Clarion Magazine

Clarion News

Hurricane Katrina Relief

Clarion developers are among those affected by Katrina's devastation; at least one developer we know of has a flooded home and business losses, and of course many people are in desperate need. Donate to hurricane relief through the Red Cross.

- » PrintWindow 1.00 Beta 14
- » Gitano Birthday Sale
- » xAnalogClock v1.3
- » New Category in Web Links
- » xWordCOM 1.4
- » xTicker Demo
- » Buggy 4.1.7
- » SetupBuilder 5.0 Final Release Date
- » A Data Architecture for IT Service Management
- » New Debuger Version Available
- » FinalStep 2.00 Beta 1
- » EasyExcel 3.08
- » Debuger Class 6.0.1
- » DigitChanger 1.00 Beta 2
- » Clarion.syn For TextPad
- » Fomin Report Builder 2.94
- » List & Label August 15 Deadline
- » EasyOpenOffice 1.00
- » EasyCOMCreator 1.01

- » Clarion 6.2 Build 9047
- » FullRecord 1.00
- » EasyCOMCreator 1.00
- » iAlchemy's daFinga
- » ClaProtect Adds ASprotect to Clarion Apps
- » xDigitalClock 1.8
- » BST 4.112
- » EasyResizeAndSplit 2.10
- » EasyListPrint 1.13
- » EasyCOM2INC 2.06
- » Clarioneers Database
- » IceTips Products C6.2 Compatible
- » xXPpopup 1.4
- » List & Label 11 Upgrades
- » xPictureBrowse 2.4
- » xRuntimeStyle Manager 2.0
- » Free xFunction Library 2.2
- » xWhatsNew Class 1.9
- » xPathManager 1.4
- » xInactivity 1.4
- » xFText 2.6
- » DOS Printer 10.31
- » xAppWallpaper Manager 2.3
- » xAppWallpaper 1.8
- » MAV Direct ODBC 010 and 008 Release
- » dpQuery 2.05
- » BoTran 2.2
- » xQuickFilter 2.17
- » Big Image Tamer 1.3
- » Free Clarion Third Party Advertising
- » Office Inside 1.78

- » MessageEx C6.2 Compatible
- » SimPageOfPage Available At ClarionShop
- » EasyCOM2INC WMI 1.00
- » Fomin IPDS Report Storage
- » List Header Customizer
- » Dynalib and Clarion 6.2
- » Secwin Online Server 1.05
- » HotDates Goes Blonde
- » ClarioNET and C6.2
- » EasyCOM2INC_SQLDMO 1.01
- » Easy Class Generator 1.05

[More news]

Podcast

[Track lists, more podcasts]

Latest Free Content

[More free articles]

Latest Subscriber Content

Clarion Magazine's "No DevCon" Special

Traditionally, **September** has been the month for **Clarion DevCons**. There's no DevCon this year (maybe we'll get one in February), but you can still get **awesome deals** on Clarion

Magazine subscriptions. **Save up to $207!**

Using Local Classes Instead Of Local Routines

Local classes are often touted as a replacement for procedure routines. But what are local classes, why do you need them, and what can you do with them? Nardus Swanevelder answers these questions, and shows how easy it is to use local classes.

Posted Thursday, August 11, 2005

PDF for June & July 2005

All Clarion Magazine articles for June & July 2005 in PDF format.

Posted Friday, August 12, 2005

Speeding Development With Defaults And Small Templates

Faced with a conversion of a DOS application to Clarion 6.2, Phil Will came up with of two simple things to do with templates and the defaults file that saved a tremendous amount of time in development, and greatly eased application maintenance.

Posted Tuesday, August 16, 2005

Reports: OOP, ABC and Ignoring Templates (Part 3)

A reporting technique, mentioned in passing three years ago, comes back to haunt Steve Parker.

Posted Friday, August 19, 2005

Fetching A Web Page With WinInet

There are a few different tools you can use to get a web page from a server, and one readily available solution is to use Windows' own WinInet DLL. Skip Williams shows how it's done.

Posted Tuesday, August 23, 2005

Mixing Clarion with .NET, Part 1

The .NET framework is extremely large, growing and very well supported. Right out of the box the framework has hundreds of classes that are well thought out and which integrate seamlessly with .NET applications. These classes cover a very wide range of the things you might want to do, once Clarion.NET arrives. But there's no need to wait. In the first article in this series, Wade Hatler looks at calling .NET classes from Clarion code.

Posted Wednesday, August 31, 2005

Querying SQL Data

It sure would be handy if Clarion had a way to run an SQL query and just get back a result. Say you want the total of a bunch of records. The problem is that in Clarion you need a view to project just the amount field and...but wait! As Tom Ruby shows, you can easily use the cCwADOclass for just this situation.

Posted Wednesday, August 31, 2005

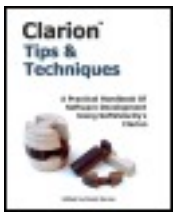
[Last 10 articles] [Last 25 articles] [All content]

Printed Books & E-Books

E-Books

E-books are another great way to get the information you want from Clarion Magazine. Your time is valuable; with our e-books, you spend less time hunting down the information you need. We're constantly collecting the best Clarion Magazine articles by top developers into themed PDFs, so you'll always have a ready reference for your favorite Clarion development topics.

Printed Books



As handy as the Clarion Magazine web site is, sometimes you just want to read articles in print. We've collected some of the best ClarionMag articles into the following print books:

- Clarion 6 Tips & Techniques Volume 1 - ISBN: 0-9689553-8-X
- Clarion 5.x Tips and Techniques, Volume 1 - ISBN: 0-9689553-5-5
- Clarion 5.x Tips and Techniques, Volume 2 - ISBN: 0-9689553-6-3
- Clarion Databases & SQL - ISBN: 0-9689553-3-9

We also publish Russ Eggen's widely-acclaimed Programming Objects in Clarion, an introduction to OOP and ABC.

From The Publisher

About Clarion Magazine

Clarion Magazine is your premier source for news about, and in-depth articles on Clarion software development. We publish articles by many of the leading developers in the Clarion community, covering subjects from everyday programming tasks to specialized

techniques you won't learn anywhere else. Whether you're just getting started with Clarion, or are a seasoned veteran, Clarion Magazine has the information *you* need.

Subscriptions

While we do publish some free content, most Clarion Magazine articles are for subscribers only. Your subscription not only gets you premium content in the form of new articles, it also includes all the back issues. Our search engine lets you do simple or complex searches on both articles and news items. Subscribers can also post questions and comments directly to articles.

Satisfaction Guaranteed

For just pennies per day you can have this wealth of Clarion development information at your fingertips. Your Clarion magazine subscription will more than pay for itself - you have my personal guarantee.

Dave Harms

Clarion Magazine

Clarion News

Search the news archive

Clarion Magazine's "No DevCon" Sale

There's no DevCon this year, but you can still get awesome deals on Clarion Magazine subscriptions. Save up to $207!
Posted Thursday, September 01, 2005

xDigitalClock 1.9

This release of xDigitalClock is a bugfix only, correcting a GPF in programscompiled in DLL mode under the last builds of Clarion 6.1
Posted Thursday, September 01, 2005

SetupBuilder 5.0 Released

After 40 builds released to more than 2000 beta testers around the world, SetupBuilder 5.0 is now available. SetupBuilder 5 is a powerful and easy to use Installation Authoring and Configuration Management environment for developers and organizations deploying applications to the Microsoft Windows Platform, including Windows Vista. SetupBuilder 5 has been designed from the ground up to deliver a perfect fusion of installation authoring, web deployment, configuration management, and scripting technologies.
Posted Thursday, September 01, 2005

EIP Template Version 2

AuDkus is glad to announce the release of the EIP Template version 2. The upgrade is free for all current users. This has been a major upgrade with many improvements and features

requested by the users. EditCalendarClass; EditCheckClass; EditColorClass; EditDropComboClass; EditDropListClass; EditentryClass; EditFileClass; EditMultiSelectClass; EditSpinClass; EditTextClass. The EditCalendarClass is a new Class added to Clarion 6 which enable the use of a Calendar. This class is not available to Clarion versions prior to version 6. Instead The EIP Template comes with an additional class called EditCallProcedureClass which enables you to call any procedure which could contain for example a calendar or a calculator from any third part developer. You can also use this class to call your own procedure when the user is in Edit in Place mode.
Posted Thursday, September 01, 2005

xWordCOM 1.5

xWordCOM 1.5 is now available. Changes include: InsertPageBreak method fixed; New InsertBreak method; New GetOptions method; Various view and option methods.
Posted Thursday, September 01, 2005

xMisc 1.2

xMisc is a set of miscellaneous classes, including xDriveClassGetting; xDigitalConvertClass; xLinkClass; xRegistryClass; xFilesOperationsClass; xHintClass. Price: $59.
Posted Thursday, September 01, 2005

PrintWindow 1.00 Beta 14

PrintWindow lets you get a screen print by simply dropping a button control on your window. You use your own report with customized parameters, which means you can choose your previewer (or none), and how many tabs per page (only one or as many as fit). You can customize the aspect of the printed controls to mimic the full window, or to use a image as background (like when you fill a Form), for example, not drawing the entries or group boxes, etc. When you take a "screenshot" to print, you get exactly what you see, but PrintWindow gives you several advantages: If the window doesn't fit on the screen, it will fully print anyway; You can customize what controls you want to print on the window, therefore if some controls you don't want to be printed, you just select them in a dialog window; You can also customize several aspects of what you are going to print, for example, entry fields with or without box, the whole print in black and white or at full color, etc.; Since you use the internal report generator, your window will end up in a WMF vector file, which you can shrink or enlarge without quality loss (as opposite to a screenshot, which is a bitmap by definition); It is far easier for your end user to just press a

button and use the regular previewer, than capture the screen and see how to print it; As PrintWindow uses a regular report, you can send your window to the printer, to just the screen, to a PDF, or to any kind of file your report system is capable of exporting to.
Posted Friday, August 26, 2005

Gitano Birthday Sale

50% OFF on Gitano's bundles and gReg and a free gift valued at $119. Sale ends on Sunday, August 28th. Source Code Bundle regular $999 now $499.50. Developer's Bundle regular $659 now $329.50. gReg regular $299 now $149.50
Posted Thursday, August 25, 2005

xAnalogClock v1.3

xAnalogClock is a simple class and control template to display time like an analog Clock. New in this version: Now compatible with Clarion 6.2; New Installation kits created by SetupBuilder 5.0.
Posted Thursday, August 25, 2005

New Category in Web Links

There is a new sub-category called "Database GUI Front End" under "Software Tools" in the "Web Links" section at the Clarion Developer web site. There are currently five links in this sub-category. You can add your favorite "Database GUI Front End" to whatever is your database of choice. Most databases are listed as their own categories and all "Web Link" entries support up to seven categories. And if your specific database of choice is not listed as a category, let Roberto know, and he will add it. "Web Links" is an area that is open to all. This includes anonymous users.
Posted Thursday, August 25, 2005

xWordCOM 1.4

xWordCOM 1.4 is now available. This COM interface library helps you to create Word documents directly from Clarion applications. You have complete control over document formatting, including table creation and graphic file insertion.You can make search and replacement in the documents, save the document in various formats, merge documents, print, execute macros and much more. xWordCOM supports MS Word'2003, MS Word'2002, MS Word'2000 and MS Word'97. New in this version:Now compatible with Clarion 6.2 (build 9047); New Installation kits created by SetupBuilder 5.0.
Posted Thursday, August 25, 2005

xTicker Demo

Kurt Pawlikowski has a demo for a ticker tape control called xTicker, and is looking for some input. Contact Kurt at kurtt@pinrod.com or (773) 284-9500.
Posted Thursday, August 25, 2005

Buggy 4.1.7

An update to the bug tracking tool "Buggy" is now available for all registered users. The evaluation edition was also updated.
Posted Thursday, August 25, 2005

SetupBuilder 5.0 Final Release Date

Linder Software plans to release SetupBuilder 5.0 Final Release (build 1244) on Wednesday, August 31, 2005. This will be a full install. You have to uninstall the previous alpha and beta versions before installing the final release. Your current product serial number and a "Maintenance Plan Key" is required in order to install SetupBuilder 5. Maintenance Plan Keys will go out to all registered SetupBuilder 5 users shortly.
Posted Thursday, August 25, 2005

