do this because once one detail sets it to true, all others after it in `ROH:Parameters` will print too.

So, how do I check all of the desired variables and do I check them in the proper order?

In `TakeRecord`, after Parent Call, after the `FirstLoop` check, but inside the `If PrintTotals` condition above, I place the following:

(A warning before checking the code: I use an implicit variable. This has received an official editorial "Ack!" and you should extend your subscription before following this vile example.)

```
!Check each requested variable
Loop i# = 1 to Clip(Len(ROH:Parameters))
  Case ROH:Parameters[i#]
  Of ‘U’
    Do CheckUserSort
  Of ‘V’
    Do CheckVendor
  Of ‘D’
    Do CheckDepartment
  End
End
```

This code checks each user-selected variable, in the order set by the user. If totals were requested and if the particular parameter is on the user’s list, the appropriate routine is called.

Notice that as I loop when any variable check forces a band to print, all remaining parameters are checked and, because of `ForcePrint`, are printed also (i.e., even though the current and previous values may not differ). That is, if another variable occurs later in `ROH:Parameters`, it will be forced to print because I set `ForcePrint` when the prior variable changes value.

This means that in `AskPreview`, all I need to do is force all subtotals to print is the following:

```
If PrintTotals
  ForcePrint = 1
  Loop i# = 1 to Clip(Len(ROH:Parameters))
    Case ROH:Parameters[i#]
    Of ‘U’
      Do CheckUserSort
    Of ‘V’
      Do CheckVendor
    Of ‘D’
```

```
      Do CheckDepartment
    End
  End
End
Print(RPT:GrandTotals)
```

Totals, if wanted, are printed in all the appropriate places.

Summary

Understanding that a report is a `Print()`-in-a-loop makes many things much easier. The initial amount of work also makes it clear why so many Clarion developers swear by the CPCS templates. CPCS does all of this at the template level. Indeed, I have heard from CPCS users who weren't even aware that an ABC report was actually a Process.

More importantly, understanding that a report is a `Print()`-in-a-loop removes much of the mystery surrounding reports. If a report is just a fancy loop then I can do just about anything I want and only need to remember to Print() the band containing the variables I have affected.

Reborn Free CLARION online

published by CoveComm Inc.

Search

Clarion MAGAZINE

Holiday Special
25% Discount
on
APM & AFE

Clarion News

December 22, 2000

SoftVelocity Licenses Linder SetupBuilder

SetupBuilder for Clarion is the new install tool that replaces the Wise for Clarion installer in Enterprise Edition. Written in Clarion 5.5 by Linder Software, SetupBuilder is a Rapid Setup Development tool for Windows 95/98 and Windows NT4/2000, and offers a user-friendly visual development environment that does not require knowledge of a script language. All users who purchased a new license of Clarion 5.5 Enterprise Edition or who upgraded from 5.x Professional Edition can receive a free copy of SetupBuilder for Clarion. If you upgraded from 5.0 EE to 5.5 EE you are a licensed user of the "Wise for Clarion" installer, but you can upgrade to SetupBuilder for Clarion for a nominal charge of $25.00 for electronic delivery or $40.00 for shipped media. To obtain your copy of SetupBuilder for Clarion, contact us for instructions to download, or to confirm your mailing address if you wish to receive SetupBuilder on disk media format.

C55 PD 1-Touch Date Tools Update

Version 5.22 of PD 1-Touch Date Tools for Clarion 5 is now available for download. This release matches recent update to the C55 version. It adds auto population of popup calendars for date fields in an application either with a setting on the global extension or an override setting on the auto-populated procedure extension template. The popup calendar displays directly above or below the date entry and button depending on the entry's location on the window. The date entry fields and popup calendar automatically include Quicken and other extended hot keys, range limiting if set in the dictionary field user options or in a virtual SetRange method, colored holiday displays, schedule display, and more. There is no charge for this update for those with current licenses.

Lodestar Software/DeveloperPLUS Holiday Schedule

The offices of Lodestar Software & DeveloperPLUS will be closed 18-Dec-00 through 02-Jan-01 for the holidays. Support will be handled through email during this period although there may be a slight delay from day to day. Happy Holidays and a very happy, prosperous New Year to all.

HTML Designer Update

A new build of HTML Designer has been posted. This build makes use of its own API interface to the compiled HTML API from Microsoft. Also included is a wizard to add the extension template to all the procedures automatically. This file is password protected, so make use of the same password you received from ClarionShop. Also included in this update is a wizard utility to add the extension template to all the procedures except report and source procedures.

