Clarion Magazine

Optimizing DLL Loading - Rebasing Your DLLs

If you create DLLs with Clarion, then you'll definitely want to read this series of articles. By default, Clarion 32 bit DLLs load more slowly than they need to, and are not sharable between processes. In this installment Carl Barnes shows how to easily rebase your own application's DLLs.

Posted Tuesday, November 06, 2001

Checkboxes For Many-to-Many Relationships

There are a lot of ways to show many-to-many relationships between two tables, and some take more work on the part of the user than others. Dave Harms shows how to create a linking record between tables with a single mouse click.

Posted Friday, November 09, 2001

The Clarion Challenge - Using C in Clarion

We've already received a number of entries in our latest Clarion challenge, which asks you to integrate some C code into a Clarion application. Get your entry in now - the deadline is Friday, November 16th.

Posted Wednesday, November 14, 2001

Checkboxes For Many-to-Many Relationships: The Source Code

There are a lot of ways to show many-to-many relationships between two tables, and some take more work on the part of the user than others. This week Dave Harms explores the inner workings of a derived browse class to manage these relationships.

News

Alison Neal Profiled In INN Bio

VCRFlash 2.1 Released

The Sylkie Web Site

SealSoft Releases xNotes Class v1.0

Clarion Photo Gallery Update

Gitano Software Back From Vacation

Updates from Gitano Software Now Available

SysIP Released

ExpressFlash 2 Supports Outlook 2000

INN Bio Features Richard Rogers

Icetips Wizards Now

Posted Thursday, November 15, 2001

Calling By Address, STARTing By Address

What would you do if someone gave you a project that involved either calling or STARTing a procedure in a DLL which was loaded dynamically at runtime, with some procedures using the Pascal calling convention, some the C calling convention, and even one procedure with an MDI window that needed to be started on its own thread? If you're Jim Kane, you write a class that handles all of this. Part 1 of 2.

Posted Friday, November 16, 2001

A New Look, And A Topical Index

Clarion Magazine has a new look, and more importantly, a new topical index, making it easier than ever to find the articles you're looking for.

Posted Tuesday, November 20, 2001

Calling By Address, STARTing By Address (Part 2)

What would you do if someone gave you a project that involved either calling or STARTing a procedure in a DLL which was loaded dynamically at runtime, with some procedures using the Pascal calling convention, some the C calling convention, and even one procedure with an MDI window that needed to be started on its own thread? If you're Jim Kane, you write a class that handles all of this. Part 2 of 2.

Posted Wednesday, November 21, 2001

Using SQL Server's Data Transformation Services (DTS)

In converting an application, changing database drivers is not the only requirement. You also have to provide a way to convert existing data. One way to do this is with Microsoft's Data Transformation Services (DTS). In this article, Ayo Ogundahunsi demonstrates the process using the Inventory example application.

Posted Tuesday, November 27, 2001

The Clarion Advisor: Sizing Windows

Clarion is a great tool for writing custom apps. And every once in a while, Andrew Guidroz II has a requirement from a customer for a window that is as unique as the individual's desktop. But how to know what size to make the window?

Posted Wednesday, November 28, 2001

Andy Ireland's COM Classes

Business Rules Manager Released

TPS.repair Template Goes Gold

ProDomus Updates and Notes

xQuickFilter v2.06 Released

Updated MySQL Templates

Optimizing DLL Loading - Rebasing Your DLLs

by Carl Barnes

Published 2001-11-06

If you want your multi-DLL applications to load as quickly as possible, or you want to share a 32 bit DLL among multiple applications, or you want to have multiple instances or your applications running, you need to know something about how Windows allocates memory for DLLs. The key is *rebasing*, the process by which Windows moves a DLL from a common and standard base address to unused memory space. In last week's installment I described the rebasing process and explained how the default Clarion approach results in slower-than-necessary load times and poor memory use. This week I'll explain how to set a new memory base for a Clarion DLL.

How to Set the Image Base in Clarion

The base address for a Clarion DLL is specified in the EXP file using the `IMAGE_BASE` statement. In a Clarion application you can enter this statement in the global embed point named 'Before the export list.'. The syntax is "`IMAGE_BASE` *`address`*" where the *`address`* value must be in multiples of 64k (65536). It is best to specify the value in hexadecimal by placing the letter "h" after the number. This makes it very easy to know you have a proper address, since 64k is `10000h`. You just have to insure your address ends in four zeros (`0000h`). An example EXP file with a base address specified is as follows:

```
NAME 'REPORTS' GUI
IMAGE_BASE 05100000h
EXPORTS
  REPORTS:INIT@F10ERRORCLASS8INICLASS  @?
  REPORTS:KILL@F                       @?
;Start of Exported Procedures
  CUSTREPORT@F                         @?
  INVOICEREPORT@F                      @?
  CUSTINVOICEREPORT@F                  @?
```

Determining What Address to Use

Don't get too worried about picking a bad base address. If the address you pick is used by another DLL, your DLL will automatically get rebased by the loader, and you'll be in the same slow boat you are in now. If the address is illegal (e.g. not on a 64k boundary) the linker will spot it, display an error and revert to using the default 00400000h.

There are a number of addresses that are not recommended, and others that are reserved by Windows and cannot be used. When the loader needs to rebase a DLL it uses the lowest memory address available. For this reason you should not base your DLLs at an address below about 0300,0000h. For example, in Figure 2 you'll see the AllFiles.DLL was rebased to address 0041,0000h just above the EXE at 0040,0000h. If you use low addresses you run the risk of your DLL conflicting with other rebased DLLs and just causing more rebasing. The TopSpeed DLLs are also based in the low address range of 005A,0000h to 0100,0000h, yet another reason to stay with higher addresses.

I suggest you put your DLLs in the address range 0500,0000h to 0FF0,0000h. In my experience this range is not normally used. This address space is 175MB in size, a large amount of space for a single app and easily allowing room for well over 100 typical DLLs. Figure 1 shows all the address ranges in the 2GB address space:

Start	End	Description
0000,0000	0000,FFFF	Reserved for Null Pointer Assignment Errors
0001,0000	003F,FFFF	WinNT – Not Recommended, used by loader for rebased DLLs, Win9x Reserved
0040,0000		Default EXE load address
005A,0000	0100,0000	TopSpeed DLLs

0100,0000	02FF,FFFF	Not Recommended, used by loader for rebased DLLs
0300,0000	04FF,FFFF	Available for use
0500,0000	0FFF,FFFF	Available for use (my recommended range)
1000,0000	10FF,FFFF	Not Recommended, default address for VC DLLs
1100,0000	11FF,FFFF	Not Recommended, default address for VB DLLs
1200,0000	1FFF,FFFF	Available for use
2000,0000	2FFF,FFFF	Not Recommended, used by OCXs
3000,0000	5FFF,FFFF	Available for use
6000,0000	69FF,FFFF	Recommended by Microsoft, I would not recommend this range since many MS and other DLLs use this range
7000,0000	7FFF,FFFF	Reserved for Windows

Figure 1. Address ranges in the 2GB address space.

The single most important thing you should do to assist yourself in picking the addresses for your DLLs is to run your program and examine it using a process viewer utility that shows all of the DLLs (modules) that are loaded into your process. Such a utility will show the address at which each DLL loaded and its size. (You can also use my CarlBase utility - see below.) You need to analyze this information to see what address ranges are in use so that you do not reuse them

and cause a load conflict.

The memory utility I used for the screen shots in this article came with the Jeffery Richter book "Programming Applications for Microsoft Windows". This utility is small, simple and perfect for the job. Its most important feature is that it highlights DLLs that were rebased by placing their original address in parentheses. Unfortunately you cannot download this utility; it only comes on the CD included with the book. The next best choice would be Dependency Walker. It's a free download and much more capable. At the end of this article is a list of other free process viewer utilities you can download.

An example

Figure 2 shows the DLLTutor example and all of the DLLs in its address space. Note that there are no DLLs loaded into my preferred address range starting at `0500,0000h` so that's a good place to start.

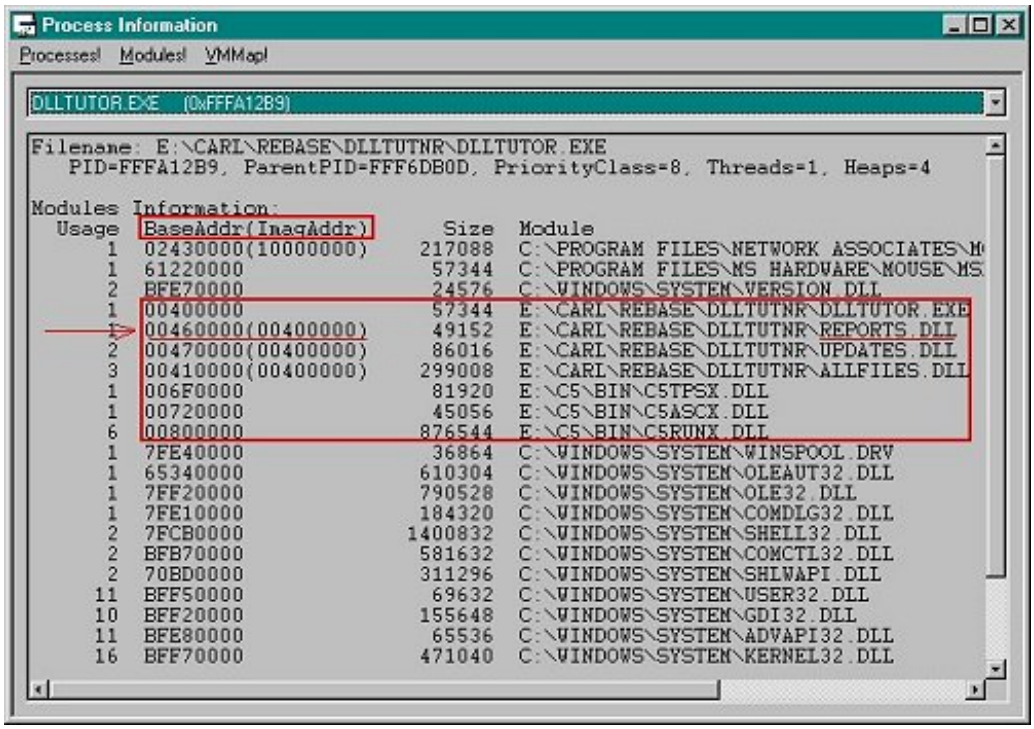

Figure 2. Modules in the DLLTutor Process

The next choice to make is how much space to allow for each DLL. In Figure 1 in the size column you can see that the Tutor DLLs (AllFiles, Updates and Reports) have sizes ranging from about 50k to 300k. I suggest you keep it simple and, in this case, allow 1MB for each DLL. Leaving these large gaps does not consume any additional memory - it's just address space, which Windows will map to physical memory. However, if you fail to leave room for your DLLs to grow in size you risk a conflict in the future that will cause a rebase. A good rule of thumb is to allow double the current size of your largest DLL, and put the DLLs on 1MB (`00000h`)

boundaries. The address region is 175MB in size so there is no problem in allowing some extra space (unless you have a really massive application).

To specify a base address for each DLL in this project I added one of the below lines to the embed point "Before the export list" for each respective DLL, and then recompiled.

```
IMAGE_BASE 05000000h ; AllFiles.DLL
IMAGE_BASE 05100000h ; Reports.DLL
IMAGE_BASE 05200000h ; Updates.DLL
```

Now when I run DllTutor and examine it under the process viewer I can see that my DLLs have been placed at the 0500 addresses I specified and are no longer rebased as indicated previously by the (400000h). Figure 3 is a screen shot of the recompiled DllTutor's process information:

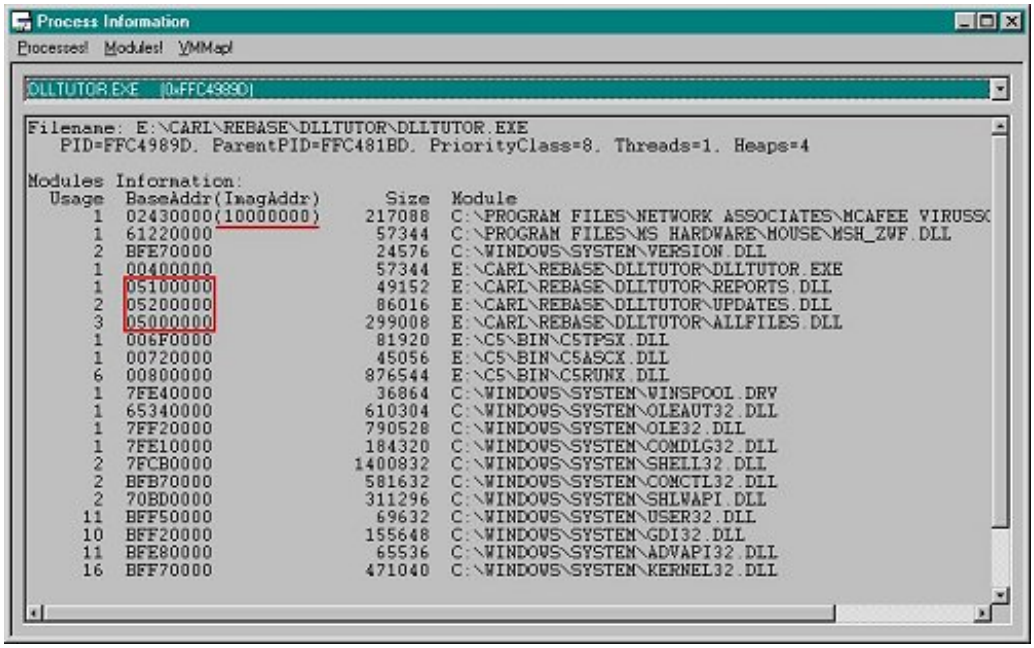

Figure 3. DllTutor after rebasing

Notice that there is still a DLL being rebased but it is not my DLL; actually, it belongs to McAfee VirusScan. I'm not sure how that got injected into my process but the address in parentheses (10000000) tells me that the programmer failed to assign the DLL a load address and ended up with the default Visual C++ base address.

The Debug Advantage

If you do not specify an image base for your DLLs you are screwed when it comes to debugging a crash where all you have is an address. Enough said? Read on for more details or skip ahead to the next section.

When the linker links your DLL it creates a MAP file that shows procedure and data names of symbols in the DLL and the address at which the linker placed them. This is useful for debugging in general, and especially when your App GPFs and Dr. Watson gives you a crash address. You can look up that crash address in the MAP and find the procedure. Having solid address information is very useful for debugging in general, for example when looking at the GPF stack dump and trying to find a return address. Below is an excerpt of the MAP from the Reports DLL (that has an image base of 0510,0000h) where you can see the entry address of each procedure:

```
5101520 CUSTINVOICEREPORT@F
 5101BA4 INVOICEREPORT@F
 51020A4 CUSTREPORT@F
 5102268 DESTRUCT@F14DLLINITIALIZER
 5102284 CONSTRUCT@F14DLLINITIALIZER
 51022D0 REPORTS:KILL@F
 51022D8 REPORTS:INIT@F10ERRORCLASS8INICLASS
 510300C $LOCALERRORS
 5103618 $LOCALINIMGR
 5103724 $DLLINITIALIZER
 5103728 $GLOBALERRORS
 510372C $INIMGR
```

If you do not specify an IMAGE_BASE for your DLL then you get the default of 0040,0000h and your MAP is written using this address for every DLL. This makes the MAP fairly useless for finding the crash address in a process with DLLs since the DLL will never load at 0040,0000h. Worse, since your DLLs will be rebased to an address determined by the Windows loader, you will have a difficult (or impossible) time determining even in which DLL the crash occurred. For the DllTutor example above a GPF in the address range of 0510,0000h to 051F,FFFFh can be reliably determined as happening in the Reports DLL.

A Dr. Watson dump will list the module addresses of modules loaded in the address space but will not show a name unless there are Microsoft debug symbol files (*.dbg) available. Having predefined your base addresses will make it easier to use a Dr. Watson dump.

Thirty-second time a template

After doing and redoing my Image Base settings on 16 DLLs about twice I was getting really tired of opening every App and changing global embeds. It’s a slow process. I wanted a way to specify all of my addresses for these DLLs in a single place. So I created an Application Extension template that reads the base address

from an INI file and places the proper code in the Before Exports embed. This template allows me to very rapidly change my mind by editing one file and kicking off a batch compile using Gordon Smith's compile manager.

The template also includes some code to protect you from mistakes like assigning an invalid address or rebasing an EXE. You *can* rebase an EXE but it's a bad idea; EXEs should always be at 40,0000h. You can go as low as 10,0000h under NT but then your application will not run under 9x.

The template's INI file is named Rebase.INI by default; you can change its name and directory. For each DLL in the application include a line in the file with AppName=BaseAddress. An example of the Rebase.INI file used to make the DllTutor is shown below:

```
[rebase]
ALLFILES=05000000h
REPORTS=05100000h
UPDATES=05200000h
```

Add the Rebase Application Extension template to each App and recompile, and you are done. If you want to change your mind simply edit the INI file and recompile. You can switch back to default addresses by deleting the "AppName=" lines and recompiling. The template will warn you that you have do not have an entry in the Rebase.INI file for your application unless you include /NoWarn=1 in the [Rebase] section.

The template is constructed as an #EXTENSION which gathers the user input and inserts a #GROUP which does the actual work of checking the address and inserting the line in the EXP. Another option for implementing rebasing is to modify the template that builds the EXP file (abbldexp.tpw or buildexp.tpw) to insert the #GROUP. If you do that you will not have to add the extension to each application, but you will have to migrate your changes to new releases of Clarion (and ensure the changes still work).

Implementing rebasing step by step

Below is a checklist for implementing rebasing in your project. (This is the hard way; I'll describe an easier way in a moment.) To keep things really simple just use the address range starting at 0500,0000h and allow 4MB per DLL. This will work great for projects with less than 44 DLLs where each requires less than 4MB to load (and 4MB is a pretty big DLL).

1. Download a process viewer and install it
2. Download and register the Rebase.TPL included below
3. View your running EXE in a process viewer. If you are using Dependency Walker

(Depends) perform a File-Open for your EXE and select Profile-Start Profiling to run it. If you are running Process Explorer run your EXE, then run PE, select View-DLLs and find your EXE in the list of processes and click on it.