A Data Architecture for IT Service Management

The document A Data Architecture for IT Service Management created by Charles T. Betz has been added to the ClarionDeveloper download section under the category Documents | Data Modeling Related. It's a lengthy (40 pages) analysis of ITSM and some related concepts from a data architecture perspective. Included is a high level entity-relationship model, definitions, discussions of general data issues as pertinent to internal IT enablement, and some initial thoughts on data/process matrixing. Extensive coverage of the Configuration Item concept. Available to registered users only.
Posted Thursday, August 25, 2005

New Debuger Version Available

A new version of Debuger (yes, it is spelled that way) is now available from Mark Goldberg. Note: The Debuger was originally created by Skip Williams, and includes code from an article by Alan Telford, see credits in the source code. This version is compatible with C5, C55 as well as C6+. A number of changes have been made recently, and some of

these are only present in the beta version (at www.monolithcc.com/clarion/debuger_beta.zip). Changes include: Automatically LINK in the ASCII driver; C6+ support for HOWMANY and WHAT(Stru,N,Dim) in the DumpQue method; Refined automatic messages when .INIT is run; Add a system to manage the module_debuger reference used by the AssertHook2 replacement, this could solve potential problems when using multiple debugers, and the most recently .init'd has been de-instantiated (BETA); Add a wild card system to allow for specific handling (and suppression of) of Assert Messages, (BETA); Fixed AssertHook2 so that when the user requests a GPF, it now use the API call, DebugBreak() (BETA); CheckError refined (BETA). Note: The beta version has some rem'd code for internal debugging.
Posted Thursday, August 25, 2005

FinalStep 2.00 Beta 1

This information is not public in the web yet. For those interested in testing the new version, there is no upgrade cost for current users of FinalStep 1.x. Changes include: Different wallpapers for Browse, Form, Frame or other windows (internal or user defined); For frame, now you can choose between fixed wallpaper, user defined wallpaper or random wallpaper; 49 original photographs in JPG format to use as random Frame wallpapers; Hook for STOP; Hook for HALT; All major routines now global procedures (Up to 10% smaller programs generated); Specific code to take advantage of Clarion 6.x; A new set of bubble like buttons (six colors); Three new sets of buttons (golden metal, silver metal and plastic); Local override for global "extras" generation.
Posted Thursday, August 25, 2005

EasyExcel 3.08

EasyExcel 3.08 is now available. Includes a fix to the WriteQueue method, a new PastSpecial method, and an updated Queue2Excel template. This is version for Clarion 5.5, 6.1 (9034) and 6.2 (9047). Free for all registered customers. Price: $129
Posted Tuesday, August 16, 2005

Debuger Class 6.0.1

Debuger Class 6.0.1 is now available. This is a SB5 install that simply places source in %root%\libsrc and %root%\template, totaling three files, plus an update tech reference PDF. The only significant change is that the DebugOut method has been deprecated in favor of the new Message method. The reason for this change is so that those that use MESSAGE() as a debug statement now have a safe alternative, yet really don't have to change anything. Takes the same parameters as the Clarion MESSAGE() statement. Those

that are used to DebugOut won't need to change anything, those that don't like DebugOut can use Message instead.
Posted Tuesday, August 16, 2005

DigitChanger 1.00 Beta 2

DigitChanger changes the number of digits of the amounts in your window, reports and tables via a global setting. For clarion 5.5 or 6.x; ABC (Legacy at request); Multi-DLL support; 100% source code (no black boxes).
Posted Tuesday, August 16, 2005

Clarion.syn For TextPad

An updated Clarion syntax file for TextPad is now available from ComSoft7.
Posted Tuesday, August 16, 2005

Fomin Report Builder 2.94

Fomin Report Builder 2.94 is now available. List of changes is as follows: The "frbuser.hlp" file has considerably changed; Context sensitive help is now available from the new frbuser.hlp file; Some stability problems where successfully eliminated by revised source code of the floating "Property Box" window, which is running in a different thread; Some minor changes and fixes included.
Posted Tuesday, August 16, 2005

List & Label August 15 Deadline

All List & Label users should have already had an email from combit reminding you that the deadline is fast approaching to secure your discount of 20% off the latest version of List & Label. For more information about the new features of List & Label v11 (especially multi-child tables which has been a request for a long time), please visit http://www.combit.net. Solace Software is processing upgrade orders at the same price as Combit, plus if you order your upgrade through Solace you will get the upgrade templates to v11 which are being worked on at the moment.
Posted Thursday, August 11, 2005

EasyOpenOffice 1.00

EasyOpenOffice is a set of classes and templates allowing you to exchange data between

Clarion applications and OpenOffice Calc and Writer. This release has basic features (you can open and close Calc and Writer, pass/get the data form your Clarion application, format cells in Calc, create tables and set font properties in Writer etc.). More features to come. Clarion 5.5 or Clarion 6.1/6.2. ABC, Legacy class template support. 32-bits only. Price: $149
Posted Wednesday, August 10, 2005

EasyCOMCreator 1.01

EasyCOMCreator 1.01 is now available. Changes include: Compilation fixes; Main window appearance; New ProgID and VersionIndependendtProgID properties; C++ console test app example for the Math object.
Posted Wednesday, August 10, 2005

Clarion 6.2 Build 9047

SoftVelocity has released an update for Clarion 6.2. The new release is build 9047 and it corrects several recently reported problems. The installs are patches and must be applied against release 9046.
Posted Monday, August 08, 2005

FullRecord 1.00

FullRecord 1.00 is now available. This is a full-featured auditing system for your Clarion or ABC programs, 5.5 or 6.x, ABC or Legacy. Multi-DLL support, 100% source code. FullRecord can record operations from any type of file (ISAM, SQL, even IMDD if needed). The auditing file may be ISAM or SQL. DIM fields are supported (full 4 dimensions support); OVERed fields are supported; Unlimited memos support per file. There is only ONE file for auditing storage. BLOB recovering is supported for clarion 6.x. Available at ClarionShop US#99, or get the FullRecord + FinalStep bundle for US$169.
Posted Monday, August 08, 2005

EasyCOMCreator 1.00

The EasyCOMCreator utility is used to create a Component Object Model (COM) dynamic-link library (DLL) automatically using Clarion. It allows you to manage your Projects, Objects, Methods and Parameters and generates all the necessary CLW/PRJ files (including a test application) automatically. After you compile your Project you will get your COM server DLL. To use it, you need to register it using the REGSVR32.EXE utility

and you are done. Requires Windows 98 or higher, Clarion C61-9034 or higher. Demo version available (limited to 5 methods in all).
Posted Monday, August 08, 2005

iAlchemy's daFinga

iAlchemy has released daFinga, a template/library designed around the APC Biopod biometric fingerprint readers. ABC or Clarion template chain. Enroll multiple fingers per user, identify, list, delete. All functions are accessible either through a simplified control/extension templates or can be hand coded as simple functions. $179.00 USD (electronic delivery).
Posted Monday, August 08, 2005

ClaProtect Adds ASprotect to Clarion Apps

ClaProtect is a set of templates and classes that allows you to use all features of ASprotect, a popular and powerful program protection system. ASprotect lets you encrypt important code, make keys limited by user, date, hardware (machine id), or function. ASprotect includes anti-debugger/CRC protection and packs executables. It uses 512/768/1024bit RSA keys.
Posted Monday, August 08, 2005

Clarion Magazine

Clarion Magazine's "No DevCon" Special

It's September, and for years, that meant it was time for the Florida DevCon. SoftVelocity resumed the tradition last year with a great show, but right now the folks at SV have their heads down coding C7 and Clarion.NET, and we're all going to have to wait a little longer for the next conference (maybe February).

DevCons are a great time to meet with other Clarion developers, get up to date on the latest Clarion technologies, and get some great deals on Clarion-related products. We can't get everybody together in one room, but getting Clarion developers up to speed on the latest Clarion technologies is what we do best. And until September 16, you can get DevCon-level savings on your Clarion Magazine subscription or renewal!

Three Year Subscription - Save up to $207!

You can now get a **three year subscription**, including all **back issues***, for **$189** (or **renew** your current subscription for three years for only **$169**) *and* get a **free copy** of our newest book, **Clarion 6 Tips & Techniques, Vol 1**** (just add the book to your shopping cart along with and the discount will be applied at the checkout). Subscribe/renew now.

Two Year Subscription - Save up to $76!

You can now get a **two year subscription**, including all back issues*, for just **$149** (or **renew** your current subscription for only **$129**). Subscribe/renew now.

One Year Subscription - Save up to $50!

You can now get a **one year subscription**, including all back issues*, for just **$99** (or **renew** your current subscription for only **$79**). Subscribe/renew now.

Prices go up September 16, 2005!

* Clarion Magazine began publication in February, 1999.

** Regular price $49.95. Shipping extra

Clarion Magazine

Using Local Classes Instead Of Local Routines

by Nardus Swanevelder

Published 2005-08-11

At DevCon 2004 Nik Johnson presented an overview of business rules, local classes and triggers. In a previous article I have explained how business rules work, and how to extend the Clarion Business Rule class to be able to change your business rules at runtime. In this article I'll discuss the use of local classes as a replacement for local routines.

But what are local classes, why do I need them, and what can I do with them? I have been using the Clarion procedure routines and it is more than adequate isn't it?

Before I continue I must make it clear that this article is not going to teach you object oriented programming (OOP) theory. It will show you the benefits of local classes, and how you can change your procedure routines into local classes.

That said, I don't think it will hurt anyone if I recap some OOP basics. I found the following definitions in Bruce Johnson's presentation at Devcon 2004 (reprinted here with permission) and I think they give a very nice overview of what classes, properties and methods are.

> **Classes ("Libraries")**
>
> Classes are like libraries of code. They can be designed to perform a generic task, or a specific one. Typically a class will have a specific task in mind (drawing a graph, making a network connection and so on.)
>
> **Properties ("Variables")**
>
> In order to conceptualize a difference between regular global data, and data that belongs to the class, we call the class data "properties." In other words properties are just variables that belong to the class.
>
> **Methods ("procedures")**
>
> If properties are just variables that belong to the class, then methods are just procedures that belong to the class.
>
> In summary, classes are collections of properties and methods (variables & procedures) that are used in your program.

Okay, I remember what OOP is all about, but why do I need local classes? To answer this lets look at some of the advantages of OOP as applied to local classes.

- Routines can be divided into methods, which can be called individually
- Method data is private to that method and cannot be accessed by external procedures and or methods
- Methods can communicate with each other
- New properties and methods can easily be added whenever necessary
- Breaking up code into a class with methods can make it easier to maintain and modify that code

So let's begin with what you need to create a local class.

Defining the class

The first step is to define the class in the Local Data embed point. Figure 1 shows you where to find the Local Data embed point.

