Clarion Magazine

PDF for December, 2002

All Clarion Magazine articles for December, 2002 in PDF format.

Posted Friday, January 03, 2003

Integrating The Clarion Report Writer Into Your Applications (Part 1)

The Clarion Report Writer is a great tool for creating standalone or ad-hoc reports, and is not only for developers, but for end users armed with a copy of the Clarion Report Writer and enough technical savvy about your database schema to use the tool as well. And as Ben Brady shows, it's easy to integrate Report Writer reports into your application.

Posted Monday, January 06, 2003

Calling A Clarion Application With PHP

PHP is a hugely popular scripting language used mainly for web development; Clarion is a Windows language used mostly for business software development. What do they have to do with each other? In this article, Ville Vahtera shows how to create a Clarion application which can be executed from, and have its output captured by, PHP.

Posted Monday, January 06, 2003

Weekly PDF For January 5-11, 2003

All ClarionMag articles for January 5-11, 2003 in PDF format.

Posted Tuesday, January 14, 2003

Integrating The Clarion Report Writer Into Your Applications (Part 2)

The Clarion Report Writer is a great tool for creating standalone or ad-hoc reports, and is not only for developers, but for end users armed with a copy of the Clarion Report Writer and enough technical savvy about your database schema to use the tool as well. And as Ben Brady shows, it's easy to integrate Report Writer reports into your application. Part 2 of 2.

Posted Thursday, January 16, 2003

Making The E-Mail Connection

Brice Schagane shows how to use John Hickey's Internet Link Template

News

- DCTBuilder 1.09
- PDF-Tools Now Supports JBIG Format
- ImageEx 2 Beta 4
- HTML Designer For Clarion Updated
- EasyVersion 2.00 Released
- Metabase Web Site Update
- C6EA Info On SV Web Site
- Virtual EIP holiday schedule
- PDF-Tools SDK Version 2.5 Released
- New Year Special Offer - 50% Off BackFlash
- Developer Graphics Special (Save $100)
- INN Bio & News For 14-Jan-2003
- xTransparent Window v1.3
- Sneak Preview Of cpTracker 2003
- C55 Threading Problems Analysis

to open an email message editor in the user's email program using a specified email address, subject, and body text.

Posted Friday, January 17, 2003

Data Structures and Algorithms Part XIV - A Queue Is A Queue Is A Queue?

You know about Clarion Queues. But what about the "traditional" queue, as known to C++ programmers? Alison Neal explains.

Posted Friday, January 17, 2003

Icons *In* List Box

Including a balance column or an aging column isn't quite obvious enough for the typical user, even if a contrasting color is set. Some other kind of visual indicator is necessary, and some iconic eye candy is just what Dr. Parker prescribes.

Posted Friday, January 17, 2003

Weekly PDF For January 12-18, 2003

All ClarionMag articles for January 12-18, 2003 in PDF format.

Posted Friday, January 17, 2003

Debug De Program With Debuger

Looking for another way to debug your programs? Skip Williams' Debuger class (yes, that's how it's spelled) logs system messages, detects duplicate messages, and will even conditionally invoke the Clarion debugger.

Posted Thursday, January 23, 2003

A Calculator Class And Template

Nardus Swanevelder's search for a free XP-like Clarion calculator was fruitless, but he found some free Clarion 4 source code. Nardus shows he he converted that source into a class and templates.

Posted Friday, January 24, 2003

Book Review: Programming PHP

PHP, an open source scripting language, has become hugely popular for web site development. Clarion developers are noticing, and so is SoftVelocity, as evidenced by their announcement of the forthcoming Clarion/PHP product. If you're hunting for more information on PHP, this book from O'Reilly is a great place to start.

Posted Friday, January 24, 2003

Data Structures and Algorithms Part XV - Priority Q

In this second of two parts, Alison Neal continues her examination of traditional (i.e. not Clarion) Queues. This week's subject: the Priority

New Web Site For DOS Printer

CWODBC Version 1.9

DictionaryBuilder 1.1

TDAN Issue 23.0

OutlookFUSE 1.1 released

PlugIT Memory Leak Checking for Clarion 5/5.5 ABC And Legacy Templates

Fomin Report Builder v.2.85

Clarion 6 EA Has Shipped!!

Sterling Data Templates And Clarion 6

BoxSoft Vacation Schedule

ZipApp 1.1a Released

SealSoft xFText v2.1

RADVenture Referential Integrity Wizard

MyODBC/MySQL Updates

INN Bio For 24-Dec-2002

Read/Write Excel Files

XML Support

Firebird 1.0.2 Update Released

OutlookFUSE 1.1.0 Demo

New TimeSavers Scheduler Beta And Picture Gallery

Search the news archive

Queue.

Posted Thursday, January 30, 2003

Getting Dynamic With Report Writer

This action packed episode picks up where Ben Brady ("Integrating Clarion Report Writer Into Your Applications") leaves off. Ben showed how to integrate the Report Writer engine into standard Clarion apps. In this article, Dr. Parker shows how to add more flexibility to the process of using Report Writer from an app.

Posted Thursday, January 30, 2003

Book Review: SQL In A Nutshell

O'Reilly's SQL In A Nutshell is a desktop reference for SQL as implemented in SQL Server, MySQL, Oracle and PostgreSQL. This is a slightly older work, first published in December 2000, but it still has some useful information. Reviewed by Dave Harms.

Posted Friday, January 31, 2003

Looking for more? Check out the **site index**, or **search the back issues**. This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

Integrating The Clarion Report Writer Into Your Applications (Part 1)

by Ben E Brady

Published 2003-01-06

The Clarion Report Writer (CRW) is a great tool for creating standalone or ad-hoc reports, and is not only for developers, but for end users armed with a copy of the Clarion Report Writer and enough technical savvy about your database schema to use the tool as well.

This article will show you how to integrate CRW and the CRW ReportEngine class (which lets you control the report writer by method calls) into your application, within the Clarion IDE and Application Generator, using embedded code. It will provide a framework in which to add an unlimited number of reports to your application, thereby allowing you or your end users to expand upon the functionality without the need for re-compiling each time you wish to provide new reporting capability.

A little history

The Clarion Report Writer, in one incarnation or another, has been bundled with the Clarion development environment since 1991. Back in those early days I got very familiar with CRW in my DOS-based projects, and was able to create some reports that would have been a bit more than ‘hairy’ if I’d had to create them as Clarion report procedures, without writing any code. It became one of the staple tools in my development arsenal.

In my opinion, CRW was so powerful that I decided to create a third party Clarion product called CLA2RW, which would allow developers to incorporate CRW reports into their CPD 2.1 Designer-based applications. I also created a version that would work with LPM from Logix Development; both versions were very well received by the CPD 2.1 development community.

As the "Reporter", as it is known, progressed through the CDD 3.0 and CFD 3.1 releases, I continued to use it as my secret weapon in extracting information from legacy database systems (whether written in Clarion or not) due to the Reporter's ability to read multiple data formats, and create file relationships at run time which were not based upon hard-coded relationships. I found this especially useful when extracting data in order to perform data migration - I could easily create an ASCII output file using the Report Writer and, if necessary, could use the output of one report to create additional reports that are not possible using just the files defined in a particular database schema.

During my initial development efforts moving from DOS to Windows, I tried using an early version of CRW for Windows bundled with Clarion for Windows 2.0. I found the product too cumbersome to use and prone to errors which usually resulted in GPF messages. As a result, I continued to rely upon CRW 3.101, the last DOS-based version of the tool that would read TPS files and legacy Clarion .DAT files.

Recently, I embarked upon a project that required the migration of an existing legacy DOS application – that I had authored circa 1992 – to the Windows platform. This project had used CRW almost exclusively to produce reports, and the rewrite gave me the opportunity to learn something interesting about CRW in C5.5.

Up to this point I had largely ignored CRW and chose to code my Clarion Windows application reports as report procedures, fighting with the Report Formatter in the Clarion IDE. This particular project had so many reports done in CRW from CPD 2.1 that I decided to take another look at the Windows version of CRW. I am glad that I gave CRW a second chance.

Jumping in with both feet

I fired up the Clarion Report Writer to try it out and was pleasantly surprised by the ease of formatting, the creation of user defined file relationships, the ability to import all of the information in my Data Dictionary, and many other features that had obviously improved over the years.

After I had created several reports I wanted to integrate them into my application in a way that would be easily understood by the end user. Enter the Clarion `ReportEngine` class, which you can find declared in the libsrc directory in RWPRLIB.INC. The `ReportEngine` class itself is available only in binary form – you either create an instance of it, or of a derived class, in order to use it.

The first order of business was to ensure that my application that would be easily understood by the end user. After some thought I came up with the following file structure:

```
REPORTER             FILE,DRIVER('TOPSPEED'),|
```

```
        NAME(GLO:Reporter_),PRE(CRW),BINDABLE,CREATE,THREAD
KeyReportName           KEY(CRW:ReportName),NOCASE,OPT
Record                  RECORD,PRE()
ReportName                 CSTRING(41)
Description                CSTRING(129)
ReportLibrary              CSTRING(256)
ReportLibraryPassword      CSTRING(65)
Device                     CSTRING(65)
Copies                     BYTE
NumberOfCopies             BYTE
Preview                    BYTE
NumberOfPages              BYTE
DateRange                  BYTE
Password                   BYTE
                        END
                    END
```

This file format provided the necessary storage for information required to interface to CRW. Most of these fields are obvious in their usage, however there are a couple that require some additional explanation. The Report Library can have a password to protect against modification, and the report itself can have a password to control access. The `ReportEngine` class methods can accept the Report Library password to access the library, however if the report is password protected the `ReportEngine` class will prompt the user for the password at the time the report is requested.

Using this file definition I then created a browse/form pair of procedures using the Browse Procedure Wizard, and called the browse from my application's main menu. The list box format only contains the `ReportName` and `Description` fields, as shown in Figure 1.

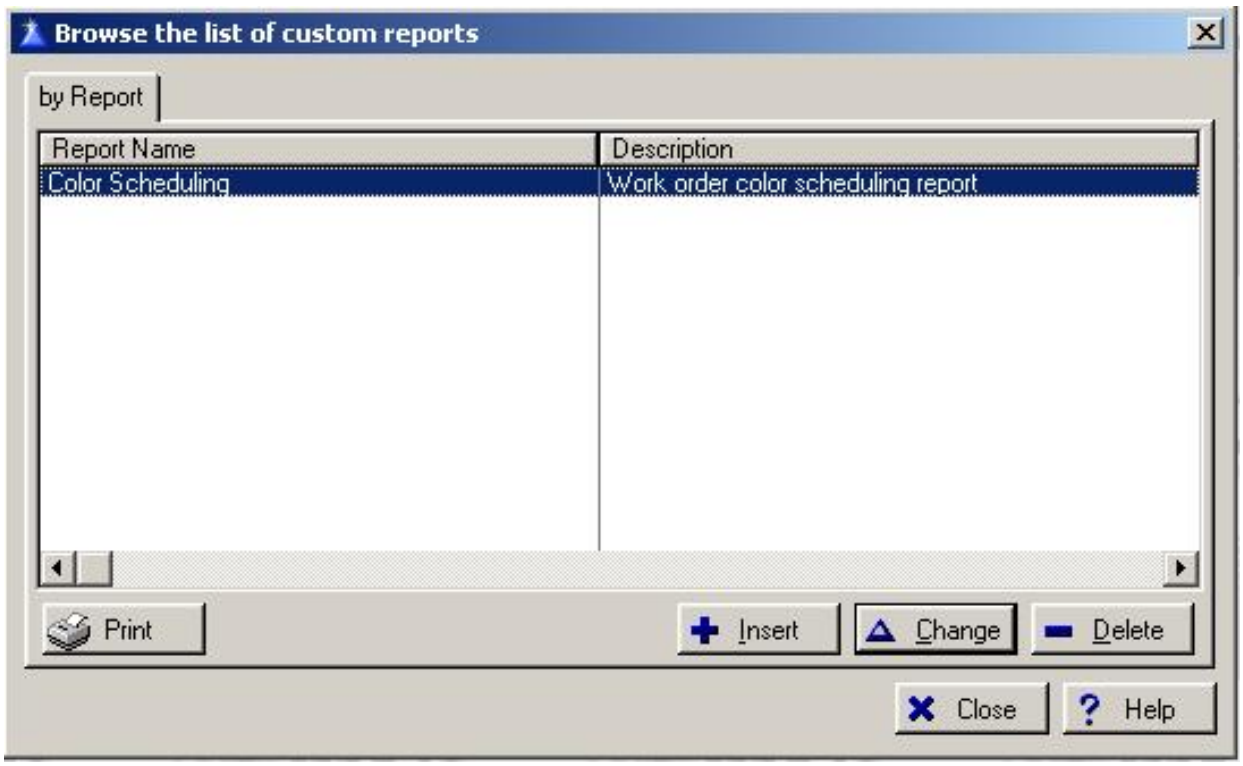


Figure 1. Reporter file browse

I added a **Print** button to the browse which would be used to execute the selected report.

The form contains the additional fields necessary to complete the record entry for storing the parameters necessary to interface to the `ReportEngine` (see Figure 2).

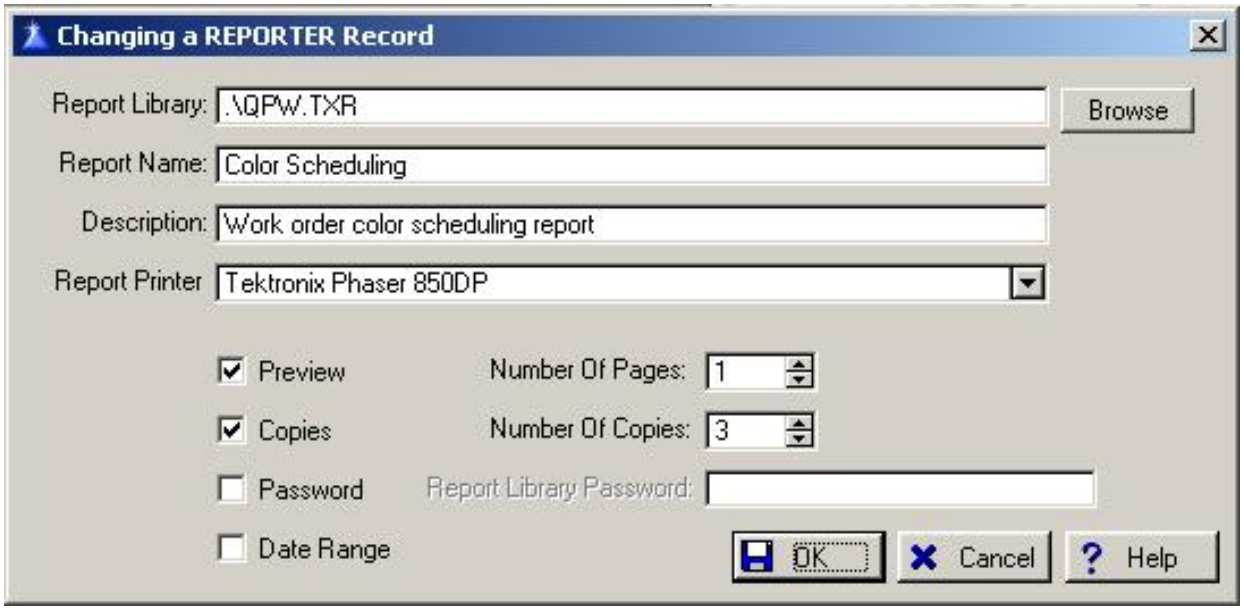


Figure 2. Reporter file entry form

I added a lookup button, using the DOS file lookup template, to browse the list of files in order to allow the user to select the location for the Report Library file used for the report definition. I also placed a drop down combo field (without the use of the control template) to provide the user with access to the list of printers, enumerated by a Windows API call, as installed on the computer where the application is being run. This helps to eliminate errors that would more than likely occur as a result of having the user type in the printer names. The printer names must be entered *exactly* in order for the ReportEngine to send output to the correct device.

OOP - A fly in the ointment

Everything was going according to plan as I reviewed the Report Writer documentation (see 55RW.PDF in your Clarion installation's doc subdirectory). I found there are actually two ways you can implement the integration of the Clarion Report Writer into your application. One is to call the C55PRNTX.EXE (32 bit) or C55PRINT.EXE (16 bit) file with a set of parameters, either on the command line or in a special .INI file. Initially, I decided to try the .INI file method, but later opted to use the direct `ReportEngine` method calling interface instead, for reasons which will become clearer later on. A deciding factor was that INI files are not very network friendly.

Unfortunately, I found the examples in the documentation to be riddled with errors and

omissions. The documentation tells you that you can accomplish this type of integration, however the examples leave a bit to be desired.

Looking at the examples provided with Clarion for the Report Writer, I was able to create a small test application in order to test the Report Engine methods. They seemed to work, however there was one small issue that was not addressed in the example files. I use variable filenames in my data dictionary, based upon a strategy outlined by the prolific Dr. Steve Parker in a previous Clarion Magazine article. I also use a combination of .INI file settings for workstation-specific settings and a single record control file stored in the data file location to store system specific setting which need to be applied across all users. Again, I do this using a modification of a methodology outlined by Dr. Parker in yet another Clarion Magazine article. (Thanks Steve!)

Each time I called the report to be executed, a dialog box would open prompting me to locate the file(s) upon which I wished to report. The reason for this behavior turned out to be that the ReportEngine does not know the values of the variable file names contained in the data dictionary imported into the library at the time it was created. Once again, I consulted the documentation for details on calling the ReportEngine methods.

It turns out the `ReportEngine` method I needed to use is called `ResolveVariableFilename`.

The example for the `ResolveVariableFilename` method looked to be fairly straightforward. However, when I attempted to implement the procedure call as documented, I found that the global variable for the filename would get replaced (as indicated by the proper return code from the procedure) but the file location dialog box would continue to prompt for the proper file location. I read the documentation until I was fairly blue in the face and then I started noticing some inconsistencies.