4. You will need to decide on what address range(s) you will use by looking in the process viewer to see what addresses are currently in use. You do not want to conflict with already specified addresses. If you dynamically load DLLs try to get them all loaded. I suggest using the address range starting at `0500,0000h`.
5. Create a Rebase.INI file with `AppName=` for each App and open it in Notepad or an editor (I like UltraEdit (www.ultraedit.com). Here's an easy way to create this file:
   a. Open a DOS command prompt in your project directory
   b. Type: `Dir *.App /on/b > Rebase.INI` to create a file with all of your app file name.
   c. Edit Rebase.INI created above. Search and Replace ".app" with "=05000000h". Insert the INI section header "[Rebase]" above the first line. Delete any Apps which are EXEs.
6. Assign each AppName= a unique load address. You need to look at the process view to see the size of each DLL and allow space for growth. I would suggest allowing 4MB (0040,0000h) space per DLL. Using 4MB the first 12 addresses in my preferred range are: `05000000h, 05400000h, 05800000h, 05C00000h, 06000000h, 06400000h, 06800000h, 06C00000h, 07000000h, 07400000h, 07800000h, 07C00000h`
7. Add the Rebase Extension template to each App and compile
8. Repeat step 3 and view your running EXE in a process viewer. Verify that DLLs loaded at the address you expected.

Implementing rebasing gets easier

After trying out the above method on a few beta readers I found there was some resistance. "You said this was easy and took just one line of code, but you didn't say anything about hexadecimal math!" To make all this address picking easier I created a utility that does the entire job. Lacking a better name I called it "CarlBase". (If I was from Louisiana and followed the Cajun Naming Convention I might have called it "Bouillabase".)

The CarlBase utility does not require you to view your running processes. It works by reading the PE header information written by the linker for the Windows loader. In many ways this is a superior approach since it works with dynamically loaded DLLs *and* OCXs. All you need to do is place is every DLL or OCX used by your EXE in a single directory. Typically this is your install directory.

Here are the steps for rebasing with CarlBase:

1. Download the latest CarlBase from www.carlbarnes.com
2. Download the Rebase Template included below
3. Put your EXEs and DLLs in a directory

4. Run CarlBase and pick the directory that contains all your EXEs, APPs, and OCXs. Figure 4 shows the DllTutor example.

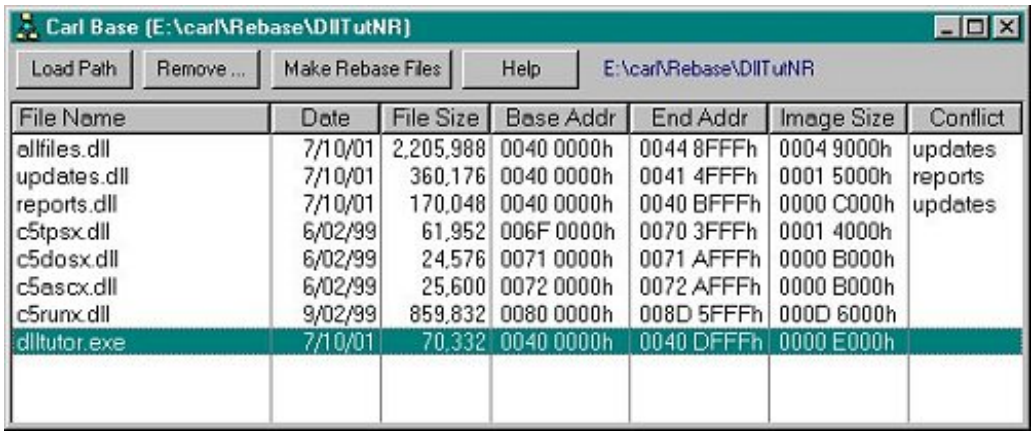

Figure 4. CarlBase run on the DLLTutor application

1. Remove any DLLs you did not create by clicking on them and pressing delete, or clicking the Remove button. In Figure 4, the c5xxxxx DLLs should be removed.
2. Press the "Make Rebase Files" button and the screen in Figure 5 will be displayed.

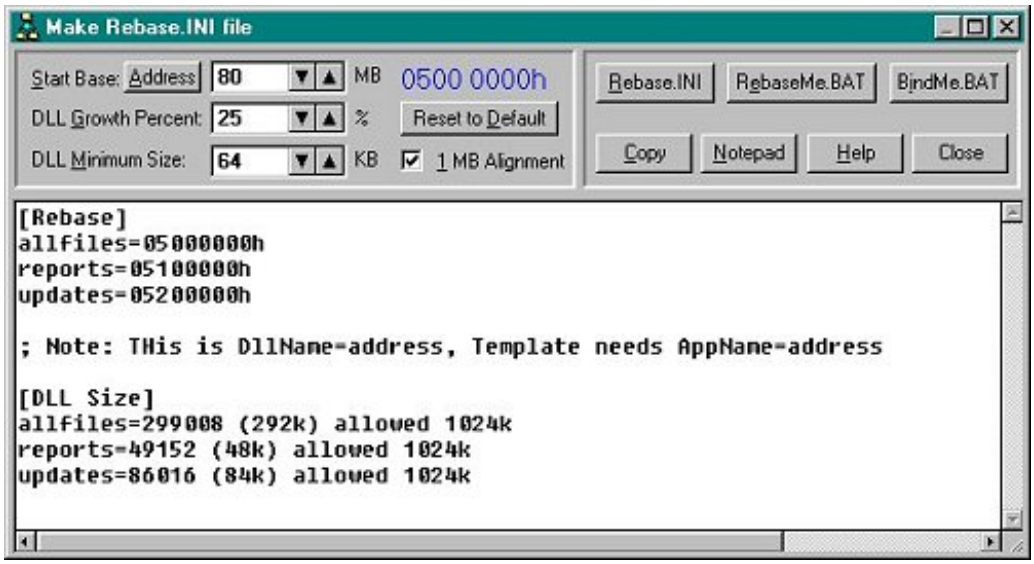

Figure 5. Creating the rebase files

1. On the dialog in Figure 5 you can choose the starting base address, the amount of growth to allow for and the minimum size to allow. I like to put my DLLs on 1MB alignment so there is a checkbox for that.
2. Press the "Rebase.INI" button and the CarlBase utility will make that file and display it in the text box as show above.
3. Press the Copy button, then the Notepad button. In the open Notepad press Paste, then save the contents to your .App file directory as a text file called Rebase.INI. (Guess I should add a 'Save to file' option.)
4. Add the Rebase global extension template to each application and compile
5. That's it. You should look at your App in a process viewer to be sure everything

loaded correctly.

Summary

The default linking of Clarion DLLs makes every DLL try to load at address `0040,0000h` which causes Windows to rebase the DLL. If you take the time to assign a good base address for the linker, your DLLs will load faster, consume less memory resources, be sharable with multiple instances, and be easier to debug. It's amazing that a single line in the project can do this much.

Download the source

Resources

CarlBase	A utility I wrote to help you easily generate a correct Rebase.TPL file and explore addresses used by DLLs and OCXs.
Dependency Walker	Dependency Walker is the best utility for this job. It shows the original and actual base address and an immense amount of other information including a complete tree of DLL usage dependencies. To get this information you must open your EXE in DW then select Start Profiling from the Profile menu. This utility is part of the Windows Platform SDK or may be freely downloaded from `http://www.dependencywalker.com/`. This is a must have utility. You can read about the origins of this utility in this MSDN article.
Process Explorer	A free utility, Process Explorer It shows the base address at which each DLL was loaded and size of the DLL. (It defaults to showing handles so to see DLLs you must select "View DLLs" from the View menu.) It does not show the original address specified by the linker. This tool has many other features and is only 77K.

Process Info	Process Info is the utility used for the screen shots in this article and a nice lightweight utility that clearly identifies the DLLs that were rebased to a different address. It is included with the book *Programming Applications for Microsoft Windows* by Jeffery Richter.
PE Explorer	PE Explorer is a $69 utility that has a process viewer, dependency tracer and many more features like a disassembler. I have not tried it but the website makes it look good and you get a 30-day free trial.
SystemWorks	According to Mike Pickus, Norton SystemWorks includes a process viewer.

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

Checkboxes For Many-to-Many Relationships

by Dave Harms

Published 2001-11-09

I'm in the process of adding a topical index to Clarion Magazine. That's fairly tedious work – it means going to my database of every article published in Clarion Magazine and Clarion Online (there are over 750) and assigning each article to one or more categories. I want this process to be as quick and painless as possible. Ideally I want to show a list of articles on the left, and a list of categories on the right, and I want to be able to assign an article to a category with a single mouse click and show that connection with a checkbox.

To some extent it's possible to do this with the EIP templates. Earlier I implemented a similar requirement with the approach Pete Halsted outlined in his EIP checkbox article with acceptable results, except that setting a checkbox took three mouse clicks: one to select the record, another to enable EIP for that record, and a third to change the checkbox selection.

To get those three mouse clicks down to one (some days every click counts!) I wrote a class, and added one line of embed code to my browse window. In this article I'll explain how to use that class to easily add this functionality to your own many-to-many browses.

The sample application

For purposes of discussion, and to avoid having to part with any Clarion Magazine data, I'll use a sample application built on the Clarion SCHOOL.DCT. (This is a 5.0b application, and while there are some differences in the class code used between 5.0b and 5.5, the supplied class will work with both versions.)

My sample application uses four tables (unaltered) from the SCHOOL application: Courses, Classes, Students, and Enrollments. There are some nine courses in the Courses table, such as English Composition, Microcomputers, Algebra, and so forth. Each course is available in one or more classes, and often in two, such as Mondays/Wednesdays, or Tuesdays/Thursdays. Each student can be enrolled in one

or more classes, so the many-to-many relationship is between the Students and Classes tables, and the linking table is called Enrollments.

> **NOTE**: If you're not familiar with many-to-many relationships, I highly recommend you read Tom Ruby's two-part series on the subject.

My sample application is called ENROLL.APP, and is shown in Figure 1. As you can see, the students are listed on the left side, and the courses are on the right side. The course browse also has a checkbox in the first column; to create or remove a student/course link, you just click the checkbox. Whenever you select a new student, the course list is updated to reflect that students chosen courses.

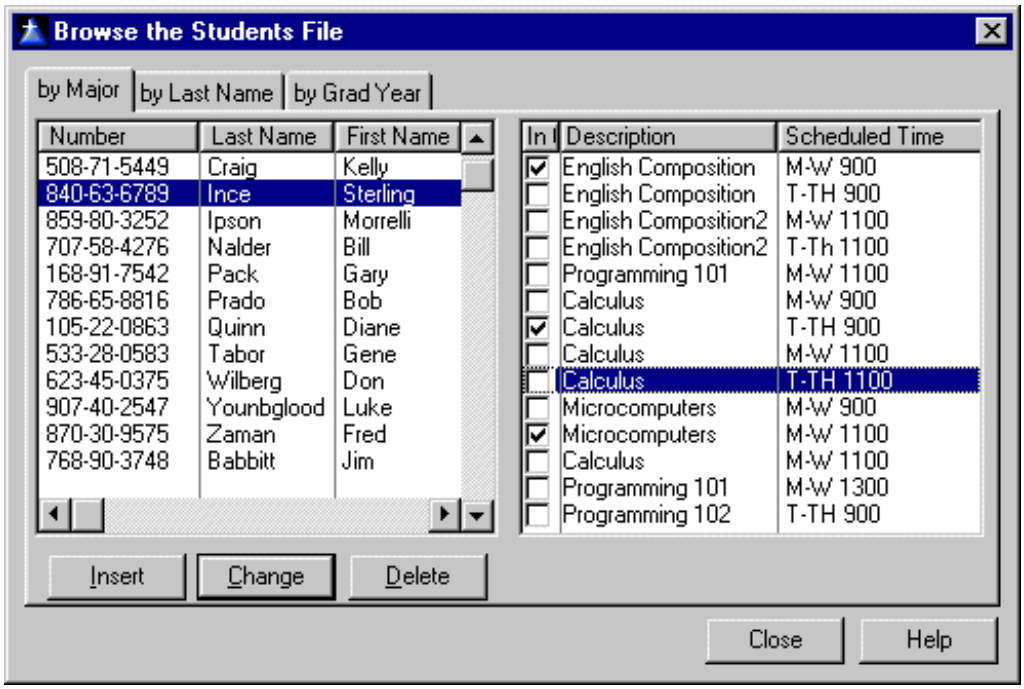

Figure 1. Managing a many-to-many relationship with one-click checkboxes

When I started writing this code I first created a standalone class to manage the checkboxes, but I quickly realized that a lot of the behavior I wanted to influence, and some of the variables I needed to use, were part of the ABC `BrowseClass`. So rather than create a separate class, I derived my class from `BrowseClass`. This made it possible for me to easily add new behavior to the `BrowseClass` without disrupting existing browse behavior.

You only need to embed one line of source code in your application; this code simply passes references to the derived browse class, including the linking table's `FileManager` object, the linking table's keys and fields, and a few other required fields. I'll explain the internal workings of the derived class next week.

Knowing your left from your right

Most importantly, if you want to follow along in this article, you need to wrap your head around my personal concept of "Left" and "Right." In my own little world, the left table is (naturally) on the left side of the window, and (arbitrarily) does *not* have a checkbox indicating a link between the left and right tables. The right table is (again, naturally) the table on the right side of the window, and (again, arbitrarily) *does* have a checkbox which indicates a link between the left and right tables. Okay, everybody who followed that, raise your right foot. Thank you.

The requirements for my checkbox many-to-many class are as follows:

- When the user selects a different left record, refresh the right table
- When a record on the right table is displayed, look for a linking record that matches the primary IDs from both left and right tables. If found, display a checked icon, otherwise display an unchecked icon.
- When the user clicks on a checkbox, either delete an existing linking record, if present, or create a new linking record, if none is present. Redisplay the row

Several additional requirements logically follow:

- Both the left and right tables need to have a primary key, which is a unique identifier for each record. This is always a good idea. See Tom Ruby's article on Managing Complexity, Part 3 for more on primary keys.
- Both primary key fields will need to be present in their respective browses, which means they need to be in the displayed fields *or* in the hot fields list.
- The linking table will need to be in the procedure's Other Tables list.

How to make it work

The `cciBrowseClass` is fairly easy to use. First, you need to copy the `ccibrows.clw` and `ccibrows.inc` files (available at the bottom of this page) into your Clarion 5.5 `libsrc` directory. This class is ABC compatible, so the next time you open an application (or refresh the class list) Clarion will add it to its list of available classes and automatically handle any compile/link issues.

You'll need two browses, one on the left, and one on the right. In the ENROLL application, the left browse is the student list, and the right browse is the class list. In the sample application, the left browse already has the student number displayed, so you don't need to add this to the hot field list. That's the only change you'd possibly have to make to the left browse.

You want the checkbox to be displayed on the right hand browse. Go to local data and create a `BYTE` field called `InClass`. Then from the window bring up the Listbox Properties for the right hand browse. Populate the `InClass` field in the first column of the browse. Set the picture to `@p p` - Pete Halsted points out that this picture will

always display a blank space. On the Appearance tab, set the Icon to Normal, as shown in Figure 2, or Transparent, if you want the highlight bar to include the icon.

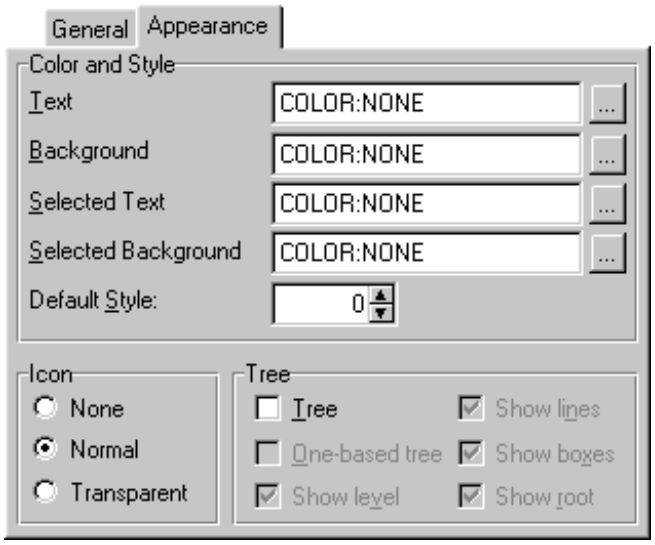

Figure 2. Setting the icon attribute

On the right hand browse you have a few additional changes to make, so if you haven't already, bring up the Actions tab for that browse. On the Default Behavior subtab click on the Reset Fields button. You want to add the *left hand browse's primary key field* to this list, as in Figure 3. Actually any field will do, but you know already that this field has to be in the list, so it's a safe bet. Whenever this Reset Field value changes, the browse will automatically be refreshed, which is what you want.

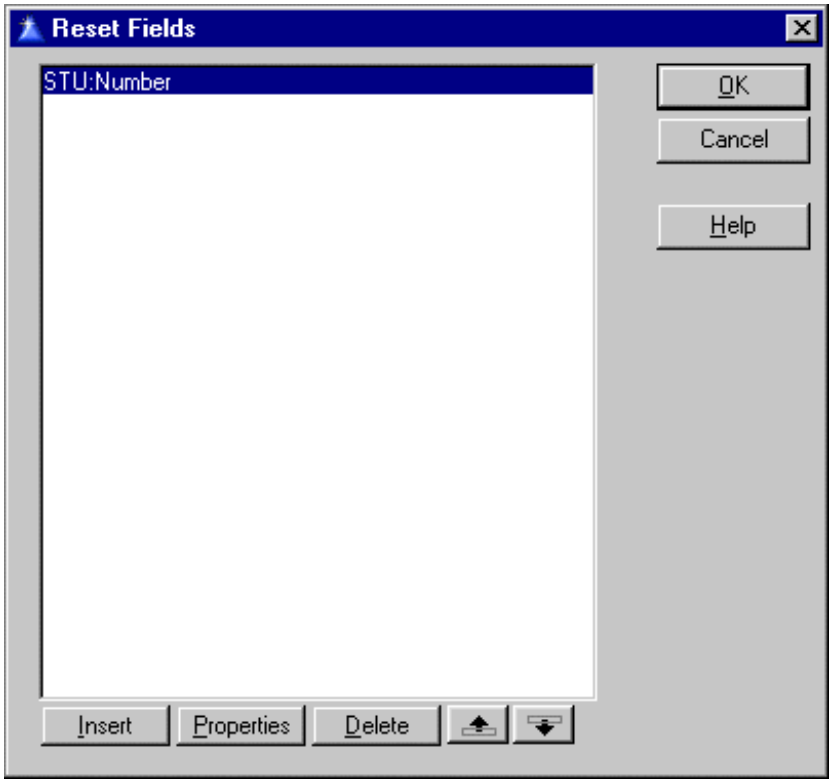

Figure 3. Adding a reset field

As with the left browse, the right browse must either display its primary key field, or have it in the hot fields list. Figure 4 shows the `CLA:ClassNumber` in the right hand browse's hot fields list.