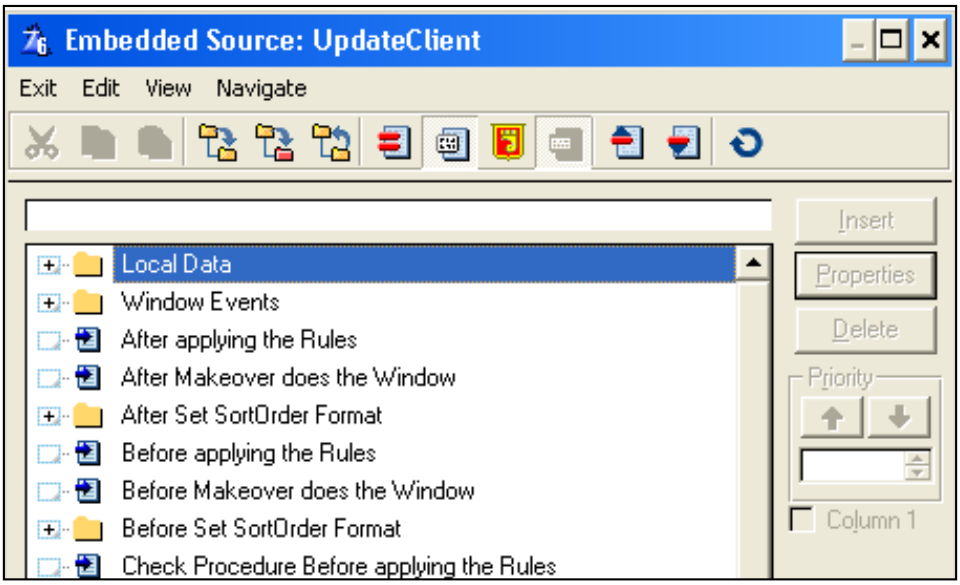


Figure 1. Embed Points for Local Class in Local Data

Here's an example of a local class definition:

```
MyLocal         Class()
!Global Property for all the methods in the class called MyLocal
Variable1         Long(0)
!Methods in MyLocal
DoSomething       Procedure()
                  End
```

You can also derive local classes from other classes. For instance, I use SysList from Solid Software, and if I want to use a local derived class I might define the class this way:

```
MyLocal           Class(SysListClass)
!Global Properties for all the methods in the class called MyLocal
ReturnValue         Byte
OpenProgram         CString(256)
FileNameToUse       CString(1025),DIM(5)
TempPath            CString(513)
PopUpString         String(256)
Operation           CString(256)  !Open / Print / Explore
```

```
!Methods in MyClass
Check_Command          Procedure(Signed ItemIndex)
Run_ShellExecute       Procedure(String FileName)
GetFileType            Procedure(String FileName)
GetOpenfileCommand     Procedure()
                     End
```

As Bruce pointed out in his definition, methods are just procedures that belong to the class. In the rest of the article I will use the word "procedure," but you can read it as "method" as well.

Creating the procedures

The next step is to create the procedures (methods). In the Local Embeds of your procedure scroll down to the bottom of the list, and you will find an Embed point called Local Procedures. Look carefully, it is Local Procedures not Procedure Routines. This is where you add your code for your methods.

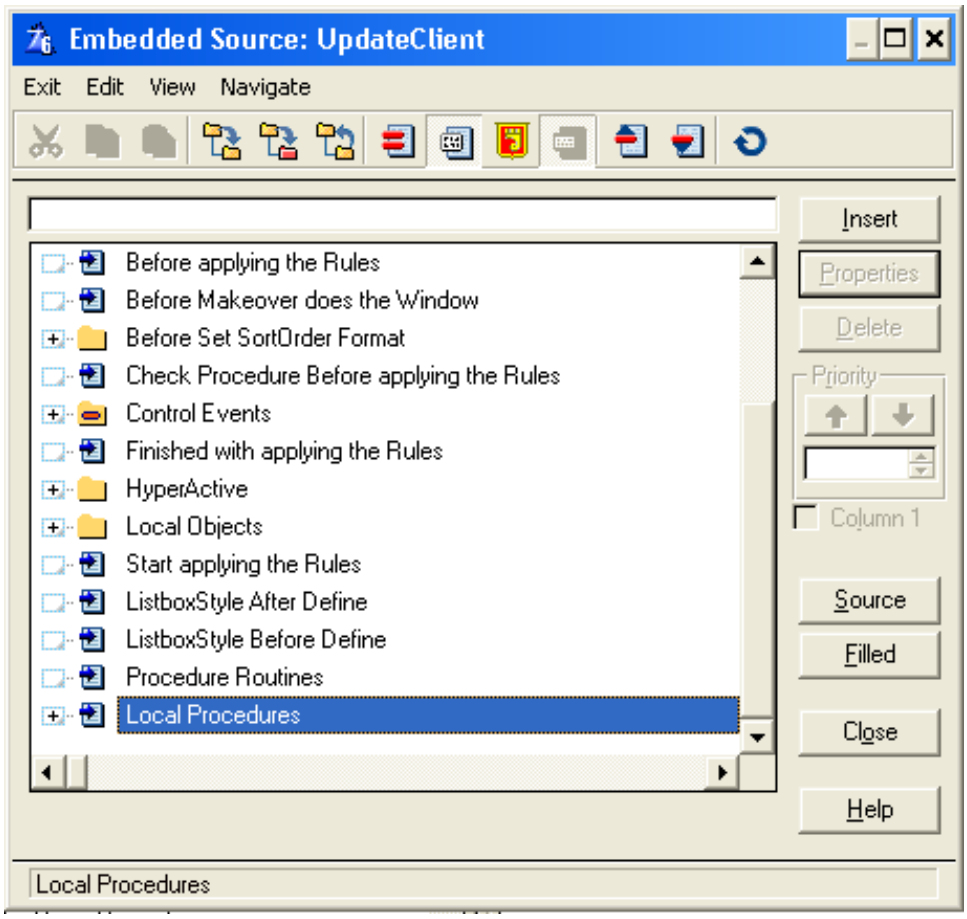


Figure 2. Embed Points for Local Class in Local Procedures

Using the first class definition I showed, `MyLocal`, you can add the following code to this embed point:

```
MyLocal.DoSomething        Procedure()
!Define any properties/variables local to this method/procedure.
Flag        Byte()
    Code
    Flag = 0
    If Flag = 0 and MyLocal.Variable1 = 0
```

```
        Message('Hello World')
    Else
       Message('Bye World')
    End
```

To recap, you have defined your local class and you added your method(s) to the Local procedure. Okay, that's cool but how do you use this?

Let's assume you want to display a Hello World message when the user clicks on one button, and another message if the user clicks on a second button. In the accepted embed points for the buttons you can add the following code:

```
CASE ACCEPTED()
    OF ?BtnDisplayHelloWorld
            MyLocal.Variable1 = 0
    MyLocal.DoSomething()    !This will display Hello World
    OF ?BtnDontDisplayHelloWorld
            MyLocal.Variable1 = 1
     MyLocal.DoSomething()   !This will display Bye World
```

So what is the advantage of this? You don't have to create all your properties as local data variables and try to keep track of them. You can create a property that is global to the class (`Local.Variable1`) or you can create a property that is local to a method (`Flag`). You can reduce the size of your routines due to the fact that you can split your routine code into methods and call the methods as part of the class.

I hope by now you are thinking that this is easy.

A simple example

The example code that I will look at is based on a small demo app available as a download with this article

Open the example application called Proplan.app. Go to the `UpdateProjects()` procedure and open the Clarion Embeds editor. Under the Local Data Embed you will find the following code

```
MyLocal              Class()
SetProjectNumber       Procedure()
                     End
```

This declares a method called `SetProjectNumber` in the class called `MyLocal`.

What does this the `SetProjectNumber` procedure do? It calculates the Project Number based on the Project Area and the Project Type. If you scroll down in the Clarion Embed Editor to Local Procedures you will find the following code:

```
MyLocal.SetProjectNumber          Procedure()
Flag        Byte()

   Code
   Flag = 0
   !Check if Area has Characters for Project Number
   LPROARE:ProjectArea = PRO:ProjectArea
   Access:LProjArea.TryFetch(LPROARE:ProjAreaKey)
```

```
If LPROARE:NumberCharacters = ''
   Message('Please select an Area first or complete' |
         & ' the Area lookup information','Projects')
Else
   Flag = 1
End
!Check if Type has Characters for Project Number
LPROTYP:ProjectType = PRO:ProjectType
Access:LProjType.TryFetch(LPROTYP:ProjTypeKey)
If LPROTYP:NumberCharacters = ''
   Message('Please select an Type first or complete ' |
         & 'the Type lookup information','Projects')
Else
   Flag = 1
End
If Flag = 1
   PRO:ProjectNumber = Clip(LPROARE:NumberCharacters) |
   & '-' & Clip(LPROTYP:NumberCharacters) |
   & '-' & PRO:ProjectAutoNumber
      Display(?PRO:ProjectNumber)
End
```

I call this procedure just after the user has completed the Project Area and/or the Project Type, with the following code:

```
Local.SetProjectNumber()
```

That code is simple and straightforward – it's just a single procedure call that would otherwise look the same in a routine.

Now let's look at a more complicated example.

A more involved example

Go to the `UpdateContacts()` procedure and open the Clarion Embed Editor. Under the Local Data Embed you will find the following code

```
MyLocal              Class()
SelectError            Byte
CheckError             Byte
ContactTypeSelect      Procedure()
CheckContactType       Procedure(String PassContactType)
                     End
```

`SelectError` and `CheckError` are two properties that are global to the class. `ContactTypeSelect()` is a method that calls a Select screen so that the user can select a Contact Type.

`CheckContactType()` checks that the user has entered a valid Contact Type code. This Contact Type code is validated against another table. This method passes a property called `PassContactType`. For simplicity I pass the property by value and not by address.

Here's the code for `ContactTypeSelect()`.

```
MyLocal.ContactTypeSelect               Procedure()
ContactType         String(20)        !Local to the method
      Code
      !Set Select Error to no error
      Self.SelectError = 0
      ContactType         = ''
      !Call Lookup - External to this procedure
      ContactType = SelectContactTypeLookup()
      If ContactType <> ''
         !If it returned a value - do assignment
         CON:ContactType = Clip(ContactType)
         Display(?CON:ContactType)
      Else
         !External Lookup Proc didnt return a value - display message & Set Select
error
         Message('No valid contact type was selected','Contacts')
         Self.SelectError = 1
      End
```

The code for the `CheckContactType()` method is as follows:

```
MyLocal.CheckContactType        Procedure(String PassContactType)
   Code
   !Set Error to No Error
   Self.CheckError  = 0
   LCONTYP:ContactType = Clip(PassContactType)
   If Access:LContactType.Fetch(LCONTYP:CONTACTTYPEKEY)
      Self.CheckError = 1
   Else
      !Do nothing - no error
   End
```

That's nice but what does it give me?

When the user completes the Contact Type I first execute `CheckContactType()` to check if it is a valid entry. If it is not valid then I call `ContactTypeSelect()` to display a Select screen with valid values.

Here is the code that executes if the user completes the Contact Type Field.

```
If QuickWindow{Prop:AcceptAll} = False
   !User has entered a Contact Type - need to check it
   !Call local method to check contact type
   MyLocal.CheckContactType(CON:ContactType)
   If MyLocal.CheckError = 1
      !If not a valid contact type
      CON:ContactType = ''
      !Call local procedure to do a lookup
      MyLocal.ContactTypeSelect()
      If MyLocal.SelectError = 1
         Select(?CON:ContactType)
         Cycle
      Else
         !Do nothing - valid Contact Type selected
      End
```

```
      Else
         !Do nothing - valid Contact Type
      End
   Else
      MyLocal.CheckContactType(CON:ContactType)
      If MyLocal.CheckError = 1
         Select(?CON:ContactType)
         Cycle
      End
   End
```

When the user clicks on the lookup button for Contact Type control I execute the following code:

```
      IF QuickWindow{Prop:AcceptAll} = False
         !If not AcceptAll do lookup
         CON:ContactType = ''
         !Call Local procedure
         MyLocal.ContactTypeSelect()
         If MyLocal.SelectError = 1
            Select(?CON:ContactType)
            Cycle
         Else
            !Do nothing - valid Contact Type selected
         End
      Else
         !Do nothing during AcceptAll
      End
```

I know you can accomplish this functionality with standard Clarion templates, but the idea is just to demonstrate how you can split your code into different methods and then call those methods when you need them.

By now you should have a good idea of how local classes work. The next problem for me was to remember how and where to put all the class definitions and procedural code. As always when I have to do repetitive things, I fix the problem with a template, which is included in the source download. I have tested this template with Clarion 5.5, 6.0, 6.1 and 6.2.

> **NOTE:** A commercial version of this template will also be available at ClarionShop

Using the template

Firstly you have to add the Global Application Template as per Figure 3.