In most of the examples the code looks like this:

```
PROGRAM
MAP
END
INCLUDE('RWPRLIB.INC')
RE    ReportEngine
  CODE
  IF RE.LoadReportLibrary('RWDEMO1.TXR')
     IF NOT RE.PrintReport('Report1')
          MESSAGE('Print Failed')
     END
     RE.UnloadReportLibrary()
  ELSE
     MESSAGE('Load Failed')
  END
```

The interesting thing to me was that this code actually works for the example, but I couldn't get it to work in my own application.

One of the things the documentation for the `ResolveVariableFilename` method does not tell you (and this is a critical piece of information) is that you must also use the `SetVariable` method in conjunction with it even, though you *do not* have to create runtime variables in your report library in order to reference the variables.

The example in the documentation is as follows:

```
! Class declaration (partial)
RE CLASS(ReportEngine)
ResolveVariableFilename PROCEDURE(STIRNG vname, *STRING value)|
                            ,SIGNED, VIRTUAL
...
END

! Method definition
RE.ResolveVariableFilename PROCEDURE(STRING vname, *STRING value)
  CODE
  IF label = '!avariable'
       Value = 'c:\examples\test.tps'
        RETURN TRUE
  ELSE
      RETURN FALSE
  END
```

This being the case, the proper code to make this work would be:

```
! Class declaration (partial)
RE CLASS(ReportEngine)
ResolveVariableFilename  PROCEDURE(STRING vname, |
                            *STRING value),SIGNED, VIRTUAL
...
END
...
! Code that gets executed
RE.SetVariable('avariable', 'c:\examples\test.tps')
...

! Method definition
RE.ResolveVariableFilename PROCEDURE(STRING vname, *STRING value)
  CODE
  IF vname = 'avariable'
       value = avariable
        RETURN TRUE
  ELSE
      RETURN FALSE
  END
```

As you can see there is a bit of a difference in the code, and there is quite a bit of difference in

the result. Not only is the call to `SetVariable` required, but the `ResolveVariableFileName` method now sets the `Value` variable (passed by address) to the contents of the appropriate class variable (which in my example is a global variable).

The key to my discovery was an email conversation I had with Russ Eggen. I sent him my example application and asked if he could look at it and tell me why it would not work, even though I had it coded per the examples in the documentation. Russ was gracious enough to provide me with the necessary information.

His response to me contained a example template used to call the Report Engine from a procedure, and when I generated the code for the example I saw the proper sequence of commands and the inclusion of the `SetVariable` method.

Now that the mystery was solved I could complete the implementation, which I'll explain next week.

Download the source

Ben E. Brady lives in what he affectionately refers to as the 'No-Tech capital of California', Dinuba, approximately 45 minutes southeast of Fresno in the heart of the Central Valley, with his wife Rita. A programmer for more than 30 years, he discovered Clarion 2.1 in 1989 after having programmed in several xBase dialects, BASIC and Z80 assembler. Ben is currently attending the College of the Sequoias in Visalia California, pursuing a degree in Computer Science and a teaching certificate in order to pass along some of his knowledge. Ben and Rita are also very active with the Tule Fog PC Users group in Visalia California, where they provide instruction to others in need of assistance with technology and give presentations to groups of all types. Ben is also the author of several very popular firewall reporting tools, written exclusively in Clarion for Windows, for users of personal firewall software such as BlackICE, ZoneAlarm, WinRoute Pro and XP Firewall.

Reader Comments

Add a comment

The source zip has been updated with a missing file -...

Calling A Clarion Application With PHP

by Ville Vahtera

Published 2003-01-06

PHP is a hugely popular scripting language used mainly for web development; Clarion 5.x is a Windows language used mostly for business software development. What do they have to do with each other? In this article, I will show you how to create a Clarion application which can be executed from, and have its output captured by, PHP. This provides web access to my application (and TopSpeed database) over the Internet/Intranet.

What is PHP?

PHP is a widely-used general-purpose scripting language that is especially suited for Web development. PHP statements can be embedded into HTML, just as server-side VBScript and JavaScript statements can be embedded in ASP pages.

You do not need any special third party software to generate PHP pages or scripts - they are pure text, so Notepad (that famous html editor) will do just fine for what I'm going to describe here.

To get the most from this article, you should have a web server set up and running (e.g. Apache or Xitami) and the latest version of PHP. You will also need a little experience with PHP, or at least passion to learn couple of new tricks.

Xitami

First, a few words about Xitami, which is an entry-level server for Windows 9x/NT/3.x, Unix, OpenVMS, and OS/2 platforms. Xitami is an excellent choice for inexperienced users.

If you need speed but don't want the baggage that comes with most other Web servers, Xitami

may be your answer. Xitami is built around iMatix's high performance SMT (Simple Multi-Threading) kernel. Xitami has a loyal, perhaps even fanatical following that is growing steadily due to its price tag (Xitami is free including vc++ source code, SSL source code version will cost about 99US$), and its tried and true performance. To the delight of many, Xitami resuscitates older Pentium boxes, making these dust-collectors serviceable once again.

I chose Xitami over IIS because of the security holes (and the endless security patches) inherent in IIS . Okay, you will lose the chance to use ASP scripting, but you get PHP and excellent windows CGI support, and you also get a chance to sleep peacefully!

Creating the Clarion application

For this example, I will create a simple Clarion application that will accept arguments via the command line, and write data out to the console. If I have an application named **php4clar**, for example, then I can pass it three (or more) command-line arguments like this:

```
php4clar -firstArg –secondArg -thirdArg
```

This example shows how to call the application from the command line – when calling from a PHP script, the `/PHP` switch is required as a first argument. More about that a little later.

The application will output the number of arguments sent to it, as well as the values of each of these arguments. I will use a PHP script to execute the Clarion application via the PHP exec() function.

My code starts by outputting the number of arguments that were passed in from the command line. The command line parameter array is indexed from zero onwards, and will always contain at least one value, the full path and name of the application, which is automatically added by the Clarion compiler. I use a loop to determine whether or not there was more than one argument passed in from the command line or PHP script. If, for example, I pass the arguments "-firstArg", and "-secondArg", then my application would output the following:

```
Number of arguments: 2
Argument1 : -firstArg
Argument2 : -secondArg
```

```
Loop x# = 1 To 50           ! Get first 50 arguments
   Argument = Command(x#)   ! get argument
   If Argument              ! is there anything
       ArgumentCount# += 1 ! how many arguments we have counted
       OutData1 = OutData1 & 'Argument' & x# & ' : ' & Argument & '<13,10>'
   Else
       Break ! no more arguments
   . ! end if
. ! end loop
       OutData2 = 'Number of arguments: ' & ArgumentCount# & '<13,10>'
       OutData1 = OutData2 & OutData1 & '<13,10>'
       WriteToConsole(OutData1)
```

Figure 1. Loop through 50 command line parameters

The Clarion application displays these arguments on the screen using the `WriteToConsole()` function (see the demo app). You don't have to compile your application with the console flag; it will work anyhow, but does not write anything to the console screen.

"Wait! what did you say? Clarion console application .. how did you do that?" OK, you're right. By default Clarion will compile every application as a GUI application which therefore doesn't write to the console.

Here is some more information about the difference between GUI (windows) & CUI (console) application:

> By default, a newly created process inherits the same console (if any) of its parent. Console apps therefore don't have to call anything special in order to attach to the console if they are launched from a process that is attached to a console.
>
> To handle the case where they are launched from a process that is not attached to a console (a "GUI" app), consoleapps have an explicit call to `AllocConsole()` in the startup code that gets executed before main(). This call silently fails when the app has already inherited a console.
>
> GUIapps, on the other hand, don't want a console. When they are launched from a process that is not attached to a console (a GUI app), they're fine. To handle the case where they are launched from a process that is attached to a console, GUI apps have an explicit call to FreeConsole() in the startup code that gets executed before `WinMain()`. This call silently fails when the app hasn't inherited a console.
>
> How do you get back to that console? XP has an `AttachConsole()` API call, but this application needs to work on other versions of Windows as well.
>
> To summarize, the main difference between console apps and GUI apps is whether

the app calls `AllocConsole()` (console) or `FreeConsole()` (GUI) at startup time. (Read the full message here)

So how to compile a console application with Clarion? After spending a couple of nights playing around with console APIs and trying to make my Clarion application write to the cmd prompt, I found out that with Visual C++ I can set option `/SUBSYSTEM:` to `console` or `windows`, depending on my need to compile a CUI (console) or GUI (windows) executable.

Similarly, with Clarion, it is possible to set the application type on the .EXP file `NAME` line so the compiler creates a GUI (windows) or CUI (command line) application.

To make a console app, add this in the .EXP file:

```
NAME YourAppName CUI
```

To make a windows app, add this in the .EXP file:

```
NAME YourAppName GUI
```

See the included **php4clar.exp** file for more details. Save the EXP with the same name as the source file name, into the same directory as your source code. Compile your application and you will get either a console or a GUI executable.

You'll now need to upload or copy the **php4clar.exe** file to your web server using some sort of FTP program. I recommend CuteFTP.

Change to the directory where you uploaded the **php4clar.exe** file into. To test the application, enter the following command at the cmd prompt:

```
php4clar one -two /three
```

The following output should be shown on the screen as below:

```
Number of arguments: 3
Argument1 : one
Argument2 : -two
Argument3 : /three
```

Now that my Clarion application is ready to go, I need to create the PHP scripts that I will use to access the application from the web browser.

When the client issues a request, PHP allows my program to be executed on the server among the data passed from the server to my Clarion application via operating-system command line arguments. When the script has called for the application, the output of the program constitutes

what is sent back to the client's web browser.

> **NOTE:** This page will not finish loading until program being called closes stdin, stdout, and stderr.

The Clarion application must do two things: Read the data from the HTML page as passed in by the server and PHP script, and generate in response for the PHP script to return to the server.

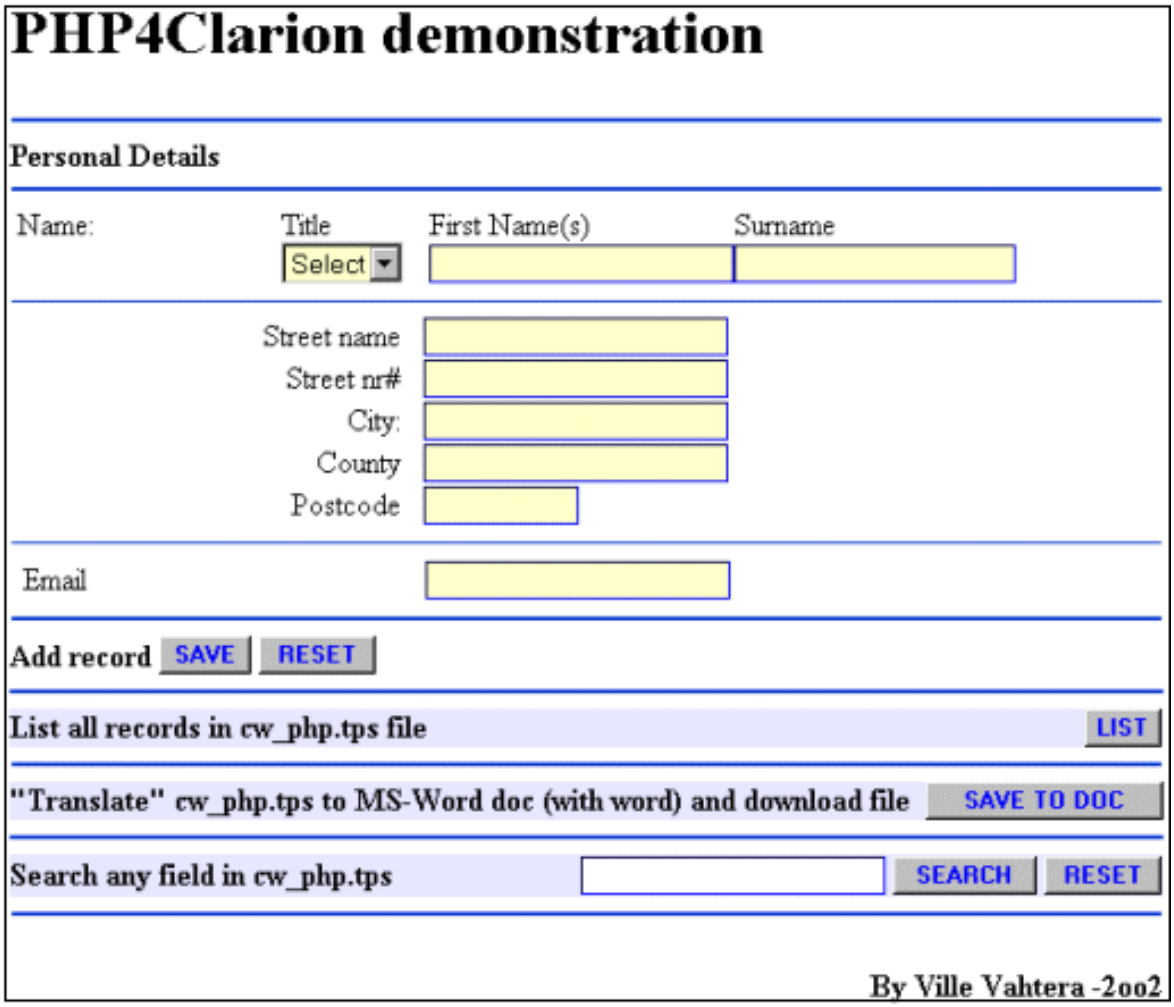


Figure 2. Index page for the demo

Creating the PHP scripts

The entire code for all of the PHP scripts is a bit too long to list, so you can download all the scripts as part of the source code and support material for this article, from end of the article.

When submitting a request (**SAVE**, **LIST**, **SAVE TO DOC** or **SEARCH**), firstly, the fields which the user fills out will be sent to a PHP script. Secondly, my script checks to see whether the `$submit` (or any other) variable contains a value.

The code `if (isset($submit))` simply prevents the script from being called without parameters.

The `$submit` variable is passed from the form submitted at the index page, with any of the submit action commands here (I have four forms on the index page):

```
<form method="POST" action="put.php">
<INPUT TYPE=hidden NAME=submit VALUE="/PHP PUT">
<form method="POST" action="get.php">
<INPUT TYPE=hidden NAME=submit VALUE="/PHP GET">
<form method="POST" action="word.php">
<INPUT TYPE=hidden NAME=submit VALUE="/PHP GET">
<form method="POST" action="get.php">
<INPUT TYPE=hidden NAME=submit VALUE="/PHP SEARCH">
```

All the form fields inside `<FORM> ... </FORM>` will be passed to the PHP script.

The "action" attribute of the form is set to the `$submit` variable. Because the `$submit` variable is holding my command for the application, I need to verify that all the parameters for the request command are filled, so this is the place where the code in the "else" braces is executed.

```
<?php
/* Look for errors in all variables and
   display an error message if one is found */
if (empty($firstname)) {
echo "<h1>You didn't fill First name!</h1><br>
 <a href=\"javascript:history.back()\">Go    back</a>\n";
}
elseif (empty($surname)) {
echo "<h1>You didn't fill Surname!</h1><br>
 <a href=\"javascript:history.back()\">Go    back</a>\n";
}
```

My **PUT.PHP** script will look for errors in all variables and display an error message if an empty variable is found.

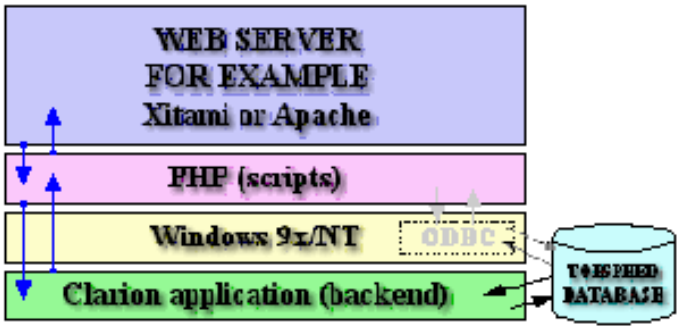


Figure 3. Using PHP between client's browser and Clarion backend

Once I enter some arguments and submit the form, the `$submit` variable will contain a value. This will cause my PHP script to execute the `exec()` code between the "if" braces.Here's the syntax of the `exec` command:

```
string exec( string command [, array output [, int return_var]])
```

It is also possible to use the `passthru()` function when the output from the command is binary data which needs to be passed directly back to the browser.

When you push the **SAVE** button on index page, this is the code (in **PUT.PHP**) which will get executed:

```
// Execute the program
$Command = exec("php4clar.exe $submit $titles $firstname
 $surname $street $house_no    $town $county $postcode
 $contact_email", $results, $error);
// Assign variables and return the current key
// and value pair from an array
while (list(,$line) = each($results))
{ // Output strings
  echo $line, "<BR>\n";
}
echo "<br><a href=\"javascript:history.back()\">Go back</a>";
if ($error)
  {
    echo "Error code: $error<BR>\n";
    exit;
  }
}
```

The `$submit` variable is automatically created (as `HIDDEN`) and given the value of the text field in my HTML form source. If no arguments were entered, I simply tell the user that they didn't enter any.

On the other hand, when the user has filled the whole form correctly, then the **PUT.PHP** script will pass all the fields to my application.

If I have filled out all the variables into the text fields of my HTML form, then, under the hood, my executed command would look like this:

```
/www-root/php4clar/php4clar.exe /PHP ↵
  PUT titlesVALUE firstnameVALUE surnameVALUE,    etc...
```

where `/PHP PUT` is the value from the `HIDDEN $submit` field and the other variables are part of the visible form.

I don't have to specify the full path to my **php4clar.exe** file, just because the scripts are in the

same directory. In this example, my files are located in the **www-root/php4clar** directory. You should change this to match the path to the **php4clar.exe** file on your server.

Conclusion

As you can see from the example and source code in this article, PHP allows developers to execute external programs as independent threads, via a script. The example described in this article is quite elementary, but with a bit of work you could create a Clarion application that runs commands, or does some other neat stuff like build a control over running processes, request reports (see my word 'report' example), it's all up to your imagination. Always make sure that your security settings are correct; for example, do not allow directory browse, and never give open-access to your internal applications via any sort of script, in other words, grant access to all authenticated users, and denied to all anonymous users. And of course study the PHP manual!