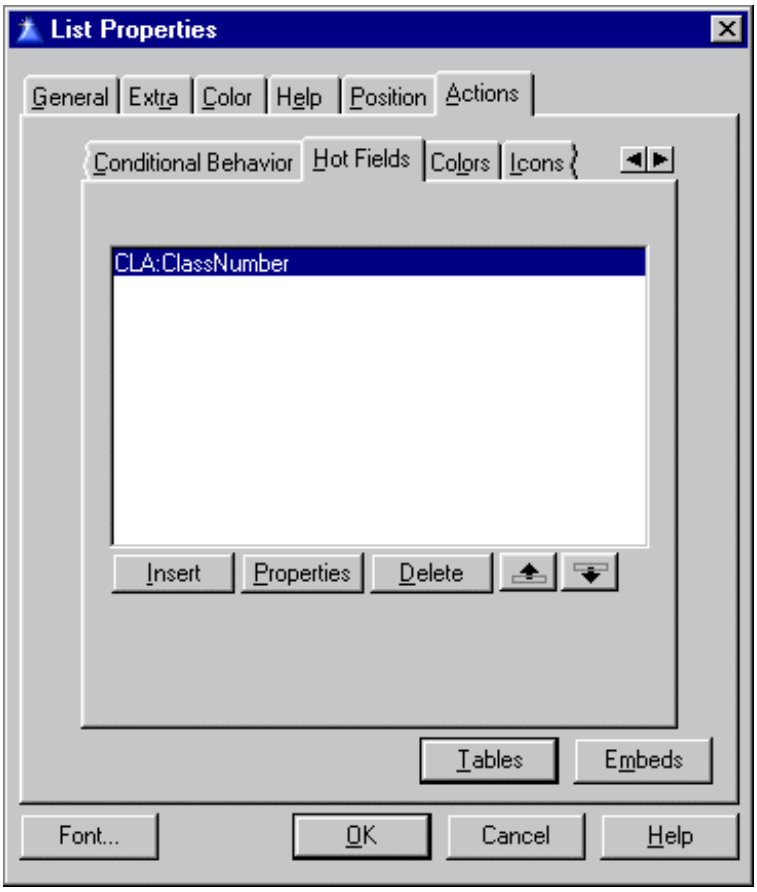

Figure 4. Adding the right hand primary key field to the hot fields list

Now go to the Icons subtab. Here you'll specify what icons the browse should use to display the actual checkbox. Set `off.ico` as the default icon, and then click on Insert to add a conditional icon. Figure 5 shows the conditional icon entry dialog popped up over the icons subtab. You probably don't have `on.ico` and `off.ico`, so I've included them in the downloadable source zip. You can also use any other icons you wish.

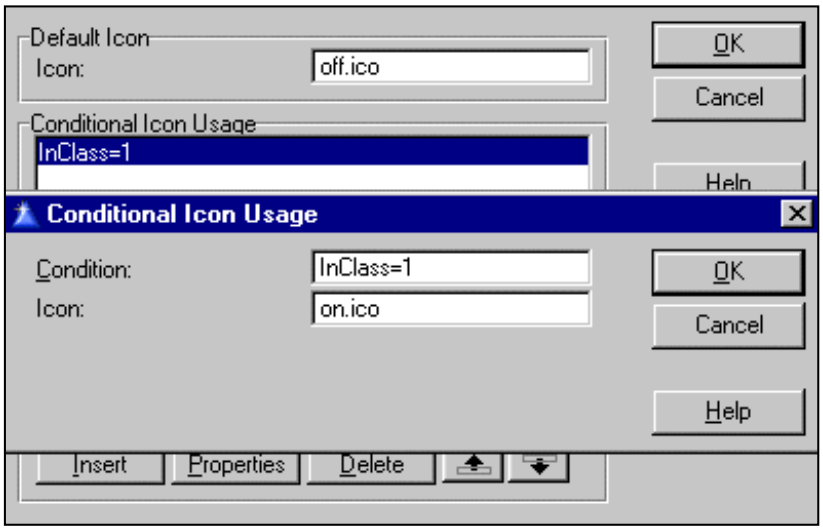

Figure 5. Setting the conditional icon usage

Next, on the Actions tab for the right browse, go to the Classes subtab, as shown in Figure 6. Uncheck Use Default `ABC:BrowseClass`. This will enable the Base Class drop list. From that list choose `cciBrowseClass` (it should be at the bottom of the list). I also recommend you change the object name to something easily remembered; I used `ClassesBrowse`.

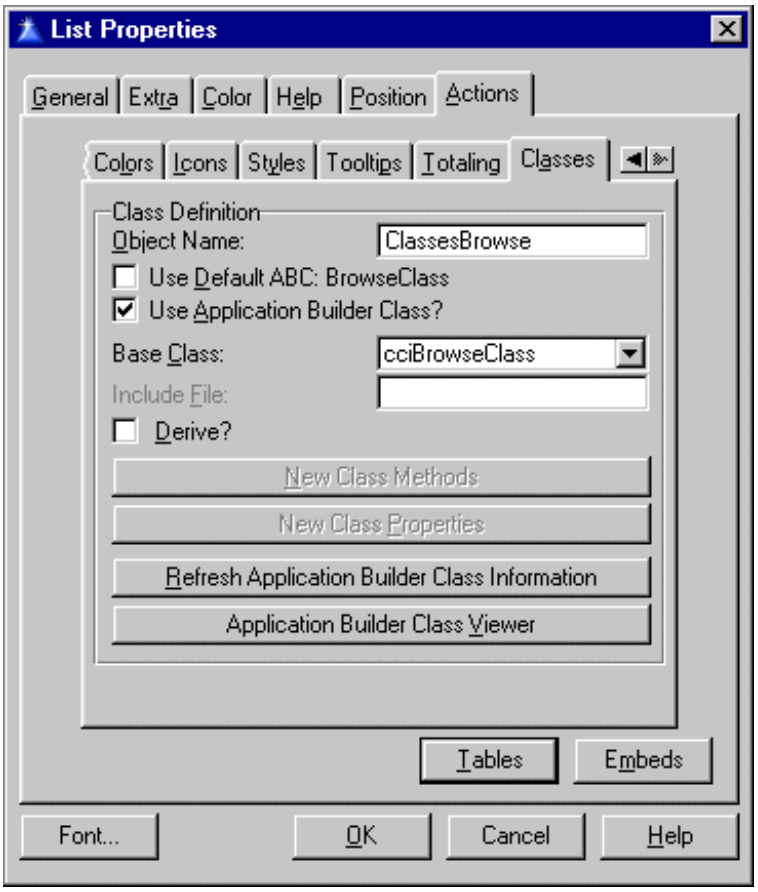

Figure 6. The right hand table's Actions/Classes settings

You're almost done! Next, go to the embeditor and find the Window Manager `Init`

method embed. I use an embed near the end, say at the call to `PrepareAlerts`. Add the following code:

```
ClassesBrowse.Init( |
   access:Enrollment, |  ! Linking FileManager
   ENR:StuSeq,        |  ! Linking File key
   ENR:StudentNumber, |  ! Linking File left field
   ENR:ClassNumber,   |  ! Linking file right field
   STU:Number,        |  ! Left file primary key field
   CLA:ClassNumber,   |  ! Right file primary key field
   InClass)              ! Local used to show icon
```

You're done! If you've set everything up correctly, you should be able to compile and run the application, and create many-to-many links simply by clicking on the checkboxes on the classes browse.

That's all for this week. Next time I'll explore the class code and show what's happening under the hood.

Download the source

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment

Example app fixed - the source I originally included with...
One more fix - there was a spurious ApplyFilter method in...
Dave, this is incredibly cool stuff! Not only does it...

Clarion Magazine

The Clarion Challenge - Using C in Clarion

Published 2001-11-14

You probably know that the Clarion environment includes the TopSpeed C compiler, which allows you to include C source files with any Clarion application. But have you ever actually tried using C code in a Clarion application? If you haven't, now's the time, and if you have, the following should be even easier.

Here's the challenge: Create a Clarion application (as an APP or source with PRJ, your choice) which calls a C function. You will pass the C function two values. The first parameter is a REAL (i.e. 3.14), and the second is a STRING containing a numeric value (i.e. 4.25). Add the two values in the C function and return the result as a REAL.

If you have any questions, post them as comments below. Send your completed applications or projects (with all necessary source) to editor@clarionmag.com.

Entries will be judged for compactness, elegance, and not least, the ability to do math.

Reader Comments

Add a comment

It may make life easier if the challenge specifies a REAL...
I agree with Gordon, in fact the Clarion documentation...
Dam! I just realised that my code was using a *cstring...
It should be a cstring - my apologies.
Changing the requirements at this late date? You're making...
<bg>

Checkboxes For Many-to-Many Relationships: The Source Code

by Dave Harms

Published 2001-11-15

Last week I introduced a class called `cciBrowseClass`, which is derived from the ABC `BrowseClass`. I explained how to use this class to add single-click many-to-many relationships between two tables. This week I'll look at the internal workings of that class, and explain how it leverages the power of `BrowseClass`.

The class declaration

Figure 1 shows the source listing for the `cciBrowseClass` declaration. This declaration follows the pattern of the ABC classes. In particular note the `!ABCIncludeFile` declaration, which tells the IDE that this class is to be considered as ABC compatible. When you start Clarion, or refresh the Class list from a class template dialog within the IDE, Clarion reads all of the .INC files in the `libsrc` directory, and parses those beginning with `!ABCIncludeFile`, adding the class labels to its internal list of available classes.

Figure 1.

```
!ABCIncludeFile

OMIT('_EndOfInclude_',_cciBrowsePresent_)
_cciBrowsePresent_ EQUATE(1)

   include('abbrowse.inc')

dataQ       queue,type
ID             long
OrigValue      long
CurrValue      long
            end


cciBrowseClass CLASS(BrowseClass),TYPE,↵
```

```
  MODULE('ccibrows.clw'),LINK('ccibrows.clw',↵
  _ABCLinkMode_),DLL(_ABCDllMode_)
DebugQ              &queue
Debug               byte(0)
DataQ               &DataQ,protected
IconField           &byte,protected
RightPrimaryID      &long,protected
LinkLeftField       &long,protected
LinkRightField      &long,protected
LinkKey             &key
LeftPrimaryID       &long,protected
LinkFM              &FileManager,protected
!--- methods ---
ApplyFilter         PROCEDURE,VIRTUAL
DebugMsg            procedure(String msg)
Init                procedure(FileManager LinkFM,↵
  Key LinkKey,*long LinkLeftField,*long LinkRightField,↵
  *long LeftPrimaryID,*long RightPrimaryID,*byte IconField)
Kill                procedure,virtual
Next                PROCEDURE,BYTE,VIRTUAL
RedisplayRecord     procedure
SetQueueRecord      PROCEDURE,VIRTUAL
TakeEvent           PROCEDURE,virtual

             end

_EndOfInclude_
```

This declaration must be `INCLUDE`d before you can use any of the classes. ABC does this automatically, but its possible to have multiple includes, which would cause duplicate symbol errors. The `OMIT` statement prevents the declaration from being included more than once (in Clarion 5.5 you can also use the `ONCE` attribute on the `INCLUDE` statement to prevent this).

You'll notice a number of references in the class's data section. These mainly reference the table that stores the linking data between the left and right tables (see Part 1 for a discussion of what I mean by "left" and "right"). There are a few other data elements for primarily internal use. Figure 2 lists the class's data.

Figure 2. The `cciBrowseClass` data elements

Field	Description

`DebugQ`	A simple queue declaration, used for some quick and dirty debugging. Discussed in the article text.
`Debug`	A byte flag indicating whether calls to `DebugMsg` will result in messages being added to `DebugQ`
`DataQ`	An internal queue used to cache status of the link between two records (and therefore the icon to display)
`IconField`	A reference to the local variable which is displayed as an icon in the right hand list box
`RightPrimaryID`	A reference to the primary key field in the right hand table
`LinkLeftField`	A reference to the linking table field that corresponds to the left table's primary key
`LinkRightField`	A reference to the linking table field that corresponds to the right table's primary key
`LinkKey`	A reference to the key that is used for lookups in the linking table (it must contain both `LinkLeftField` and `LinkRightField`)
`LeftPrimaryID`	A reference to the left table's primary key field
`LinkFM`	A reference to the linking table's file manager

The Init method

Before `cciBrowseClass` will do any work for you, you need to call its `Init` method. This is the *only* code you actually need to write and embed in the procedure. Here's

what I used in the ENROLL application:

```
ClassesBrowse.Init( |
   access:Enrollment, |  ! Linking FileManager
   ENR:StuSeq,        |  ! Linking File key
   ENR:StudentNumber, |  ! Linking File left field
   ENR:ClassNumber,   |  ! Linking file right field
   CLA:ClassNumber,   |  ! Right file primary key
   InClass)             ! Local used to show icon
```

Remember that Enrollments is the name of the linking table, and it has both `StudentNumber` and `ClassNumber` fields, to link students with classes. The `StuSeq` key contains both these fields; you could also use the `SeqStu` key which has the same two fields in reverse order. All you need from this key is a way to retrieve a single record based on the combination of primary IDs from the left and right tables. You also pass in the primary key field for the right table. This is the field, not the value; remember that since the right browse will be displaying the icons, you just need to know the field to get the value for the currently displayed record. Finally, you pass the byte variable used to display the checkbox icons in the browse.

Here's the source for the `Init` method:

```
cciBrowseClass.Init    procedure(FileManager LinkFM,↵
     Key LinkKey,*long LinkLeftField,↵
     *long LinkRightField,*long LeftPrimaryID,↵
     *long RightPrimaryID,*byte IconField)

ListControl    long

   code
   self.LinkFM &= LinkFM
   self.LinkKey &= LinkKey
   self.RightPrimaryID &= RightPrimaryID
   self.LeftPrimaryID &= LeftPrimaryID
   self.LinkLeftField &= LinkLeftField
   self.LinkRightField &= LinkRightField
   self.DataQ &= new DataQ
   self.IconField &= IconField
   self.RetainRow = 1
```

Most of the `Init` code involves assigning the passed `FileManager`, key, and fields to their respective references. The last line of the method sets the browse's (remember, this class is derived from `BrowseClass`) `RetainRow` property, which ensures that the selector bar remains on the line where I've just clicked, rather than jumping to the following line after the icon changes.

I also create the DataQ queue; that means I also have to be sure I dispose of the queue when the procedure is done. I do this in the Kill method:

```
cciBrowseClass.Kill        procedure
   code
   free(self.DataQ)
   dispose(self.DataQ)
   parent.kill()
```

Although I have to add some embed code for a specialized Init method, the Kill method doesn't take any special parameters, so I've simply overridden the base Kill method, ensuring that my code will be called automatically by the WindowManager. That's because Kill is a virtual method. It already exists in the base class, but because I've added an identical Kill declaration to cciBrowseClass, when I create an object which is an instance of cciBrowseClass, any code in the base BrowseClass which calls Kill will call my method instead of its own. For more on virtual methods see my ABCs of OOP article on the subject (http://www.clarionmag.com/cmag/v1/v1n5abcsofoop_part3.html).

Of course, any time you override a virtual method you should call the parent method as well, unless you have a specific reason not to. If I didn't call the parent Kill method, all the objects BrowseClass creates wouldn't be disposed of, and at best I'd have a memory leak.

I was reminded of the importance of calling the parent method while testing a procedure which used cciBrowseClass. In an earlier version I'd thought of using the data cache to store all potential changes, and then writing these changes whenever the browse's filter changed. I tried doing this with the ApplyFilter virtual method, but when I abandoned this approach I made the mistake of commenting out the method source, including the parent call! What do you suppose happened?

```
cciBrowseClass.ApplyFilter          PROCEDURE
   code
!   self.SaveChanges()
!   parent.ApplyFilter()
```

Without the call to the parent ApplyFilter method, the base BrowseClass never got to execute its own code which set the filter on the browse. As a result, when I set a filter on that browse, the filter never took effect. Ouch! This problem was present in the first and second releases of the source code from Part 1, but has now been corrected.

Almost all of the cciBrowseClass's code is contained in two methods: SetQueueRecord, and TakeEvent, both of which are, like Kill, virtual methods.

The SetQueueRecord method

The `SetQueueRecord` method is called by the `BrowseClass` after that class retrieves each record from the table. Typically you use `SetQueueRecord` to set up the value of any local variables which you're displaying in the browse, and this is also where browse icons are set. In the ENROLL application, where the first column is set to show an icon, the AppGen creates the code shown in Figure 2.

```
ClassesBrowse.SetQueueRecord PROCEDURE

! Start of "Browser Method Data Section"
! [Priority 5000]

! End of "Browser Method Data Section"
  CODE
  ! Start of "Browser Method Executable Code Section"
  ! [Priority 1300]

  ! Parent Call
  PARENT.SetQueueRecord()
  ! [Priority 5500]

  IF (InClass=1)
    SELF.Q.InClass_Icon = 2
  ELSE
    SELF.Q.InClass_Icon = 1
  END
  ! [Priority 8000]

  ! End of "Browser Method Executable Code Section"
```

Figure 2. The generated SetQueueRecord as seen in the embeditor

The InClass local variable is the one populated in the checkbox column in the right hand browse; when you set up the icon, you specified that the browse was to display off.ico by default, and on.ico when InClass was equal to 1. The following is the generated browse queue definition incorporating the icon field:

```
Queue:Browse          QUEUE
InClass                 LIKE(InClass)
InClass_Icon            SHORT
COU:Description         LIKE(COU:Description)
CLA:ScheduledTime       LIKE(CLA:ScheduledTime)
CLA:ClassNumber         LIKE(CLA:ClassNumber)
COU:Number              LIKE(COU:Number)
Mark                    BYTE
ViewPosition            STRING(1024)
                      END
```

The InClass_Icon is an automatically generated, and its this field that determines which icon will be displayed. The Help states the following about the I (icon) attribute

in a list box format string:

> An I (PROPLIST:Icon) indicates an icon displays in the column, at the left edge of the column (prepended to the data). An icon number is contained in a LONG field immediately following the data field in the QUEUE (or FROM attribute string). The LONG field contains a number that refers to an entry in a list of icons associated with the LIST control through the PROP:IconList runtime property. If an asterisk is also specified for color, this LONG must follow all the color information.

Since you don't actually want to display the `InList` variable, you use a picture of `@p p` in the list box format string (as explained in Part 1). All that matters here is the icon. The only missing piece of this puzzle is the assignment of icons to the list box, and that happens in the `WindowManager`'s generated `Init` method:

```
?List{Prop:IconList,1} = '~off.ico'
?List{Prop:IconList,2} = '~on.ico'
```

These icons have also been added, by the ABC browse template, to the application's project so they can be linked in; the ~ character tells the application to look for the icons internally, rather than on disk.