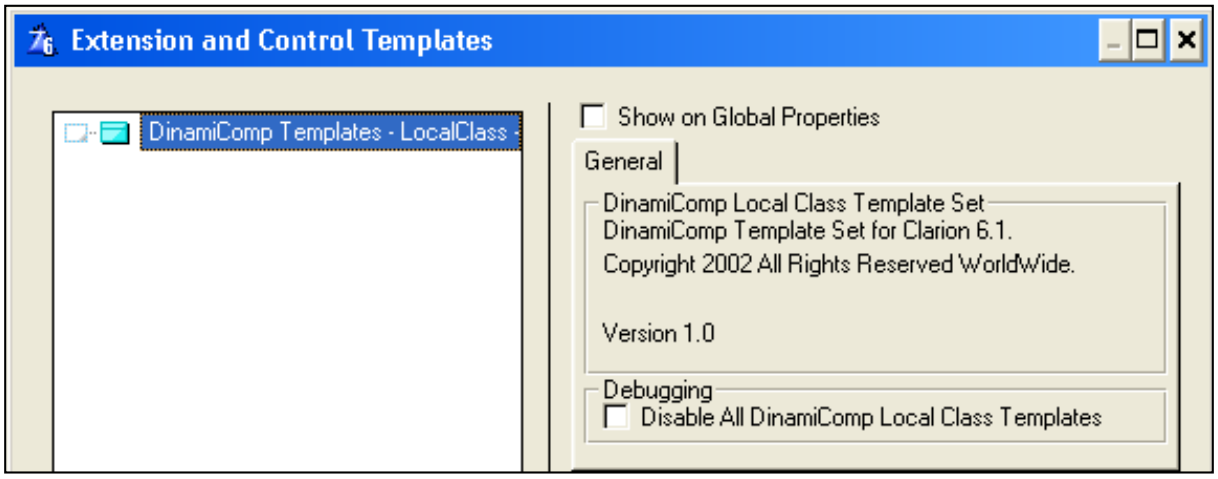


Figure 3. Global Extension for DinamiComp Local Calls template set.

The Global Template gives you the option to globally disable all of the template's code in all of the procedures in your application.

The next step is to add the Local DinamiComp Class Extension Template to the procedure where you want to use the Local Classes.

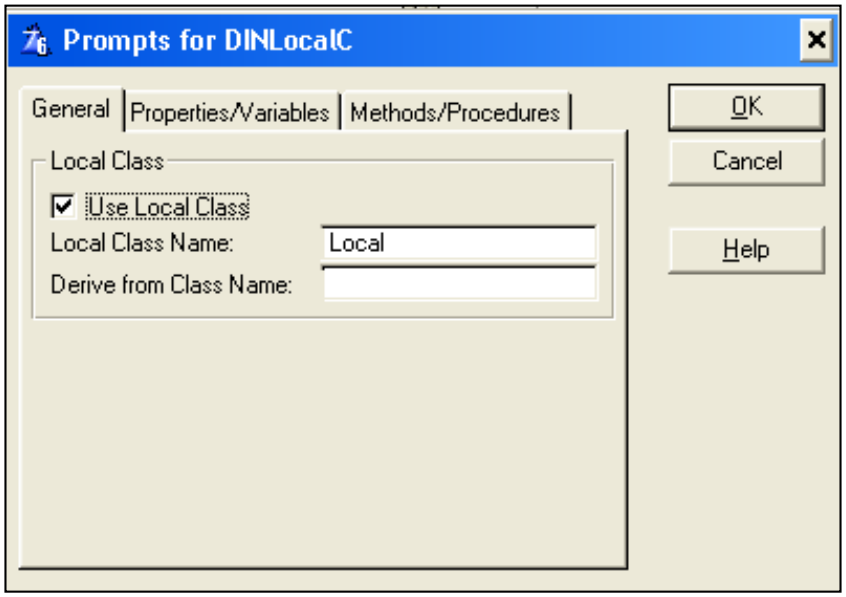


Figure 4. Local Extension for DinamiComp Local Calls template set.

All that needs to be entered on the General Tab is the class's name (defaults to `MyLocal`) and you can specify if you want to derive from another class.

The Properties/Variables tab is the place where you add all your properties that need to be global to the class.

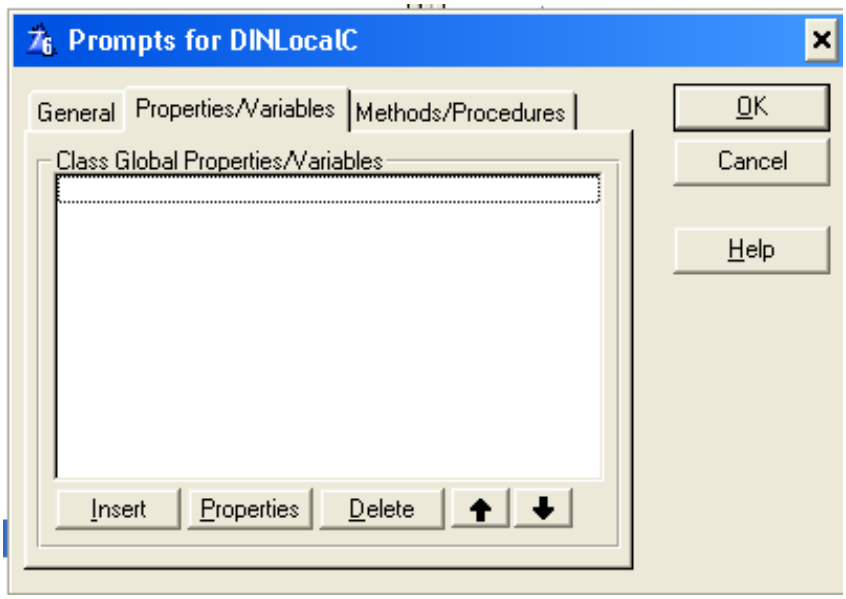


Figure 5. Properties/variables that is Global to the class

As Figure 6 shows, you have various options available to you when you add a "global" class variable. You have to give the property a name, choose between `Signed`, `String`, `CString`, `Long` and `Byte`. For `CString` and `String` you need to specify the size, and for `Signed`, `Long` and `Byte` you can specify an initial value. You can also define a property as dimensioned. The last option on the screen gives you the option to add a comment per property. The comment will be added to your source code.

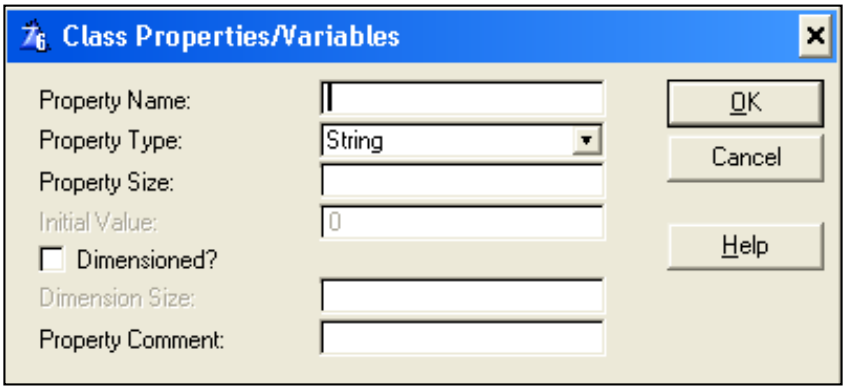


Figure 6. Properties for Global Class Variables

The last tab that needs to be populated is the Methods/Procedures tab. This is the place where you add all your methods for the Local Class. See Figure 7 for an example.

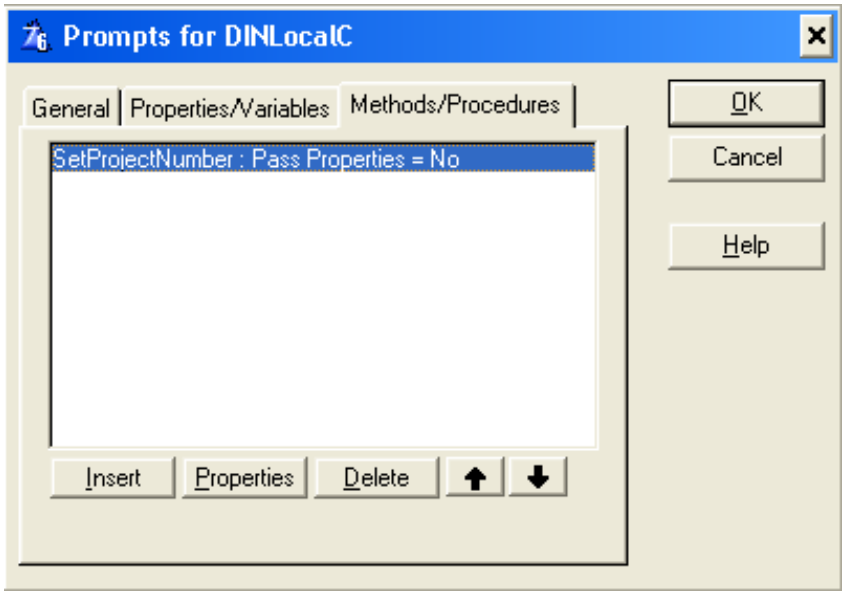


Figure 7. Procedures for the Local Class

You can add any number of methods to your local class by clicking on the Insert button. You will then be prompted with the screen as in Figure 8.

The first value to enter is the method's name, and after that you have to specify the method type (None, Derived, Virtual, Private). As well, you have to specify if this method is going to pass any variables.

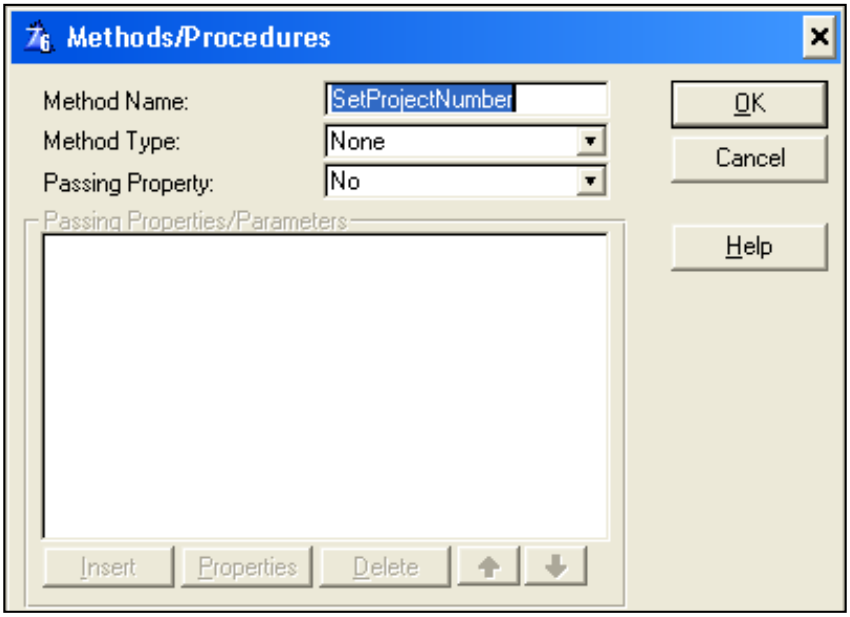


Figure 8. Detail for the Methods in your Local Class

If the method is going to receive parameters (see Figure 9) you need to give each parameter a name and specify the type. Please note that the template, as written, only allows you to pass parameters by value, so `CString` is not an option here as a `CString` can not (natively) be passed by value, only by address.