Related Links

- http://www.php.net (PHP home page)
- http://www.dzsoft.com/dzphp.htm (Excellent PHP editor for Windows)

Related Books

- PHP and MySQL Web Development
- Professional PHP Programming

Download the source

Ville Vahtera lives near Helsinki, Finland, and is an active participant in the Clarion newsgroups. Ville was first impressed by Clarion back in the DOS days. He has been using Clarion as his primary development environment ever since version 4. When not in front of a computer, Ville enjoys listening to music, going regularly to the gym, swimming, playing badminton, and spending time with his daughters.

Reader Comments

Add a comment

Integrating The Clarion Report Writer Into Your Applications (Part 2)

by Ben E Brady

Published 2003-01-16

Last week I explained how I created a test application that allowed me to pass parameters stored in a data file for predefined ReportWriter reports to the Clarion Report Engine. The main difficulty I had was getting the Report Engine to recognize the variables used for the path to the data files contained in the data dictionary. Once I had that sorted out, I was ready to implement my solution. This week I'll introduce you to the embed code that does the work.

In my test application, in the **Browse Reports Window - Local Data - Other Declarations** data embed I entered the following code:

```
MyRE CLASS(ReportEngine) ! instantiate the ReportEngine object
ResolveVariableFilename PROCEDURE(STRING Vname, |
                          *STRING Value),SIGNED,VIRTUAL
   END
```

In the **Browse Reports Window - Print Button Accepted** code embed I placed the following:

```
SavePath = PATH()
IF EXISTS(CLIP(CRW:ReportLibrary))
  IF CRW:DateRange
     GetDateRange
  END
  IF MyRE.LoadReportLibrary(CLIP(CRW:ReportLibrary),|
    CLIP(CRW:ReportLibraryPassword))
    ! Open and load the report library
    MyRE.SetVariable('GLO:AcctRep_  ',CLIP(GLO:AcctRep_))
    MyRE.SetVariable('GLO:AuthAlt_  ',CLIP(GLO:AuthAlt_))
    MyRE.SetVariable('GLO:Billing_  ',CLIP(GLO:Billing_))
    MyRE.SetVariable('GLO:Colorstd_ ',CLIP(GLO:Colorstd_))
    MyRE.SetVariable('GLO:Contacts_ ',CLIP(GLO:Contacts_))
```

```
    MyRE.SetVariable('GLO:Customer_ ',CLIP(GLO:Customer_))
    MyRE.SetVariable('GLO:Delivery_ ',CLIP(GLO:Delivery_))
    MyRE.SetVariable('GLO:Forms_    ',CLIP(GLO:Forms_))
    MyRE.SetVariable('GLO:Frmsizes_ ',CLIP(GLO:Frmsizes_))
    MyRE.SetVariable('GLO:FVndrchrg_',CLIP(GLO:FVndrchrg_))
    MyRE.SetVariable('GLO:Orders_   ',CLIP(GLO:Orders_))
    MyRE.SetVariable('GLO:OVndrchrg_',CLIP(GLO:OVndrchrg_))
    MyRE.SetVariable('GLO:Package_  ',CLIP(GLO:Package_))
    MyRE.SetVariable('GLO:POHeader_ ',CLIP(GLO:POHeader_))
    MyRE.SetVariable('GLO:PODetail_ ',CLIP(GLO:PODetail_))
    MyRE.SetVariable('GLO:Press_    ',CLIP(GLO:Press_))
    MyRE.SetVariable('GLO:QPWSys_   ',CLIP(GLO:QPWSys_))
    MyRE.SetVariable('GLO:Reporter_ ',CLIP(GLO:Reporter_))
    MyRE.SetVariable('GLO:Runsizes_ ',CLIP(GLO:Runsizes_))
    MyRE.SetVariable('GLO:Shellcnt_ ',CLIP(GLO:Shellcnt_))
    MyRE.SetVariable('GLO:Shipper_  ',CLIP(GLO:Shipper_))
    MyRE.SetVariable('GLO:Shipping_ ',CLIP(GLO:Shipping_))
    MyRE.SetVariable('GLO:VContacts_',CLIP(GLO:VContacts_))
    MyRE.SetVariable('GLO:Vendors_  ',CLIP(GLO:Vendors_))
    IF CRW:Preview
       MyRE.SetPreview(CRW:NumberOfPages)
    ELSE
       MyRE.SetPreview(0)
    END
    IF CRW:DateRange
       MyRE.SetVariable('ReportStartDate',|
         CLIP(FORMAT(GLO:ReportStartDate,@d17)))
       MyRE.SetVariable('ReportEndDate'  ,|
         CLIP(FORMAT(GLO:ReportEndDate,@d17)))
    END
    IF CRW:Copies
       MyRE.SetNumberOfCopies(CRW:NumberOfCopies)
    ELSE
       MyRE.SetNumberOfCopies(GLO:ReportCopies)
    END
    IF CRW:Device
      MyRE.SetPrinter(CLIP(CRW:Device))
    ELSE
      MyRE.SetPrinter(CLIP(GLO:ReportPrinter))
    END
    IF NOT MyRE.PrintReport(CLIP(CRW:ReportName))
       MESSAGE('Report Printing Failed', |
         'Report Writer Report Printing Failed for ' |
         & CRW:ReportName, ICON:Exclamation)
    END
    MyRE.UnloadReportLibrary
  ELSE
    MESSAGE('The Report Writer library file could not be loaded.', |
        'Report Writer Library File Load Failure', ICON:Exclamation)
  END
ELSE
  MESSAGE('The Report Library file is not currently accessible.', |
      'Report Writer Library File - Access Restricted', ICON:Exclamation)
END
SETPATH(SavePath)
```

And in the **Browse Reports Window - Local Procedures** embed I placed the following code to enable the use of the `ResolveVariableFilename` method:

```
MyRE.ResolveVariableFilename  PROCEDURE(STRING vname, |
                                        *STRING value)
RetVal SIGNED(0)
  CODE
  RetVal = FALSE
    CASE UPPER(vname)
  OF 'GLO:ACCTREP_'
    value = GLO:AcctRep_
    RetVal = TRUE
 OF 'GLO:AUTHALT_'
    value = GLO:AuthAlt_
    RetVal = TRUE
 OF 'GLO:BILLING_'
    value = GLO:Billing_
    RetVal = TRUE
 ! And so on and so forth for all other
 ! file name variables

 END
 RETURN RetVal
```

What is not obvious about the `ResolveVariableFilename` method is that it is called by the Report Engine itself. Implementing the code as shown above *overrides* the default method in the ReportEngine. As you can see there are not any calls to it in the Print Button embed code. In my mind I think you have to consider it a method to set properties even though the prototype appears as if you would call it explicitly like any other Clarion procedure.

There are other Report Engine methods that you may find useful in addition to the ones listed above, such as `SetReportFilter` and `SetReportOrder` (beware of the errors in the documentation on this one as well – someone forgot to proofread the page). I haven't implemented them in the code listings above as I tend to use the filtering and sort ordering within the Report Writer. You could certainly use these if you had a way to capture the filtering and field ordering information in such a way that your user could specify it in a form and then pass it along to the `ReportEngine`.

You should note that while it appears it would be useful to put all of this report calling code into a template, doing so would not meet the main objective, which is the addition of unlimited reports to the application without recompiling. If you were to use a template to specify a new procedure you would have to re-compile each time you wanted to add a report. As much as I like the idea of a template, there are times when creating a template is not the best idea, and it is my opinion this is one example of when not to do so.

Using the data file structure listed above, in conjunction with the code listings provided, you can now implement any Clarion Report Writer report library, in any location on your hard disk

or on your network, even though you may be using variable file naming in your data dictionary.

Don't forget to add the C55PRLBX.LIB to your application project properties (as a trick I use `C%V%PRLB%X%.LIB` as the file name in the project file in the event that I need to switch between 16 and 32 bit targets) and distribute the C55PRLBX.DLL or C55PRLB.DLL file with your application, along with the .TXR file containing your reports. There is no need to distribute your data dictionary to the end user unless you want to also sell them a license of the Clarion Report Writer and allow them to modify the report library.

If you need to add reports to your application in the future, you can simply modify the Report Library and have your user update their file. Better still, keep the report library you initially distributed with your application as your standard default report module, add the additional reports to a new report library, and distribute the additional report library to implement the new reports. All your user has to do is enter a new record into the REPORTER.TPS file for each new report contained in the Report Library and reference the location of the new .TXR file in the record entry.

> **Note:** The name of the report must match the name of the report contained in the report library. Spaces are not allowed and are automatically replaced with underscores.

If you have the data files located on a network it is advisable to install the default and any new TXR files into the same folder as your data files so everyone running the system can have access to them as necessary.

If you have specialized reports such as payroll or other sensitive information, you can have the user install them onto the local hard drive, preferably in the same folder where your application is installed, and provide a layer of security by limiting access to the report library containing the restricted report. This is the reason for the `IF Exists(CRW:ReportLibrary)` clause at the top of the **Print** button embed. If the report library file does not exist on the network drive when a user tries to access the report, the ReportEngine code does not get executed and the user is presented with a message describing the restricted access.

Another important point to note is that if you have your data files and your report library files located on a network drive, or in a location other than your application on your local hard disk, you *must* save the current path prior to loading the CRW report library you are going to use. You must do this because the ReportEngine method `LoadReportLibrary` intrinsically changes the path to the location of the report library being opened, and will not automatically restore the path for you when the `UnloadReportLibrary` method is called to close the report library. Restore the path after you unload the report library. Failure to do so could

potentially result in your application not being able to find data files or .INI settings once the report has been processed, and will most assuredly result in a very ungracious halt of the application.

Although I have been using Clarion for nearly 15 years I still find features and tricks to make my development tasks easier and enhance the functionality of my applications for my users. The Clarion Report Writer for Windows, and the ReportEngine, have regained a place in my toolbox instead of being forever relegated to the dusty corners of my hard disk. I hope you'll find this reporting technique as useful as I have.

Download the source

Ben E. Brady lives in what he affectionately refers to as the 'No-Tech capital of California', Dinuba, approximately 45 minutes southeast of Fresno in the heart of the Central Valley, with his wife Rita. A programmer for more than 30 years, he discovered Clarion 2.1 in 1989 after having programmed in several xBase dialects, BASIC and Z80 assembler. Ben is currently attending the College of the Sequoias in Visalia California, pursuing a degree in Computer Science and a teaching certificate in order to pass along some of his knowledge. Ben and Rita are also very active with the Tule Fog PC Users group in Visalia California, where they provide instruction to others in need of assistance with technology and give presentations to groups of all types. Ben is also the author of several very popular firewall reporting tools, written exclusively in Clarion for Windows, for users of personal firewall software such as BlackICE, ZoneAlarm, WinRoute Pro and XP Firewall.

Reader Comments

Add a comment

Hi Ben, Thanks for the article! Something to consider...

Making The E-Mail Connection

by Brice Schagane

Published 2003-01-17

Recently, I had a user request a function that would open a New Message window in his default mail client. He also wanted to have the message initialized with specific file loaded data.

At first, I thought this would require something like OLE or DDE. But after review, I discovered that I already had the tool I needed. I frequently use John Hickey's Internet Link Template to create controls which will open a hypertext transfer protocol (HTTP) session using a specified site address. Using this template, I can also create controls, which will open an email message editor in the user's email program using a specified email address. Before continuing, you should review John Hickey's article Making The Internet Connection (freely available as part of the Clarion Online archive).

The "mailto" protocol

After a little research, I discovered that initializing the email message wasn't that difficult. The code behind John's template utilizes Windows' `ShellExecute` API function. When using the `ShellExecute` function to open a client's email system, the supplied address must comply with the "mailto" protocol.

The syntax for the mailto protocol is pretty straightforward:

mailto:*sAddress*[*sHeaders*]

sAddress – This can be one or more valid e-mail addresses separated by a semicolon.

sHeaders – This token is optional, but can contain one or more name-value pairs. The first pair should be prefixed by a "?" and any additional pairs should be prefixed by a "&". The name-value pairs can include any of the following:

subject Text to appear in the message subject line.

body Text to appear in the message body.

CC Addresses to be included in the "cc" section of the message.

BCC Addresses to be included in the "bcc" section of the message.

Special characters may be represented in the message body through the use of URL encoding. For example, a carriage return can be represented by `%0D` and a linefeed can be represented by `%0A`.

Putting it all together

When using the Internet Link control, I specified that I would be linking to E-Mail. For the address, I chose to use a local variable. I did this by placing an ! in front of the variable name (for example, `!Loc:myAddress`).

Within the Internet Link Template, there is a variable labeled `URLBuffer`. The `URLBuffer` is defined as a `CSTRING(256)`. I modified the template source code, changing the variable definition to a `CSTRING(512)`. If I had a need to send larger emails, the size of this variable should be increased accordingly.

Finally, before the template code is called, I simply fill the local variable with a valid email address and any valid name-value pair(s) that I wish to use. This is shown in the following example.

```
Loc:myAddress = Schagane@mis.net?subject=Feedback&body=Clarion Rocks!
```

That was easy! In all its simplicity, John Hickey's template has proven useful many times. The beauty of it is that it can easily be adapted to handle much more than just Internet Links. But that's another story…

Brice Schagane works for the Kentucky Transportation Cabinet. He also runs a small computer company by the name of Ghost Solutions, Inc. Brice has been using Clarion since 1997.

Reader Comments

Add a comment

This is Nice and Easy.....is it possible to add an...
John: Sorry, but the mailto protocol does not support...

Clarion Magazine

Clarion News

Published 2001-11-21

DCTBuilder 1.09

A new version of DCTBuilder is now available - this release fixes problems with field type conversion, and adds exporting of query results.

Posted Monday, January 27, 2003

PDF-Tools Now Supports JBIG Format

JBIG image format support is now included in the Image-XChange SDK. (JBIG provides a to state-of-the-art lossless compression ratio for high resolution bi-level images, at around three to four times better compression than GIF on typical 300 dpi documents. JBIG supports hierarchical "progressive" encoding, that means it is possible to encode a low resolution image first, followed by resolution enhancement data. See www.jpeg.org/jbighomepage.html for more on JBIG.) As other PDF-Tools SDKs and PDF products automatically inherit this improved functionality - the current version now available for download includes this update. This is a free update. Also you can view, manipulate and save from/to JBIG in Image-XChange - even from Clarion with no OCX required (same with all supported formats).

Posted Monday, January 27, 2003

ImageEx 2 Beta 4

ImageEx 2 beta 4 is now available. If no bugs are reported within the next weeks, this will become the gold release. New features include: Mouse wheel support for the viewer and panner control; Fixed hotspots (don't scale, don't move); Several new event handlers (virtual methods) for all controls; "NoWrap" feature for the panner; Twenty six new draw modes for the bitmap class; Text transformations (rotate, scale & skew); Clipped drawing of images. The updated documentation contains a detailed list of changes. ImageEx is available for purchase at www.clarionshop.com for US$149 (beta price, will go up to US$ 199 on gold release).

Posted Monday, January 27, 2003

HTML Designer For Clarion Updated

HTML Designer Version 1.03 (Beta) Build 69 is now available. This is a FULL Installation and includes all the required Microsoft installations (9.2 Mb). New users can purchase the product via www.clarionshop.com; Current users can use the password supplied by Clarionshop to extract and install HTML Designer.

Posted Monday, January 27, 2003

EasyVersion 2.00 Released

EasyVersion 2.00 is now available, along with a new demo. This release includes a number of changes and bug fixes, as well as several new features. Registered users of version 1.xx can upgrade for a discounted price at ClarionShop.

Posted Monday, January 27, 2003

Metabase Web Site Update

The Whitemarsh Metabase has had two minor updates since the last full release. Most recent changes include: RLC - Changed Resource Lifecycle Node Database Schema to Resource Lifecycle Node Database Object Assignment; SDM, IDM, ODM - Import SQL DDL procedures have increased accuracy, and newly added error logs to store DDL errors; Auto Creation (importation) of both DBMS and ANSI SQL data types in the IDM and ODM; Auto Creation of a complete set of default data; The Operational Data Model module now supports the import and export of [unique] INDEXes when reading and writing SQL DDL; Added six re-engineering windows to database objects; Fixed a number of small display errors in a number of modules.

Posted Monday, January 27, 2003

C6EA Info On SV Web Site

You can now read more about Clarion 6 EA on the SoftVelocity web site. There are three PDFs on the new threading model, plus all the currently available C6 documentation.

Posted Thursday, January 23, 2003

Virtual EIP holiday schedule

Alan Telford is taking an eight week break from programming. Anyone needing help with the Virtual EIP templates can try Alan's home email address (adcatelford@maxnet.co.nz), which he says he may occasionally check if he's feeling energetic.

Posted Friday, January 17, 2003

PDF-Tools SDK Version 2.5 Released

PDF-Tools SDK version 2.5 for Clarion is now available. This version is all new and will not update or alter or remove previous versions. New features include: Watermarks; Page Rotation; Support for all previewers - Clarion, TinTools RPM etc.; Convert Images direct to PDF. (Image formats supported BMP, DCX, GIF, JPEG, JNG, PCX, PNG, TGA, TIFF,

AMF/EMF/WMF) JBIG due FEB 2003; PDF File passwording and encryption, and more. This is a free upgrade for all existing users. There will be a further update in the next two weeks adding the ability to create and add bookmarks (outlines) directly from your Clarion Reports with just a tick in the template based on certain report values, as well as a Crop function for existing pages.

Posted Friday, January 17, 2003

New Year Special Offer - 50% Off BackFlash

Save 50% on Sterling Data's BackFlash if you order by midnight, Saturday, January 18, 2003. BackFlash provides a simple one-button backup facility that makes backup easy. Compression by the high performance 32 bit ZLIB DLL - this is royalty free and can be distributed by you as part of your BackFlash EXE without restriction. BackFlash will work out how many floppies are required and notify the user before starting. Each set of backup disks will be given a unique serial number to ensure that only disks from the same set are used on restore and also that the most recent backup is being used. Any drive and path can be specified for backup - so large capacity drives such as ZIP drives could be used, or drives somewhere else on the network. Other features include an unattended backup facility, logging, last backup date/time stored in INI file for use by other apps, multi-language support, and more.