New Listbox Runtime Sort Template

The new RABSORT extension template adds listbox column sorting by clicking on the column header. The template automatically picks the fields from the listbox and automatically creates the sort orders. The default allows the user to ShiftMouseLeft on the Listbox column header to sort. During the sort process, the currently selected record is retained. This template currently does not allow any other custom sort orders other than those created by the template itself. Also it is not advised to use this template on listboxes with more than 15,000 records. This template is in beta and will be available from www.clarionshop.com for $59 discounted to $39 during the beta stage.

CapeSoft File Explorer Beta 2 Released

CapeSoft's newest effort comes in the form of a template wrapper around some common (and free) OCXs, including the Internet Explorer OCX which ships with Windows, the Adobe PDF viewer, and the Microsoft Media player. This means that any format supported by Internet Explorer, Media Player or Adobe Reader are supported. And you can view, or play these files directly in your application. Supported file types include HTM, HTML, PDF, AVI, MPG, WAV, SND, MP3 plus a whole lot of other obscure formats. This template allows you to simply drop the document viewer, or media control, directly into your application. Templates include some standard Clarion buttons to Play, Stop, Next Page, Last Page and so on. Beta 2 Improvements include: support for the HTML editing control; new methods (Edit, Save and PrintMe) added to the FileExplorer Class; events now trigger when a video, or audio, file completes playing; assorted other minor bug fixes and improvements. The normal price for File Explorer is $99, but it's currently on a Special Price of $69 during the beta program. Beta users will automatically get a free upgrade to the gold release, and beyond.

NetTalk Beta 9 Available

Beta 9 of NetTalk is now available. NetTalk allows you to build both robust TCP/IP communications between your Clarion applications, and also allows you to build generic TCP/IP (sockets) servers and clients. New in this beta: improved support for late versions of Windows NT and also Windows 2000; fixed bug so W2K, NetTalk, and OCXs happily work together; added news send and receive objects; added DIP client functionality. Also available on the web site is the DIP server, in both source code (requires WinEvent) and Executable format. DIP (Dynamic IP addressing) allows two machines who don't know each other's IP address to connect automatically. NetREFRESH updates browse screens on a LAN when the data underneath changes. No need to poll the database, if one user makes a change, it's reflected immediately on all browses. NetTalk will usually cost $299, but will be priced at $199 for the duration of the beta program. Beta users get free upgrades to the gold release, and beyond.

Special Agent 1.29 Released

This release of Special Agent fixes some bugs, and documents the examples. It also includes an update for Clarion 5.5 users.

Capesoft Office Messenger Version 1.1b Released

Office Messenger is not an Accessory, but rather a separate instant messenger-type utility. CapeSoft Office Messenger is used to pass messages from one employee to another inside the CapeSoft offices. This is a serverless system, yet all options can be set by a single administrator without going to each machine. Includes a phone book and many other features. This release fixes some bugs and has improved support for Windows 2000. A free 30 day demo download is available.

RPM and AFE 25% Holiday Discount

Just a reminder that the 25% discount on RPM and AFE is available through the end of the year.

December 12, 2000

25% Holiday Discount On RPM & AFE

Lodestar Software is offering a 25% discount on its Report and Presentation Manager and Automated Fax Engine products. RPM provides report presentation tools including viewing, paging, text search, printing, page numbering, document archiving, report merging/appending, and more. AFE includes all the tools necessary for integrating faxing into your application. Features include a developer's FaxMan fax engine license, 500 annual distribution licenses, support for unlimited fax servers on a network, cover sheet editor, variety of templates to support different faxing needs, multiple modem support, fax scheduling, fax grouping, and more.

PD 1-Touch Date Tools and PD Drops C55 Update

ProDomus is please to announce the release of a minor C55 upgrade to PD 1-Touch Date Time and Scheduling tools. This tool now includes a popup button class implemented by the ABC templates. Features include: popup Buttons may be created at run time for all date entries with a single global template entry; calendars may have range limits using a dictionary field options entry; calendars are displayed immediately above or below the entry; the button is skipped when tabbing from the date entry field - it is called with the AltDown key or by clicking the button; all calendars use colors for display of holidays and scheduled dates; all calendars display the week of the year according to local calculation rules; all calendars have a popup menu to jump to dates using a variety of calculations; all calendars have advanced navigation features including Quicken type keys. These calendars add an alternative to the previously provided auto-populated drop down calendars. ProDomus has also has new updated files for C55 PD Drops that include minor fixes resulting from the gold release.

HTML Designer Template

HTML Designer from Riebens Systems is intended to be a complete (D)HTML help environment geared for the Clarion Application Developer. The first part of this suite of utilities and tools (Version 1) consists of the Compiled HTML Help Source Extractor templates. These templates will be available from http://www.Clarionshop.com from 12 December 2000 onwards at a price of $99. During the BETA stage the price will be $49. Unfortunately no refund policy can be used as the product consists of a template that is *not* blackboxed but open for reproduction and modification. The HTML Designer suite allows the Clarion Application Developer to make use of the Standard Help, Tip and Message entry options on Screen Controls and Fields to enter information and a Wizard Utility Template will create the needed source code for the Compiled HTML Help. This source code is "Compile ready" in that with use of the Microsoft Help Workshop you only need three steps to create the Compiled HTML help file:1. Load the Project file into the Workshop; 2. Press the Compile button; 3. Sit and watch the file being created.