As you can see, the `InClass` variable itself needn't ever be used – you could just set the value of the `SELF.Q.InClass_Icon` field directly. But since everything downstream of `InClass` is handled by the templates, it's just as easy to set the value of the local variable at the appropriate point, which is in `SetQueueRecord`.

There are actually three different `SetQueueRecord` methods involved here. The base `SetQueueRecord` is declared as part of `BrowseClass`. Since you're using `cciBrowseClass`, that class's `SetQueueRecord` is next in line, and finally the application generates another `SetQueueRecord` as part of the procedure's generated source.

Because these are all virtual methods, the "outermost" `SetQueueRecord` is the one that will be called first, and it will (or should!) call its parent, and so on. The sequence of calls is as follows:

1. `ClassesBrowse.SetQueueRecord`
2. `cciBrowseClass.SetQueueRecord`
3. `BrowseClass.SetQueueRecord`

ClassesBrowse looks like this:

```
ClassesBrowse.SetQueueRecord PROCEDURE

  CODE
```

```
PARENT.SetQueueRecord()
IF (InClass=1)
  SELF.Q.InClass_Icon = 2
ELSE
  SELF.Q.InClass_Icon = 1
END
```

The first call is to the parent method, which is cciBrowseClass.SetQueueRecord:

```
cciBrowseClass.SetQueueRecord PROCEDURE()
   code
   ! Look for the value in the queue
   if self.LeftPrimaryID > 0
      self.DataQ.ID = self.RightPrimaryID
      get(self.DataQ,self.DataQ.ID)
      if errorcode()
         self.LinkLeftField = self.LeftPrimaryID
         self.LinkRightField = self.RightPrimaryID
         if self.LinkFM.Fetch(self.LinkKey)|
             = level:benign
            ! If not found, then get it from the data
            ! file
            self.IconField = 1
         else
            self.IconField = 0
         end
         ! Add this value to the queue
         self.DataQ.ID = self.RightPrimaryID
         self.DataQ.CurrValue = self.IconField
         self.DataQ.OrigValue = self.DataQ.CurrValue
         add(self.DataQ,self.DataQ.ID)
      else
         ! Get the value from the queue
         self.IconField = self.DataQ.CurrValue
      end
   end
   parent.SetQueueRecord()
```

This method first looks in its internal queue for a record that matches the combination of left and right table primary key fields. If the value is in the queue, then the code simply retrieves the value and assigns it to that byte icon field; otherwise it looks in the linking table for a matching record, and sets the icon field appropriately. This method then calls the parent `SetQueueRecord`:

```
BrowseClass.SetQueueRecord PROCEDURE
  CODE
    SELF.Fields.AssignLeftToRight
    SELF.ListQueue.SetViewPosition(POSITION(SELF.View))
```

The "left" and "right" as seen by the `BrowseClass` are *not* the same as the "left" and "right" I'm describing in this series of articles; here they refer to the view and the queue contents. You really don't need to worry about this code – I've just included it so you can follow the thread of execution.

After `BrowseClass.SetQueueRecord` completes, control passes back to `cciBrowseClass.SetQueueRecord`, and then immediately back to `ClassesBrowse.SetQueueRecord`. Now the `InClass` variable, which is `self.IconField` to the class, has the correct value, and the generated code assigns the appropriate icon for display.

The TakeEvent method

That's how to display the icon; to change it, you need to intercept the user clicking on the browse's icon field. That's easiest to do with the `TakeEvent` method:

```
cciBrowseClass.TakeEvent PROCEDURE
lc    long
   code
  compile('***',_c55_)
   lc = self.ilc.getControl()
  ***
  omit('***',_c55_)
   lc = self.ListControl
  ***
   if field() = lc and |
     event() = event:Accepted |
     and keycode() = MouseLeft |
     and lc{proplist:mouseuprow} = |
     lc{proplist:mousedownrow} |
     and lc{proplist:mouseupfield} |
     = lc{proplist:mousedownfield} |
     and lc{proplist:mousedownfield} = 1
      ! Get the current record
      self.UpdateViewRecord()
      ! Update the buffer
      self.UpdateBuffer()
      if (self.IconField)
         ! If the link exists, remove it
```

```
        self.LinkLeftField = self.LeftPrimaryID
        self.LinkRightField = self.RightPrimaryID
        if self.LinkFM.Fetch(self.LinkKey) = level:benign
          compile('***',_c55_)
           self.linkFM.DeleteRecord(0)
          ***
          omit('***',_c55_)
           delete(self.LinkFM.File)
          ***
        end
      else
        !  Create the link
        self.LinkLeftField = self.LeftPrimaryID
        self.LinkRightField = self.RightPrimaryID
        self.LinkFM.TryInsert()
      end
      self.RedisplayRecord()
   end
   parent.TakeEvent()
```

I ran into a minor difficulty when porting this class from Clarion 5.5 back to 5.0. There have been some changes in the classes in 5.5, and the way you refer to a browse's list control is no longer the same, and 5.5 also added a `DeleteRecord` to the file manager. I needed code that would compile under both releases, so I used `COMPILE` and `OMIT` statements to conditionally include/exclude code based on the version of Clarion.

The first thing `TakeRecord` needs to do is determine where you've clicked on the list box. It tests to see that the control is the list box, that between the time you clicked down and then released the mouse button you didn't change the column or the row, and that you clicked on the first column of the list box. If all of that matches, then its okay to go ahead and toggle the link.

The call to `UpdateViewRecord` regets the data from the database, and the call to `UpdateBuffer` ensures that the browse class is working with that data in its queue. Then its just a matter of creating or deleting the linking record.

Finally, the call to the `RedisplayRecord` method triggers the display of the new value as an icon, via a subsequent call to `SetQueueRecord`. This method just mimics what happens when you return from an update form.

```
cciBrowseClass.RedisplayRecord PROCEDURE()
ChangeIt    byte(ChangeRecord)
Finished    byte(RequestCompleted)
```

```
   code
   self.ResetFromAsk(ChangeIt,Finished)
```

This class works well for my purposes, but it does have some restrictions. First, it only responds to single clicks on the icon field. That could be a problem if you want to prevent the user from inadvertently creating a link, but in my case the speed and convenience of the single click far outweigh any risk. As well, this class assumes that you will only put the linking checkbox in column 1. That's not a bad assumption, but it may not always be accurate. You could change this column to a class property to get around that limitation.

Debugging

I haven't shown any debugging code so far, but if you look at the source in the downloadable zip you'll occasionally see something like this:

```
self.DebugMsg('cciBrowseClass.SetQueueRecord |
  (' & self.IconField & '/' & self.RightPrimaryID & ')')
```

This code calls the class's DebugMsg method:

```
cciBrowseClass.DebugMsg     procedure(String msg)
   code
   if self.Debug and ~(self.DebugQ &= null)
      self.DebugQ = msg
      add(self.DebugQ)
   end
```

All this class does is add a string to a queue, which can be displayed on the procedure window. To use this debug feature, just create a queue with a single string field, and populate a list box on your procedure's window. Set the list box's `FROM` attribute to the queue's name. After you call the `cciBrowseClass`'s `Init` method, add the following code, substituting your own browse and queue names:

```
<browse object name>.DebugQ &= <my local debug queue>
```

In Clarion, you can use a queue's label for the first field in the queue. That means that this code:

```
MyQ = 'some data'
ADD(MyQ)
```

is functionally identical to:

```
MyQ.myField = 'some data'
```

```
ADD(MyQ)
```

The `cciBrowseClass`'s `DebugMsg` method uses this feature to add messages to the queue you specify, and which you can display in your own procedure.

Summary

When I started writing what became `cciBrowseClass`, I began with the idea of a class that would be an adjunct to the existing ABC BrowseClass. I quickly realized that much of what I wanted to do would be a lot easier if I just derived my class from `BrowseClass`. And in the end, it really didn't take very much code, or particularly complicated code, to get the effect that I wanted. The hardest part, as usual, was figuring out what the `BrowseClass` was doing in the first place.

Download the source

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment

Calling By Address, STARTing By Address

by Jim Kane

Published 2001-11-16

Recently a project came along that involved either calling or STARTing a procedure in a DLL which was loaded dynamically at runtime. Some of the procedures to be called used the Pascal calling convention, and some used the C calling convention. This DLL contained an MDI window that needed to be STARTed, and was a legacy DLL that needed to be run from an ABC application. Now how is that for a project?

My first thought was scheduling some vacation time in the hopes some one else would get the project done before I came back. Unfortunately this was a rush project so any vacation probably would not have been approved, and since I had already used the "my grandmother died and I have to go to the funeral in Florida" routine a few times I was stuck!

Normally when I need to call something by address I use a little TopSpeed C, but that involves typing both a typedef and a prototype for each function to be called. Likewise I've seen a number of compiler tricks to call by address, but they also involve creating a prototype and something extra. Being a charter member of the Lazy Programmer's Club, and knowing the project at hand involved a sizable number (hundreds actually) of prototypes, I wanted another option.

What I wanted was a nice simple class that loads a DLL and calls a procedure in it, without requiring prototypes. The Clarion CALL function will do that, but it won't let me pass parameters or get back values. I had something more like this in mind:

```
!1=show debug messages
If ~CallDllCl.Init('RPCRT4.DLL',1)
  If ~CallDllCl.Call('UuidCreate',address(Uuid))
    Message('Got the UUID')
  End
End
CallDllCl.kill()
```

This example loads a DLL called RPCRT4.DLL and calls its `UuidCreate` procedure. The UuidCreate procedure takes one parameter which is passed by address.

In the downloadable source for this article is a small project that uses the `CallDllCl` class to call the `UuidCreate` function and generate UUIDs. They come in handy for COM work and for replication where an autoincremented field alone is not enough. The sample application is called UUIDGEN; besides `UuidCreate` it calls a few other functions by address to display a nicely formatted UUID and/or paste it to the clipboard. Essentially that is all there is to calling by address using the `CallDllCl` class – just call the class's `Init` method to load the DLL, call the DLL functions as desired, then `Kill` the class.

To meet my other needs I also needed a `Ccall` method to handle calling using the C calling convention, and a `Start('ProcName')` method to handling starting a DLL procedure containing a MDI window.

Now that I knew where I wanted to go, it was time to quit trying to escape and get to it! If you're not particularly interested in how the class works, you may wish to skip to the section concerning calling `Start` dynamically.

How the class works

Before you can call a DLL's methods you have to tell Windows to load the DLL. That just involves one call to the API `LoadLibrary` function. The `Init` method does this and saves the `hDLL` or handle to the DLL that `LoadLibrary` returns in a class member variable.

```
callDLLclType.Init   Procedure(|
       string pDllPath, byte pDebug=0)
```

```
cPath cstring(File:maxfilepath),auto
  Code
  SELF.Debug=pDebug
  cpath=clip(pDllPath)
  SELF.hDLL=loadlibrary(cpath)
  if ~SELF.hDLL then
    SELF.TakeError('LoadLibrary failed for: ' |
      & cpath)
    return return:fatal
  end
  Return Return:benign
```

Another API call that will get the same type of handle `LoadLibrary` returns is `GetModuleHandle`. If you know the DLL you need to call is in memory and will not be unloaded, you can call `GetModuleHandle` instead of `LoadLibrary`. However, unlike `LoadLibrary`, `GetModuleHandle` does not increase the reference count on the DLL so it is quite possible for other code to unload the DLL before you are done with it, resulting in GPFs. Because of that danger, I do not use `GetModuleHandle`. If `LoadLibrary` finds the requested DLL in memory, it does not load another copy anyway. In my opinion the potential benefits of `GetModuleHandle` are minimal and the dangers real, so I choose not to use it.

The only other quirk of `LoadLibrary` is that it is path sensitive. To explain, if you were to call `LoadLibrary('C:\path1\myDLL.DLL')` followed by `LoadLibrary('C:\path2\myDLL.DLL)`, two copies of the DLL would be loaded into memory, even if `myDLL.DLL` in `path1` and `path2` was identical. To avoid that problem, do not specify a path or always specify the same path to avoid a duplicate, unless for some unusual reason you really do want two copies of the DLL.

If the DLL isn't in the specified path, or there was no specified path, `LoadLibrary` looks in the directory the program was loaded from, then the Windows path, the system path, and then the path stored in the environment. The first DLL found in that search is loaded. If you ever want to find out what would be loaded (without actually loading the DLL), or if you just want to know if a DLL of a given name exists anywhere, use the `SearchPath` API function. `SearchPath` can be very handy at times.

In just about all the classes I write, I add a debug member variable and a TakeError method. The TakeError code is like this:

```
CallDllClType.TakeError     Procedure(string pError)
  code
  SELF.Errorstr=pError
  If SELF.Debug then message(pError).
  Return
```

If Debug is non-zero then the message is displayed. I turn debug mode on or off as needed.

Getting the address for a procedure in a loaded DLL involves just one API call as well. In addition, I save the address for future use in a queue so it only has to be looked up once.

```
CallDllclType.GetAddress  Procedure(string procname)
cProcName cstring(80),auto
lpaddress long,auto
  code
  SELF.AddressQ.procname=procname
  get(SELF.AddressQ, SELF.AddressQ.Procname)
  if ~Errorcode() then
    return SELF.AddressQ.lpAddress
  end
  cProcName=clip(procname)
  lpAddress=GetProcAddress(SELF.hDLL,cProcName)
  if lpAddress then
    SELF.AddressQ.procname=procname
    SELF.AddressQ.lpAddress=lpaddress
    add(SELF.AddressQ,SELF.Addressq.procname)
  else
    SELF.TakeError('get adress failed for ' & cProcName)
    Return SELF.eProcFail
  end
  return lpaddress
```

First the code looks in the AddressQ to see if the address has be looked

up before; if not, it calls GetProcAddress. Be aware that GetProcAddress is case sensitive so be sure to get the case correct in the procedure name passed to GetAddress. By the way, the AddressQ is created in the constructor and disposed in the destructor.

In some cases the error code returned by GetAddress or subsequent functions to be discussed can become confused with values returned by the called procedure. For example, many procedures return 0 to indicate success. If GetAddress returned 0 for failure, things could get confusing. To prevent this confusion, I defined a member variable eProcFail; this can be set to any convenient value and it is returned when GetAddress fails. The default value of eProcFail is 0; this works well for most API functions, which usually indicate failure by returning 0.

Once the DLL is loaded and you have the function's address, the only thing left is to call the function. This is a two step process. The first step is to put the parameters plus the return address on the stack. The second step is to jump or transfer control to the address of the function to be called. Fortunately I found a way to use the TopSpeed calling convention to my advantage. Here's an example of calling a function using the Pascal calling convention with two parameters.

```
callDLLcltype.call_p2   procedure(|
   string procname, long p1, long p2)
lpaddress long,auto
  code
  lpaddress=SELF.GetAddress(procname)
  if ~lpaddress then Return SELF.eProcFail.
  return callA_p2(lpaddress,0,0,0,p2, p1)
```

The first two lines get the function's address (with GetAddress) and return an error if this fails. The function CallA_p2 uses the Topspeed calling convention to pass two parameters to the DLL function, but itself takes five parameters. The first parameter using the TopSpeed calling convention goes into the EAX register. This means the address to be called is put into the EAX register. The TopSpeed calling convention puts the next three parameters into other registers and are not of interest here. I just pass zero to fill the parameter spots. The next two parameters go onto the stack right where they are needed. Since the TopSpeed calling convention puts parameters on the stack in the

opposite order from the Pascal calling convention, the parameters are reversed (p2 then p1). The TopSpeed Programmer's Guide has a more detailed explanation of how the TopSpeed calling convention works, but this will do for now.

At this point all I need to do is transfer control to the address, lpAddress, currently in the EAX register. To do that I used one line of assembler code in a procedure called CallA.A:

```
segment CallA_Text('CODE',29H)
public CallA:
    jmp  eax
```

The segment line specifies that the following lines contain code (as opposed to data), and this data is byte aligned, 32 bit, and can be combined with other code segments. For virtually all purposes ('CODE',29H) works. The public CallA is just a label for the code. The third line is what transfers control to the address in the EAX register. Whenever you use this class in your code, just press the project button in the IDE, highlight the external source branch in the tree and press the add file button. When prompted for a file name type in Calla.a. That causes the IDE to call the assembler and assemble that little bit of assembler code for you when you make the application, or project. You could also write a template to add it and the class at the same time, but it never seemed worth the effort to me.

Notice that all of the call methods take only longs as a parameter. This is not a problem. If you need to pass data by value, whether it be a byte, short or long, just pass the value in the long. All parameters when put on the stack are placed as 32 bit (long) values anyway. The stack is one size, 32 bit, so there is no problem. If you are passing data by address, for example a CString, or any variable type when a return value is expected, pass in the variables address (using the Clarion ADDRESS function). In the UuidCreate function, the UUID group is passed by address. You *must* know whether to pass by address or by value. A wrong choice will result in a GPF most of the time. But on the bright side, you do not need to type a prototype. Not having to type prototypes is a good thing! If you do not agree I'll have no choice but to revoke your membership in the Lazy Programmer's Club.

The call methods all assume the DLL's function is returning a long. If the function you are calling does not return anything, just ignore the return value. If the function you are calling returns a byte or short, copy the long result to a byte or short, or mask off the high order bits with BAND before using the return result.

Now you know how to call DLL functions using the TopSpeed and Pascal calling conventions. Next week I'll show you to use the C calling convention, and how to START dynamically loaded DLL procedures.

Download the source

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Clarion Magazine

A New Look, And A Topical Index

Published 2001-11-20

This week brings a number of changes to Clarion Magazine. First, there's the new visual design. The ClarionMag site now makes more extensive use of style sheets, and will probably look best under IE, although we've gone to some effort to make things readable with various versions of Netscape as well.

More importantly, article pages now have navigation aids to help you find your way around the site. With more than a million words and over 850 articles by scores of authors, ClarionMag.com is an incredible source of information for Clarion developers. The difficulty is finding the information you need, when you need it. At the top of each article page you'll now find a topic list. Click on any link in this list, and you'll be taken to the topical index, which will show you a list of available subtopics. Click on a subtopic to get a list of available articles.

As well, each article now also lists related subcategories and articles on the right side of the page. Let me know if you find these links helpful. The topical index is under constant revision; if you think an article should be assigned to or removed from a category, I'd like to hear about.

Although a number of people have tested the new layout, there are still a few rough edges. If you find something that isn't working properly, email me. You can also post your comments at the bottom of this page.