Posted Friday, January 17, 2003

Developer Graphics Special (Save $100)

Gitano Software is offering a Developer Graphics Special, including: Application Logo; Application Icon; Splash Screen; Product Box Image; 20% off coupon for additional custom image work. All images are distributed as multilayer PSD and JPG (You may choose a different format instead of JPG). The application icon is distributed as a single ICO resource file containing 48x48, 32x32, 24x24, XP, 16mil and 256 formats. You get also get three test logo images to select one from and the rest of the images will based on your selection. The splash screen will contain room for you to add dynamic data at runtime. Use the 20% off coupon to get additional work for your application such as toolbar icons. If purchased separately the cost of the package would be $299+.

Posted Friday, January 17, 2003

INN Bio & News For 14-Jan-2003

INN's first bio of 2003 features one of Clarion's better-known template authors. A listing and labelling expert, he's also a dad, and when not working, he enjoys watching birds from his back yard - both those birds created by nature and those created by the Royal Air Force.

Posted Friday, January 17, 2003

xTransparent Window v1.3

New in xTransparent Window v1.3 is template modification - Transparent on Window Lost/Gain Focus events is enabled now. New demo and install kit are available.

Posted Friday, January 17, 2003

Sneak Preview Of cpTracker 2003

The Berthume Software web site now features screenshots showing much of the new cpTracker project management software interface and capabilities. A fully functional demo of cpTracker 2003 with data to play with will be available soon.

Posted Friday, January 17, 2003

C55 Threading Problems Analysis

In this newsgroup message, Owen Brunker has provided an analysis of and possible solution for threading problems in Clarion 5.x.

Posted Friday, January 17, 2003

New Web Site For DOS Printer

DOS Printer now has an advertisement-free web site. DOS Printer takes text files one character at a time and creates a report using Epson control characters to determine font characteristics. The resulting windows report can be printed to any windows device or emailed.

Posted Friday, January 17, 2003

CWODBC Version 1.9

Dan Pressnell has released CWODBC version 1.9. This version fixes a bug that can sometimes happen with the GetColumnValue method if you use a column number that is greater than the number of columns returned. As well, there is now a method called FillQueueWithResultByName. This allows your queue fields not to match exactly the order of the columns in the select statement. Also, you can have fields in the queue that are not in the select statement at all. (This can be handy for things like icons in browses, etc.)

Posted Friday, January 17, 2003

DictionaryBuilder 1.1

Peter Rakké's DictionaryBuilder creates a dictionary from any ODBC connection, complete with keys and relationships. Used by Erik Pepping and Peter Rakké in RAD Race 2002.

Posted Friday, January 17, 2003

TDAN Issue 23.0

The new issue of The Data Administration Newsletter (Issue 23.0) is now available on-line. This issue includes: ; 12 new articles; 4 new feature columns; Dozens of new data links; many new recommended readings; updateds book center - new reviews, books listed; updated article archive, conference center and company and product center, and more.

Posted Friday, January 17, 2003

OutlookFUSE 1.1 released

ThinkData has released Version 1.1 of OutlookFUSE. The new version features improved support for COM date/time values, support for Outlook XP event handling, more feature rich examples, and examples of late binding to other COM objects including PowerPoint, Excel, and ADO. The demonstration applications have been updated

Posted Friday, January 17, 2003

PlugIT Memory Leak Checking for Clarion 5/5.5 ABC And Legacy Templates

ThinkData, Inc., in conjunction with Plugware Solutions.com Ltd., has released PlugIT, a memory leak checker and thread safe memory allocation engine for Clarion 5/5.5 ABC and Legacy Templates. PlugIT is a template-based solution giving Clarion programmers the ability to check existing or brand new applications for memory leaks in a matter of minutes. PlugIT also checks third party generated code for memory leaks. PlugIT overrides key portions of the Clarion runtime library (RTL) and monitors those functions to determine when an allocated block of memory is not deallocated. PlugIT then outputs the location of the memory leak to a log file. PlugIT is extremely simple to use as well - it is as easy as adding a global extension template to your existing and future applications and recompiling. Demo available. Features include: Memory leak checking including log file generation and module name/line number support allowing programmers to find the leaks and fix them quickly; Thread safe memory allocation in Clarion 5/5.5 ABC and Legacy; Easy template interface; Windows GDI (graphics device interface) leak checking; Windows COM (component object model) leak checking; Full source code and examples included; Enterprise Site License allows unlimited installations and support contacts for an organization; Free technical support via online forum. Single user license is US$199.

Posted Friday, January 17, 2003

Fomin Report Builder v.2.85

Fomin Report Builder v.2.85 is now available. Registered users are welcome to use the "Live Update" service. This is a free update as before (since 1997, when the first version was released).

Posted Friday, January 17, 2003

Clarion 6 EA Has Shipped!!

The Early Access release of Clarion 6 has shipped, and developers are beginning to receive their copies!

Posted Friday, January 17, 2003

Sterling Data Templates And Clarion 6

Sterling Data's update policy with regard to C6 will be the same as with previous Clarion version releases - no charge will be made. The only requirement will be that customers will be using the latest *major* template version - e.g. IMPEX 5, 5.1 and 5.2. Customers will be

emailed with the free upgrade details. IMPEX 4 users will need to upgrade, but in most cases no upgrade will be needed because all source code is supplied automatically when you buy a template (with the exception of ExcelBond).

Posted Thursday, January 02, 2003

BoxSoft Vacation Schedule

The BoxSoft office will be closed from January 1st through the 13th. Shortly after the office reopens look for the new version of SuperSecurity. This release includes numerous updates for password and logon handling, better support for Clarionet, enhanced documentation, and many other goodies. BoxSoft will also be testing its products against the C6 Early Access release.

Posted Thursday, January 02, 2003

ZipApp 1.1a Released

ZipApp 1.1a is now available. This release includes a new option to list archives (in selected parent directory and ZipApp directory) and optionally open the file (using your default archive program) or delete the file. If you have ZipApp click on the version number to go to the download page.

Posted Thursday, January 02, 2003

SealSoft xFText v2.1

New in xFText v2.1: Small changes in template; xFText methods now available from other DLLs in a MultiDll program. New demo and install kit available.

Posted Thursday, January 02, 2003

RADVenture Referential Integrity Wizard

RADventure's Referential Integrity Wizard ($99) enables you to automatically create an application from a data-dictionary which checks if all parent-child relations defined in this dictionary are still intact.

Posted Thursday, January 02, 2003

MyODBC/MySQL Updates

Jorge Miranda reports that MyODBC 3.51.05 is out, and MySQL 4.06 too.

Posted Thursday, January 02, 2003

INN Bio For 24-Dec-2002

This week's Icetips News Network bio features Jolly Old Saint Nik! Well, the Clarion version, anyway. A former aircraft wing inspector and distillery VP, he certainly could qualify as a relative of St. Nick, now, couldn't he? He's worked in flying through the air (just like Mr. Claus), and certainly a distillery makes people happy (just like Mr. Claus). Well, OK, maybe a stretch, but it's still a great bio. And don't miss the picture at the end: the bio-ee, Tom Hebenstreit, Jim Kane and Andrew Guidroz, all looking their best. (Sue doesn't pick 'em, she

just prints 'em!)

Posted Thursday, January 02, 2003

Read/Write Excel Files

With Alexander Ageev's Excel utility you can read and write Excel files without having Excel on the system. No OLE/DDE used. Command line and manual modes are available. Includes examples for Clarion 5.5, and ABC and Legacy templates. Native XLS format supports Excel 97/200/XP.

Posted Thursday, January 02, 2003

XML Support

Alexander Ageev has released the XML Support product, which includes five base classes that allow you to: Read and parse any XML file or URL; Store data as it was read (as stream) or as it was structured (as objects in objects like a DOM); Create new structured XML document and edit data/structure of new or presence document; Write any data to file, create valid XML data structure; Analyze real document schema and generate source to write the same. The product includes ABC and Legacy templates, documentation, ABC and Legacy examples, and an XML analyzer utility.

Posted Thursday, January 02, 2003

Firebird 1.0.2 Update Released

Kelvin Chua reports that Firebird 1.0.2 is now available. This release represents the first maintenance update following the release of the v1.0 product, and includes the following bugs fixes (for all platforms): 64-bit file i/o is now properly supported under Linux; There was problem with connection strings on Unix platforms that could lead to database corruption; Table name aliases are now allowed in INSERT statements; String expression evaluation now throws an error if the expression could be greater than 64k. Previously an error was thrown if the expression evaluated to a possible size of greater than 32k; Minor problems with two-phase commit were fixed; INT64 datatype now supported correctly in arrays; SF Bug #545725 - internal connections to database now reported as user; SF Bug #538201 - crash with extract from null date; SF Bug #534208 - select failed when udf was returning blob; SF Bug #533915 - Deferred work trashed the contents of RDB$RUNTIME; SF Bug #526204 - GPRE Cobol Variable problems fixed.

Posted Thursday, January 02, 2003

OutlookFUSE 1.1.0 Demo

The OutlookFUSE v.1.1.0 demo is now available - this version of OutlookFUSE has the following new features: Complete support for Outlook XP events as Clarion derived class methods; Enhanced demo application demonstrating Plugware COM late binding interfaces to Powerpoint, Excel, and Access using ADO. Demo includes source code. The final release of OutlookFUSE v.1.1.0 with new documentation will be available the week after Christmas.

Posted Thursday, January 02, 2003

New TimeSavers Scheduler Beta And Picture Gallery

A new TimeSavers Scheduler beta is now available. Also on the site: a gallery of screen captures of programs using TimeSavers Scheduler. If you'd like to share your own work, please send screen captures and a brief description of the application it is used in to john@clarioncentral.com.

Posted Thursday, January 02, 2003

Reader Comments

Add a comment

Data Structures and Algorithms Part XIV - A Queue Is A Queue Is A Queue?

by Alison Neal

Published 2003-01-17

I promised in my previous article that I would discuss the "traditional" Queue data structure, and show how it worked. Before I get started it is probably wise to mention that the Clarion Queue structure does not behave in the same manner as the traditional Queue structure that C/C++ programmers work with. Fundamentally if you talk about a Queue structure to a C/C++ programmer you're talking about an entirely different animal, but without knowing more about the nuts and bolts of what's going on under the hood of Clarion, it is impossible for me to comment on the implementation differences.

The traditional Queue is a data structure that is very similar to the List and Stack, which I covered in earlier articles. The Queue is a chain of data nodes like the List. However, the Queue is the reverse of the Stack. With the Stack I only ever care about the last data element added to it, like a Stack of books on the floor, but with the Queue I am always most interested in the head.

The Queue reflects the Queue (or line) in nature. For instance, when you Queue (line up) at McDonalds, the first person in the Queue, or the element at the front of the Queue, is always the most important.

The queue

The Queue structure itself is very simple, and looks remarkably similar to the List and the Stack:

```
QNode       CLASS,TYPE
NodeVal     ULONG(0)
```

```
nextNode    &QNode
            END
head        &QNode
tail        &QNode
curr        &QNode
Qlen        LONG(0) !Keeps a count of the Q elements
```

The traditional Queue requires the following methods:

`Init`	Initialise the Queue
`Kill`	Dispose of the Queue
`Enqueue`	Add an element to the Queue
`Front`	Return the element at the front of the Queue.
`IsEmpty`	Check to see if the Queue is empty
`DeQueue`	Remove the head of the queue
`Length`	Return the Length of the Queue

The `Init` and `isEmpty` methods are the same for the Queue as they are for most of the simple structures:

The `Init` method just initialises the relevant parts, setting the head of the Queue and the tail of the Queue to `NULL` and the count of the Queue elements to zero:

```
myQueue.Init                  PROCEDURE()
   CODE
   SELF.head &= NULL
   SELF.tail &= NULL
   SELF.QLen = 0
```

The `isEmpty` method checks to see if the head of the Queue is currently `NULL` and if so returns `TRUE`:

```
myQueue.isEmpty                   PROCEDURE()
  CODE
  RETURN CHOOSE(SELF.head &= NULL, TRUE, FALSE)
```

The `EnQueue` and `DeQueue` methods are the most difficult of the Queue methods. This is my code for the `EnQueue` method, which adds a value to the Queue:

```
myQueue.enQueue                   PROCEDURE(ULONG YourVal)
  CODE
```

```
  SELF.Curr &= NEW(QNode)
? ASSERT(~SELF.Curr &= NULL)
  SELF.Curr.NodeVal = YourVal
  IF SELF.head &= NULL
    SELF.head &= SELF.Curr
  ELSE
    SELF.tail.nextNode &= SELF.Curr
  END
  SELF.tail &= SELF.Curr
  SELF.QLen += 1
```

Here's a quick example, adding the numbers 500, 60, and 657. First a call is made to the `enQueue` method passing the value 500. A new node is automatically allocated, and the value 500 assigned. Now if the head is currently `NULL`, which means that no elements have been added to the Queue previously, then the `Head` is made to equal the current `Node`. The `Tail` is also made to equal the current `Node`, and the Queue length is incremented by 1. Figure 1 shows the Queue structure at this stage.

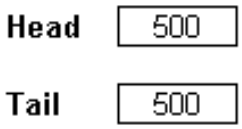


Figure 1.

When a call is made to enQueue passing the value 60, another new node is created, but as the head of the Queue is not `NULL`, this time the tail node is made to point at the new `Node`. So, at the end of the `IF` statement, both the tail and head look like Figure 2.

Figure 2.

After the `IF` statement, the `Tail` is then made to refer to the new `Node`, giving Figure 3.

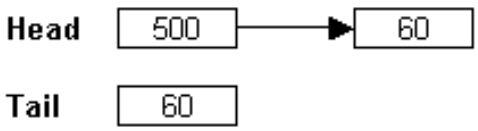


Figure 3.

The next number I want to add is 657, so with the call to the `enQueue` method a new node is created with the number 657 assigned to it. Again in the `IF` statement, the `Tail.nextNode` is made to refer to the new `Node`, giving Figure 4.

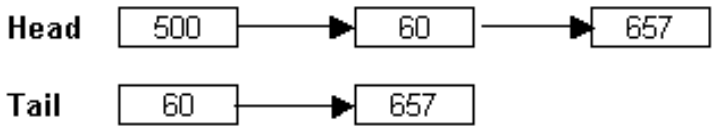


Figure 4.

Again, after the `IF` statement, the tail is made to refer to the newest `Node`, giving Figure 5.

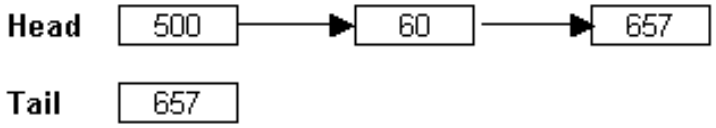


Figure 5.

At this point the ordering principle of the Queue becomes reasonably clear. The guiding principle is First-In-First-Out (FIFO), unlike the Stack which is Last-In-First-Out (LIFO).

Here is my code for the deQueue method:

```
myQueue.deQueue                  PROCEDURE()
V  ULONG(0)
  CODE
? ASSERT(~SELF.isEmpty())
  V = SELF.head.NodeVal
  SELF.Curr &= SELF.head
  SELF.head &= SELF.Curr.NextNode
  SELF.Qlen -= 1
  DISPOSE(SELF.Curr)
  RETURN V
```

Here's how the deQueue method works. To start, note that no value is passed to the method, as the Queue structure is such that it only cares about the head node, and so the assumption is automatically made that only the head node is to be deleted.

A copy of the value stored in the head node is assigned to the local variable V (500). SELF.Curr is made to reference the head node, which contains the value 500, and the head node is made to reference the next one in the chain (value 60). This frees the node containing 500 to be disposed, giving Figure 6.

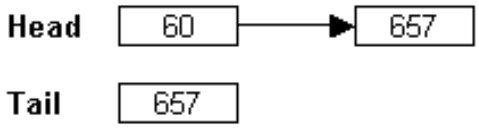


Figure 6.

The first node that I added (500) is now gone from the structure.

The other methods that I have included in my Queue implementation include the Front method and the Length method. The front method just returns the value of the head node:

```
myQueue.front                  PROCEDURE()
  CODE
? ASSERT(~SELF.isEmpty())
  RETURN SELF.head.nodeVal
```

The Length Method just returns the number of elements currently stored in the Queue:

```
myQueue.QLength                  PROCEDURE()
  CODE
  RETURN SELF.Qlen
```

The last of the methods is the Kill method, which just disposes of the memory allocated by calling the deQueue method:

```
myQueue.Kill                  PROCEDURE()
   CODE
   LOOP UNTIL SELF.isEmpty()
     SELF.deQueue()
   END
```

Summary

The in-built Clarion Queue structure provides infinitely more functionality than the traditional Queue structure used by C/C++ programmers, but without knowing the nuts and bolts of the Clarion Queue, it's extremely difficult to comment on its efficiencies in comparison to the traditional Queue and other available data structures.

The traditional Queue could be expanded to include such things as a Find method, however the Queue by nature places the stress on the head node, and so if I break pattern and start analyzing the data in the other nodes, I then question whether this structure is still really a Queue, or whether it has become a linked list instead. Also, the Queue suffers from the same efficiency overhead afflicting the Linked List. If I were to search the Queue for a value then I would have to visit every subsequent node until I found the value I was looking for. As I've

already discussed, Trees are far more efficient for searching, than the Queue, Stack or List.

In my next article I will continue on the theme of Queues, and introduce a derivation of the Queue called the Priority Queue.

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

Add a comment

Icons *In* List Box

by Steven Parker

Published 2003-01-17

In a customer file, I have customers with credit balances. When browsing the customer file, I want to highlight those customers. How do I do that?

On another browse, I need to highlight customers with past due balances. How do I do *that*?

In a check register, how do I show checks that have cleared or, in a student placement office app, highlight jobs a student has selected for printing?

Including a balance column or an aging column isn't quite obvious enough for the typical user, even if a contrasting color is set. Some other kind of visual indicator is necessary. I need some eye candy here because simply displaying the information in a browse column sometimes just isn't enough (yes, this is the voice of experience).