Crystal Clear Class Offers Crystal Reports Interface
Crystal Clear Class is a pure Clarion source ABC compliant class (no dll black boxes) that duplicates all the functionality provided by SoftVelocity's Crystal Reports class interface and adds additional features, including: the ability to embed Crystal report preview inside of the Clarion window and to fit the whole client area or a selected rectangle; export report to selected file format or to MAPI; log on to the SQL server; pass parameters and formulas to the report; change database at run time; set a printer for the print job. Crystal Clear Class is C5 compatible. Current price is $79.95. Demo available.

New Data Conversion Template Released
Vasiliy Goncharenko’s Data Conversion template is now out of beta. This product handles automatic datafile conversion when a dictionary changes. Features include: automatic registration of data dictionary changes; automatic data file conversion; embed points to tune up the conversion code, including custom field assignment; converting several versions of a file at once; "Internal Converter" mode for simple one-application projects, when converter is inside of application; "External Converter" mode for a complex projects, when several applications uses one data dictionary, and converter is common to all applications in the project; backup of old versions of data files; fast transition from a "Internal Converter" to "External Converter" modes and back, without need for manual coding; 16 bit and 32 bit support, Local and Stand-Alone run-time libraries, Legacy and ABC templates set; multi-tables support (TopSpeed database driver).

December 5, 2000

SearchFlash 2.0 Released
The SearchFlash QBE template has been updated with the following new features: saving tags in a tag file; batch copy of tagged records to another file; developer-defined QBE conditions; manual editing of a QBE search string by the end user, allowing use of built-in Clarion functions; option to show only tagged records after a search. SearchFlash 2.0 is compatible with Clarion 2,4,5,& 5.5. Demo available.

Topspeed Turnpike Renamed
The TopSpeed Turnpike has been renamed: it is now called the Clarion Connection. It’s still on the same server; it just has a new name.

ABC Free Templates Updated
The ABC Free Templates and Tools have been updated to fix the Copy Button template for use in Clarion 5.5 Gold.

Solace ReDesigner Demo Update
The Solace Redesigner demo has been updated to fix a problem when saving modified screen designs or changed properties. Note that the beta program for the templates closes on December 10, 2000.

RPM55 Gold Now Available
RPM55 for C5.5 Gold is available for download. This is a free upgrade from RPM5 using your current password and serial number. PNet55 and AFE55 for C5.5 Gold will be available shortly.

Clarion Handy Tools New Features

New features in build O of the Clarion Handy Tools include: a new download wizard; C5.5 no-browse edit form; wizard tab buttons; use of proxy server in FTP; disabling Windows-style list box marking; separate DLL for email and FTP; improved compile manager; new demo apps; a file synchronizer; and various bug fixes.

Clarion Third Party Profile Exchange

Dave Troxell’s Profile Exchange now contains 212 product profiles and 180 vendor profiles. Requires Product Scope 32 PRO.

Reborn Free CLARION online

published by CoveComm Inc.

Clarion MAGAZINE

Search

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Finding Lost Files: A Redirection Class

by Jeff Slarve

ClarionMag Holiday Schedule
(Dec 22,2000)

Interfacing Satellite Forms Applications and Clarion for Windows
(Dec 22,2000)

Completely Dynamic Report Orders and Breaks Part 2
(Dec 22,2000)

December 2000 News
(Dec 22,2000)

Finding Lost Files: A Redirection Class
(Dec 22,2000)

Clarion Essentials CBT From SoftVelocity
(Dec 22,2000)

Freebie: Data Is Executed Policy - A Case Study
(Dec 22,2000)

The last several applications that I have written made use of images that needed to be loaded at run time from filenames stored in a database.

In a perfect world, this would be no problem. Just store the filename in a database, display the file as needed in an image control, and be done with it. However, in a networked environment, several nagging issues pop up that can cause significant difficulty.

For one thing, you can’t count on drive names or path names to be the same from one workstation to the next. Many small businesses don’t have a full time administrator that can make sure that everything is set up in a way that you can trust for your program to run smoothly. The path `D:\TheFolder\Images\File.jpg` on one machine could be `M:\Apps\TheFolder\Images\File.jpg` on another machine.