Dave Harms

Publisher

Reader Comments

Add a comment

Very COOL look.
Wow! That's a big change! And I'm sure it will continue...
Well done. The topical index really exposes how well...
Thanks, everyone! I'm glad you like the new look, and the...
I cannot read any of the screens in the new web format. ...
Allen, can you be more specific?
Dave, congrats for the new look, it´s very nice and very...
Very cool look and feel, Dave! Anytime you want to...
Wow, thanks Mark! Hang on, where's my rate sheet...
Thanks, Pablo!
The new look is very good, but also very unreadable. I...
This might be obvious to most of you, but it wasnt to me....
I've gone back to fixed fonts - clearly I have a bit more...
Bless your hearts for all this work you've done for us,...
Colour scheme very pleasing to the eye and makes it easy to...

Calling By Address, STARTing By Address (Part 2)

by Jim Kane

Published 2001-11-21

Last week I introduced a class that you can use to load DLLs at runtime and call those DLL functions without having to write prototypes. Unlike the Clarion CALL function, this class lets you pass parameters to those dynamically loaded DLL functions, and get return values. I showed how to use that class to call functions with the Topspeed and Pascal calling conventions. This week I'll look at the C calling convention, and STARTing dynamically loaded DLL procedures.

The C calling convention presents more of a challenge; after the procedure you call returns the parameters (p1 and p2 in last week's example) are still on the stack and need to be removed. To do that I wrote another very short assembler piece called FixStack. FixStack needs to add eight bytes (two parameters, each four bytes) to the stack pointer; in effect, this removes the two parameters from the stack. The only complication is the return address for FixStack itself is on the stack before the parameters to be removed. To get the return address off the stack, FixStack pops its return address into the ECX register, then moves the stack pointer the number of bytes specified in EBX and jumps to the return address. The code is called like this:

```
callDLLcltype.ccall_p2       procedure(|
          string procname, long p1, long p2)
lpaddress long,auto
  code
  lpaddress=SELF.GetAddress(procname)
  if ~lpaddress then Return SELF.eProcFail.
```

```
  return fixstack(callA_p2(lpaddress,0,0,0,p2, p1),8,0,0)
```

The return result from CallA_p2 goes in to the EAX register since it is parameter 1. The number 8 (the number of bytes to remove from the stack) goes into EBX , and the last two dummy parameters go into ECX and EDX. By putting the dummy parameters into these registers, the compiler marks the corresponding registers as dirty and doesn't mind the fact that FixStack uses them. The assembler code for FixStack is as follows:

```
public FIXSTACK:
    pop    ecx
    add    esp,ebx
    jmp    ecx
```

None of the sample code with this article uses the C calling convention but it has been well tested calling some Palm OS functions used in making Palm Conduits.

STARTing a Procedure

The last challenge I faced was calling a legacy EXE from a ABC program and starting a MDI window in the legacy app. Here's what I did. The first step was to open the legacy app and press the project button. I highlight the top line of the project and press the Properties button, and changed the target type from EXE to DLL. Before recompiling, to make life easier, I went to the procedure I wanted to call and in the prototype entry control on the procedure properties window I added the ,Name('PROCNAME') attribute to the prototype line, where PROCNAME was the name of the procedure as I wanted to call it from the ABC application. That meant I did not have to deal with case or name mangling. In my sample app, the procedure is called MdiWindow, with the adjusted prototype as follows::

```
MdiWindow(string cmdline),name('MDIWINDOW')
```

Now I recompiled to make the legacy DLL. I did not make anything external, since this DLL will not share any data with the caller.

To test, I created an ABC program named Caller.app. On the main menu, I added an item called 'Legacy'. In the embed code for when

accepted, for the Legacy menu item, I added this line:

```
ThreadNumber#=|
  callDLLcl.start('MDIWINDOW',0,'LEGACY DLL')
```

The first parameter is the procedure name and is case sensitive. Since I used the `Name` attribute in the DLL, I know it is upper case without having to use LibMaker to examine the DLL directly. The 0 means use a default stack size. The last parameter is a string passed to the procedure. The class can handle starting a procedure with 0,1,2 or 3 string parameters as can the underlying Clarion `START` command.

In the data section I declared the class:

```
callDLLcl callDLLcltype
```

In the procedure's `Thiswindow.init` embed point I added the initialization code:

```
X#=CallDLLcl.init('Legacy.DLL',1)
```

You can add any additional error checking you wish. The ‘1’ turns on debug messages so if there is an error, a message will be displayed. This one line loads the legacy DLL into memory. To clean up when the procedure finishes, I added the following in the embed for the procedure's `ThisWindow.Kill` method:

```
CallDLLcl.kill()
```

In the global embeds at before global includes I add an include so that the class would compile:

```
Include('callDLLcl.inc')
```

Although normally this class needs its assembler code, but the Start method doesn’t need any assembler so adding `calla.a` to the project is optional.

After compiling, I click the menu item to call the legacy DLL and up it popped, displaying the passed parameter.

The only code for the Start method is simple and very similar to that for the call methods:

```
CallDllClType.Start   procedure(String pMdiForm, |
   long StackSize=0,<string pS1>,<String pS2>,<String pS3>)
lpMdiForm long,auto
  code
  lpMdiForm = SELF.GetAddress(pMdiForm)
  if lpMdiForm=SELF.eProcFail then
    SELF.TakeError('Start address resolution failed')
    return 0
  end
  if OMITTED(4) and Omitted(5) and Omitted(6) then
    Return CLASTART(lpMdiForm, Stacksize)
  end
  if ~Omitted(4) and Omitted(5) and Omitted(6) then
    Return CLASTART1(lpMdiForm, StackSize, pS1)
  End
```

And so on for other parameter configurations. Remember that all class methods have an implicit first parameter which is the method's class, so the OMITTED count is one higher than you might think.

As you can see, Start calls getAddress to turn the string parameter name into a address; if there's a problem, the code calls TakeError. After figuring out how many string parameters have been passed it just calls a variant of the CLASTART library procedure to invoke the Clarion Start procedure. The prototypes tell the compiler to expect an address rather than a procedure name:

```
expect an address rather than a procedure name:
   module('')
     clastart(long lpproc,long pStackSize)|
        ,long,proc,name('CLA$START')
     clastart1(long lpproc,long pStackSize,|
        string param1),long,proc,name('CLA$START1')
     clastart2(long lpproc,long pStackSize,|
        string param1, string param2),long,|
        proc,name('CLA$START2')
```

```
    clastart3(long lpproc,long pStackSize,string param1,|
       string param2, string param3),long,|
       proc,name('CLA$START3')
end
```

CLA$START is the name exported by the runtime for the Clarion Start procedure.

I found that in C5B starting a MDI procedure in a dynamically loaded DLL containing threaded file definitions could cause a GPF when the program ended; the bug has been fixed in C55. . Alexey Solovjev was kind enough to trace down the GPF in C5B and said it was in code that checked that all files were closed. Since it is only a check that is GPFing, as long as you close all files in your code before the program terminates so the Clarion runtime does not have to, there does not seem to be a problem.

To get by in C5B I found I could avoid the GPF by not explicitly unloading the dynamically loaded DLL. As a result the call to `CallDllCl.kill` is commented out. The code could be put back in after switching to C55. When `Kill` is not explicitly called, windows unloads the DLL for you when the program terminates. This avoids the GPF. There is no memory leak. The only potential drawback is you cannot unload the DLL while the program is still running. I normally leave the DLL loaded anyway, since it is not particularly convenient to determine when the `Started` procedure has ended and unloading the DLL while the code in the DLL is still running leads to a certain GPF.

Another possible use for these prototypes is that they make it possible to store the address of a procedure in a long in a queue and start the corresponding procedure by address rather than using the procedure name.

The `CallDLL` class in the download contains a few other features. Perhaps the most important additional feature is the ability to call a procedure when the windows API provides the address of a function to call, and I do not need to load a DLL and determine the address. This is in contrast to the usual case where I know the name of a DLL and the name of the procedure in the DLL, and need to load the DLL and *then* determine the address of the procedure from the name of the procedure.

This situation occurs, for example, in an ISAPI extension where when IIS provides the address of several different functions to call. There is no need to load a DLL or determine the address – it is supplied. To allow for this possibility, call the `init` method with a blank DLL name. This causes the class to set a member variable (`IgnoreHDLL`) to true. When `IgnoreHDLL` is set to true, the class doesn't try to load or unload a DLL as it normally would. If I want to call the function I know the address of by its name, I need to get the address I have into the class's address queue. This can be done by calling the `addAddress` method. Once the class knows the functions name and its address, any of the normal functions to call by address such as `Call(procedurename,...)` can be used to call the target function using the address the windows API provided.

Lastly, if you obtain the address of a procedure or data variable located in another DLL using the Clarion Address function, the address returned is the address of the memory location where the procedure's or data's address is stored and not the data itself. It just takes one line of code to do a quick `memcpy` to retrieve the true address. This process is frequently called dereferencing the variable. The `callDLLcl` method has a method to wrap this as well:

```
DeRefPointer procedure(long lpPointer, *long pDeref)
```

The parameter `lpPointer` is typically returned from `Address` and the `pDeref` output variable is the "true" address that should be used with API calls.

Summary

I think you'll find this class to be quite useful. It will dynamically load DLLs at runtime, can handle Clarion (TopSpeed), Pascal, and C calling conventions, and can also call the Clarion Start function. And, with just a little assembler magic, it manages all this without requiring you to create a single procedure prototype.

Download the source

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard

University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Clarion Magazine

Using SQL Server's Data Transformation Services (DTS)

by Ayo Ogundahunsi

Published 2001-11-27

In my previous articles I demonstrated how to convert the Inventory application from TopSpeed file format to a more powerful Database Engine – Microsoft SQL Server. In converting an application, changing database drivers is not the only requirement. You also have to provide a way to convert existing data. In most cases, existing clients are probably running applications using the TopSpeed database driver, and transition to the upgrade should be easy and accurate.

One way to convert data is to use Microsoft's Data Transformation Services (DTS). As the name indicates, DTS transforms data from one form to the other. DTS is a very powerful and effective tool, and comes bundled with SQL Server versions 7 and 2000. In this article I will demonstrate how to use DTS to move data from the Inventory example to the converted SQL Server application.

Using DTS

You can use DTS to move data from Data/ODBC Sources to SQL Server and vice versa. For example, you can move data from an Oracle Database to a Sybase Database without going through SQL Server at all.

One of the strengths of DTS is its ability to assemble tasks or functions and connections to heterogeneous systems, and synchronize all these together into what is called a package. The tasks can be importing or exporting data, or sending an email as soon as a particular task is completed. The actions, processes, and settings created for transforming data can be saved in SQL Server as a package, or externally as a Visual Basic Script (VB Script) that can be run from a Visual Basic Application.

Improving the database

It's quite simple to set up DTS to transform the data contained in the four files of the Inventory application (INVHIST.TPS, PRODUCTS.TPS, VENDORS.TPS, and

ZIPCODES.TPS). As usual, I will be using the terms "files" and "tables" interchangeably, however I will specifically use "files" when I am talking about TopSpeed data, and "tables" when I am referring to SQL Server data. I will also introduce some additional tables that will make the Inventory Application's database design more practical.

The `VENDORS` file contains City, State and ZipCode fields. It also contains a field called `ID_ZIPCODES` that is to be the linking field with the `ZIPCODES` files. To follow normalization rules, and achieve better performance, I will remove the `City`, `State`, and `ZipCode` fields from the `VENDORS` files. I'll also create separate `City` and `State` tables to replace the fields in `Vendors`, but this data will also be linked to `Vendors` via `ZipCode`. While this is a good idea, understand that if you have to enter data into another table, say a `Customer` table you must know the zip code for the customer or you will not be able to add the record.

The interrelation will now be:

```
STATE <---->> CITY <---->> ZIPCODES <---->> VENDORS
```

Every zip code belongs to some city, and each city is in a state, so as long as you have the zip code stored in the Vendor table you can easily get the rest of the information. The idea is to try to eliminate redundant or repeated data as much as possible. This is called Data Normalization. Please read Tom Ruby's article Five Rules for Managing Complexity: Part 2 for more information on normalization.

Updating the ZipCodes file

The `ZIPCODES` file in the original Inventory application contains about 3,924 zip codes. This is not even half of the zip codes in the United States, but it is possible to add data from an external table with enough zip codes that gives a realistic figure. You can download a more accurate zip code file from Steve Parker's website. This file contains about 48,254 records.

In Part 2 of this Data Conversion series I'll show how to import a text file containing the 50 US states as well as their abbreviations into a SQL Server database, with the `CITY` table filled accordingly, and the linking IDs appropriately inserted. For now, I am going to do a straight import of the four files directly into the SQL Server backend.

Connecting to data sources

When connecting to a Data source, what readily comes to my mind is Open Data Base Connectivity (ODBC). I can use ODBC with DTS, but I can also connect to a data source by using OLE DB Drivers.

Microsoft made some improvements to ODBC by creating what is know as OLE DB

which is a way of providing data access at a component level with an interface that is not restricted to relational data only (as implemented in ODBC), but any other kind of non-structured data that can include spreadsheets, email, etc. The driver for OLE DB is know as an OLE DB Provider.

Microsoft ships an OLE DB Provider (Microsoft OLE DB Provider for ODBC) with SQL Server.

See the following sites for more information on ODBC and OLE DB:

- http://www.microsoft.com/data/oledb/default.htm
- http://www.oledb.com/ole-db/guide.html

Data conversion options

There are two popular ways to convert data from the TopSpeed files to SQL Server tables. One is by using Data Transformation Services (DTS), and the other is by adding TopSpeed Data Source as a Linked Server to the SQL Server Environment (more about this in a future article) using OLE DB. With a linked server, you can send SQL to the database on the linked server and it will behave as if it were an SQL Server database. However, the response to SQL requests is not as fast as if an SQL Server database is being accessed directly. In this article, I will limit my explanation to DTS.

TopSpeed ODBC Driver

When you install the Enterprise Edition of Clarion, you automatically install the TopSpeed ODBC Driver (developer version) as well. The developer version of the driver *cannot* be deployed with any application; if you want your customers to use it you need to contact Soft Velocity Sales for ODBC license requirements.

Nevertheless, the developer version is appropriate for the purposes of this article. Since this is an ODBC driver, the first step is to create a Data Source Name (DSN).

> **WARNING:** It is important to mention here that any attempt to do anything too complicated with this TopSpeed ODBC Driver freezes my system. For example when I tried to create a linked server to TopSpeed, the process froze my SQL Server, and nothing worked until I rebooted the PC. Probably this is a security feature by SoftVelocity, but it is strange that I was unable to run my Enterprise Manager after this. Shutting down the SQL Server service and restarting it did not make any difference either. I had to restart my PC.

You can add a DSN by using the ODBC Data Source Administrator as shown in Figure 1.

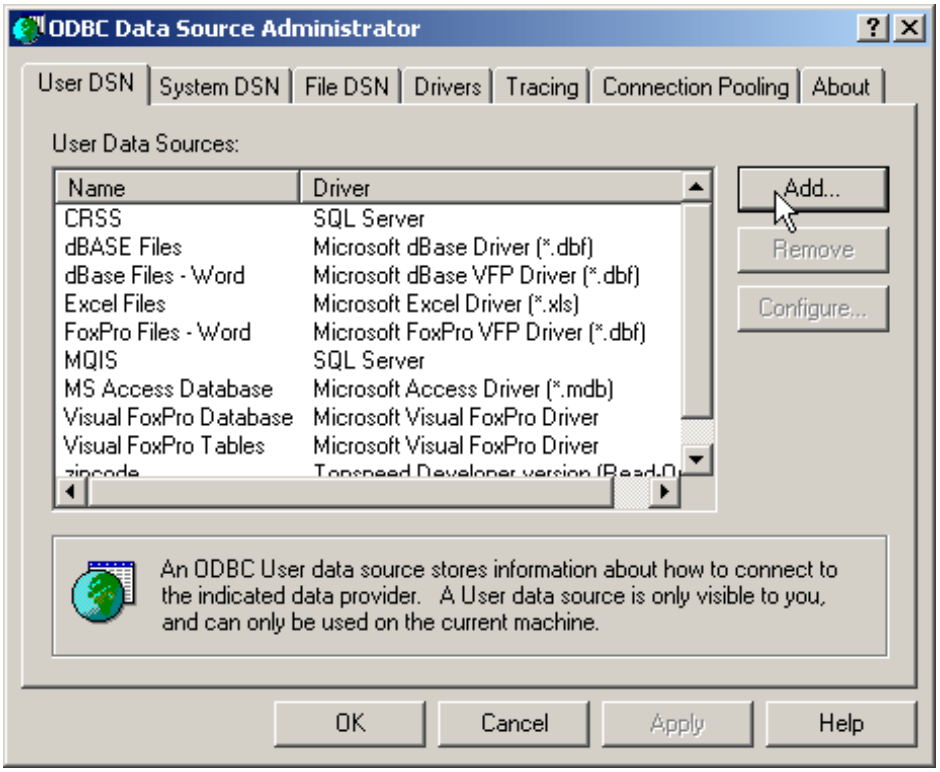

Figure 1. Adding a DSN, step 1

Two kinds of the TopSpeed ODBC Driver are usually installed. One is Read-Only. Either will work for this purpose (See Figure 2).

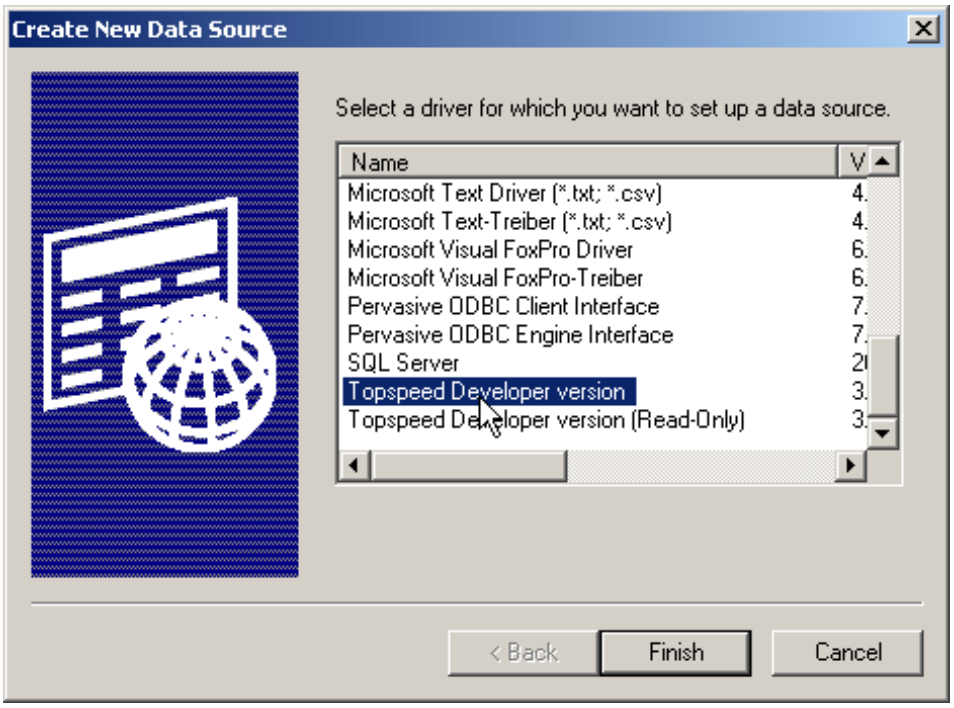

Figure 2. Adding a DSN, step 2

You then specify the physical location of the TPS data file., as shown in Figure 3.