When Clarion for Windows was new, I used a column that displayed a character only when the item was selected. Figure 1 shows an X in the first column for two records selected for printing.

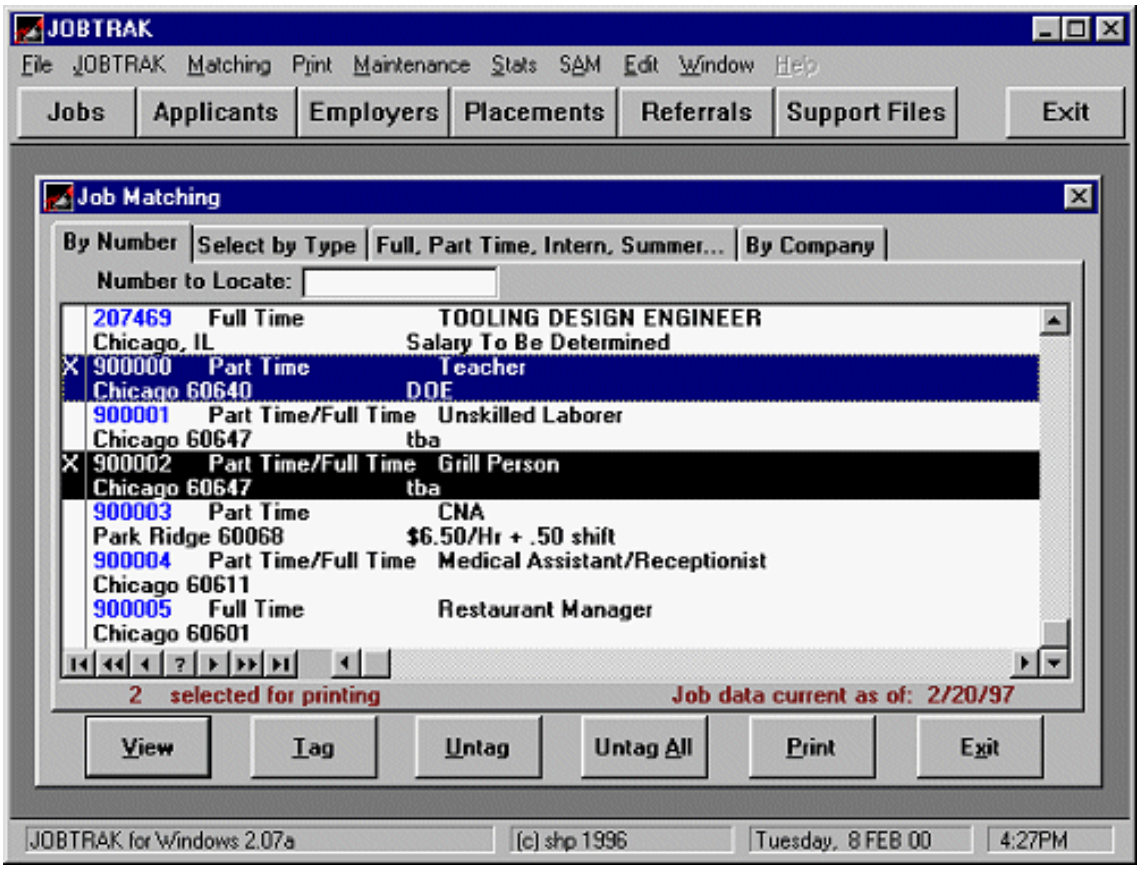


Figure 1. Highlighting selected item in early Clarion application

Later, icon support was added to list boxes and I did something with a bit more visual appeal by using an icon (read, "much harder for the end user to miss"):

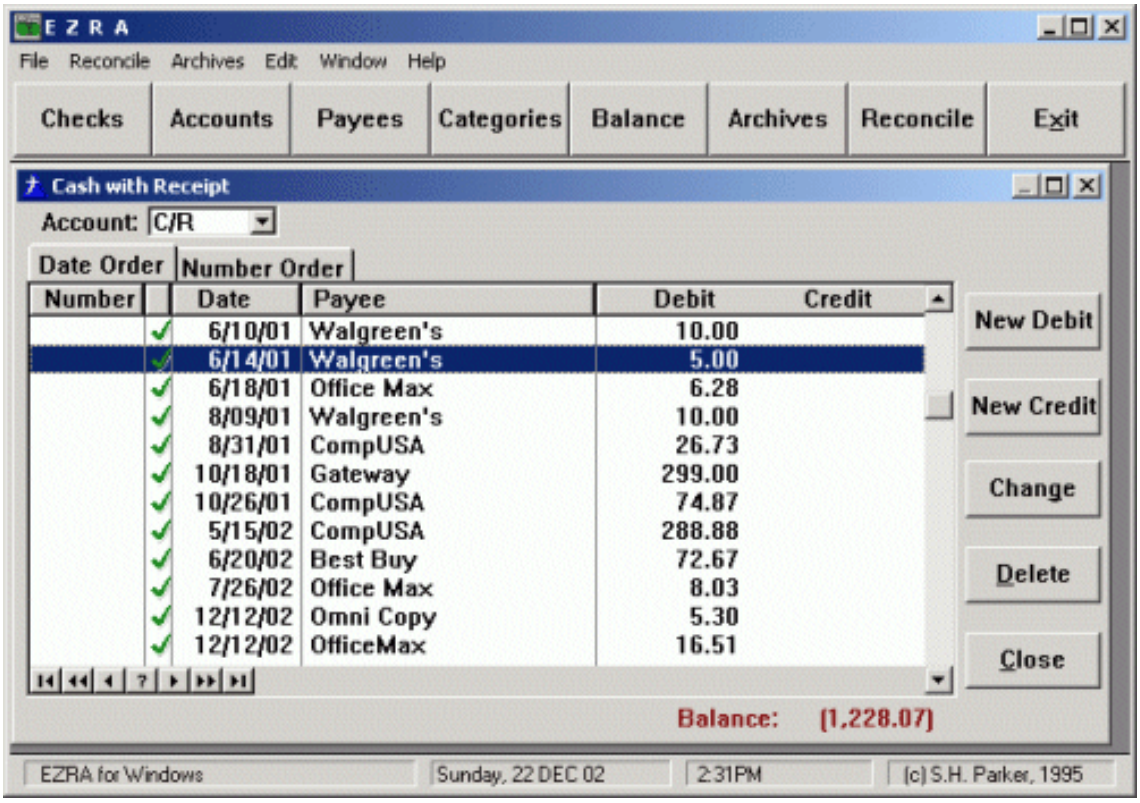


Figure 2. Icon in a list box

The question is how to apply techniques like icons to all these different cases. A credit balance might be a file field (`CUS:Balance`) or a runtime computed field. Aged balances are most likely to be in a file field, as is check status in a register. Jobs for printing are dynamically selected at runtime and one would expect this list to reset after printing or closing the browse (see List Box Marking).

File fields, computed fields, dynamically selected list rows … quite a mix of needs it would seem.

In fact, all of these cases have a very important commonality. In each case, there is a testable condition. If the condition is true, display something, otherwise, display nothing.

The existence of a testable condition suggests the following code to display an icon in a list box line:

```
Case CHK:Clear
Of 'Y'
  Clear_ = '~J:\CLARION\IMAGES\CHECK.ICO'
Else
  Clear_ = ''
End
```

or

```
Balance = CUS:PreviousBalance + CUS:Debits - CUS:Credits
If Balance > 0
 CrBal_ = ''
Else
 CrBal_ = '~J:\CLARION\IMAGES\PLUS.ICO'
End
```

These codelets would go in the **Format an element of the browse queue** embed (a/k/a **SetQueueRecord, Before parent call**).

While this works perfectly well (I have been using this code since CDD3), it almost seems like work. On second thought, it actually *looks* like work. One would certainly think that the conditional display of an image is something that the templates would support.

Indeed, this is the case.

List Box Icons the Easy Way

Right click on the list and select **List Box Format**. Then, highlight the field which is to display the icon. I usually use a local string field of one character for this purpose. In the list box, I set

its width at six or seven and delete the **Heading text**. The key to making the template do the work for you is on the `Appearance tab` where you enable icons:

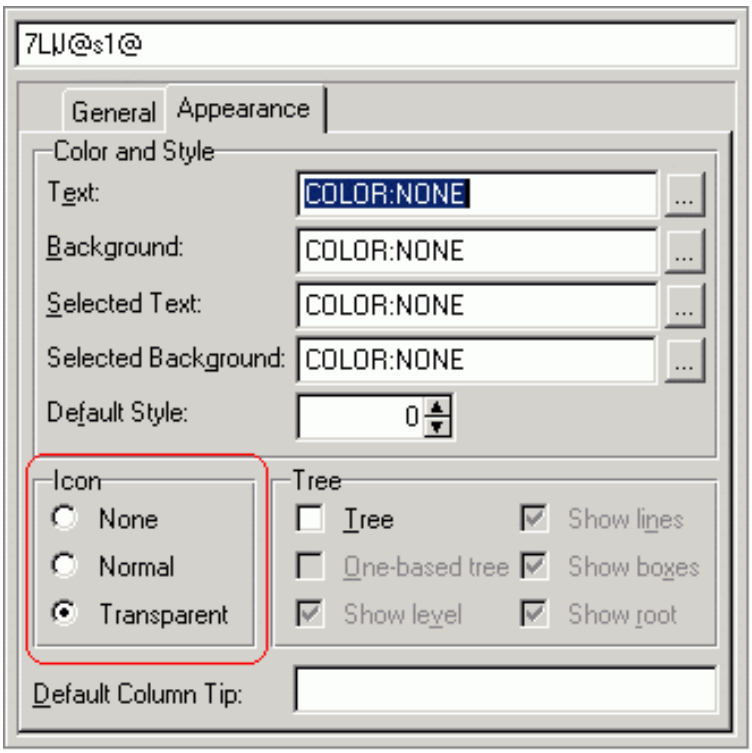


Figure 3. Setting up a list icon

Now the template knows that you want to be able to use an icon in this list box column.

Press "Ok" to save and right click the list box again. This time, select the **Actions** option. One of the tabs within **Actions** should be **Icons** and the list box columns which have icons enabled should appear in a list:

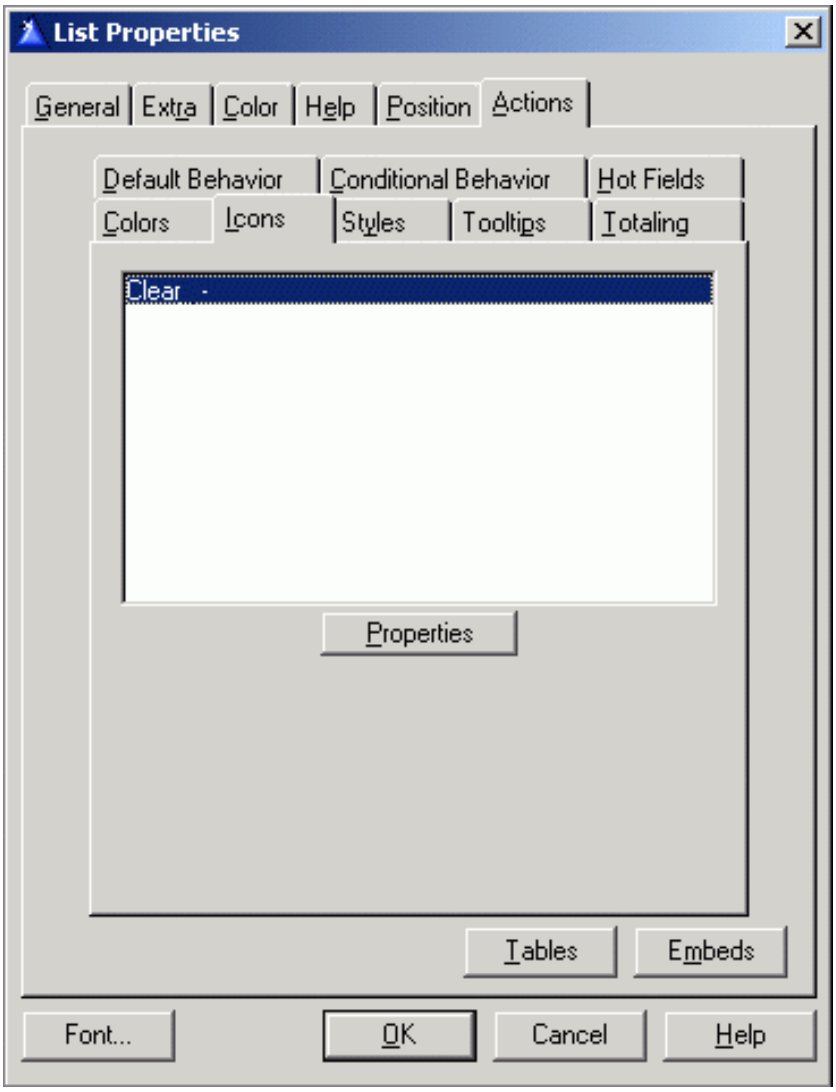


Figure 4. Columns with icons enabled

Press **Properties**. In the cases described, I want to use **Conditional Icon Usage**, because I want to change what is displayed based on a condition. So, press **Properties** at the bottom. Then simply create an expression and name the icon:

Figure 5. Conditional icon setup

Because the **Default Icon** is blank, nothing shows under "normal" conditions (i.e., when the condition entered is not met). But when, for example `CHE:Clear` is "`Y`," `check.ico` will be displayed.

I haven't actually tried it, but I see no reason why multiple conditions can't be used. So, I could have a single browse display one icon if the customer has a credit balance and another if the customer is past due.

The Legacy/Clarion templates generate the following code into the **Format an element of the browse queue** embed:

```
BRW1::Clear_ = Clear_
IF (Che:Clear = 'Y')
  BRW1::Clear_:Icon = 1
ELSE
  BRW1::Clear_:Icon = 0
END
```

ABC generates similar code into the **SetQueueRecord** embed:

```
IF (Che:Clear = 'Y')
  SELF.Q.Clear__Icon = 1
ELSE
  SELF.Q.Clear__Icon = 0
END
```

The ABC code looks very much like the code I originally embedded manually. That is, the templates generate essentially the same code I used to type. Might as well let the template do the typing, eh?

Summary

Many things are difficult to do, even in Clarion. Displaying an icon in a list box or displaying an icon when the record meets certain conditions is not one of them. The "hard" part is finding the relevant template prompts and, now, not even that is hard. Finding suitable icons is another matter…

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

Debug De Program With Debuger

by Skip Williams

Published 2003-01-23

In all of my years of writing programs, I have constantly fought with bugs and ways to detect and eliminate them. Although Clarion has an adequate debugger, I usually don't use it, probably because I have never taken the time to get really comfortable with it.

What I wanted was a way to selectively watch what was happening in the program as it ran. I used `stop()` or `message()` statements, or I would create and display a debug queue in a list box as the program executed. All of these methods were cumbersome and tended to change things within the program (e.g. errorcodes). I then wrote a `Debug` procedure that I included in every program, and that took the same form as the `message()` statement. `Debug` would DDE over to a client program that displayed my debug information in a list box. Now, with well-placed debug statements, I could really watch the execution of the program as it happened as well as see the variables that I was interested in. It worked, but was somewhat slow, and again, the DDE commands could change what was really going on in the program.

There had to be another way.

Then I ran across DebugView from SysInternals. Here was a viewer that captured and displayed system-wide debug messages that were sent to the system by any program using the API call `OutputDebugString()`. DebugView also had a lot of options that really intrigued me... but more on those later. Now that I had a way to externally view the debug messages, I needed a way to easily create them in my applications during the debugging process.

To create the debug messages, I wrote a class – called `Debuger` (yes, I know that the name is missing one "g") – that I could use in any application, hand-coded or created with the

Application Generator, that would call `OutputDebugString()` to send my debug information to DebugView. I prefer `OutputDebugString()` over everything else that I have tried. It seems to be very fast and, if there is no viewer to see the message, nothing bad happens – the message is just disregarded.

Being an inveterate tinkerer, I kept playing with the class to include various options and functions, including a way to break the program and evoke the Clarion debugger from within the source code. I'll explain that in a little while, but first, let me describe the `Debuger` class and its various options using hand-coded examples. I have also included a very simple control template in the source download that will make the class easier to use in AppGen. It will do the include, init, and cleanup for you.

db.INIT

To instantiate the `Debuger` class, I use an include, and a declaration in the data section or in the **After Global Includes** embed.

```
  Include('debuger.inc')
db Debuger
```

I initialize the class after the `Code` statement or in the **Program Startup** embed, using this code:

```
db.init('ProgramName')
```

The `'ProgramName'` string is simply passed to the external DebugView program so that you will have an indication of which application is sending the debug output. This string can be anything you want, if you use the control template in appgen, the template automatically uses the application name.

There are two other optional parameters on the init that you may want to use as well. If the second parameter is set to `TRUE` (the default is `FALSE`) debug messages will be sent regardless of the compiler debug options. If omitted or `FALSE`, any debug messages are sent only if the debug option in the app's project is set to something other than 'off'. This allows you to put debug messages throughout the program and turn them all off when it comes time to ship.

I write a lot of one-shots and quick'n dirtys for file conversions and the like. In those programs, I set this parameter to true and compile with debug off for better execution speed. I use `Debuger` to display information in DebugView instead of opening a window, doing an `Accept` loop, etc just to get information on how the program is doing during its execution.

Also, if you launch the program with **/Debuger** on the command line, `Debuger` will be turned on regardless of the application's compile options. If you get in a bind in the field and need to see `Debuger` output without recompiling, this is the key.