One way to deal with different file names is to refer to files by their Universal Naming Convention (UNC) names, which have this format:

```
\\servername\sharename\filepath\file
```

Although UNC would be a good solution, it is not at all easy to obtain the UNC name of a file. This is especially true if the "server" happens to be the workstation from which you’re working, because the API calls to get UNC names require that you have Administrative rights. UNC would be fine for a power user, but I haven’t found a practical way to reliably retrieve the UNC name of a file. The API procedure `SHBrowseForFolder()` could be used, but in my opinion it requires too much of the user to make it work as it should.

Another thing that I have read about UNC is that it puts more of a demand on system resources because it has to resolve the host name every time a file is referenced. Supposedly it is more efficient to use a mapped drive letter, but I am not certain about that.

Either way, I really did try to figure out a reliable to store the UNC names, but wasn't comfortable with the hoops that I was having to jump through as a programmer.

Another option is to store just the plain filename, or the "relative path" of a file, but there are no built in `FileDialog()` functions which retrieve only the filename, or the relative path.

Enter the Redirection Class

I always thought that the redirection file in the Clarion environment was really cool. If a certain file type is defined in the redirection file, then the IDE knows to search in that spot for the file without having to know in advance the fully qualified path of the file in question. When you use the file dialog to look up an image on a window in the IDE, and that image file is found within defined constraints of the redirection file, only the filename itself is stored. This is immensely useful, especially when you swap APP files with other developers or move Clarion to another drive, because otherwise the filenames would be invalid.

I figured that if I could mimic the redirection file behavior of the Clarion IDE, then it would solve a lot of problems. I wouldn't have to worry about storing the path to the file, nor would I have to worry about UNC.

At the end of this article you can download the redirection class and an example application. That application uses a redirection file that looks like this:

```
[Common]
*.TPL = .;YadaYada
*.txt = .;yadayada;%ROOT%\test2
*.tst = %ROOT%\Test2
```

With this redirection file (test.red) I can, for example, put any files with the .txt extension in the current directory, the `yadayada` subdirectory, or the application's `test2` subdirectory, and the application will be able to find it.

Methods

Here are some of the redirection class's methods (in alphabetical order):

`AddDefaultMacros Procedure,Virtual`

Adds the `%ROOT%` and `%PATH%` substitution macros so that they can be used in the redirection file. This is a virtual method that can be overridden and customized. `%ROOT%` is set to the location of the application, and `%PATH%` is the application's working directory. These could be different locations if the application's shortcut is configured for a different working directory.

`AddMacro Procedure(String pMacro,String pReplacement)`

Adds a macro such as `%PATH%` to the redirection object.

Usage: JSR.AddMacro('PATH', LongPath())

`Construct Procedure`

Automatic constructor

`Destruct Procedure,Virtual`

Automatic destructor. If overridden, the `PARENT.Destruct()` should be called.

`FileDialog Procedure(String pTitle, |`

`*String pFileName,String pExtensions,| Long pFlags,| Byte ReturnRelativePath=False),Byte`

Just like the standard Clarion file dialog, but it has a switch to only return the path relative to the application's directory, if desired.

```
FindFilePath Procedure(String pShortName,|
        *String pFullName,|
        Byte Suggest=False),Byte
```

Attempts to locate a file within the defines of the redirection file. Returns `FALSE` if the file is not found, unless the `SUGGEST` flag is set to `TRUE`. If the `SUGGEST` flag is set to true, then it will suggest a path where a file might get created.

```
ReduceFileName Procedure(String pLongName,|
        Byte ReturnRelativePath=False),String
```

Reduces the fully qualified pathname of a file to that just the filename, just like the Clarion IDE does when you select a file that is found within the redirection file

```
ParseRedirFile Procedure(String pRedirFileName,
        String pSection,Byte Validate=False),Byte,Proc
```

Parses a redirection file for use by the redirection object. You can optionally set `VALIDATE` to `TRUE`. If this is done, then it will offer to create the paths that are defined in the redirection file, but don't exist yet.

`ParseRedirString Procedure(String pRedirString,|`

`Byte Validate=False),Byte,Proc`

This is the same as `ParseRedirFile()`, except that you can pass a string instead of a file. This is handy to use in a config file.

> **NOTE:** As with `ParseRedirFile()` you can call this method multiple times with multiple strings. This could allow you to have a "standard" redirection, and an additional "user" redirection.

`ParseLine Procedure(String pLine),Private`

Called by the other `Parse..()` functions. Private

`ReplaceMacros Procedure(String pText),String,Private`

Called by the `Parse..()` functions. If there are macros, then this method replaces them with the valid path.

`Reset Procedure`

Clears any current redir data that might have been read. You might use this if you allow the user to modify the redirection file and need to re-parse it.