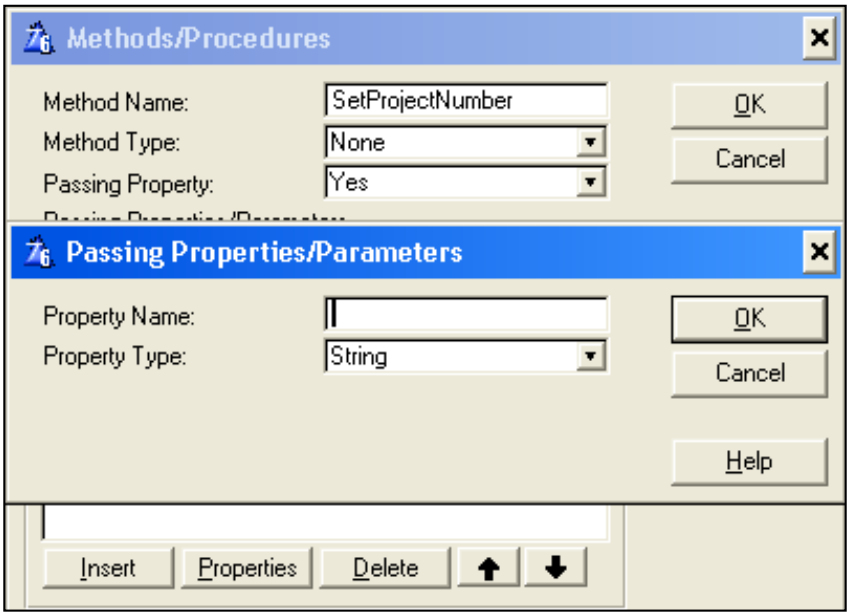
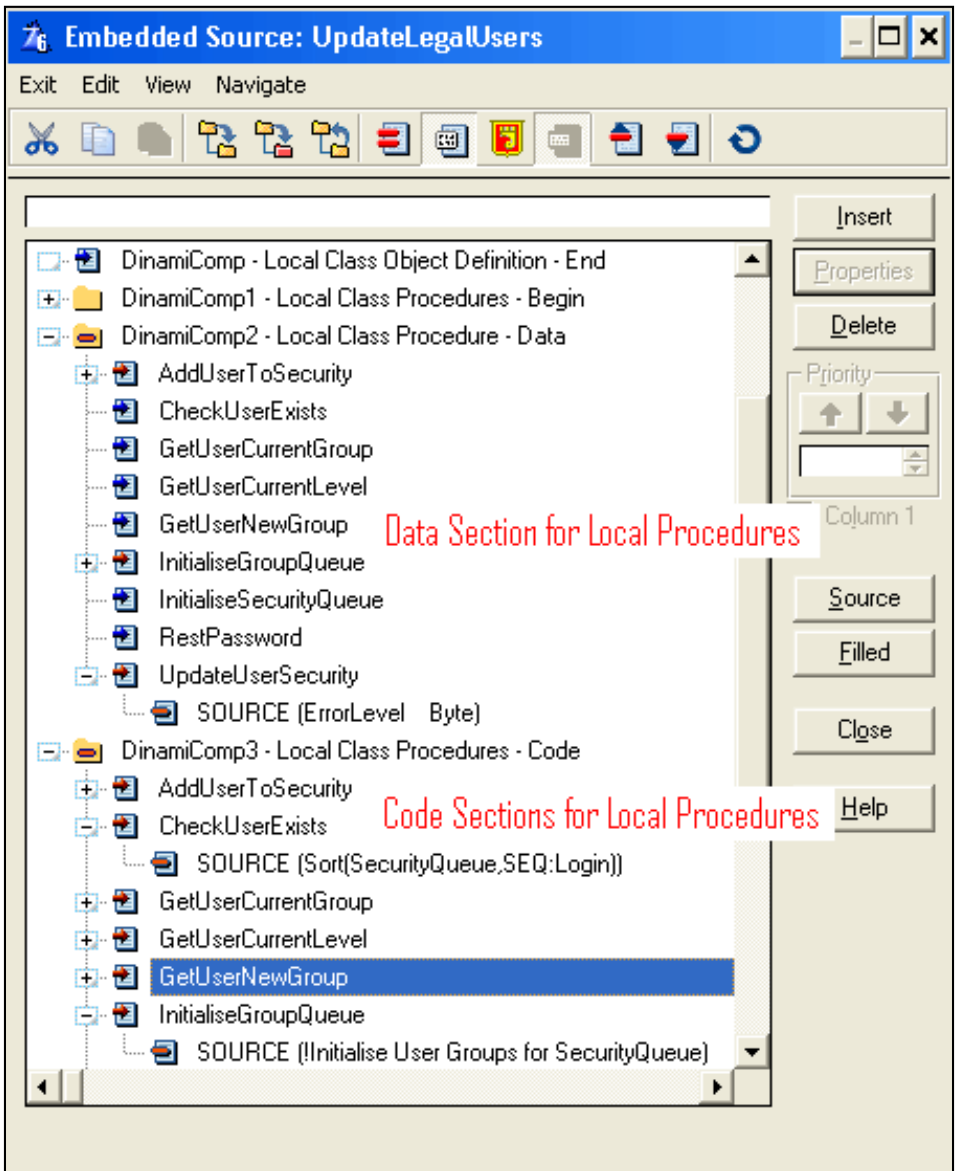


Figure 9Creating a property.

So what does this template do for you? It creates the Local Class Object Definitions (Procedures and Variables) as well as the necessary embed points for your Local Class Procedures. See Figure 10 for an example.



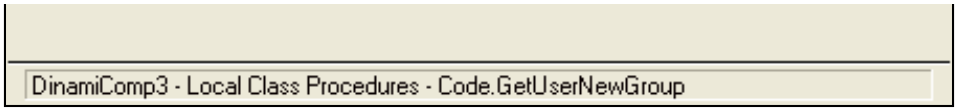


Figure 10. Sample of Embed points for all procedures added via DinamiComp Template

All that is left to do for you is to add the code to your procedures and call your procedures from your code.

Summary

In this article you have learned the advantages of changing your routines into local classes. You have also learned how to create and use local classes in your procedures.

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. He has been programming in Clarion since 1989, and holds B.Com and MBA degrees. In his spare time Nardus lectures Financial Management to B. Com Hons students at North-West University.

Reader Comments

Add a comment

- » In the Legacy templates prior to C6 the Local Procedures...
- » Thanks for that Carl, I can't remember when last did I use...

Clarion Magazine

Speeding Development With Defaults And Small Templates

by Phil Will

Published 2005-08-16

Over the last couple of weeks, I have been converting an old DOS application called Cybele to Clarion 6.2, something I had been thinking of doing for several years but had put off. Cybele had been doing it's thing without interruption since it was written in 1993, but the time came when there were enough issues and new possibilities to warrant an upgrade. As it worked out, I thought of two simple things I could do with templates and the defaults file that I was sure would save a tremendous amount of time both in development and future maintenance. While some of the specifics may not be relevant to others, some might find the concepts useful.

The application has a number of reports and a help system for each window. Each report has a standard layout with copyright information, a real estate property name, a management company name, a date and time when the report is run, range limit options that should be displayed, and page numbers. The progress window for each report needed an MDI attribute, a consistent font (8 pt. MS Sans Serif), space for a pause button control template and for a couple of range limit fields. The progress window, like other windows, needed to be named in a way that was consistent with menus, report names, and help topics. Obviously I need a way to set some default AppGen values.

Defaults file

The defaults.clw file found in the Clarion Libsrc directory has definitions for windows and reports which can be selected when you are creating new reports. In the past, I have usually ignored the fact that I could create my own definitions; I thought of reports as something that could be easily modified after adding them to a procedure. In this case, it occurred to me that I could customize these not only as standard layouts, but I could also include some application specific fields that I would otherwise have to add after they were selected.

Progress window

The first definition I created was for the progress window. For this to work with the ABCChain, I quickly discovered that the Progress:Thermometer control needed the `#ORIG()` attribute or else Clarion would display a generation error saying that the progress control cold not be found. If you examine the generating code, you will find it looking for the `%ControlOriginal` template symbol which references the contents of the `#ORIG` attribute. The `#ORIG` and `#SEQ` attributes also contain template instance references which are sometimes handy to know about if you're ever copying window structures from one procedure to another, or if you inadvertently delete something like a delete button and need to restore it with a template references.

```
!!> CYBELE Progress
ProgressWindow WINDOW('Progress...'),AT(,,167,102),|
```

```
    FONT('MS Sans Serif',8,,FONT:regular,CHARSET:ANSI),|
        CENTER,TIMER(1),GRAY,DOUBLE,MDI
PROGRESS,USE(Progress:Thermometer),VERTICAL,SMOOTH,AT(5,5,31,91),|
    RANGE(0,100),#ORIG(Progress:Thermometer)
STRING(''),AT(41,13,119,10),USE(?Progress:UserString),|
    LEFT,#ORIG(?Progress:UserString)
STRING(''),AT(41,31,119,10),USE(?Progress:PctText),|
    TRN,LEFT,#ORIG(?Progress:PctText)
BUTTON('Cancel'),AT(120,84,45,15),USE(?Progress:Cancel),|
    LEFT,#ORIG(?Progress:Cancel)
     END
```

If you're not familiar with the defaults file, note that the name for the window that appears in the window selection list is contained in the line beginning with `!!>`.

Report

For the default report, I wanted a consistent title format, a range/filter description, a run date and time, a property name, copyright information, and a management company name. These are shown in bold in the structure below. The typical report also totals entries in the detail, so I included a break structure as well.

```
!!> CYBELE (Paper size Letter - Portrait)
Report REPORT,AT(500,1500,7500,8750),PAPER(PAPER:LETTER),PRE(RPT),|
FONT('Arial',8,,FONT:regular),THOUS
       HEADER,AT(500,500,7500,1000)
         STRING(@D17),AT(333,135),USE(lDate),LEFT,#ORIG(lDate)
         STRING('Title'),AT(1188,73,6125,),USE(?pdTitle),TRN,CENTER,|
         FONT(,14,,FONT:bold),#ORIG(?pdTitle)
         STRING('Run:'),AT(10,125),USE(?pdRun),#ORIG(?pdRun)
         STRING(@T7),AT(396,313),USE(lTime),LEFT,#ORIG(ltime)
         STRING(@s80),AT(1188,313,6125,177),USE(lFromTo),CENTER,#ORIG(lFromTo)
         STRING(@s40),AT(0,552,2865,177),USE(CBC:Property),LEFT,#ORIG(CBC:Property)
         STRING(@N4),AT(7177,552),PAGENO,USE(?pdPageNo),LEFT,#ORIG(?pdPageNo)
         LINE,AT(0,1000,7500,0),USE(?pdLine7),COLOR(COLOR:Black),#ORIG(?pdLine7)
         STRING('Page:'),AT(6677,552),USE(?pdPage),#ORIG(pdPage)
         LINE,AT(0,1000,7500,0),USE(?pdLine2),COLOR(COLOR:Black),#ORIG(?pdLine2)
       END
ReportBreak BREAK(BreakField)
         HEADER,AT(0,0,7500,)
         END
Detail   DETAIL,AT(,,7500,)
         END
         FOOTER,AT(0,0,7500,)
         END
       END
       FOOTER,AT(500,10250,7500,750)
         LINE,AT(31,10,7438,0),USE(?pdLine5),COLOR(COLOR:Black),#ORIG(?pdLine5)
         STRING('Copyright'),AT(31,52),USE(?Copyright),TRN,#ORIG(?Copyright)
         STRING(@s40),AT(5375,52,,208),USE(CBC:Company),RIGHT,#ORIG(CBC:Company)
       END
       FORM,AT(500,500,7500,10000)
       END
     END
```

Figure 1 shows the resulting report in the report formatter.

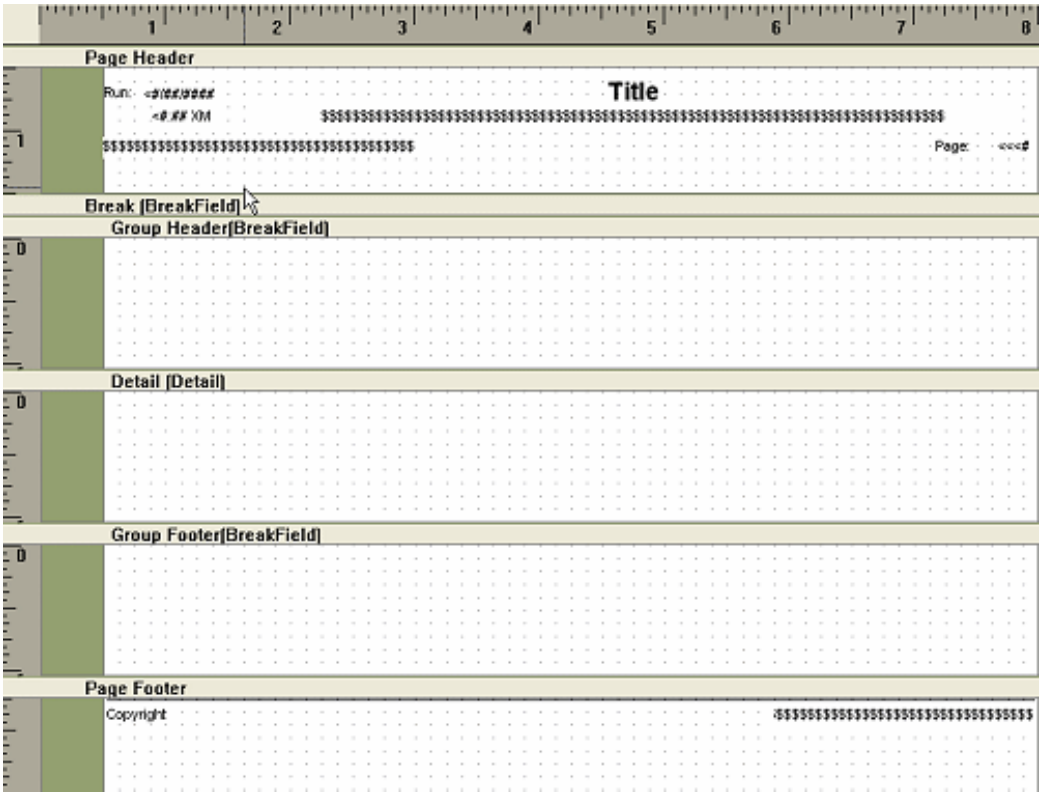


Figure 1. The new default report in band view

Templates using ProcedureDescription

Now I could populate the new default progress window and report in a report by creating a new report procedure and using the Window and Report buttons. The next logical step was to create an extension template that creates needed local date, gets the configuration file, assigns values to the date and time, and assigns a copyright string to the copyright control.