Sometimes programs will get into a loop, and I would like to stop within the loop and see what is happening. The third parameter on the init indicates how many duplicate messages in a row Debuger will allow before issuing a `message()` statement with **Continue**, **Halt**, and **Debug** buttons. The default is 50. If set to zero, no checking for duplicate messages takes place.

```
loop i# = 1 to forever
  db.debugout('This is a duplicate message')
end
```

In addition to each entry going to `OutputDebugString()`, after every 50 iterations of the loop, you will get a messagebox saying '**This is a duplicate message**' where you can decide if you want to continue or not.

db.Debugout

Now that you have Debuger class initialized, it's time to use it. The main procedure to simply get debug output to the viewer is `Debugout`

```
Debugout Procedure(string stuff,<string header>,<string flag>)
```

And this is how you would typically use it.

```
Db.debugout('message text','message header')
```

Yes, I know the parameters seem to be backward, but I prototyped them that way to emulate the `message()` function where the title of the message box is the second parameter. `Debugout` formats the output as 'myprogram – Message header – Message text'. The second parameter, as in `message()`, may be omitted.

The third parameter is optional. If set to `TRUE`, then, in addition to the debug message going to the debug viewer, a `message('Message text','Message header')` is also produced with the option to **Continue**, **Halt**, or **Debug**.

```
If glo.snafu > 999 then |
  db.debugout('I can't go on','I have reached my limit',true).
```

The fourth parameter is also optional, and if set to `TRUE` will force that particular debug message to be produced regardless of compile debug option or init option.

```
If errorcode() = msg.fubar then |
```

```
db.debugout(error(),|
 'I have fallen and I cant get up',true,true).
```

If `errorcode()` does indeed equal `msg.fubar` , the `db.debugout()` will always cause a debug message to be sent (since the fourth parameter is true) and a `message()` to appear (since the third parameter is also true) .

I use `debugout()` to display the contents of variables, errorcodes, and sometimes just to indicate that a particular piece of code is being executed – `db.debugout('here')`. I have also been known to put `debugout('Enter/Exit MyProcedure')` statements at the entrance and exit to procedures and routines to be able to trace execution through them as well.

In summary, put the `debugout()` statements in those areas of the program where you are having trouble, display critical variables or errors, or just to watch your program flow.

db.DebugBreak

The last method is `DebugBreak` This causes the application to break and the default debugger (usually Clarion) to be invoked.

```
If errorcode() = msg.FileDroppedOffEndOfEarth|
 then db.debugbreak.
```

This helps if you want to immediately invoke the debugger and see all of your variables. Note that since `debugbreak()` is issued within the `Debuger` class, the statement pointed to by the debugger will not necessarily indicate where you were when you broke the program.

```
db.Kill
```

Finally, the `db.kill` should be issued at the end of the program, or in the 'Program Ended' embed. This produces a debug message indicating that the program has ended.

DebugView

Now that you are producing debug messages from your application, it is time to take a look at the viewer. Several are available, but the one that I like best is DebugView from www.sysinternals.com. They also have a variety of other neat utilities that are worth taking a look at as well.

DebugView has a very small footprint and comes with a help file that explains all of the various options. I just want to highlight a few here.

I suggest you put DebugView in your startup menu on your development machine so that it always available. It requires no setup and is only about 16k so you can also easily run it from

diskette at a client location.

Since DebugView displays all debug messages system wide, you may get some really strange stuff showing up in the viewer. Some other programs (like VNC) produce debug information just because they can. This can be very distracting, but DebugView has a filtering process to let you filter out the noise and only display debug output from your program. The Debuger class appends each of its debug messages with a backquote ` (the quote under the tilde ~). I set up DebugView to ignore any message that doesn't have this quote as the first character (see Figure 1).

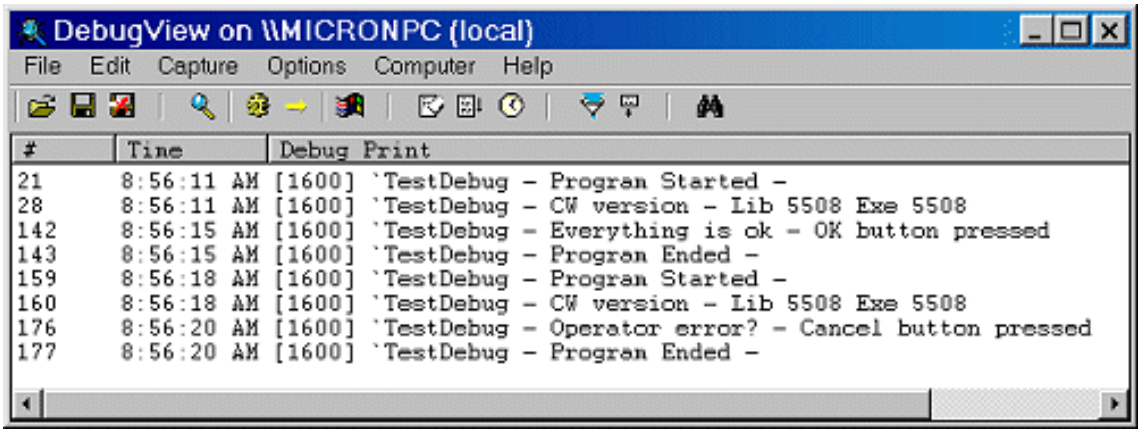


Figure 1. DebugView set to only display messages beginning with `

You can also highlight debug messages in color. For example, you might want to highlight all debug messages in red that contain the word 'Fatal'.

You can save the debug messages to a file, and can add comments into the debug messages as well.

If you start DebugView on a machine that is reachable via TCP/IP with a **/c** command line option, you can, using your local DebugView, capture and view in real time messages from that remote PC just by pointing DebugView to its IP address or domain name.

Summary

The Debug class is not a silver bullet to kill all of your program bugs, but it does give you an easy way to place debug statements in your app to view variables, program conditions, or just notes to yourself as the program executes. It requires very little overhead and can be turned on or off easily.

The main effort is deciding where to place the `Debugout()` statements and what information to display that will help you in the debugging process.

Download the source, register the template, download and install DebugView from www.sysinternals.com and give it a try on the sample app. I think that you will find that it will save you many hours of time when debugging an application.

Download the source

Skip Williams' first job was programming mainframes for a bank. He did that for 10+ years then became a VP of Technical Services and moved into a management job. After several bank mergers, Skip became a programmer again and helped start Global Trade Information Services in Columbia, SC, using Clarion programming tools. Skip has been using Clarion since 1986, when he was persuaded by Bruce Barrington, at Comdex, to try this new language Bruce had developed.

Reader Comments

Add a comment

Topics

A Calculator Class And Template

by Nardus Swanevelder

Published 2003-01-24

In a previous article I wrote about how to implement the Page-of-Pages functionality in a report by adding on to the Clarion ABC classes. In this article I will look at how to take the source code for a Clarion 4 calculator, change it into a class, and then create a template to use this calculator as a control lookup button in an application. I could have changed the calculator into a template only, but Clarion source code is easier to maintain when it's not mixed in with template source.

I already had a calculator in my applications, but it didn't look like the Windows calculator, and I didn't want to rewrite mine completely. Thus I began to search for a cheap calculator. There are some good ones to buy, but once again being from South Africa I had to look for a free calculator for cost reasons. This search turned out to be more or less fruitless, but eventually I did find an older calculator in a file called CalcC4.zip, put into the public domain by Castle Computer Technologies. This calculator had most of the basic features that was required, but as I wanted a class, and I wanted some additional features, I began coding.

The first step was to create an app where the calculator could be tested and refined. The second step was to take the corrected calculator code and change it into a class, and the third step was to write a template to be able to use the class in the application with no extra coding.

Step 1 – Changing the calculator code

The calculator I found was written in Clarion for Windows version 4. My idea was to make this calculator resemble the XP calculator, and also have the same features as the XP calculator. The C4 calculator contained some features that were not necessary and it also lacked some other features. To test the code I created a small C5.5 app and added a source procedure to call the calculator. I began removing unnecessary features and adding new

features.

I found two bugs during the testing process. The first bug caused a problem when a user would press the memory recall button after the divide operator button was pressed. The memory recall would not recall any amount, and the calculation could not be completed. Please note that my idea here is not to explain the working of the calculator code, but to explain the development process. But to show how easy it was to fix this problem, here's the code I added to the first accept loop.

```
OF ?Recall
  Number = Memory
  Cycle
```

The second bug had to do with inputting numerical data after the decimal point. I fixed this by changing the variable `Number` from `Real` to `String(15)`.

For each of the new features I added a button and some code:

```
!Copy to clipboard
OF ?ClipB
   setclipboard(Number)
   Cycle
!Memory Clear
Of ?MClear
   Memory = 0
   Number = 0
   NewNumber = 0
   DecimalPoint = False
!Backspace (Delete the last entered key)
OF ?BSKey
   Len# = LEN(CLIP(Number))
   IF SUB(Clip(Number),Len#,1) = '.'
      DecimalPoint = False
   End
   Number = SUB(Clip(Number),1,Len#-1)
   IF Number = 0
      Number = '0'
      DecimalPoint = False
   END
   Cycle
!CE (Clear the value entered after the last
!    operator was pressed)
OF ?EClear
   Number = 0
   NewNumber = False
   DecimalPoint = False
   Cycle
```

Step 2 – Change the calculator code into a class

To change the code into a class I used the free ObjectWriter application from Capesoft. I will not explain the process to change the code into a class as the documentation from Capesoft on Object writer is clear on this process.

Here's the **ProCalc.Inc** file as created by ObjectWriter:

```
!  omit('***',_ABCLinkMode_=1)
!_ABCDllMode_  EQUATE(0)
!_ABCLinkMode_ EQUATE(1)
!  ***
OMIT('_EndOfInclude_',_ProCalc_)
_ProCalc_ EQUATE(1)
! Generated by CapeSoft's Object Writer
!------------------------------------------------------------
!Class ProCalc
!
!------------------------------------------------------------
ProCalc              Class(),Type,Module('ProCalc.Clw')
! Properties
ReturnNumber           REAL
! Methods
Init                   PROCEDURE
                     END ! Class Definition
_EndOfInclude_
```

The `Omit`, `Equate`, and `_EndOfInclude_` lines are simply instructions which prevent the .INC file from being included incorrectly.

You have to declare the new class, starting with the name of the new class (`ProCalc`). Because there is no parent class the name of the parent is empty – `Class()` . After that follow some compiler directives.

It is also possible to declare data that will be available to the class as a whole, hence the variable `ReturnNumber`. This number is used to return the result of the Init procedure back to the calling procedure.

The next file you have to look at is **ProCalc.Clw**. Due to the length of this file I have not quoted all of the code:

```
 Member()
_ABCDllMode_  EQUATE(0)
_ABCLinkMode_ EQUATE(1)
      Map
      End ! map
      Include('Calc.inc')
!------------------------------------
ProCalc.Init            PROCEDURE
Number     STRING(15)
Operand    REAL
```

```
Memory     REAL
Operation  REAL
NewNumber  BYTE
DecimalPoint    BYTE
Digit      BYTE
CalculatorS WINDOW('Calculator'),AT(80,7,127,119),IMM, |
             ICON('calc.ico'), WALLPAPER('Clouds.wmf'),!
             TILED,TIMER(100),SYSTEM,GRAY,RESIZE,AUTO
...
END     !End Window
  CODE
  OPEN(CalculatorS)
  NewNumber = True
  DecimalPoint = False
  Operation = ?Equal
  ACCEPT
  ...
END                          !End ACCEPT
  Self.ReturnNumber = Number
  CLOSE(CalculatorS)
```

Note that all methods that were defined in the .INC file have to be coded in the .CLW file. In this case I am only showing the `Init` method.

When you look at the second last line you will see the following instruction `Self.ReturnNumber = Number`. This is where the procedure returns the calculator result.

Step 3 – Writing a template to use the class as a control lookup

My idea was to create a control lookup template that could be dropped on a form, that would call the calculator and then return the result to a variable. I ended up creating three templates: a global extension template, a local extension template, and a control lookup extension template.

The purpose of the global extension template is to add the **Procalc.inc** and **Procalc.clw** file to the application, as well as control the compilation of the calculator code (you can instruct this template to either include the code for the calculator or exclude the code from your app.) The local extension template instantiates the calculator class in the local procedure. And the a control template that executes the Init procedure and returns the value to the local variable.

Here's a detailed look at each of these templates.

Global Extension Template

This template organizes the inclusion of the calculator in the application. The first variable that is used, `%UseCalculator`, determines if the calculator code will be included in the application. The next variable contains the local variable that will be used for instantiating the

calculator class in each procedure. This will default to `LCLCalc` but can be changed to anything you like.

```
#TEMPLATE(ProVectus,'ProVectus Calculator'),FAMILY('ABC')
#!-------------------------------------------------------
#! (c) 2002 By
#! ProVectus Software
#! All Rights Reserved World Wide
#!
#!
#! This is for free and may not be sold
#!-------------------------------------------------------
#!-------------------------------------------------------
#! Global Extension Template
#!-------------------------------------------------------
#EXTENSION(ProVectusGlobals,'ProVectus Globals'),APPLICATION
#SHEET
  #TAB('ProCalculator')
    #BOXED(''),Section,AT(,65,,)
    #PROMPT('Include Calculator Objects',CHECK),%UseCalculator,AT(10)
    #ENABLE(%UseCalculator=%TRUE)
          #PROMPT('Local Variable Name',@s20),%LCLCalc,Default('LCalc')
        #ENDENABLE
    #ENDBOXED
  #ENDTAB
#ENDSHEET
#!-------------------------------------------------------
#AT(%CustomGlobalDeclarations)
  #IF (%UseCalculator = %TRUE)
    #Project('ProCalc.clw')
  #ENDIF
#ENDAT
#!-------------------------------------------------------
#AT(%AfterGlobalIncludes)
  #IF (%UseCalculator = %TRUE)
  include('ProCalc.inc')
  #ENDIF
#ENDAT
#!-------------------------------------------------------
```

Local extension template

This template instantiates the calculator class in the local procedure by adding the local variable defined in the global extension template to the local procedure.

```
#!-------------------------------------------------------
#! Local Extension Template
#!-------------------------------------------------------
#EXTENSION(ProVectusCalc,'ProVectus Local Declarations')↵
   ,PROCEDURE,REQ(ProVectusGlobals)
#SHEET
  #TAB('General')
    #BOXED(''),section
```

```
    #Display('This template add Class to local Procedure')
    #ENDBOXED
  #ENDTAB
#ENDSHEET
#AT(%DataSection),PRIORITY(4000)
%LCLCalc    ProCalc
#ENDAT
```

Control Lookup extension template

I used the Clarion 5.5 "Trigger an Entry Control Lookup" template in the ABC classes as an example to develop this specific template.

This template gives you the ability to add a button to a screen which will bring up the calculator when the user clicks on it. The template then assigns the result from the calculation to a defined screen variable.

After you populate the control lookup template on your screen you have to specify the "Control with Lookup" variable on the Actions tab.

When the user clicks on the button the calculator class method `Init` is executed. The following statement assigns the result to the control variable: `%ControlUse = %LCLCalc.ReturnNumber`. `ControlUse` is the variable you specified at the **Control with Lookup** prompt.

The following statement in this template controls the availability of this control template under the **Populate, Control Template** menu in the screen formatter:
`%ControlInstance=%ActiveTemplateInstance`

```
#!------------------------------------------------------
#! Control Lookup Extension Template
#!------------------------------------------------------
#CONTROL(ProCalculatorLookup,'ProVectus Popup Calculator-Control')↵
   ,DESCRIPTION('(ProVectus) Popup Calculator'),MULTI,HLP('')↵
   ,REQ(ProVectusGlobals)
    CONTROLS
         BUTTON('...'),AT(,,12,12),USE(?PopCalculator)
     END
#BOXED('Field Lookup Button Prompts')
  #PROMPT('Control with lookup:',CONTROL),%ProControlToLookup
#ENDBOXED
#ATSTART
 #DECLARE(%ProLookupControl)
  #FOR(%Control),WHERE(%ControlInstance=%ActiveTemplateInstance)
    #SET(%ProLookupControl,%Control)
  #ENDFOR
  #FIX(%Control,%ProControlToLookup)
  #IF(%ControlType<>'ENTRY')
    #ERROR(%Procedure & ↵
```

```
      'Error: File Lookup needs to refer to Entry Control')
  #ENDIF
#ENDAT
#AT(%ControlEventHandling,%ProLookupControl,'Accepted')
    #FIX(%Control,%ProControlToLookup)
    #IF (%UseCalculator = %True)
    %LCLCalc.Init()
    %ControlUse = %LCLCalc.ReturnNumber
    Display(%ProControlToLookup)
    #ENDIF
#ENDAT
```

Implementation

How do you implement this in your app? Just follow these directions:

- Copy the **Procalc.clw** file and the **Procalc.inc** file to the Clarion **libsrc** or **3rdParty\libsrc** folder.
- Copy the **Procalc.tpl** file to your Clarion **template** or **3rdParty\template** folder and register the template with Clarion.
- Open the **Clubmgr app** and add the **ProVectus** global extension,
- Tick the **Include the calculator objects** checkbox
- Add the local **Provectus** extension to the `UpdateTransactions` procedure. There are no prompts to be completed
- Click on the **Populate** menu and then choose **Control Template**. Populate this on the form, right click on the button and complete the **Control with lookup** prompt, specifying the control to receive the total.
- Compile the app and test

In summary, I took a Clarion 4 calculator source procedure, changed the look and feel as well as the functionality, then changed this source into a class (with the help of Capesoft's Object Writer) and lastly developed three templates to assist in the use of the class in an application. The advantage of this class and template set is that you only have to change code in one place if you want to change the appearance or functionality of your calculator.

Download the source

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

You don't mention anything about using this in a multi-APP...
I have developed it in a multi-dll app and only tested it...

Book Review: Programming PHP

by David Harms

Published 2003-01-24

Programming PHP

By Rasmus Lerdorf, Kevin Tatroe
Published by O'Reilly, March 2002
1-56592-610-2,
524 pages, $39.95 US, $61.95 CA

PHP, an open source scripting language, has become hugely popular for web site development. It has also started showing up on the radar screen of Clarion developers. Clarion Magazine has published several PHP-related articles, and of course Clarion/PHP (a product similar to Clarion/ASP) is somewhere on the horizon.