`ValidateRedirQ Procedure(Byte Verbose=False,|`

`Byte CreateFolders=False),Byte,Proc`

Validates any redir stuff that had been parsed

Usage

To use this class, you have to do the following:

1. Create an instance of the redirection class in your application:
   ```
   JSR JSRedirClass
   ```
2. Parse the redirection file, specifying the section to use:
   ```
   RedFile = '.\Test.Red'
   JSR.ParseRedirFile(RedFile,'Common',1)
   ```

Call the `FindFilePath` method, passing the file name you're looking for, and a string to hold the full path and name of the file, if found. At its simplest, the function call looks something like this:

```
if JSR.FindFilePath(FileToFind,FoundFile)
  ! do something with FoundFile
```

If the method is successful, it will return true, and your application can go ahead and use the file.

The placement of the folders within the redirection file dictates the order that the folders will be searched. E.G.:

```
*.TXT = c:\;c:\temp
```

will cause c:\ to be searched first.

The same idea holds true on the order of addition of redirection files. If you want to change the order that things are found, then you need to call `RESET()` and re-parse the files in the order that you want.

The only other caveat is that you need at least Clarion 5.0 or better, as the redirection class makes use of the `MATCH()` function.

Download the redirection class

Download the redirection demo

Jeff Slarve is an independent software developer and the creator of the critically-acclaimed In Back automated file safeguard utility. Jeff has been a Clarion developer since 1991, and is a member of the group formerly known as Team TopSpeed.

Main Menu | Log In | Subscribe | Links | FAQ | Advertising

Reborn Free CLARION online

published by CoveComm Inc.

Search

Clarion MAGAZINE

Main Page

COL Archive

Log In
Subscribe
Renewals

Frequently Asked Questions

Site Index
Article Index
Author Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion Essentials CBT From SoftVelocity

Reviewed by Tom Hebenstreit, Reviews Editor

ClarionMag Holiday Schedule
(Dec 22,2000)

Interfacing Satellite Forms Applications and Clarion for Windows
(Dec 22,2000)

Completely Dynamic Report Orders and Breaks Part 2
(Dec 22,2000)

December 2000 News
(Dec 22,2000)

Finding Lost Files: A Redirection Class
(Dec 22,2000)

Clarion Essentials CBT From SoftVelocity
(Dec 22,2000)

Freebie: Data Is Executed Policy - A Case Study
(Dec 22,2000)

Let’s get the basic question out of the way right now - what the heck is CBT? It is an acronym for "Computer-Based Training." And that is a fancy way of saying that it is an instructional course designed for use by individuals, on their own time, with the computer substituting for a real live instructor. In a nutshell, you click, you learn.

What are the advantages of CBT? Well, the biggest one is that, unlike a physical class where there is limited time and a fixed daily format, you can easily fit a CBT course into your own schedule and time requirements. You can also concentrate on particular topics for as long as you personally need to, without having an instructor dictating the pace.

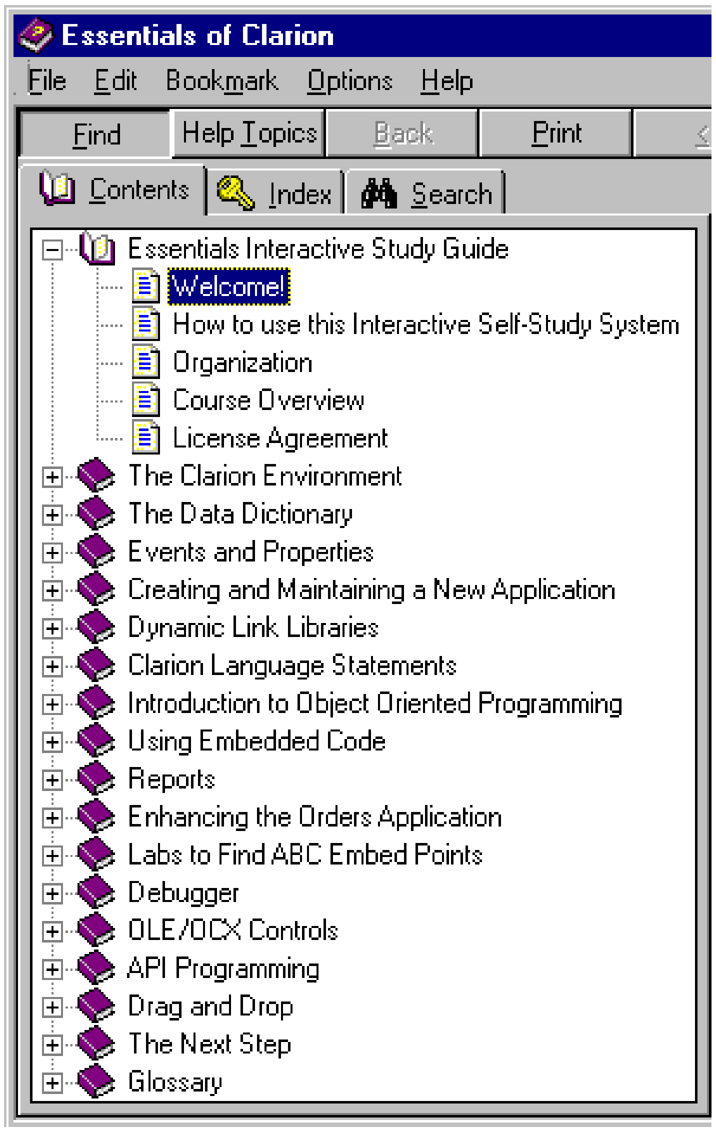
The downside? Well, you don’t get to take a vacation (er… make that a business trip) and you don’t get the personal interaction and attention that a good instructor can provide.