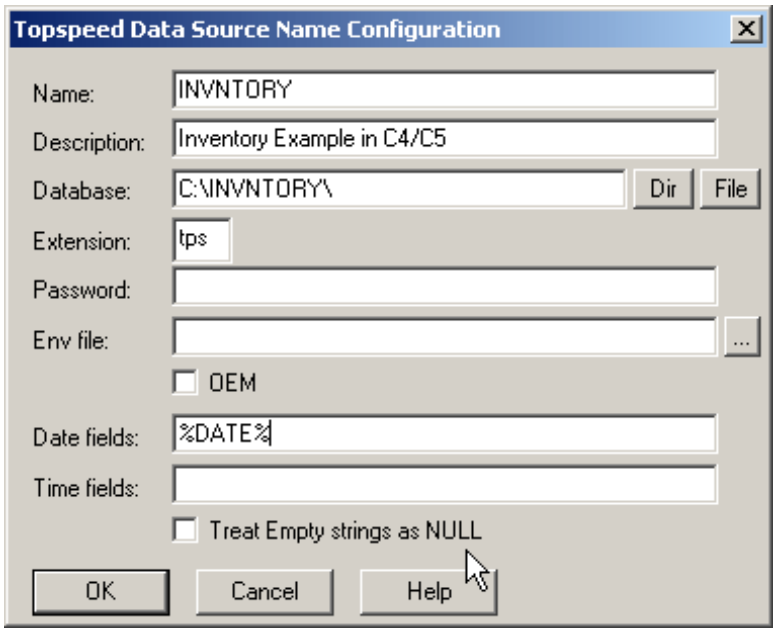

Figure 3. Adding a DSN, step 3

In Figure 3, the `%DATE` shown in the Date Fields allows you to automatically convert a Clarion Date field which is defined as a `LONG` data type to an ODBC date data type. The same applies to `TIME` fields.

Understanding DTS

A DTS package contains all the rules required to automate the process of transforming data from one storage format to another. Note that I didn't say "from one SQL backend to another." You can use DTS to transform data from an Excel spreadsheet to SQL Server or another backend e.g. Oracle, Sybase, DB2, etc, or, to a flat file system like TopSpeed, dBase, or an ASCII file, and vice versa.

In order to transform data, you need a data connection to the source where data is coming from as well as the destination where the data is to be converted. Default connections are available for Microsoft Access, Excel, Paradox, Oracle (using an ODBC driver for Oracle installed with SQL Server), Text Files, and HTML. Also available is connection to any ODBC Driver as well as the OLE DB Provider for SQL Server.

DTS works in the form of a process-flow, and you create this visually. To build this process-flow diagram, you drop tasks into the designer and connect them together, as shown in Figure 4.

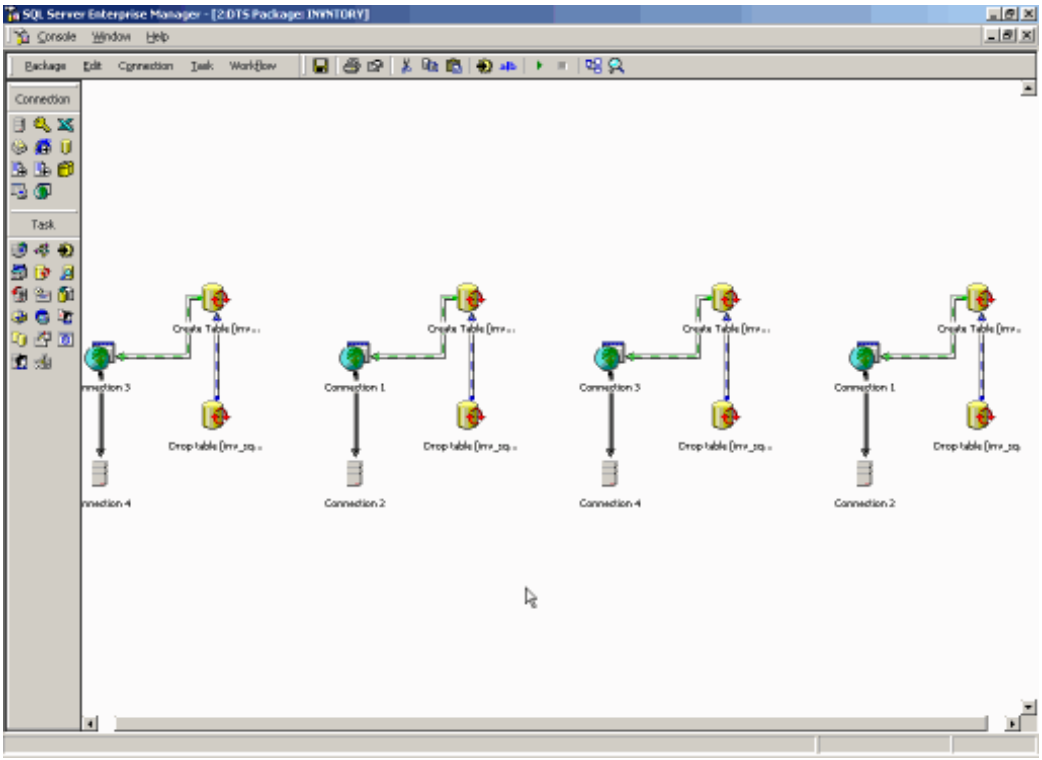

Figure 4. The DTS Designer

Different tasks can be linked together as a chain of events to be executed one after the other. Some of these tasks are:

- Connecting to an FTP site and transferring files
- Automatically sending an email once a task has been completed
- Running a Microsoft Message Queue Task
- Calling a stored procedure,
- Executing an SQL task,
- Copying a database

A very useful white paper on DTS is available at the MDSN web site. Another site dedicated solely to DTS is: http://www.sqldts.com/

Creating a package

Within the Microsoft SQL Server Program menu, there is a sub-menu – "Import and Export Data". This brings up an easy-to-understand wizard that takes you through the process of setting up the package.

In setting up the package, for the Data Source select "TopSpeed Developer Version," and the DSN you just created (see Figure 5).

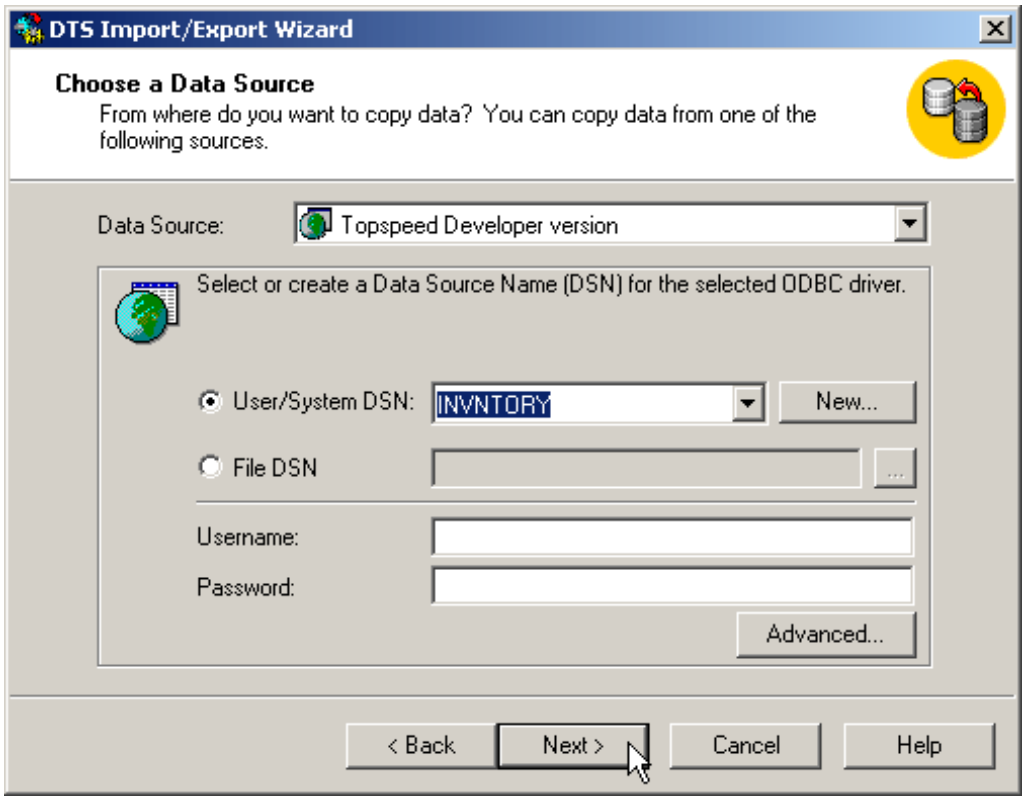

Figure 5. Choosing a Data Source

After selecting a Data Source Name (DSN), proceed by clicking the Next button. You do not need to fill the other fields like Username/Password since you didn't fill these while creating the data source.

In configuring the destination, select the "Microsoft OLE DB Provider for SQL Server". Remember not to choose an ODBC connection as this requires creating a DSN entry for SQL Server, which is unnecessary since the OLE DB driver is adequate, and more efficient.

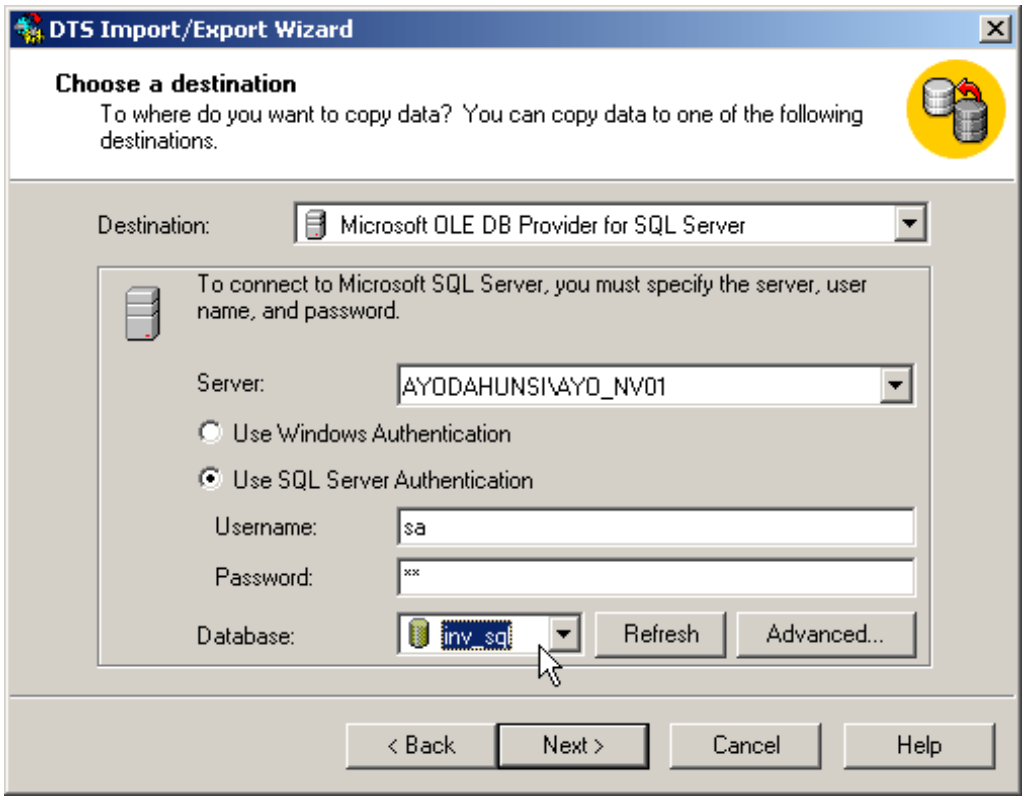

Figure 6. Choosing a Destination

In choosing a destination, you have to fill in the Username and Password fields. The database to be selected is the one I described in Part 2 of the article Migrating the Inventory Example to SQL Server.

The next form on the wizard (see Figure 7) shows how to do a straight copy by selecting the default radio button (Copy table(s) and view(s) from the source database), or by writing a SELECT statement to retrieve a specific record set to be transformed (Use a query to specify the data to transfer).

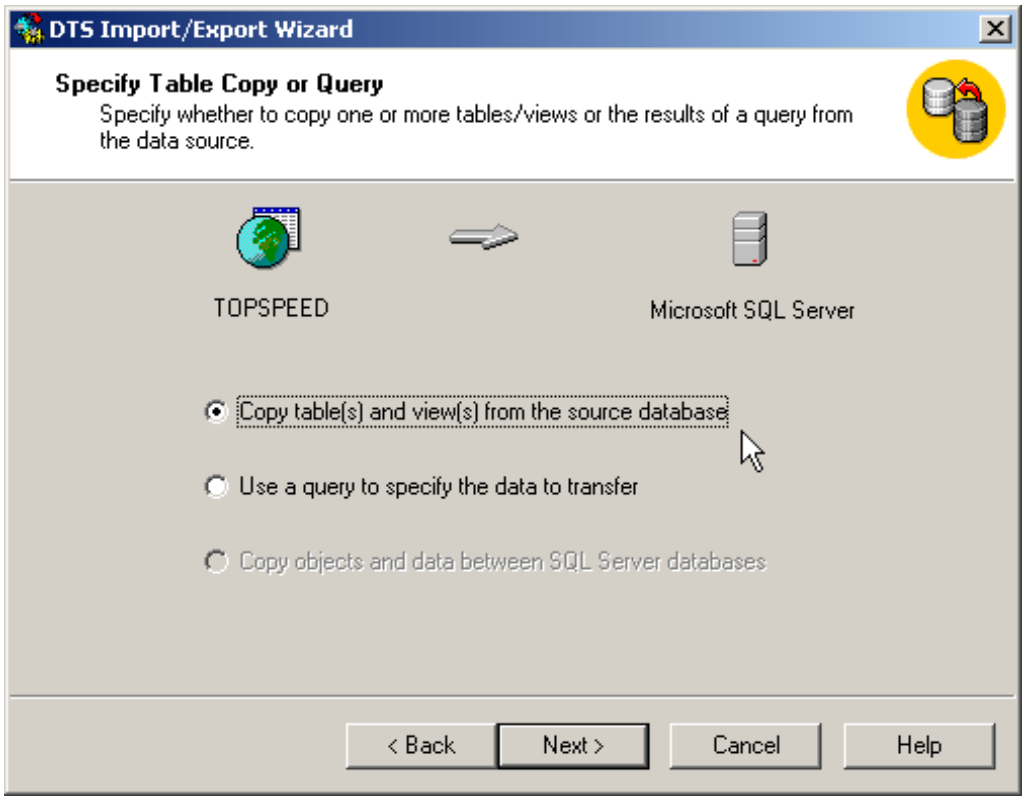

Figure 7. Specifying Data selection method

The next step is to map fields in the TopSpeed files to tables in SQL Server, as in Figure 8.

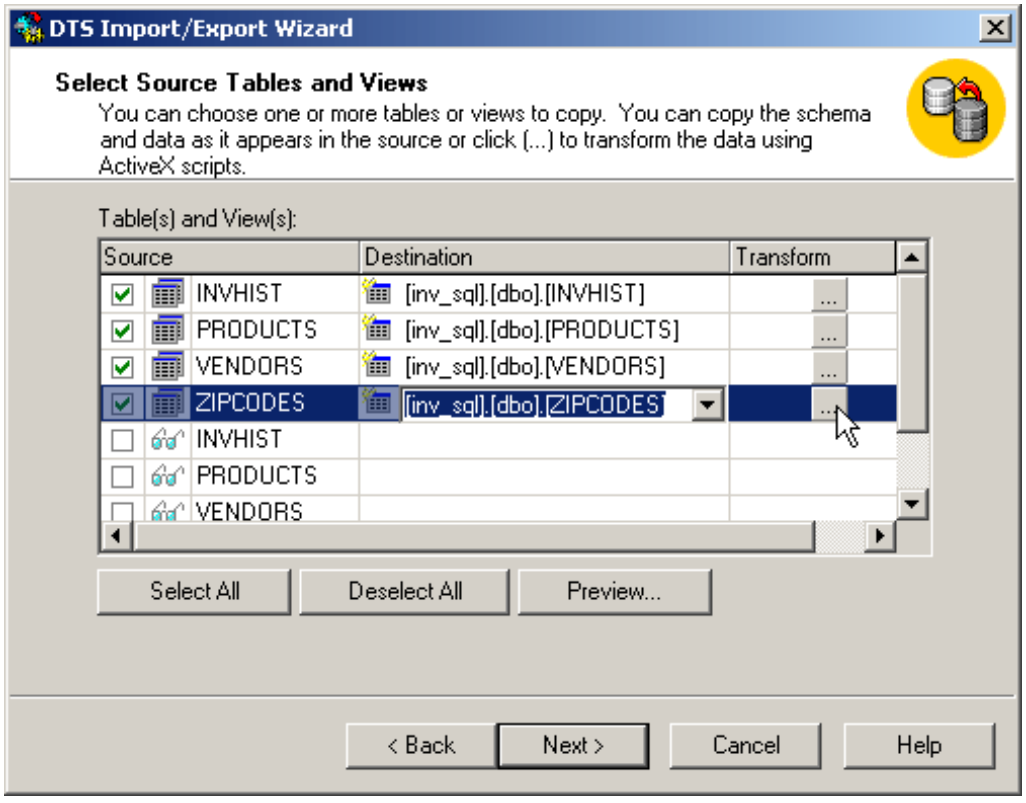

Figure 8. Mapping tables

When you click on the ellipsis button in the Transform column as shown in Figure 8, you can specify how the destination table is used. This means the destination table(s) can be deleted (DROP in SQL terms) and recreated before fields (called *columns* in SQL) and the records (called *rows* in SQL) are updated sequentially with data from the TopSpeed files (see Figure 9).