In addition, I wanted the template to handle assignment of a name for the report so that the proper name would appear in the windows print dialog. I also wanted the template to assign a report title, window title, and help topic to the progress window. I didn't see a need for these values to be different from each other, so decided to use the procedure description from the procedure properties for all four strings.

Report Extension and Year Include Files

The specifics of the template are shown below. It first populates local data. At the beginning of generation, it then finds the original equate names (`%ControlOriginal` or `%ReportControlOriginal`) for the copyright and title controls and assigns them to a new symbol in case the equates have been change – a good practice if the template might be used by other developers.

The template then inserts code into three embeds, one after the window has been opened, one in the procedure's `Init` method, and one after the report has been opened. In the first, the window title and help topic are assigned to the progress window using the procedure description (`%ProcedureDescription`). In the `Init` section, the date and time are assigned and the configuration file is opened and retrieved, here using another procedure call. In the `OpenReport`

embed, the report name and report title are assigned using the `%ProcedureDescription` string. To further simplify maintenance, the copyright uses two year equates, `FirstPublished` and `ThisYear` for copyright dates. `FirstPublished` is located in a global include file that contains a number of application specific equates. The `ThisYear` equate is in a file called Year.inc located in the libsrc directory, and is referenced by a number of my applications.

Figure 2 shows two reports the use procedure descriptions for titles and help topics.

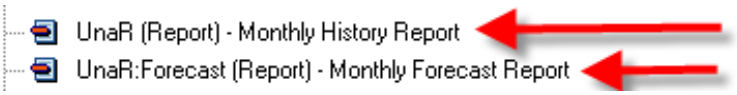


Figure 2. Using procedure descriptions for titles and help topics

Where reports had filters, I populated filter options on the progress window the old fashioned way, using a pause control which is modified to have the default attribute and always start paused, and some hand embed code to set the filters to the displayed range string..

```
#TEMPLATE(Cybele,'Cybele Application Templates'),FAMILY('ABC')
#!-----------------------------------------------------------------------------
#EXTENSION(CybeleReport,'Cybele Report Extension'),PROCEDURE,REPORT
#!-----------------------------------------------------------------------------
#LOCALDATA
lDate       LONG
lTime       LONG
lFromTo     STRING(80)
BreakField  BYTE
#ENDLOCALDATA
#!-----------------------------------------------------------------------------
#ATSTART
  #DECLARE(%pdCopyRight)
  #DECLARE(%pdReportTitle)
  #FOR(%ReportControl)
    #CASE(UPPER(%ReportControlOriginal))
    #OF('?COPYRIGHT')
      #SET(%pdCopyRight,%ReportControl)
    #OF('?PDTITLE')
      #SET(%pdReportTitle,%ReportControl)
    #ENDCASE
  #ENDFOR
#ENDAT
#!-----------------------------------------------------------------------------
#AT(%WindowManagerMethodCodeSection,'Open','(*Window pWindow,<*Window
pOwner>)'),PRIORITY(7500)
  #IF(%ProcedureDescription)
pWindow{PROP:Text}='%ProcedureDescription'
pWindow{PROP:Hlp}='%ProcedureDescription'
  #ENDIF
#ENDAT
#!-----------------------------------------------------------------------------
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(7500)
lDate=TODAY()
lTime=CLOCK()
IF CBCS:Open()
  MESSAGE('Unable to open configuration file','Error',ICON:Exclamation)
```

```
  RETURN Level:Notify
END
#ENDAT
#!-----------------------------------------------------------------------------
#AT(%WindowManagerMethodCodeSection,'OpenReport','(),BYTE'),PRIORITY(6300)
  #IF(%ProcedureDescription)
%Report{PROP:Text}='%ProcedureDescription'
  #ENDIF
  #IF(%pdCopyright)
%Report$%pdCopyright{PROP:Text}='PDLIM <0169> Copyright '|
    &FirstPublished&CHOOSE(ThisYear<>FirstPublished,'-'&ThisYear,' ')|
        &' ProDomus, Inc. All rights reserved.'
  #ENDIF
  #IF(%pdReportTitle)
%Report$%pdReportTitle{PROP:Text}='%ProcedureDescription'
  #ENDIF
#ENDAT
#!-----------------------------------------------------------------------------
```

Window extension

The window extension is similar to the report extension in terms of assigning the procedure description to the window title and help topic. In this case, I added a prompt to generate opening the configuration file only where needed. I also have an option to turn off resizing, as resizing is rarely needed in this particular application. Normally, all new windows are generated with resizing turned on.

```
#EXTENSION(CybeleWindow,'Cybele Window Extension'),PROCEDURE,WINDOW
#DISPLAY('Assigns Procedure Description to Window Title and Help Topic')
  #PROMPT('Open CBC',CHECK),%OpenCBCCk,AT(10)
  #PROMPT('Turn Resize off',CHECK),%pdResizeOff,DEFAULT(1)
#!-----------------------------------------------------------------------------
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(7500)
  #IF(%OpenCBCCk)
IF CBCS:Open()
  MESSAGE('Unable to open configuration file','Error',ICON:Exclamation)
  RETURN Level:Notify
END
  #ENDIF
#ENDAT
#!-----------------------------------------------------------------------------
#AT(%WindowManagerMethodCodeSection,'Open','(*Window pWindow,<*Window
pOwner>)'),PRIORITY(7500)
  #IF(%ProcedureDescription)
pWindow{PROP:Text}='%ProcedureDescription'
pWindow{PROP:Hlp}='%ProcedureDescription'
  #ENDIF
  #IF(%pdResizeOff)
pWindow{PROP:Resize}=FALSE
  #ENDIF
#ENDAT
#!-----------------------------------------------------------------------------
```

Internationalization

Internationalization is not needed for this application. If it were a requirement and if you were using the `TranslatorClass`, you would find that the strings are in fact assigned before windows and reports are translated by the `TranslateWindow` method. The only addition that might be considered is to create a TRN (Translation File) with equates for the copyright string. If you were translating Help topic strings or allowing users to customize stings, then you would also need a way to distinguish help topics from other strings in the translation file with the same text.

Conclusion

I can't say specifically how much time this saved except that it was significant and that it eliminated a considerable amount of otherwise tedious effort. The application also has a consistency in naming that has been hard to match and maintain in other applications. I would hope that the few of these tips such as the use of the `#ORIG` and `#SEQ` attributes and year equates might help other members of the Lazy Programmers Society.

If you are using an early version of Clarion, you will need to use a different embed from the `Open(*Window pwindow,<*Window pOwner>)` which is a later addition to the `WindowManager` library.

Download the source

Philip S. Will is President of ProDomus, Inc, a SoftVelocity Third Party Accessories Partner and Clarion applications developer. He has been a presenter at several Clarion conferences, and has published articles on Internationalization and template writing. His principal third party products include PD Lookups, PD Translator Plus, and PD 1-Touch Date Tools. Philip has been coding in Clarion since 1991.

Reader Comments

Add a comment

Clarion Magazine

Reports: OOP, ABC and Ignoring Templates (Part 3)

by Steven Parker

Published 2005-08-19

This may be a record for hiatus between article parts. Reports: OOP, ABC and Ignoring Templates (Part 2) appeared three years ago, in August 2002. A technique mentioned there, entirely in passing, has come back to haunt me. So another part is necessary.

In that article, I described the `OpenReport` method as "the point at which I can make final adjustments to report properties." The "final adjustment" I had been discussing at the time was a title page (or band). Almost casually, I pointed out that I could programmatically set form feed after group breaks at this point also:

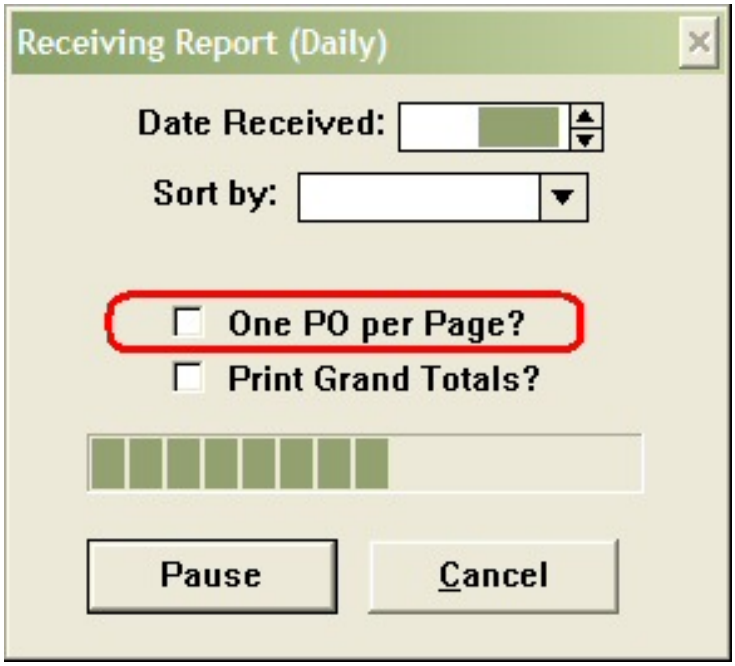


Figure 1. Setting PageAfter at Runtime

The code to effect this is:

```
IF ~ReturnValue
  SetTarget(Report)
  If OnePOPerPage
    ?break5{Prop:PageAfter} = True
  Else
    ?break5{Prop:PageAfter} = False
  End
  SetTarget
End
```

There is a problem, however, and it took my users three years to discover it. The problem is that when I set `OnePOPerPage`, forcing `PageAfter` on the group footer (or any band, I suppose), I get a blank page at the end of the report.

Because I always tested the report with the "Print Grand Totals?" option checked (see Figure 1), I never noticed a blank page. ("Print Grand Totals?" causes a band to be printed in `.AskPreview`. Using this method for "post loop" processing is discussed in Part 2.)

Once I'd confirmed this, I did a fast and dirty test. I removed all my code and turned on the `PageAfter` property in the template.