Now that I’ve cleared that up, let’s take a look at the recently released Clarion Essentials CBT course from SoftVelocity.

What does it cover?

As a step up from the original Clarion Foundations CBT, which concentrated mainly on learning the basics of the Clarion language and IDE, the Essentials course tackles more advanced concepts and features of the Clarion IDE and language. In the bigger picture of SoftVelocity educational courses, it falls squarely between the beginning Foundations course and the advanced Mastery course (which isn’t available as CBT as of this writing). Figure 1 shows the table of contents for the essentials CBT.

Figure 1. Table of contents for the Clarion Essentials CBT



Essentials Interactive Study Guide
Welcome!
How to use this Interactive Self-Study System
Organization
Course Overview
License Agreement
The Clarion Environment
The Data Dictionary
Events and Properties
Creating and Maintaining a New Application
Dynamic Link Libraries
Clarion Language Statements
Introduction to Object Oriented Programming
Using Embedded Code
Reports
Enhancing the Orders Application
Labs to Find ABC Embed Points
Debugger
OLE/OCX Controls
API Programming
Drag and Drop
The Next Step
Glossary

Note: In order to reduce the width of the sample Essentials CBT images, I've split the single window into two figures. Keep in mind while viewing them that you would normally see both parts on the screen at the same time.

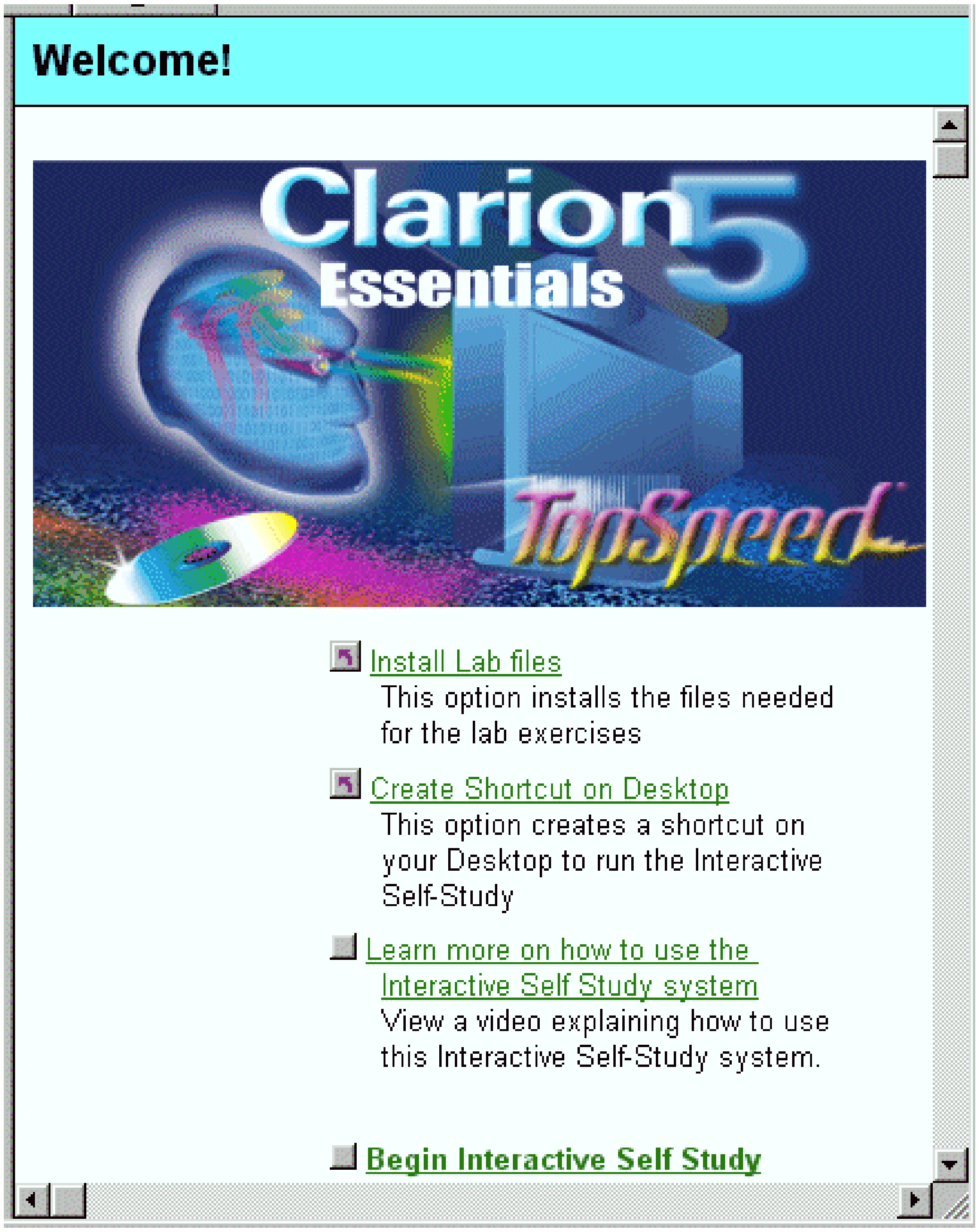
As you can see, a wide range of topics are covered within the course. The number of topics may seem a bit daunting at first, but remember that the biggest benefit of using CBT is that you can move at your own speed and according to your own requirements. As a side benefit, this should also help to extend the useful life of the training course since you can tackle individual lessons based upon your needs of the moment. Not interested in Drag and Drop right now? No problem, just skip that lesson. But it will still be waiting for you in a few months when you decide to utilize that feature in a future project. And unlike a physical class, you won't be stuck trying to remember what the instructor said way after the fact.