So what does PHP do? According to the authors of O'Reilly's Programming PHP, you can use PHP three different ways: for server-side scripting; for command-line scripting; and for GUI applications (using the PHP-GTK toolkit). For the most part, however, PHP is used for server-side scripting, and the book's content reflects this focus, although it does provide detail on just about every aspect of the language. This is as you'd expect, since one of the authors is Rasmus Lerdorf, who first conceived PHP in 1994.

The book begins with a solid grounding in PHP language basics. PHP is an easy language to learn, but you'll want to pay special attention to things like the subtleties of variable handling

(a PHP variable can, for instance, easily be assigned a different data type during program execution), operators (in addition to the = operator, there is an equality operator (==) and an identical operator (===), and memory allocation, specifically the “copy on write” functionality that doesn’t allocate memory for copied variables until the content of the copied variable is changed.

Besides language basics, the book also dedicates chapters to function prototyping, string handling (including several styles of regular expressions), arrays, and objects. Yes, like Clarion, PHP is a hybrid language, with procedural and object-oriented capabilities. And PHP adds a few extensions that Clarion doesn’t yet have, including introspection, which is the ability to look, using code, at an otherwise unknown class and determine its methods and properties, and serialization, which converts a class to a byte stream for storage and, later, recovery.

With these chapters under your belt, you’ll be well on your way to writing some useful PHP code. And that’s where the rest of the book takes you.

The chapter titled Web Techniques is one of the most concise treatments of forms and session handling you’re likely to encounter. Somehow the authors find space to include topics like HTTP authentication and file uploads, a tribute in part to how easy many of these tasks are in PHP. Still, if you’re brand new to web development, you might want to look into additional reading material – some of these topics could easily be chapters in themselves.

Similarly the following chapter on database access will not, as the authors warn, tell you everything you need to know about creating web database applications. This chapter focuses on the PEAR DB abstraction layer (included with PHP), and includes topics such as prepared statements and sequences (a.k.a. autonumbered primary keys). The sample application section shows how to create a three related tables, connect to the database (MySQL, PostgreSQL, or Oracle 8i were tested) from PHP, and update the three tables. This is also one of the more visually appealing web applications I’ve seen in a programming book.

The chapter on graphics explains how to generate images at runtime, using the GD extension. GD is not included with PHP but this open source product is readily available. Drawing with GD is a bit less complicated than, say, using the GDI, since you don’t have to worry about tricks like double buffering – all images are drawn completely before being sent to the client.

The PDF chapter covers the use of the pdflib extension, which is not open source, but free for personal and non-commercial usage. The examples show how to create text and images, patterns, templates, bookmarks, thumbnails, links and more. The XML chapter shows how to create and read XML files from PHP. As with the database and web techniques chapters, you’ll want to look elsewhere for detailed background information – XML is a vast topic in its

own right. This chapter also shows how to create RPC and SOAP servers with PHP.

The remaining chapters cover security issues (a short treatment), tips and tricks, performance tuning, creating PHP extensions, and the ins and outs of running PHP on Windows.

The preface states that the book assumes readers have a working knowledge of HTML – a few programming fundamentals wouldn't hurt either, as the authors waste little time diving into the code. For Clarion developers with that basic HTML knowledge, and wanting a leg up on PHP, it's hard to imagine a better book.

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

For those interested in looking at free or commerical PHP...

Data Structures and Algorithms Part XV - Priority Q

by Alison Neal

Published 2003-01-30

In my previous article, I looked at the Queue structure as used by C/C++ programmers. These Queues are First In First Out (FIFO) structures, and are not quite the same animal as a Clarion QUEUE.

In this article I'll expand on the subject of traditional Queues, and discuss the Priority Queue. The main use of the Priority Queue Abstract Data Type (ADT) is for scheduling. Think about your PC for a minute, and how it knows in which order processes should be performed. Your PC is fundamentally using a form of Priority Queue, where each element in the Queue has a priority assigned. Your PC will process those tasks with the highest priority first (priority 3 before priority 1 tasks), and then where two or more tasks have the same highest priority, it should ideally process the task, with that priority, that has been waiting in the Queue the longest.

Sounds simple enough, doesn't it?

Like all abstract concepts, though, the implementation of this data type is left completely in the programmer's hands. Thus, the Priority Queue can be implemented in numerous ways.

PRORITY	TASKS		
	1	2	3
	GPF Save File	GPF	GPF Print Compile

Figure 1. A Priority Queue

Looking at Figure 1 it would be easy to see where both multi-dimensional arrays and the Clarion Queue structure could build an adequate Priority Queue. The drawback with using an array is that you are generally limited to a pre-set size within Clarion. This is fine if you know the number of elements you will be holding in the Priority Queue from the beginning. Dynamic arrays can be achieved using STRINGS as arrays, and creating a NEW STRING each time the structure needs to grow. To put it another way, a NEW STRING of a larger size can be created to store the current records and any new records. This has the added overhead of having to copy data in and out of memory, with each growth, and maintaining an index of each record, as a record may fill more than one string position. Also, if the array is to be maintained in any sort of order, the reshuffling process that inserts introduce is an added overhead. The array implementation is not my preferred method in Clarion for illustrative purposes.

Another option is the Clarion QUEUE structure; it is possible to create a Priority Queue from a Clarion QUEUE by storing the task and the Priority, then sorting the QUEUE by the Priority value as required. However the SORT method in Clarion, while guaranteeing that the first record out will be the highest priority, in no way guarantees that the first record out will also be the oldest of that priority. This opens the way for a task to sit in the QUEUE and never get processed, because of the way that the SORT method works.

Using the Clarion QUEUE would also have the added overhead of constant sorting, which can be slow where there are a lot of records (although Clarion QUEUEs are really quite fast). The other option would be to loop through the QUEUE, looking for the first record with the highest priority. This is also a slow alternative, as it would mean visiting every record in the QUEUE every time the next Task was required.

A work around could also be to store the record number, and then SORT the QUEUE by both priority and record number, which you would need to maintain in code. However it would guarantee that the first record out would be the oldest record for that priority.

```
Pqueue    Queue
Task        STRING(80)
Priority    BYTE(0)
RecNo       ULONG(0)
          END
SORT(Pqueue,-pQueue.Priority,pQueue.recNo)
```

Prior to adding the QUEUE record you could assign RECORDS(Pqueue) + 1 to Pqueue.RecNo (assuming that no records are ever deleted) or maintain a continuous count of the number of records added. If there are only a small number of records being used then either a static array or the Clarion QUEUE implementations would be more than sufficient.

For my implementation of the Priority Queue I have elected to use an Ordered Linked List, which may not be ideal either, but it does not have the overhead of dynamic arrays, or the

overhead of having to visit every single record in the structure, when wanting to look at the highest prioritised task. This implementation also doesn't require the constant sorting that both the Clarion QUEUE and array implementations could require, and it is one of the simpler methods for illustrative purposes. However, it does make inserts slower, as the correct position for the new task has to be located from the start.

The Priority Queue

The Ordered List Structure itself is similar to the List I covered in an earlier article, except that it now includes both a variable for the priority and for the data value (task) being stored:

```
PQNode      CLASS,TYPE
priority      BYTE
NodeVal       ULONG
NextNode      &PQNode
            END
root        &PQNode
curr        &PQNode
```

The methods required for the Priority Queue are, however, fewer in number than for a List:

Init	Initialise the Queue
Kill	Dispose of the Queue
InsertQ	Add an element to the Queue
RemQ	Remove the head of the Queue and return the data value stored there.
IsEmpty	Check to see if the Queue is empty

The Init and isEmpty methods are the same for the Priority Queue as they are for most of the simple structures: the Init method merely initialises the root to NULL, and the isEmpty method checks to see whether the root is currently NULL.

The insertQ method is a little more complicated, but doesn't really introduce anything new:

```
PQueue.insertQ    PROCEDURE(BYTE YourPri, ULONG YourVal)
  CODE
  SELF.root &= SELF.insQ(SELF.root,YourPri,YourVal)

PQueue.insQ   PROCEDURE(*QNode t,BYTE YourPri, ULONG YourVal)
  CODE
  IF ~t &= NULL
    IF t.Priority >= YourPri
```

```
      t.nextNode &= SELF.insQ(t.nextNode,YourPri,YourVal)
      RETURN t
    END
  END
  SELF.Curr &= NEW(PQNode)
? ASSERT(~ERRORCODE())
  SELF.Curr.Priority = YourPri
  SELF.Curr.NodeVal = YourVal
  SELF.Curr.nextNode &= t
  RETURN SELF.Curr
```

The `insertQ` method is originally passed the priority value and the data value to be stored. The recursive `insQ` method is then called passing the root `NULL`. In the first call the root should always be `NULL` so a new node will be allocated and the relevant values are assigned.

With the insertion of the second element, the root will not be `NULL`. If the new priority is less or equal to that currently stored, then a recursive call is made and a new node is allocated as the second link in the chain. If the priority were higher then a new `Node` would be allocated, the values assigned, and the old root node would become the second link in the chain.

The `RemQ` method is also fairly simple, as the only `Node` it will ever remove is the one with the highest priority, which is the `Node` that will always be occupying the root (or head).

```
PQueue.remQ         PROCEDURE()
tmpVal  ULONG(0)
  CODE
? ASSERT(~SELF.root &= NULL)
  tmpVal = SELF.root.NodeVal
  SELF.Curr &= SELF.root
  SELF.root &= SELF.root.nextNode
  DISPOSE(SELF.Curr)
  RETURN(tmpVal)
```

Basically, as long as the root is not `NULL` the process should take a copy of the current root node and the data value contained therein, assign the second link in the chain to the root or head position, dispose of the old root node, and return a copy of the data value that it held.

The `Kill` method just makes an iterative call to the remQ method until the head or root node is `NULL`:

```
PQueue.Kill                 PROCEDURE()
  CODE
  LOOP UNTIL SELF.root &= NULL
    SELF.remQ()
  END
```

Summary

There are a myriad of ways in which the Priority Queue can be implemented; some will be more efficient than others, and what is the most efficient will depend on the intended application. For example if the tasks are being completed as rapidly as they are being added to the Priority Queue, then my list implementation may not be a good option, and the Clarion Queue may be the better option, particularly if the Queue is going to be entirely empty at times. If the Priority Queue is frequently empty then there is no real fear of tasks being left to wait forever.

Having now discussed the Trie structure and the Priority Queue, in my next article I will build on this information and discuss Huffman's compression algorithm, which is used in PKZip, JPEG and MPEG compression.

Download the source

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

Add a comment

Getting Dynamic With Report Writer

by Steven Parker

Published 2003-01-30

This action packed episode picks up where Ben Brady ("Integrating Clarion Report Writer Into Your Applications") leaves off. Ben showed how to integrate the Report Writer engine into standard Clarion apps. I want to add more flexibility to the process of using Report Writer from an app.

Filters

Report Writer has always had the ability to filter reports dynamically. Report Writer has also always featured the capability to create "runtime" fields in the report and then to use them in the report filter.

If I select **Control | Data | Runtime Fields** and press the **Add Runtime Field** button, as Figure 1 shows, I can create what amounts to a local variable.

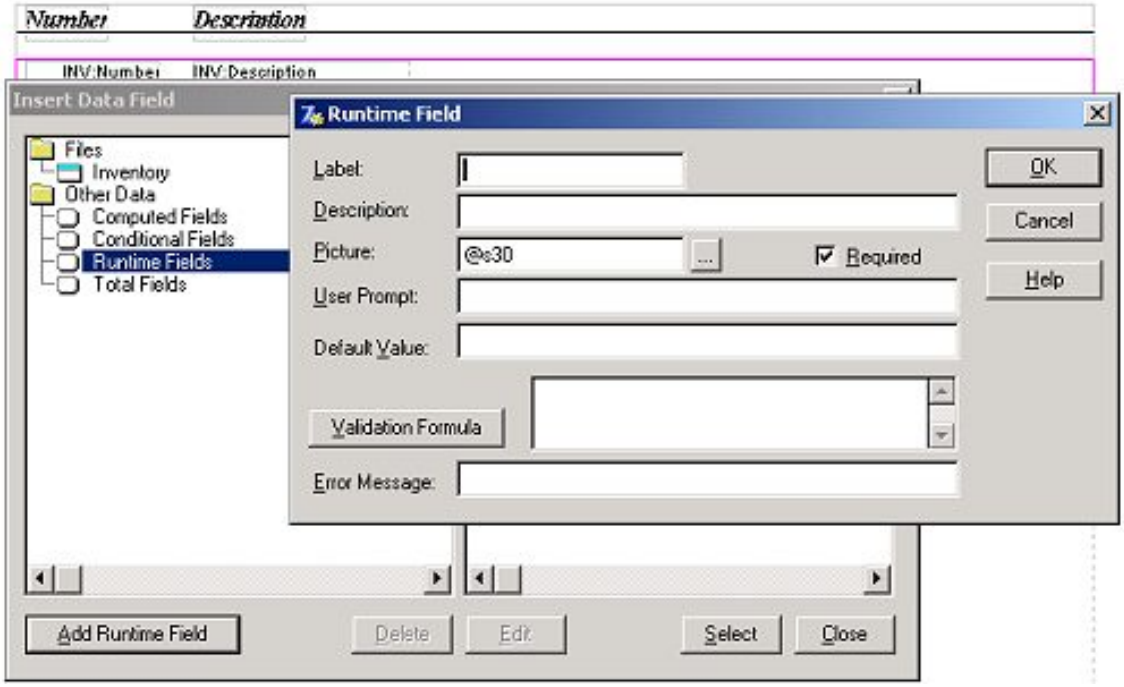


Figure 1. Creating a Runtime field

If I want to filter an inventory file by last purchase on or before a specific date, I create a Date field on which to filter:

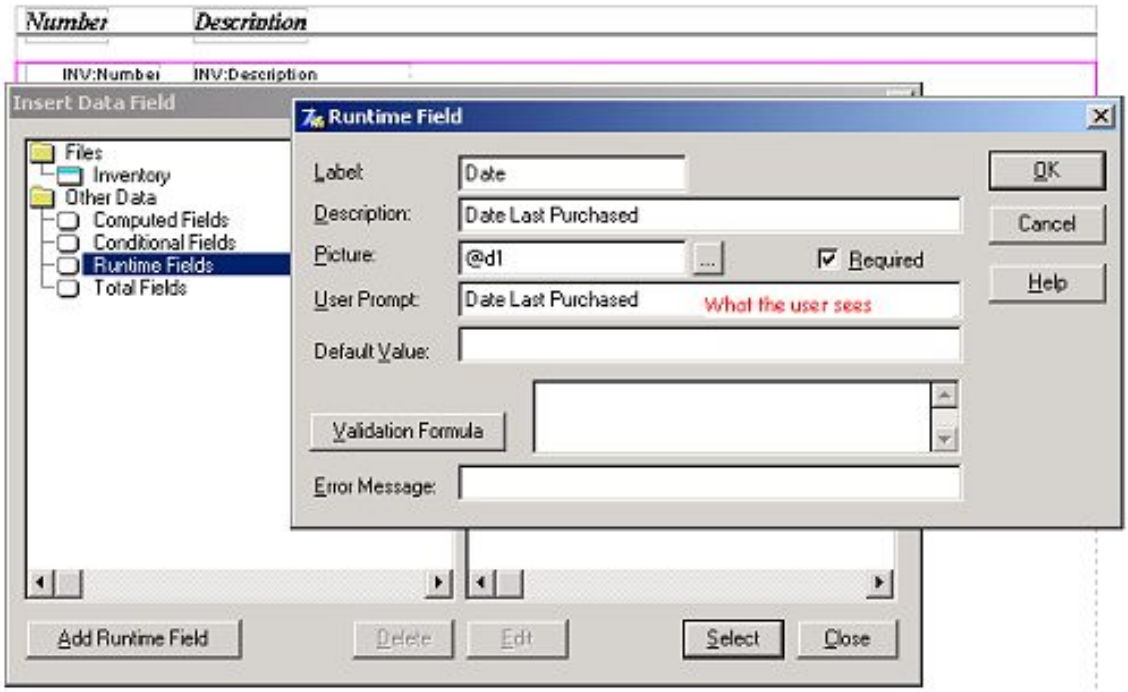


Figure 2. Setting up a Runtime field

Notice that I specify a **Label**, a description and a **Prompt**, just as if I'd created the field in the dictionary or from a procedure **Data** button. I can even make the field required.

Actually using the runtime field in a filter is simplicity itself. Just select it as one of the

components of the filter. No programming is required.

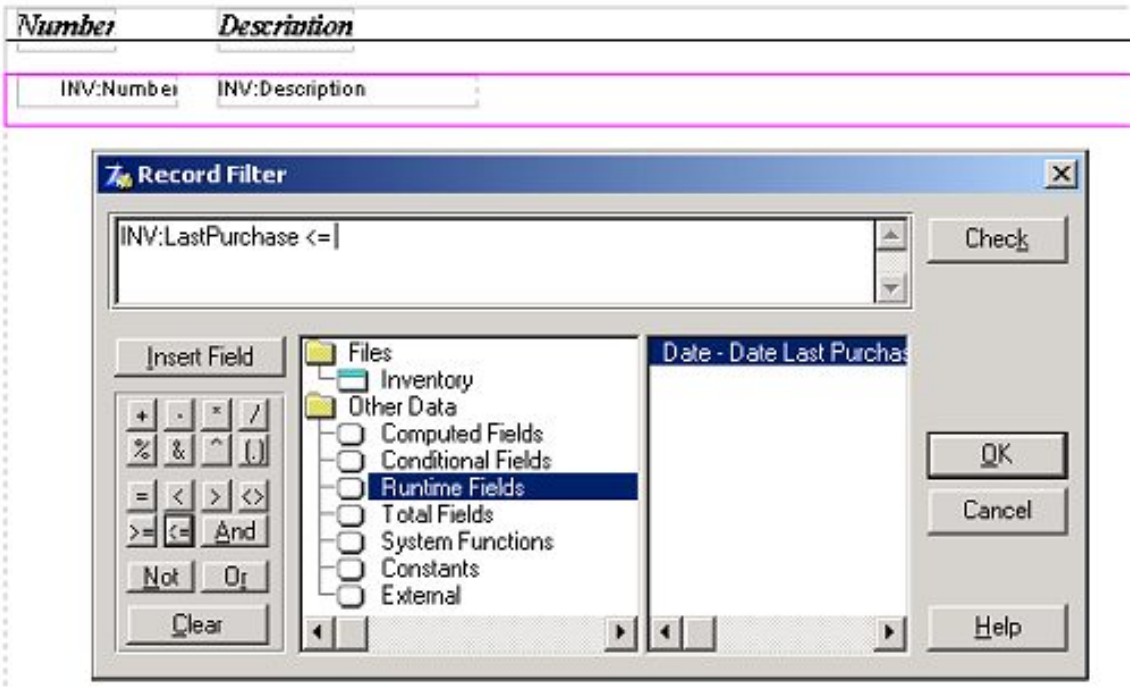


Figure 3. Set up the filter

When the report is run, sure enough, the user is asked for a last purchase date:

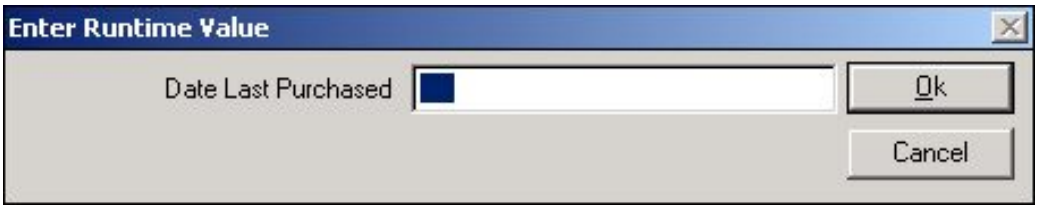


Figure 4. Runtime field, at run time

Report Writer is even intelligent enough to allow the report to be cancelled. By default, it provides a **Cancel** button, something I often forget in standard reports.