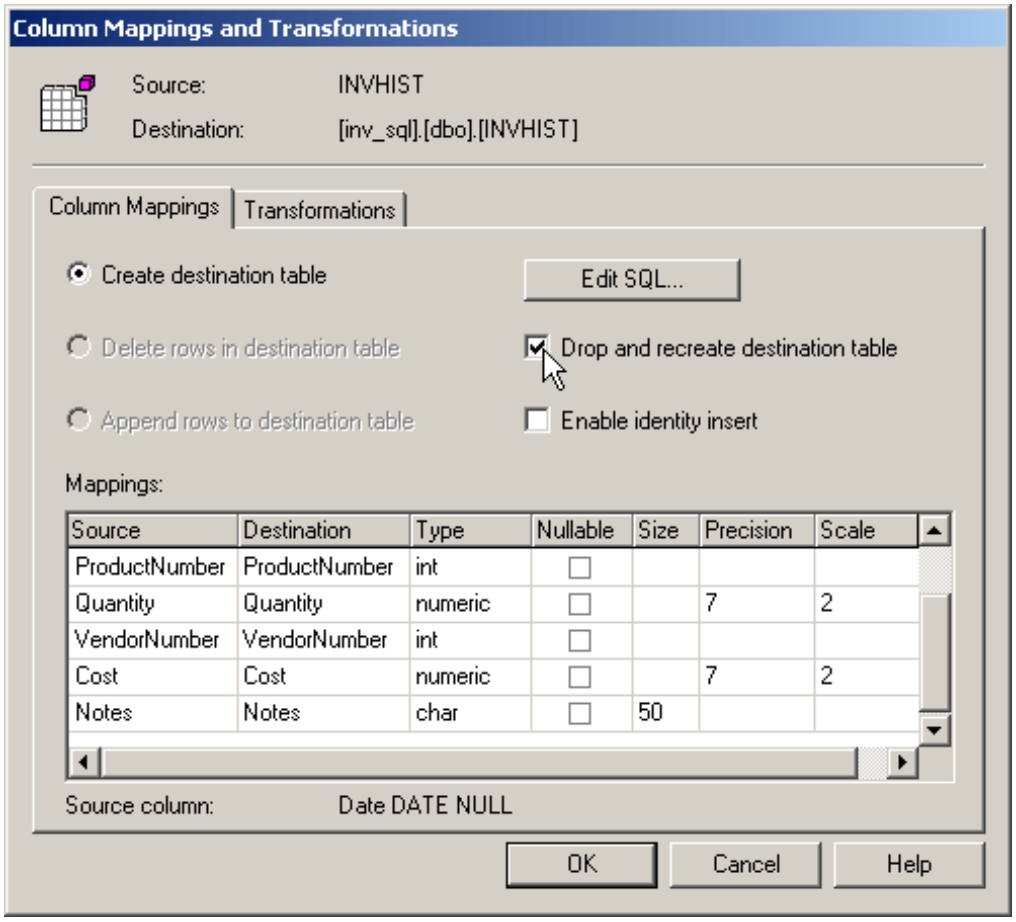

Figure 9. Pre-update Actions

There are different ways a DTS package can be saved. The most ideal way is to save it in SQL Server, so you can run it at any time via a stored procedure, as in Figure 10.

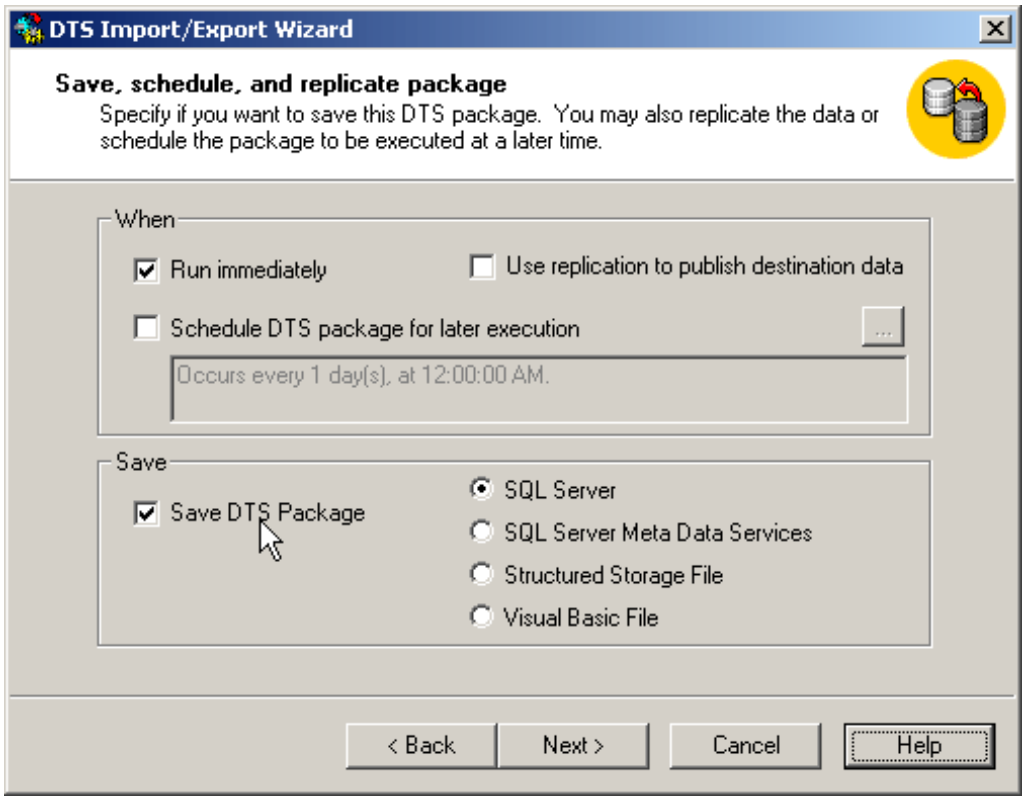

Figure 10. Save and Run

Click on Next to start the transformation process. If all goes smoothly, then a screen similar to Figure 11 appears.

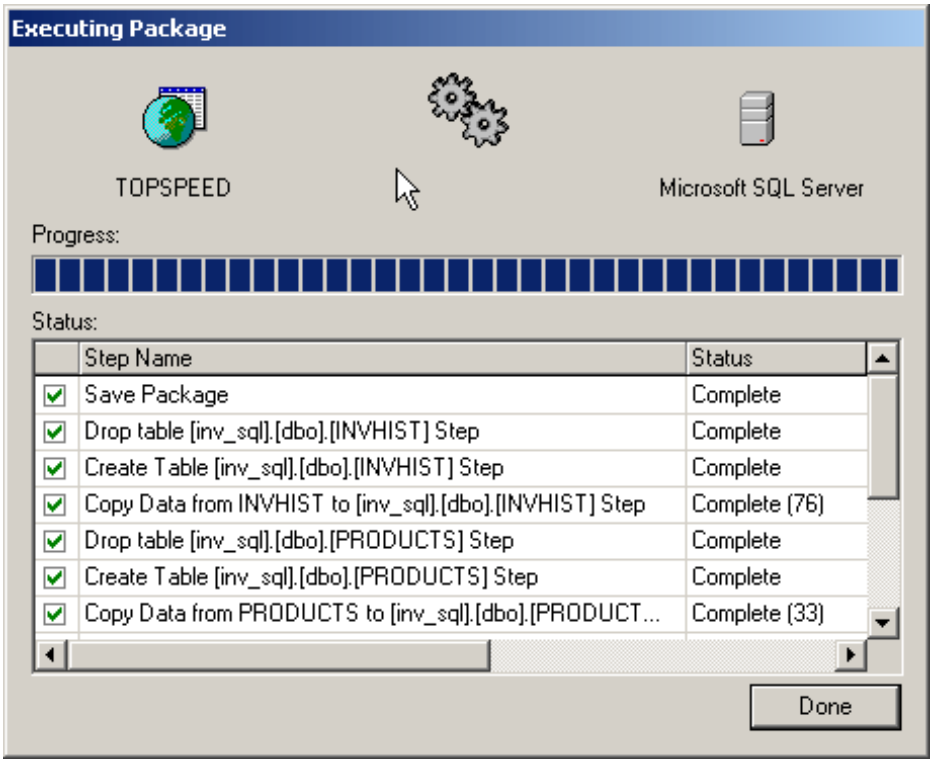

Figure 11. DTS Completed

We know all does not go smoothly most of the time. When an error happens you can

modify the transformation script to suit your environment. In the example the tasks executed are in this sequence:

1. Save Package (The package is saved in your SQL database) - This is always executed except when disk space is insufficient, or if you do not have the required database permission to save DTS packages.
2. Drop Table [*Table Name*]
3. Create Table [*Table Name*] – This task might not execute if you do not have the permission to CREATE tables, or if you have run out of disk space.
4. Copy Data

Note that tasks in 2, 3, and 4 are repetitive for all tables.

Whenever the execution of a task is unsuccessful, you will see a red "x" instead of the green check mark indicated in the first column. The severity of the failure is dependent on the kind of task being performed. For example, if (2) is unsuccessful but (3) and (4) succeed, this could be due to the fact that you checked "*Drop and recreate destination table*" as indicated in Figure 9 when the table in non-existent in the database. It could also be that you do not have the database permission to DROP tables in which case, you could end up with duplicated data if you really wanted to start with a blank table before your data is transformed. This applies to (3) as well.

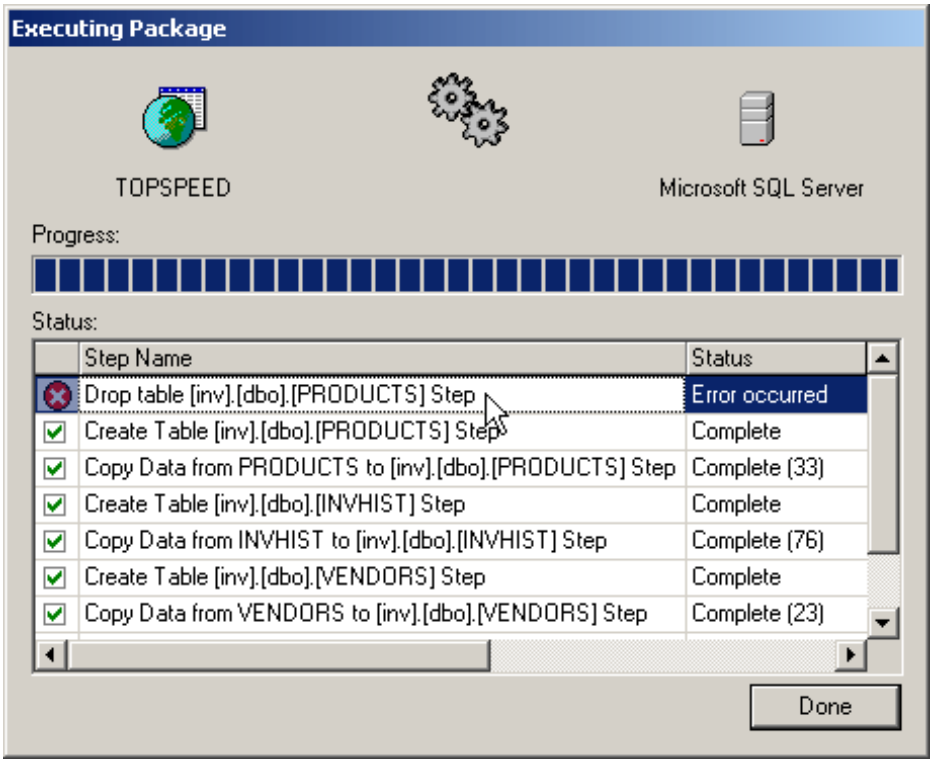

Figure 12. DROP Error in DTS Package

On the other hand, if (4) does not succeed, then no transformation has been done; this can happen with data type conversions. See the next section for more on how to

resolve this. Another notable cause of failure can come from a column with a unique index being populated with a duplicate values.

Whenever the status of a task indicates an error, double-clicking on the task will display the reason for the error.

A VB Script example

When you are transforming data from, for example, a Btrieve file to a SQL Server table, you are likely to run into Date field conversion problems, in which case DTS will not transform the data. When this happens, you have to modify the Transformation script as shown in Figure 13. (Note that this figure is similar to Figure 9; you get to it by clicking the "Transformations" tab.). Your language of choice for editing can either be VB Script, or JavaScript.

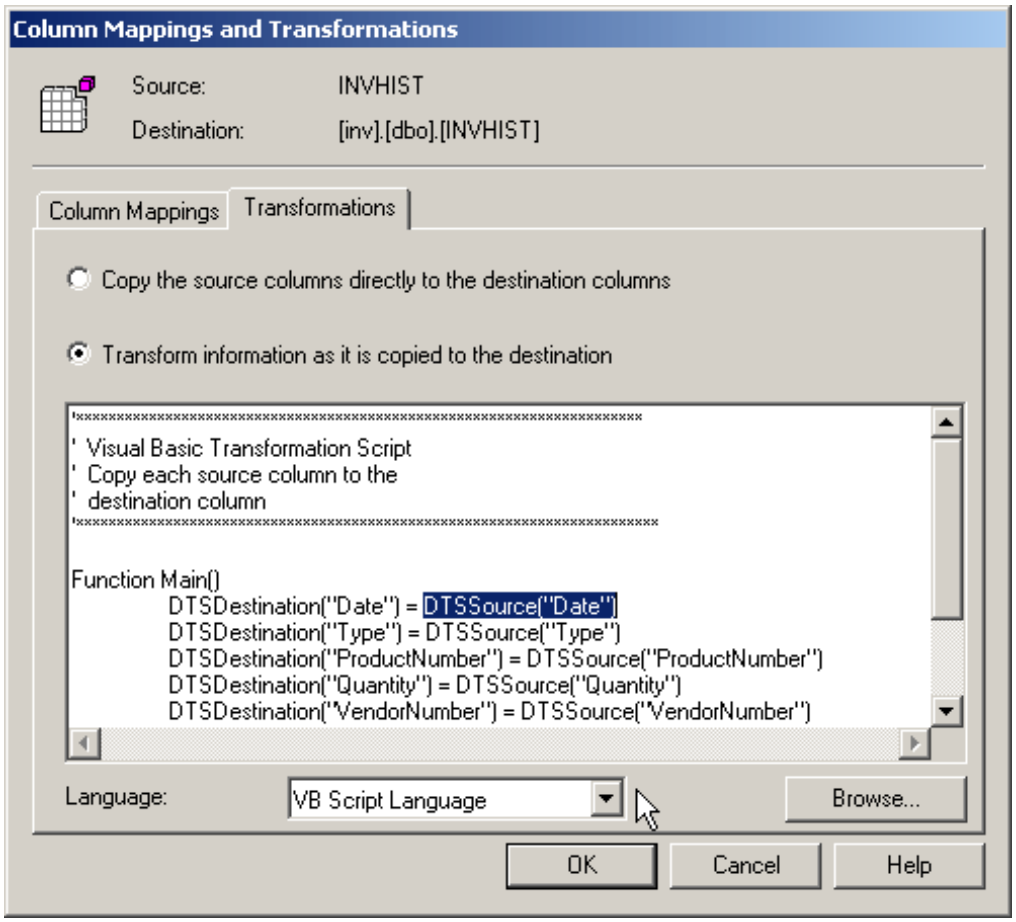

Figure 13. Modifying column mappings, step 1

If you are converting a file with a structure where date and time information is stored in two separate fields, it makes sense to merge the fields together and update the corresponding SQL Server `DateTime` column.

For example, assume there is another time field called `TIME` in the `INVHIST` file; the line containing the selected text as shown in Figure 13 is:

```
DTSDestination("Date") = DTSSource("Date")
```

Modifying the script, you will now have:

```
DTSDestination("Date") = CStr(DTSSource("Date")) +" "
  +CStr(FormatDateTime(DTSSource("Time"),vbShortTime))
```

I've used some VB Script functions (in bold) in order to achieve a SQL `DateTime` field format picture. `VbShortTime` is a Visual Basic constant which allows you to display time using the 24-hour format (HH:MM).

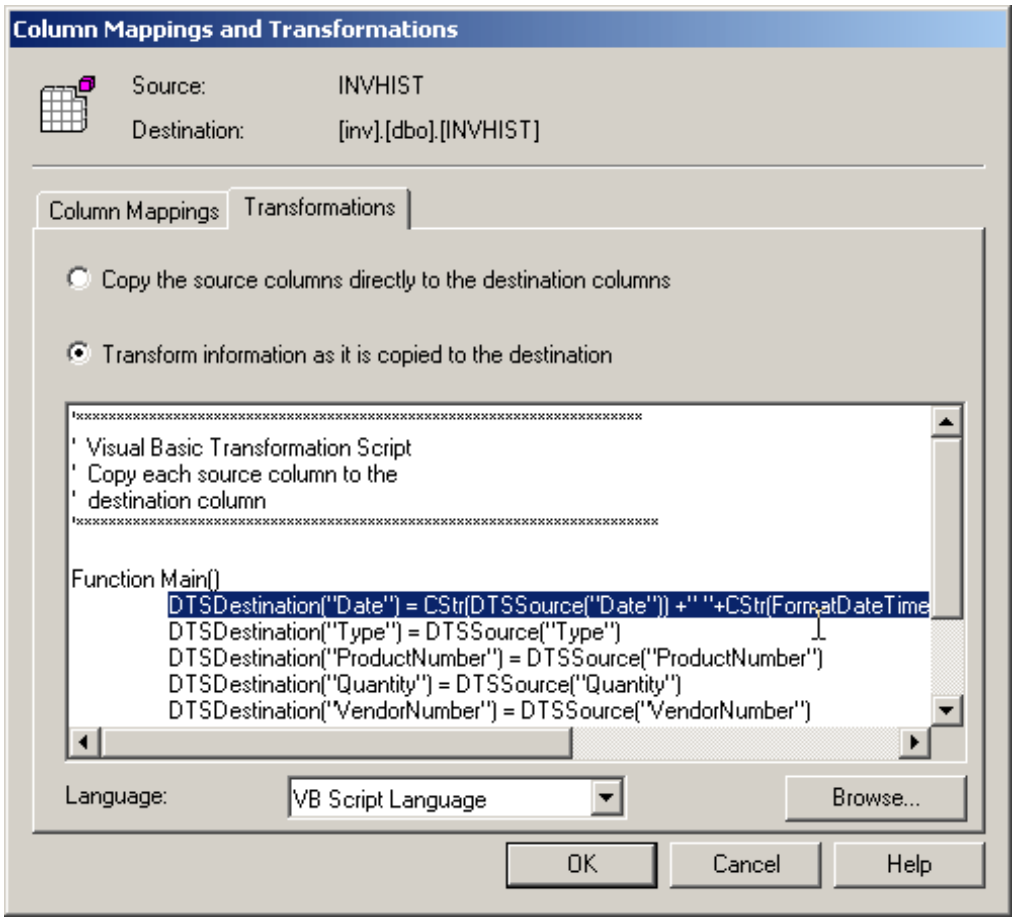

Figure 14. Modifying column mappings step 2

If you are thinking of doing a lot of your work in DTS, it is a good idea to start getting familiar with VB Script or JavaScript. You can download the VB Script HTML Help file from the Microsoft Script Technologies website.