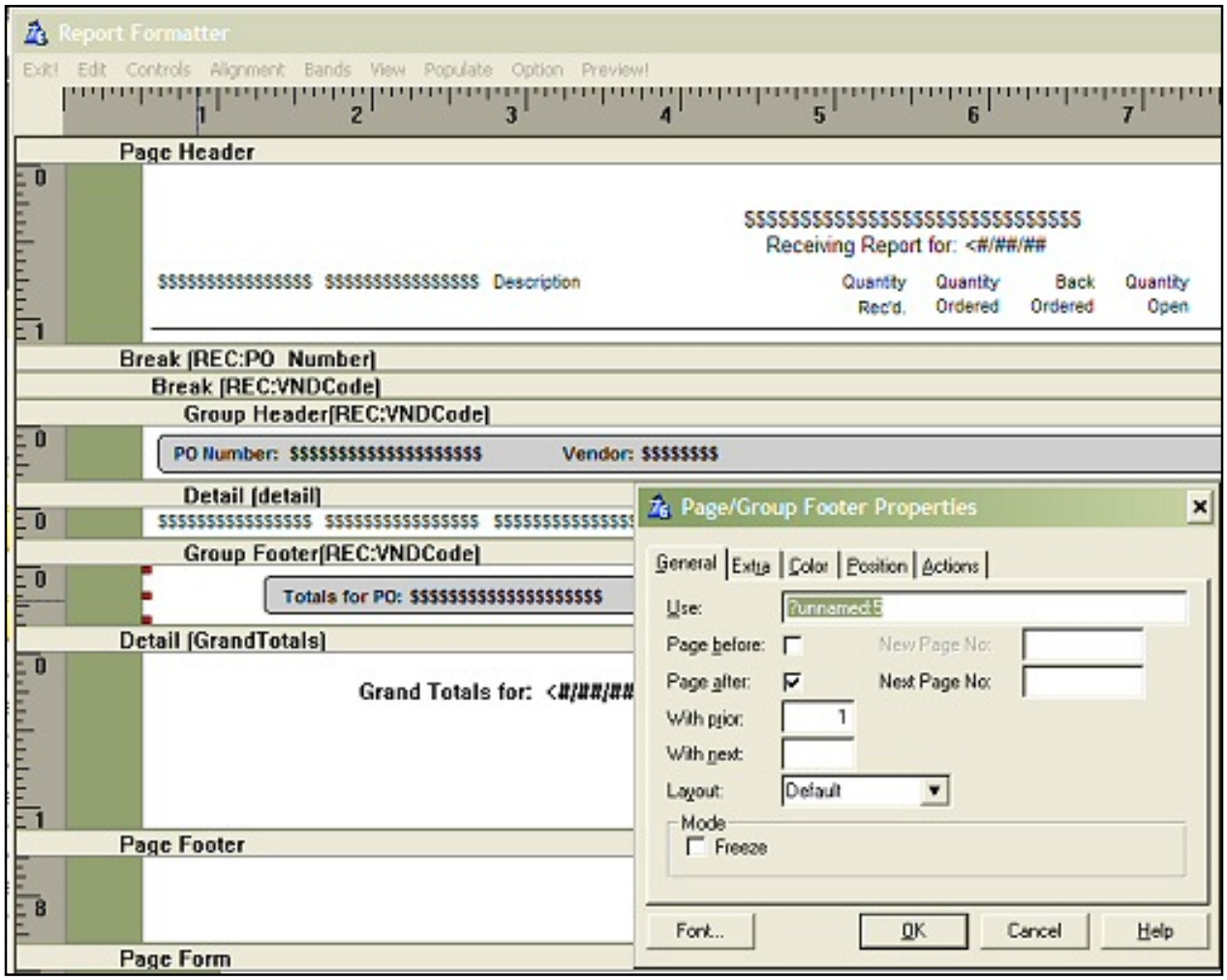


Figure 2. Template setting PageAfter

Sure enough, an extra page, blank except for the page footer.

On reflection, I understood that my grand totals band was *on* the extra page. In fact, it ought to have been on the page containing the last set of records. So this "problem" had actually been staring at me all along.

Can't go wasting tree limbs, now can I? So, what can I do? I need to turn off `PageAfter`, that's what I can do.

How to turn it off is easy:

```
?break5{Prop:PageAfter} = False
```

The question is "where?"

The answer, too, is easy, once the problem is subjected to analysis.

Since a report is "just" a `Print` statement inside a loop, were I hand coding, I would be doing something like:

```
Loop
  If Access:fileLabel.Next() = Level:Notify
    ?break5{Prop:PageAfter} = False
  End
  Print …
End
```

In other words, I want to test for `Next` failing. Unfortunately, the ABC templates provide three `.Next` methods:

1. `ThisWindow.Next(),Byte` *(Window Manager)*
2. `ThisReport.Next(),Byte` *(Process Manager)*

and

3. `ThisReport.Next PROCEDURE(BYTE ProcessRecords),Byte`

In the embed tree, `ThisWindow` is "Window Manager:"

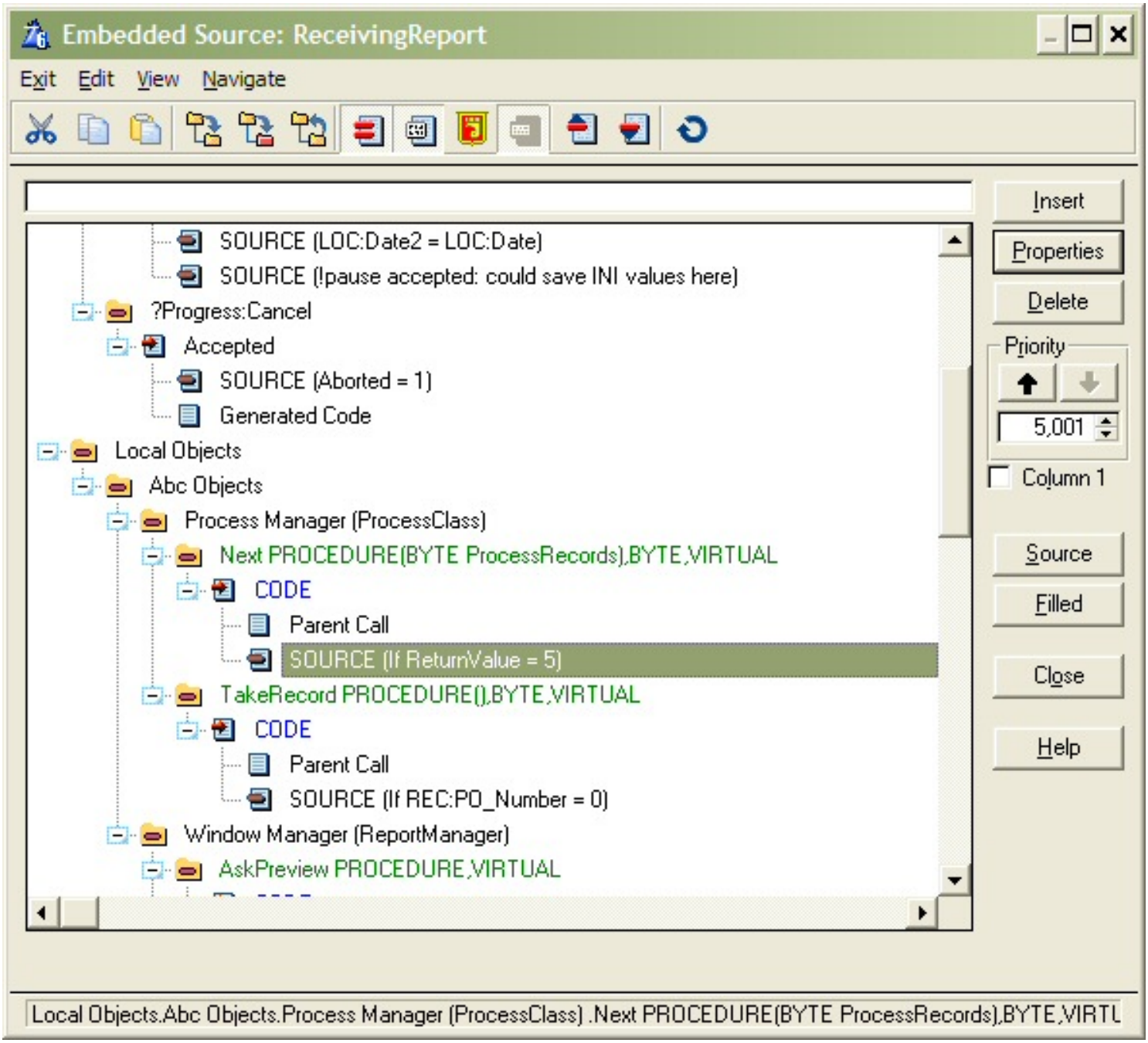


Figure 3. ThisWindow.Next() in the embed tree

ThisReport is "Process Manager:"

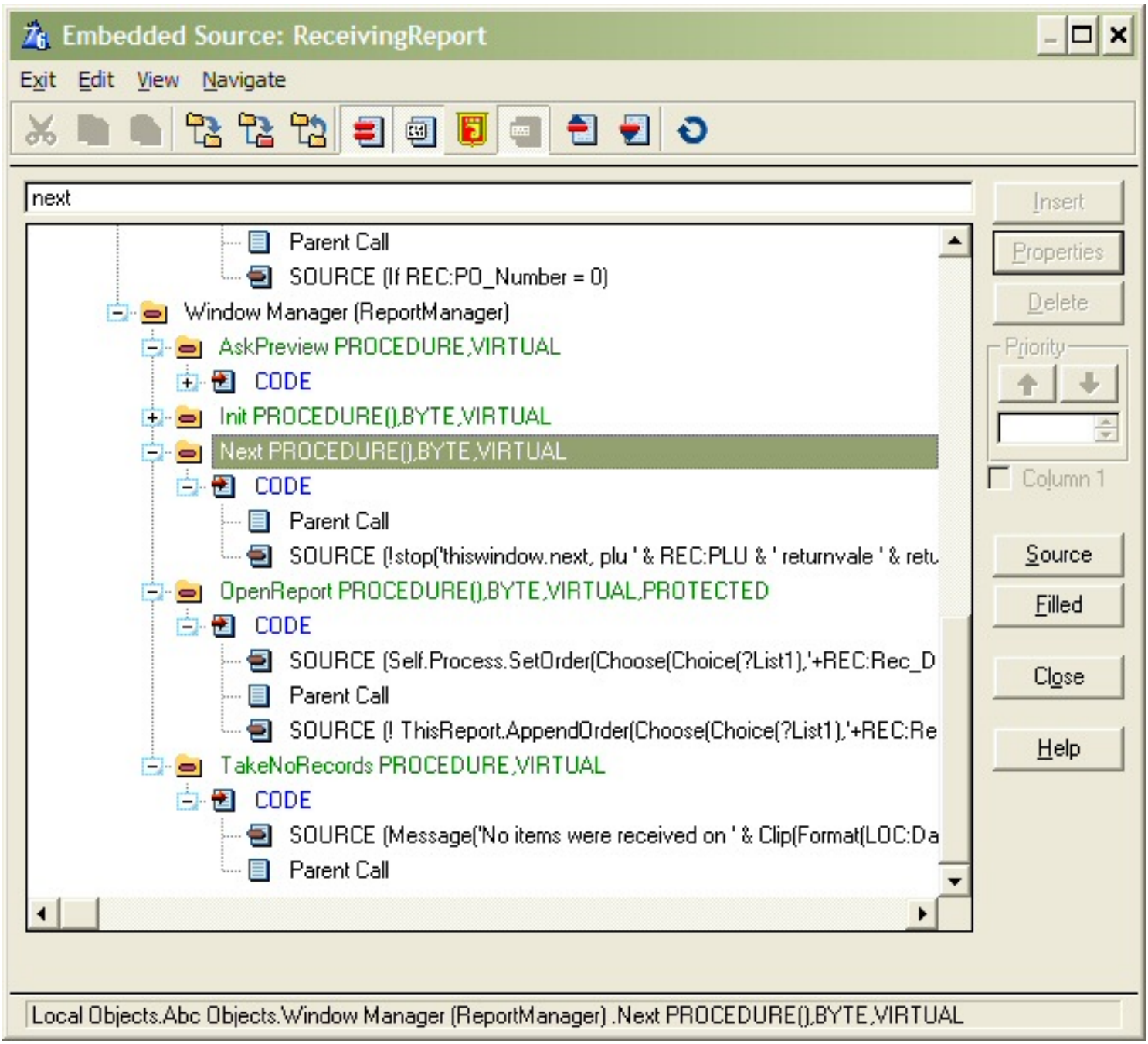


Figure 4. ThisReport.Next() in the embed tree

By judicious debugging (`Stop(ReturnValue)`) in each of these embeds, I discovered that 2 doesn't work. Better, 3 tells me that `Next` failed before 1.

So, in the embed in Figure 4, I insert:

```
If ReturnValue = Level:Notify
   SetTarget(Report)
   If OnePOPerPage
     ?break5{Prop:PageAfter} = False
   End
```

```
    SetTarget
End
```

Et voilá, no more extra page.