What does it look like?

The Essentials course is presented within the overall context of a Windows help file, with additional lab exercise documentation provided in the form of a PDF file and corresponding APP and DCT files. Many of the topics also include small AVI format movies that demonstrate step by step how to accomplish various tasks (more on the movies later).

When you first insert the Essentials disk, you see the standard welcome screen that looks like this:

Figure 2. The starting point for the Essentials CBT (the right side of the help window)



Welcome!

Install Lab files
This option installs the files needed for the lab exercises

Create Shortcut on Desktop
This option creates a shortcut on your Desktop to run the Interactive Self-Study

Learn more on how to use the Interactive Self Study system
View a video explaining how to use this Interactive Self-Study system.

Begin Interactive Self Study

You'll note that the screen prominently features the TopSpeed name, and that holds true for most of the course materials. The Essentials CBT was being wrapped up during the transition of Clarion from TopSpeed to SoftVelocity, and obviously they didn't want to hold up the initial release just to go back and redo everything for a name change. Just translate TopSpeed to SoftVelocity everywhere you see it, and you'll do fine.

Getting back to Figure 2, a nice touch is the prominent link for learning how to use the CBT course itself. Some training disks of this type simply assume that you know how to use a CBT course. In this case, you get a movie that demonstrates all of the essential things you need to know to be able to jump right in.

In use, the Essentials CBT basically consists of clicking through a topic in the help file to get an overview of the lesson subject, watching the occasional movie, and then taking a self-test and/or working through the related lab examples to get actual hand-on experience.

Installation

The Essentials CBT doesn't really require any installing – it can be run directly from the CD without any files being copied to your PC.

Most users, though, will want to install the lab files and sample applications to their hard drive so that they can actually work through them step-by-step. Using that option requires approximately 71 MB of hard drive space. I did find it a little strange that the PDF file for the lab work wasn't copied to the hard drive as well (it is left on the CD).

Overall, the installation program is pretty painless – basically just pick the destination folder and go. I'd recommend letting it install to the default location of `\Clarion5\CBT\Essentials`, though, as that particular path is used extensively in the help files and lab documentation. There is no harm in installing it elsewhere, but you can save yourself some needless path translating by simply letting the installer do what it wants to.

One note regarding the uninstall option of the CBT – it just didn't work on my test machines. It would remove the icons and uninstall links, but leave the actual 71 MB Essentials folder on my disk. Fortunately, everything is contained within that one folder, and it is easy to remove manually.

Where's the popcorn?

One of the nifty features of CBT is the ability to interactively watch a mini-movie as the instructor performs certain actions on-screen. At the same time, you get an audio track the logic behind the actions.

In the finest tradition of pictures being worth more than words, the Essentials CBT contains a generous helping of 83 separate movie (AVI) files. Now, they don't quite compete with *The Matrix*, but they certainly do the job.

I did notice a few places where what the instructor was saying didn't match the screen shot, but in all cases these were minor slip-ups. For example, the instructor was demonstrating how to place a print button on a browse window, and when he said to select the `PrintBrowse` Control template, the screen actually showed him selecting the `BrowsePrintButton` Control template.