Summary

Converting the Inventory application's TPS data to a SQL database is quite simple. Nevertheless, for practical purposes, say deploying an upgrade to an existing site currently running on TopSpeed files might require some level of automation. For Visual Basic (VB) applications, a DTS can be saved as a Visual Basic Script (VB

Script). This can be compiled as part of Visual Basic. As usual, the Clarion Language does not enjoy this luxury, so there is the need to provide a Clarion Application with an automation feature that can also be integrated into the upgrade. This is possible, and I will address it in upcoming articles.

Ayo Ogundahunsi presently lives in Henderson, Nevada, about ten minutes from Las Vegas. He works for Impac Medical Systems Inc., the leading company in cancer therapy software (written in Clarion). Impac has its headquarters in Mountain View, California. Ayo is married to Ayodola, and they have two boys, Darren and Joshua.

Reader Comments

Add a comment

The Clarion Advisor: Sizing Windows

by Andrew Guidroz II

Published 2001-11-28

Clarion is a great tool for writing custom apps. Every once in a while, I have a requirement from a customer for a window that is as unique as the individual's desktop.

Typically, such a customer wants a procedure to show every piece of information that will fit. The size and position of this window can vary depending on the default Windows font on that machine, whether or not the user hides the taskbar, how many other applications/windows need to be visible at the same time, and many other factors.

Every app that I deploy contains a procedure I use to get customer feedback on what size to make such a custom window. I run the application with the customer present and then resize and position the window exactly where it is needed, as shown in Figure 1. I jot down the stats that I see so I know what the window size and position needs to be.

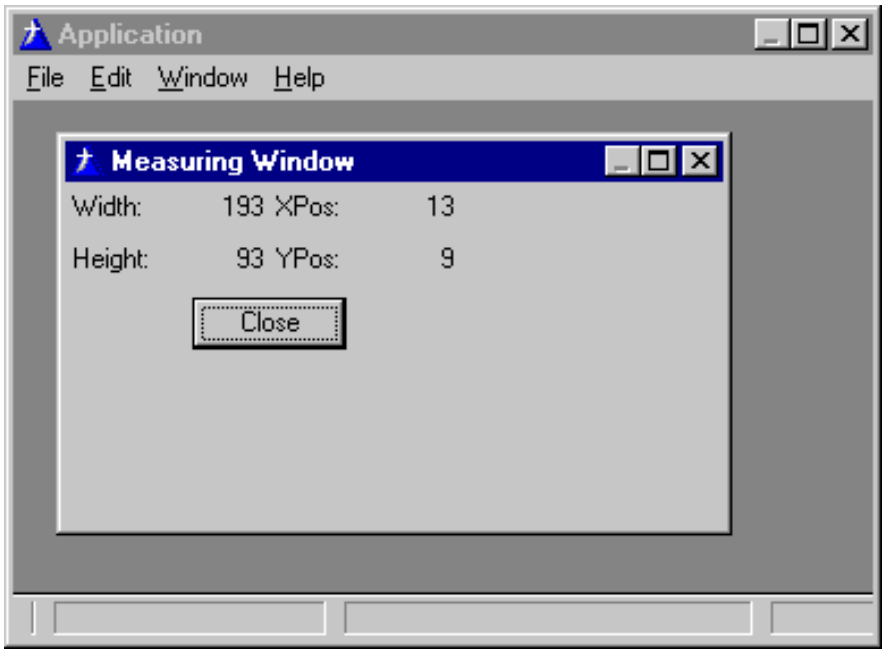

Figure 1. The resizable window

The code for this procedure is quite simple. I create local variables to hold the x and y position, length and width, and then in the window's TakeEvent method I place the following code:

```
IF EVENT() = EVENT:Sized OR EVENT() = EVENT:Moved
   Loc:WindowWidth =  ThisWindow{PROP:Width}
   Loc:WindowHeight = ThisWindow{PROP:Height}
   Loc:WindowXPos =   ThisWindow{PROP:XPos}
   Loc:WindowYPos =   ThisWindow{PROP:YPos}
   DISPLAY
END
```

Make it better!

For highly customized applications, this simple procedure can be a real timesaver. But it's only the beginning, and I'm sure you can think of ways to improve this procedure. Post your suggestions as reader comments, or email them to editor@clarionmag.com.

Download the source

Andrew Guidroz II, when he isn't traveling around the countryside watching his 2001 SEC

Champion LSU Fighting Tigers, writes software for all facets of the insurance industry. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the U.S. Andrew's Cajun website is www.coonass.com.

Reader Comments

Add a comment

Clarion Magazine

Clarion News

Published 2001-11-21

Alison Neal Profiled In INN Bio

Alison Neal, a Kiwi Clarion developer and regular on the SoftVelocity newsgroups, is the subject of this week's INN bio.

Posted Wednesday, November 28, 2001

VCRFlash 2.1 Released

Beta 2.1 of VCRFlash is now available. New features include: option to automatically save changes when scrolling using the form, or to prompt the user if they want to save changes; VCR button hot Keys can be specified in template; compatible with SearchFlash; RTF controls now updated. New demo available.

Posted Tuesday, November 27, 2001

The Sylkie Web Site

Richard Rogers has moved his web site to dbWired and now has, among other things, 26 new image sets for splash screens and about boxes, for a total of 34 sets available. These matched image sets are designed to fit perfectly into the AppInit template set, but may be used by anyone. You may not, however, upload them to other sites, or charge a fee for them. The whole website has been given a facelift and is now open and ready.

Posted Monday, November 26, 2001

SealSoft Releases xNotes Class v1.0

SealSoft's xNotes adds notes and reminders features to your applications. Features include: easy template implementation; save notes to file; load notes from file; print notes; customize fonts and colors; localization with TRN file. xNotes is $49. Demo available.

Posted Monday, November 26, 2001

Clarion Photo Gallery Update

Mike McLoughlin's Clarion Photo Gallery has three purposes: 1) To put a face to the people we chat to in newsgroups and on discussion boards; 2) To help strengthen the sense of community among Clarion developers; 3) To provide a place for developers to indicate they are looking for work.

Posted Thursday, November 22, 2001

Gitano Software Back From Vacation

Jesus Moreno is back in the office and has answered all waiting mail. If you did not received an answer to your mail, please re-send.

Posted Thursday, November 22, 2001

Updates from Gitano Software Now Available

The following Gitano utilities have been updated and are ready to be downloaded. Please use the same unlock and password information from your original registration: gREG - Build compatible. New _run code_ feature after registration; gSEC - F Build compatible. Fixed _BRW0_ bug; gBUDDY - F Build compatible; gCAL - F Build compatible, and fixed compatibility with gCalPro. You can now provide a full path for the data file; gCALC - F Build compatible; gNOTES - F Build compatible; gSREG - Compatible with all utilities; C55DLLS - F Build DLLs available.

Posted Thursday, November 22, 2001

SysIP Released

SysIP is a new product from solid.software. This is another wrapper class for the win32 API common controls. SysIP is a wrapper for the so-called IP address control, which you are most likely to know from the network control panel. There are some advantages over using a standard entry control when using specialized IP address controls:You're able to limit each of the four fields to a certain address or address range (i.e. subnet masks); Users will be able to jump from field to field by pressing ctrl+right or ctrl+left; Ability to use standard Windows "look and feel" inside your applications. You can use SysIP without writing a single line of code. SysIP also provides embed points for event handling, just like the standard Clarion templates. The SysIP runtime library also contains helper functions to convert IP addresses between the different formats: LONG, 4 BYTES and STRING. SysIP works with Clarion 5 and Clarion 5.5, ABC and legacy templates are supported, 32bit only. Comes as a .LIB and a .DLL version, supporting standalone and local runtime libraries.

Also ships with a documentation in html help format; email support and updates are free. SysIP is now also included in SysPack, together with SysAni, SysHotKey, SysTrack, SysList and SysProgress.
Posted Thursday, November 22, 2001

ExpressFlash 2 Supports Outlook 2000

Version 2.0 of ExpressFlash has just been released and now includes compatibility with Outlook 2000 as well as Outlook Express. Use ExpressFlash to link your Clarion apps to incoming email. Updated demo available.
Posted Thursday, November 22, 2001

INN Bio Features Richard Rogers

This week's featured developer bio at INN is Richard Rogers. He's been a logger, a soda jerk, a bosun's mate, and now... He's one of the Clarion crazies.
Posted Wednesday, November 21, 2001

Icetips Wizards Now Compatible With C4/C5

Icetips Software has released an updated version of Icetips Wizards, Standard Edition. There aren't really any bug fixes in this release because no bugs have been reported since the last build of the initial release. The major new feature is that the wizards now work with both Clarion4 and Clarion5. Because of the way wizards were set up in Clarion4 and Clarion5, one line in the browse, update and report procedure templates needs to be modified to make the browse, form and report wizards 100% compatible with C4 and C5. Icetips Software will do this editing, free of charge, for those who don't feel up to it. Icetips Wizards Professional Edition will be available for distribution by late December. It will include support for various third party tools, direct integration with the Icetips Reporter and a lot of new features.
Posted Monday, November 19, 2001

EasyExcel Version 1.01.1

Ingasoftplus has released a patch for the EasyExcel Class Libraries and Code Templates. This patch includes a fix to large cell ranges, and requires that you have installed EasyExcel 1.01. Registered users can download a password protected setup for this patch. After installation you must recompile all applications which use EasyExcel.
Posted Monday, November 19, 2001

ABCFree Templates And Tools Updated

The freeware ABCFree Templates and Tools have been updated. Version 2.34 changes include: new template to add generic CTRL+F (Find) support to all browses in a procedure or application; drag and drop re-ordering support to move up/down in a browse; dropped support for Catalyst Sockettools.

Posted Monday, November 19, 2001

SealSoft Releases xSearch Class

SealSoft has released xSearch, a class and control template for searching through all string fields of main and related files. Many-to-one and one-to-many relations are supported. xSearch also includes tagging features and a report extension to allow printing of tagged records. Demo available. QBE is scheduled for release 1.1.

Posted Monday, November 19, 2001

Clarion 5.5 SR7 Alpha Addresses XP Issues

Microsoft Europe has confirmed unexpected behavior under XP for programs running correctly under previous Window versions. It is confirmed that the most likely cause for the loss of backward compatibility are changes in the XP system loader. C55F had been assumed as our last C55 build, but to provide a solution for the XP compatibility problem SoftVelocity has prepared an update. This build has passed initial testing of the reported problems, but is being released in alpha format while testing continues.

Posted Monday, November 19, 2001

ProDomus Bundle Special Ends November 15th

ProDomus is offering s 30% discount on selected packages through November 15th. The PD International Pack (savings of $344) includes: PD Translator Plus Enterprise Edition ($899); PD 1-Touch Date, Time, and Scheduling Tools less Appointment Class ($150); PD Worldwide Address Formats ($99). Total before discount is $1148, after discount price is $804. A second special package offers savings on ProDomus' two most popular tools, PD Lookups and PD Date Tools. The fully internationalized Date Tools automatically populates a calendar button for every date entry, adds scrolling to both spin and entry controls, and incorporates range limits defined in the dictionary. It provides many functions and templates for handling dates, holidays, and time. Calendars

automatically translate month and day names and calculate week of the year according to the user's locale or any specified locale supported by Windows. PD Lookup Date Pack includes: PD Better Lookups ($99) and PD 1-Touch Date, Time, and Scheduling Tools less Appointment Class ($150). Regular price is $249, discounted price is $174.

Posted Tuesday, November 13, 2001

Prodomus Free Template Updated

An updated version of the PD Class Removal Template for C55 is now available.

Posted Tuesday, November 13, 2001

Beta Testers Wanted

Beta testers wanted for Clarion shareware calendar/diary system. Active beta testers will be rewarded.

Posted Friday, November 09, 2001

VCRFlash Beta 2 Released

Sterling Data's VCRFlash is now in Beta 2. VCRFlash places VCR buttons on your standard forms and allows scrolling through the file without returning to the browse. No changes needed to your existing update forms - just drop the VCR buttons straight on to them. New in Beta 2: insert button; search button; a form can be called from many browses; some small cosmetic changes. New demo available. The price will be increasing from $99 to $149 when the templates go gold at the end of this month.

Posted Thursday, November 08, 2001

Next Age Imaging Templates & Windows XP

Next Age has confirmed that Windows XP does not include the Imaging for Windows Application that has been included with every version of windows since Win 95B. This means the imaging OCX's need for the Imaging Templates are not included. According to the newest company to own the product (EiStream), you must purchase Imaging for Windows PRO in order to have the imaging function with Windows XP. For more information on this please see the EiStream web site.

Posted Thursday, November 08, 2001

Parker Profiled At INN

In the second of an ongoing series, the Icetips News Network is very

pleased to present an interview with Steve Parker. One of the more well-known guys in our Clarion world, he may have helped you solve a problem once or twice. Now you can see what he thinks of Clarion, business, and life.

Posted Thursday, November 08, 2001

Clarion Third Party Profile Exchange Updated

An update to the Clarion Third Party Profile Exchange is now available. Click on Profile Exchanges, then click on Clarion 3rd Party Online Profiles.

Posted Thursday, November 08, 2001

Nice Touch Solutions Adds ClarioNET Support

By popular demand, Nice Touch Solutions, Inc. is pleased to announce support for ClarioNET in it's line of third party products. Support for Query Wizard, Report Wizard and View Wizard is currently available. Query Wizard and Report Wizard offer the same full functionality as their desktop counterparts. View Wizard has a few limitations we hope to overcome as soon as possible. Support is currently provided for Clarion 5.5 applications using ClarioNET 1.1. ClarioNET support is offered as an add-on to your existing product license and is priced as follows: Query Wizard $79; Report Wizard $69; View Wizard $69.

Posted Tuesday, November 06, 2001

New G-Cal Build Available

A new build for G-Cal is now available. The gsDATEPLUS and gsBUSINESSPLUS functions now accept a new parameter: 1 = days, 2 = weeks, 3 = months, 4 = years.

Posted Tuesday, November 06, 2001

IceTips Report Wizard Renamed Icetips Reporter

As there is already a product called Report Wizard (from Nice Touch Solutions), Icetips Software has renamed its reporting wizard product to Icetips Reporter. Version 1.002 is now available.

Posted Tuesday, November 06, 2001

Gitano Software Closed Nov 7-14, 2001

Gitano Software will be closed from November 7-14, 2001. All support/sales/etc will be handled as soon as the office opens on the 15th.

Posted Tuesday, November 06, 2001

xDataBackup Manager v1.2

SealSoft has released xDataBackup Manager 1.2. This release includes automatic deleting of old archives. New demo and docs available now, new install will be available shortly.

Posted Tuesday, November 06, 2001

New SysList Demo

A new SysList demo is now available from solid.software. The demo now contains a procedure that shows how to display a list of files with their associated icons (just like Explorer). SysList is a wrapper class for the list view common control, available for Clarion 5 and Clarion 5.5 (ABC and Legacy, 32bit only). Features include: large icon, small icons, list and report view modes; clickable headers; drag-and-drop reordering of columns; grid-lines (in report view); flat scrollbars; label-editing; single or multiple item selection; checkboxes; callback procedures for event handling; hot-tracking; background image; draggable items, and more. Cost is $99 at ClarionShop, email support and updates included.

Posted Tuesday, November 06, 2001

Andy Ireland's COM Classes

Andy Ireland has posted a copy of his soon-to-be open source COM classes. This is code under development, copyright of Plugware Solutions.com Ltd and may only be used for your own applications. They may not be distributed or copied in any way nor can they be used in any third party product.

Posted Friday, November 02, 2001

Business Rules Manager Released

Riebens Business Rules Manager has been released. This product allows you to create and manage your business rules, and also create the Clarion code to implement the rules, for import into your application. Features include: import the Project dictionary into the business rules application for use during business rules definition; create multiple Clarion source code for each business rule defined; export business rule Clarion Source code; link the business rules source code into applications without having to write a single line of code; if business rules change, the rule change is automatically reflected next time the application is compiled; works for all versions of Clarion for Windows, 16 & 32 bit and both ABC & Legacy. Available from www.clarionshop.com.

Posted Thursday, November 01, 2001

TPS.repair Template Goes Gold

The TPS.repair Template gold release is now available at www.clarionshop.com for US$ 50. This template set generates one-click file repair and maintenance functionality for .TPS-files. Supports Clarion version 5.5; a 30 days free trial version is available. The release for Clarion versions 4 and 5 is delayed because of licensing issues regarding the redistribution of SoftVelocity's TPSFIX utility."

Posted Thursday, November 01, 2001

ProDomus Updates and Notes

Several ProDomus products have recently been updated. The Translator Plus language dictionary now contains over 23,000 phrases in 10 languages. Some languages include translations for popular third party tools such as CPCS reports and ToolCraft's Query Wizard. Many thanks to users who have contributed translation files. The Translation Assistant also now includes a dictionary synchronization process allowing the user to selectively override any conflicting translations found in the dictionary and translation files. The Source Extraction Utility has been modified to parse default cell tools tips added in C55f. It also allows you to exclude any specified attributes that you don't want added to a translation file such as help items or icons. New templates provide multi-language translation for CPCS Reports and template code to modify Query Wizard templates to do the same. These tools otherwise require separate compiles for each language - a common third party tool limitation. Another class revision allows users to edit and change translations at run time - you can see the change while in the application. ClarioNet Support has been added to C55 versions of PD Better Lookups, PD 1-Touch Date Tools, and PD Translator Plus. Translator Plus code is provided to change the client's international environment when doing international applications.

Posted Thursday, November 01, 2001

xQuickFilter v2.06 Released

SealSoft's xQuickFilter now supports ClarioNET, allowing you to add quick filters to your web applications. You can also now display in the status line the current level of enclosure of a filter.

Posted Thursday, November 01, 2001

Updated MySQL Templates

Roberto Artigas has posted an updated version of his DCT to MySQL templates, including a variation by Lee White.

Posted Thursday, November 01, 2001

Reader Comments

Add a comment