Very nice.

Very functional.

Very easy.

But not very attractive. More importantly: if I have multiple runtime fields, each one will cause a window to pop up for entry. Five filter conditions? Five windows. I'll explain how to get around that shortly, but first a word about sort orders.

Orders

One of Report Writer's *great* strengths has always been that it allows new (and often unusual) sort orders. It has always been easy to create reports with sorts not defined in the dictionary or on sorts based on joins. If I want to sort data in one file by a key or field in another, I can.

Again, those joins do not need to be defined in the dictionary (as relations). In fact, when Report Writer was introduced, Clarion didn't support relations in the dictionary.

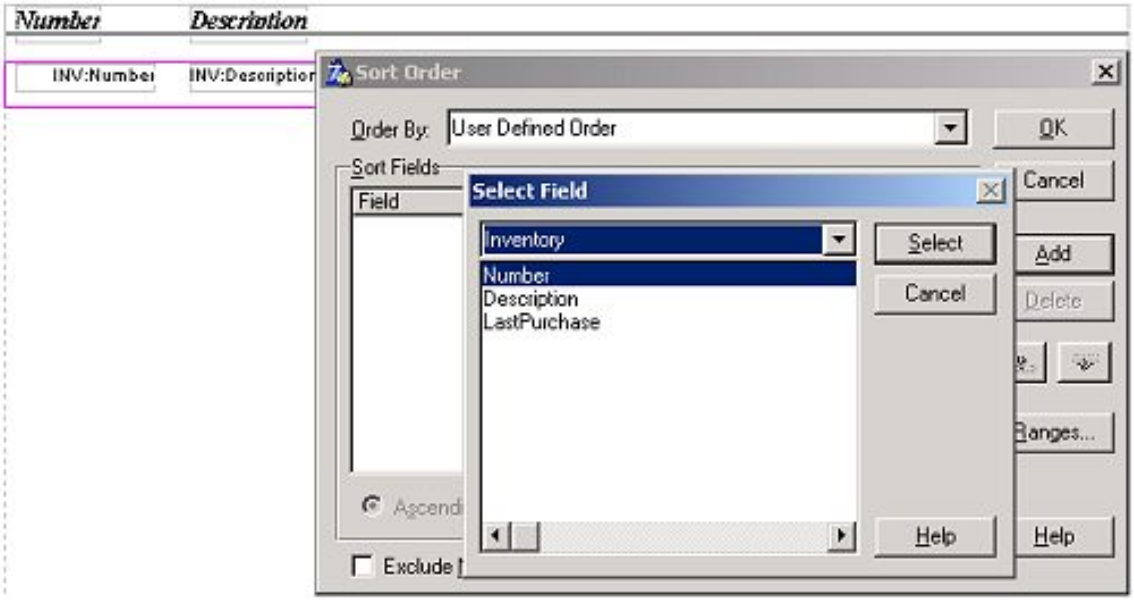


Figure 5. Custom sort order

When Report Writer was introduced, this feature was *big*. In fact, the IDE's Report Formatter couldn't duplicate this capability until the ABC templates came along.

However, only one sort order is allowed per report and that sort order is hard coded. A report, differing only in sort order from a report already created, has to be copied, renamed, updated and presented as a separate option (e.g., customers by name, by customer ID, by postal code, etc.).

Completely dynamic sort orders and filters are fairly straightforward with ABC in the Report Formatter; this is well documented for Clarion Magazine readers (see, for example, Completely Dynamic Report Orders and Breaks, Conditional Sort Orders and Page Breaks or How ABC Handles Multiple Sort Orders).

The question, then, is why can't I use a window of my own creation to capture filter data, with as many parameters as I

Which Came First?

For those of us who've been around a while, these questions about Report Writer functionality represent a curious about-face. In the past, many of us wanted the features Report Writer offered for the Report Formatter. And, in fact, TopSpeed spent a few years

want, and why can't I select sorting at runtime? Why do I need a separate report for each order?

Why not, indeed.

Wouldn't you know it? There are public methods in the Report Engine that do exactly what I want: `SetReportFilter` and `SetReportOrder` (see the Report Writer manual, pp. 180-1).

SetReportFilter

SetReportFilter (*reportname,filter*), VIRTUAL

"**SetReportFilter** overrides the report's filter, setting it to the specified *filter*."

This means that I can create a window to collect filter data and I can do so inside the IDE. I can use that window to create the filter Report Writer will use and pass it to the Report Engine.

promising that it would be possible to import Report Writer reports into the Report Formatter … in the next release.

Well, they never did make Report Writer reports importable, but the key features (ad hoc keys, sorts and breaks) certainly found their way into the ABC libraries. And, now, I find myself asking why Report Writer can't have some features from the IDE. Funny how things turn around, isn't it

Figure 6. My own filter collection window

If the user does not complete the screen (provided I *allow* that), the default filter will be used. But, if data is entered, I can create my own filter at runtime.

The code that accomplishes this is as follows:

```
RE.LoadReportLibrary('RWDemo.TXR')
RE.SetVariable('GLO:Inventory',Datapath & 'Inventory.tps')
RE.SetPreview(999999)
x = GetDate()
```

```
If x
  RE.SetReportFilter('Report2','INV:LastPurchase <= ' & x)
End
RE.PrintReport('Report2')
RE.UnloadReportLibrary
```

The lines in bold are all I require to use my own window to capture filter data at runtime. In the demo app, see the Inventory Report menu item.

Note how I check whether GetDate() was completed. If a date is returned, I call `SetReportFilter` to set the filter parameter. If no date is returned, the default filter in the report (none, in this case) is used.

While `SetReportFilter` may not have the ability to concatenate multiple filters the way the ABC `SetFilter` method does, it can take variables, as the snippet above demonstrates. This means that as long as I can create a string variable containing the filter expression I want, I can have dynamic filters every bit as complex as I can in the Report Formatter. I have tested this using only a variable and it does work as expected ("legacy" aficionados should feel right at home with this).

SetReportOrder

You can use `SetReportOrder` in a similar way to `SetReportFilter`. Its prototype is:

```
SetReportOrder(reportname,order), VIRTUAL
```

`SetReportOrder` "overrides the report's sort order, setting it to the specified *order*." You will notice a fairly significant "documentation bug" in the Report Writer manual: "filter" is substituted for "order" in a couple of places, making the spec for this method somewhat difficult to read.

Using a variable or function in the *order* parameter also works. However, it is important to remember that the variable must contain the actual order string (or, the function must return the actual order string), not just return values from selection variables.

When I loaded 'Customer Name' in the string, I got bind errors. When I loaded 'CUS:Name' into the string, it worked. I discovered this working with Clarion 6 EA (yes, the demo app works perfectly with C6), which did not return an 801 (it returned 1101, I think) and named the offending variable. Eureka!

In the sample app, however (and, I expect, many cases), I was able to design an option structure to capture the alternative sort orders:

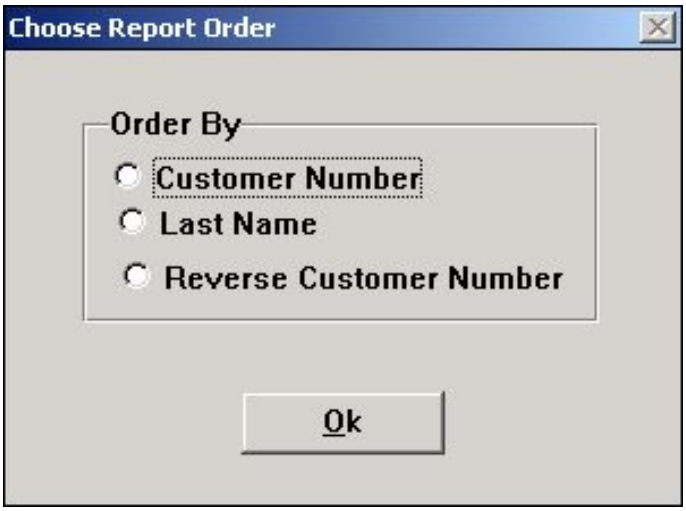


Figure 7. Getting sort orders

The code to implement runtime selection of sort orders looks like this:

```
RE.LoadReportLibrary('RWDemo.TXR')
RE.SetVariable('GLO:Customers',Datapath & 'Customers.tps')
RE.SetPreview(-1)
Case ChooseOrder()
Of 'Customer Number'
  RE.SetReportOrder('Report1','CUS:CustomerNumber')
Of 'Last Name'
  RE.SetReportOrder('Report1','CUS:Name')
Of 'Reverse Customer Number'
  RE.SetReportOrder('Report1','-CUS:CustomerNumber')
End
RE.PrintReport('Report1')
RE.UnloadReportLibrary
```

(Again, the code in bold is what affects the sorting.)

And, for the truly lazy ...

It really is very nice to be able to create custom reports without having to remake an entire project. It is often somewhat less convenient to have end users maintain a file which lists those reports. Users often don't add new reports to the file, instructions notwithstanding, they "accidentally" delete reports they want and, generally, they end up causing more … problems.

Now, when I run Report Writer, I am asked for the report library I wish to work with. Then I am asked to select the report I want to work with. If I run C55PRNTX.EXE, the same thing happens. If I supply the report library, can't I use the engine's **Select Report** window?

By omitting the report name, yes, I can:

```
RE.LoadReportLibrary('RWDemo.TXR')
RE.SetVariable('GLO:Inventory',Datapath & 'Inventory.tps')
```

```
RE.SetVariable('GLO:Customers',Datapath & 'Customers.tps')
RE.SetPreview(-1)
RE.PrintReport('')
RE.UnloadReportLibrary
```

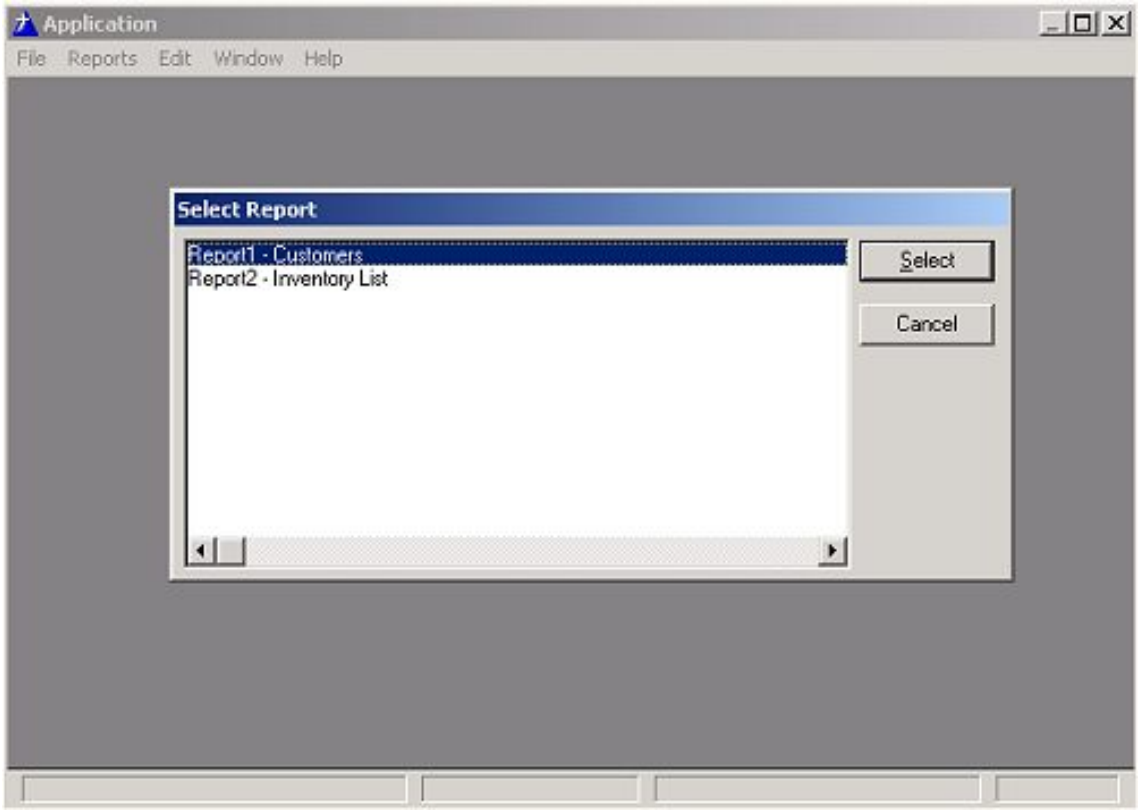


Figure 8. Using the built in pick-list

Granted, this isn't as pretty as a window I would design nor does it conveniently permit calling `SetReportOrder` or `SetReportFilter` but it *is* functional and clear. It works. It is immune to user meddling. And therefore t is perfectly adequate for many users and many purposes.

Summary

Mark Burgess first introduced technique to call Report Writer from Clarion apps in 1991. As Ben Brady demonstrates, that basic concept is as valid today as it was then.

And, now you can interface your own data collection screens and select sort orders at run time.

Download the source

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since

1993.

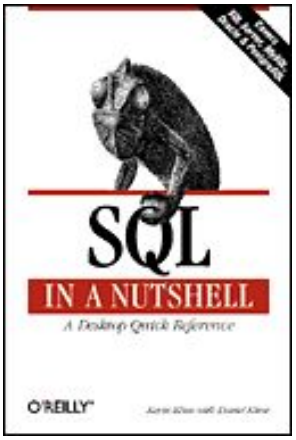
Reader Comments

Add a comment

Book Review: SQL In A Nutshell

by David Harms

Published 2003-01-31



SQL in a Nutshell

By Kevin Kline, with Daniel Kline, Ph.D.
Published by O'Reilly, December 2000
1-56592-744-3
224 pages, $29.95 US, $43.95 CA, £20.95 UK

O'Reilly's SQL In A Nutshell is a desktop reference for SQL as implemented in SQL Server, MySQL, Oracle and PostgreSQL. This is a slightly older work, first published in December 2000.

This comparison of four of the most popular SQL databases (notable exceptions being Sybase, DB2, and Interbase/Firebird) is instructive in two ways. First, it points out the startling lack of real standards in the SQL world, and second, it offers practical information for anyone wishing to develop applications which can be easily ported to a variety of databases. This is not, however, a book for someone who doesn't already know a bit about SQL. It's a reference, not a tutorial.

The touchstone for this book's comparisons is the SQL99 standard. As the authors point out, SQL database vendors typically have data types that correspond to most of the SQL99 data types, but often use different naming conventions. These data types are listed in tables in the

book; a cross reference showing each vendor's version of a given SQL99 type would be a useful addition.

The bulk of the book is taken up with an SQL command reference. Each of the SQL99 commands is discussed in detail, and here this is also a cross-reference table listing each vendor's implementation (or lack thereof). Most of this information is also available online, but not in such an organized fashion. If your concern is database portability, then you will want to compare statement syntax yourself, and not rely entirely on the descriptive text.

For instance, all four databases support the syntax `ALTER TABLE tablename ADD [COLUMN]` …. Oracle's syntax for changing a column, however, is `ALTER TABLE tablename MODIFY [COLUMN]` … while for the other three it's ALTER TABLE tablename `ALTER [COLUMN]` …. PostgreSQL is the only one of the four databases that doesn't support `ALTER TABLE tablename DROP [COLUMN]` (as of the book's publication, that is – the ability to drop columns was added in PostgreSQL 7.3). None of this is mentioned in the text, although to be fair if every difference between databases were discussed in detail, this would be a book series, not a single volume. The point is that you'll still have to do some studying if you want to be able to seamlessly install and run your apps on different SQL databases. The authors do point out many differences between vendors in the text – just don't expect these discussions to be exhaustive. There are also useful descriptions of where the vendors depart from the SQL99 specification.

Some SQL commands get fairly length treatment, including CREATE TABLE, GRANT, and SELECT. And keep an eye out everywhere for those owl and turkey images in the text – these identify, respectively, helpful information and potential problems which deserve your special attention.

The last two chapters cover SQL functions (built-in, not user-defined) and unimplemented SQL99 commands. Again, what this book (perhaps unintentionally) points out is the absolutely pitiful state of SQL standardization.

Although the information in this book is slightly dated, that may not be a bad thing for anyone wishing to develop for multiple backends, since it's unlikely that all potential clients will be running the latest version of their respective database of choice. Or it may help you decide which database will best serve your needs, and save you the grief of trying to satisfy the disparate requirements of each database vendor.

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