The only real downside to the movies was the mechanism used to play them back (and you can thank Bill Gates for that). The multi-media system used to play the movies within the help file has a rather paltry range of options, pretty much just play and stop. You can speed up and slow down the pacing within a limited range (unless you *like* learning from Alvin the Chipmunk), but unfortunately Windows doesn't retain your settings. As soon as you start another movie, you are back at the defaults. Perhaps the most aggravating thing is that you can't run a movie back just a bit to catch something you missed - you can only restart the entire movie.

On the bright side, you can manually run the CBT movies outside of the actual CBT program using many other multi-media players, such as the freely available MS Windows Media Player 7 or the Creative Labs player that comes with most SoundBlaster audio cards. When played in these programs, you often have a complete control over playback, including skipping forward and back to your hearts content.

Oh yeah, just don't put too much butter on your popcorn while watching these movies, or your mouse will get really greasy and hard to handle…

Documentation

As far as printed documentation goes, there is none. If desired, you can print the lab exercise PDF file, but be forewarned that it is 174 pages. On the plus side, it *is* 174 pages worth of good information.

The only real problem with the course documentation relates to the many references to TopSpeed rather than SoftVelocity. Hopefully SoftVelocity will update the materials at some point or at least include a sheet of paper with the CBT that lists the correct SoftVelocity web address and phone numbers.

Technical support and program requirements

Despite the many TopSpeed references, both sales and support are provided by SoftVelocity. I don't think there is a whole lot of support needed for this type of product, though. Either it works on your machine or it doesn't, and it would take a *really* stripped down machine not to have the basics of a sound card and the built-in Windows help and multimedia subsystems.

Clarion-wise, the CBT was released before Clarion 5.5 went gold, so it uses Clarion 5 throughout. In watching the SoftVelocity support forums, and in going through exercises myself, I've only seen one case where something was a bit different when using the CBT with release 5.5.

Summary

I'm a fan of well-done CBT courses, and the Clarion Essentials course works out to be a pretty good one. It covers a lot of ground in a straightforward manner and, nits aside, provides a solid and useful foundation for the topics addressed. Add in the overall advantages of convenience and self-pacing, toss in a dash of multimedia, and you have a recipe for a successful CBT course.

The bottom line: As with most educational products, the benefits you derive from the Essentials CBT are directly proportional to the amount of time and effort you put into it. With even a modicum of effort, this course should provide you with a handy boost towards the next level of Clarion expertise and productivity.

PRODUCT RATING	
Overall	▲▲▲
Ability to do the task	**Very Good**
Ease of use	**Excellent**
Ease of installation	**Very Good**
Documentation	**Good**
Technical support	**Excellent**
Black box DLLs/LIBs	**N/A**

LEGEND	
First class all the way	▲▲▲▲
More than adequate	▲▲▲
Barely adequate	▲▲
Don't even think about it	▲

The Clarion Essentials CBT course retails for US$350, and is available from SoftVelocity Sales via phone at 800-354-5444. More information on this and other SoftVelocity training offerings can also be found at their web site at: http://www.softvelocity.com/education/education.htm

Vendor Comments from Russell B. Eggen of SoftVelocity:

One of the major decisions we had to make on this product was to either base it on C5 or the as-yet unreleased 5.5. This was a very important and potentially dangerous decision. C5 would limit the effective lifespan of the product and 5.5 would lose accuracy as it was in middle of massive changes. Or delay the release of the CBT until 5.5 was released. Since that was about 6 months away at the time, [it wasn't] a viable decision.

A compromise was achieved where the lessons would be as version neutral as possible. Of course this could not apply to the visual aspects of the AVI movies as 5.5 has the improved interface.

But the lessons on Object Oriented Programming, Reports, finding unknown embed points, and more are still quite valid. As a matter of fact, the CBT product still can achieve its goal with 5.5, even assist those developers trying to learn 5.5.

We are very pleased with the feedback we have received with CBT customers, even those with 5.5. It exceeds the goals we had for the product (as did the Foundations CBT). That tells us that interest is very high for these types of products. It also states the decision to make a version neutral lesson plan was a wise one. There are plans for more titles and a bit more advanced in nature. This is in addition to an "extension" for 5.5-only features.

Of course, these products must keep the high education standards used not only in these products, but our instructor-lead classes. Providing all the features that our customers want does not do anyone any good if they don't know how to use them. This is why many lessons assume no prior knowledge.

These products must not raise new questions, as there is nothing to fall back on. We had an excellent test phase where questions were raised. The content was immediately fixed. The authors were not allowed to answer any tester's questions directly, but [had to] provide revised materials that must clear up the confusions. This was the quality measure used.

CBT products are excellent for learning new topics, especially when a student does not risk embarrassment if they truly do not understand a topic.