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

Clarion Magazine

Fetching A Web Page With WinInet

by Skip Williams

Published 2005-08-23

Recently I needed to look up about 3,000 or so IP addresses to discover the location of customers who were using our online system. All it would take was a simple program to take the list of IP addresses, and, using several available websites, pass a URL, get a web page and parse it for the IP location.

Being a member of the Lazy Programmers Society, I used a DLL that I had found in the newsgroups. This DLL would take a URL, fetch a webpage and write it to a text file. It worked great, until I tried to use it with Clarion 6.2. The DLL was compiled in 6.0 and I didn't have the source. Bummer!

"Self, no problem", I said to myself. I have NetTalk and it does this. Indeed it does, but after working with it for an hour or two and trying to get it to fit into a hand coded program to loop through a file and fetch 3,000 or so web pages, I was frustrated. NetTalk is a fine third party package for doing just about anything net-related, but it is geared to an interactive AppGened application, which mine was not.

The WININET DLL

As part of the Windows operating system, Microsoft provides a very powerful DLL called WinInet. It provides functions to communicate with Web, FTP and Gopher servers from a client application with all the features you would ever want or need.

I have been using Microsoft's WinInet.DLL for almost 10 years to do several mission critical client/server applications in Clarion, so I decided to just get back to basics and write a very simple DLL in Clarion to fetch a web page. It was to be synchronous with a minimum of error checking. It is easy enough to do asynchronous calls with WinInet and extensive error checking as well, but I wanted to keep it simple.

The resulting WebFetch.DLL uses only four basic WinInet functions:

- InternetOpen
- InternetOpenURL
- InternetReadFile
- InternetCloseHandle

The WebFetch DLL

The Clarion prototypes for the functions are straightforward.

```
module('wininet.lib')
   InternetOpen(*lpcstr,dword,*lpcstr,dword,dword)|
          ,handle,pascal,raw,NAME('InternetOpenA')|
      ,DLL(1)
   InternetOpenURL(handle,*lpcstr,dword,dword,dword|
      ,dword),handle,pascal,raw,name('InternetOpenURLA')|
          ,DLL(1)
   InternetReadFile(handle,*lpcstr,dword,*dword)|
      ,handle,pascal,raw,name('InternetReadFile')|
      ,DLL(1)
   InternetCloseHandle(handle),pascal,raw|
     ,name('InternetCloseHandle'),DLL(1)
End
```

The code for the WebFetch DLL is also fairly simple:

```
Code
OpenType = INTERNET_OPEN_TYPE_PRECONFIG
hsession = InternetOpen(UserAgent,OpenType,ProxyName,zero,zero)
if hsession = 0
   Return(1)
end
```

`InternetOpen` initializes a WinInet session with your application. You specify what kind of open you want to do. I recommend that you always use the `INTERNET_OPEN_TYPE_PRECONFIG` equate (see the source for the actual value), as this will use Internet Explorer's settings from the registery, and is helpful if you might be going through a proxy server. The User Agent parameter is where you specify the browser name to the server. You can specify whatever you want to here (sorry, "Mozilla" is already taken).

```
dwFlags = NoCaching
hconnect = InternetOpenURL(hSession,URL,zero,zero,dwFlags,dwContext)
if hconnect = 0
   Return(2)
```

```
end
```

`InternetOpenURL` connects to the specified service and URL. Since this WinInet function can be used for FTP and even Gopher, you need to make sure that you always append the URL with `HTTP://` to let `InternetOpenURL` know that its an HTTP site that you are opening. Also, setting the `dwFlags` parameter to the `NoCaching` equate should cause the function to retrieve a new copy of the page each time.

As an aside,the Wininet `InternetConnect()`, `HTTPOpenRequest()`, and `HTTPSendRequest()` APIs can be used instead of the `InternetOpenURL()` if you need to send a userid and password to your firewall, POST data to the webpage, or do additional checking for errors or data and headers being retrieved.

Notice that the first parameter is the handle passed by the `InternetOpen` function. The handle returned by `InternetOpenURL` will be used in the next WinInet function, `InternetReadFile`:

```
Loop
   hReadFile=InternetReadFile(hconnect,HTTPbuffer,bufferlen,bytesread)
   if hReadFile = true and bytesread = 0
      ! Accumulate the contents of HTTPbuffer[1:bytesread]
      ! in your buffer and try to read additional data...
   end
   webpage = webpage & HTTPbuffer[1:bytesread]
end
```

`InternetReadFile` will return the web page specified by the URL in the previous function. If `HTTPbuffer` is large enough, all data is returned by the first invocation of the `InternetReadFile`, however, if `HTTPBuffer` is smaller than the web page you are fetching, you must continue to call `InternetReadFile` until it returns a handle of true and bytes read of zero as the data is broken into chunks to fit the size of your buffer (`bufferlen`) and sent back in several transmissions . The content of `HTTPBuffer` is placed (concatenated) into second parameter variable as it is retrieved.

After you have received the web page, the only cleanup you need to do is close the connection and then close the session

```
internetclosehandle(hconnect)
Internetclosehandle(hsession)
```

Then the DLL returns to the caller:

```
return(returncode)
```

Included in the download source are two files that you need when compiling the DLL; wininet.lib, which contains all of the WinInet functions, must be added to the WebFetch.prj under Library, Object,

and Resource files. WebFetch.exp exports the `GetaPage()` procedure so that it can be called from an EXE and just needs to be present in the your application's subdirectory when WebFetch.DLL is linked..

Another important consideration is to link the WebPage.DLL in local mode if the application that uses it is linked local. It is usually not a good idea to mix the link modes between a Clarion EXE and a Clarion DLL

Using the WebFetch DLL

To use the WebFetch DLL in your application, simply add the following code into the Inside Global Map embed:

```
Module('WebFetch.lib')
   GetaPage(string pURL, *string pTextbuf),long
End
```

Next, add the file WebFetch.lib to your project under Library, Object, and Resource files.

Then, to get a webpage, call this code:

```
AURL = 'http;//www.anydomain.com'
rtn# = GetaPage(aURL,ThisPage)
```

If the DLL returns 0, the retrieved webpage will be in the second parameter (`ThisPage`). If a non-zero is returned, a 1 indicates that the `InternetOpen` failed, a 2 indicates that the `InternetOpenURL` failed, probably due to an invalid URL, and a return code of 4 indicates that the retrieved web page is larger than the target variable (`ThisPage`).

WebFetch can be compiled as an .EXE, in which case the code in the main procedure will be executed and it will run standalone, giving an easy way to test the DLL. If it is compiled as a .DLL, that code is ignored and can be removed if desired.

The download includes the source code and project files for WebFetch along with an AppGen test program and a hand coded test program and project file.

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

- » Skip - This is very cool. Thanks for writing it.
- » Please note that you must compile the PRJ first to create...

Clarion Magazine

Mixing Clarion with .NET, Part 1

by Wade Hatler

Published 2005-08-31

I'll say it right up front. I like the Clarion language a lot, and I always have. I've been using it along with a half-dozen other languages since the dawn of time, and I find Clarion's syntax and template-driven code paradigm to be extremely productive. What I've always hated about it is that it never plays nicely with the rest of the world. COM interop has always been a nightmare. Third party COM libraries are nearly non-existent, and when you do get it to work with other systems, it's seldom smooth or seamless. Such a simple task as embedding an Active-X control, or using any of the many of the thousands of COM utilities available, has always been difficult. Clarion is also abysmally bad at text processing and several other classes of problem that I seem to get a lot, which has always forced me to write utility programs in better suited languages.

That's all changing now though. It's a not very well-kept secret that Clarion is moving towards the .Net world; with the upcoming .NET-based IDE, you'll use Clarion 7 to create Win32 apps, and Clarion.NET to create .NET apps. Clarion.NET will breathe a whole new life into the Clarion language. The .NET framework is extremely large, growing and very well supported. Right out of the box the framework has hundreds of classes that are well thought out and which integrate seamlessly with .NET applications. These classes cover a very wide range of the things you might want to do, such as directory services, graphics, internet connectivity, security, XML, message services, web services, really good text processing and much, much more. You can find tons of examples all over the place. Most of the .NET languages aren't structurally a lot better than Clarion, but the framework is extremely nice.

If you're eager to start, you don't have to wait for Clarion to get their next version out. With a bit of interop code, you can get many of the benefits of the .NET world right here and now in Clarion 6. You'll find that Clarion and .NET make a powerful combination.

This series will cover in detail how to interoperate between Clarion and .NET. I'll cover calling .NET from Clarion, calling Clarion from .NET, and embedding WinForms windows and controls into Clarion applications. Most of it isn't all that hard to do once you know the secrets.

I'll be demonstrating how to do all of these things, and I'll provide a couple of utilities to do some of the grunt work. I'll use Visual Studio 2003 and Clarion 6 for my discussion, but those are not limiting factors. You can use any version of Visual Studio .NET, and any version of Clarion after Clarion 4. All of my examples will be in C#, but again you aren't limited to C#. Any .NET language will probably work with minor syntax changes. Our company uses mostly C#, but I'm constantly tempted towards VB.NET with its more Clarion-like syntax.

.NET Versions

There are currently two versions of .NET and Visual Studio (VS) available. Visual Studio 2003 along with the .NET framework 1.1 is the current active shipping release. It's very stable and has been around for some time. Visual Studio 2005, with the .NET 2.0 framework is currently in Beta II phase and will ship on November 9. If you can possibly wait until after November to ship real code to customers I strongly recommend Visual Studio 2005. There are a lot of new and cool things in 2005 and the C# language which would make it a compelling upgrade, particularly since you'll probably do it in six months anyway. The real clincher though is that the debugger in VS 2005. You can single-step and use all the features of a *real* debugger when calling .NET from Clarion. I've never been able to get it to work with 2003. It may be a solvable problem, but I don't feel compelled to waste any time on it. If you have to go with 2003, fear not. Everything I'm going to show you will work just fine in 2003, but your debugging experience will be suboptimal.

Overview

Calling Clarion from a .NET language is fairly simple. You use the .NET interop classes in a manner similar to how you would make native API calls. The only real complexities are in bypassing Clarion's unusual parameter passing mechanism, which I'll cover in Part 3, and in passing `GROUP`s, which I'll cover in Part 4.

To call any .NET procedure from Clarion, you have a few choices. You can expose any .NET class to COM, which makes it visible to any COM client. Once you've done that, you can either use Clarion's OLE control or write some Clarion wrapper classes. The latter is better, but harder to do.

You can also make a Managed C++ wrapper, which is stored inside your .NET assembly. You then call this wrapper the same way you would call an API function. This is less flexible and a tiny bit harder to do, but it works much faster than the COM option so you may want to consider it if you're calling something a lot of times.

Along the way, I'll describe two of the ways you can embed WinForms windows inside of Clarion applications, as well as embedding WinForms controls into Clarion windows.

A simple Active-X wrapper

The simplest way to call a method in .NET is to make a COM wrapper around the .NET class, and then use Clarion's COM capabilities to call that method. First, I'll demonstrate using Clarion's `OLE` control, which is adequate for simple tasks, and much better than you might expect if you used it prior to Clarion 6.

Start out by making a really simple .NET project.

In Visual Studio, Use File | New Project.

Pick Class Library from the Visual C# tab, and pick a good folder to place it in.

I also recommend you click the More button and turn the Create directory for solution checkbox off. If you don't, you end up with an annoying double directory structure.

Click OK, and you'll have a new solution with the file `Class1.cs` opened. You should immediately rename it, and change the name of the class and its constructor. I renamed it to `Interop01.cs`, with a namespace of `Interop01` and a classname of `Interop01Class`. (In VS 2005, if the first thing you do is rename the file, the class and its constructor are renamed automatically).

Whenever you start a new C# project, you'll want to immediately add a few `#using` directives to the top of the file, and if you want to do anything involving Windows you'll need a few references. For this demo:

In the Solution Explorer, right-click on References and select Add. From the list, add a reference to `System.Windows.Forms.dll`. This has a similar effect to adding a LIB to a Clarion project, and adding the associated INC file at the same time – you're including the `Systems.Windows.Forms` classes in your project. You'll probably want this in every project you ever do until the end of time, even if it's a console project, because it includes the ubiquitous `MessageBox` class. It never hurts to add it, because .NET will only pull in the assemblies you actually reference.

At the top of the file, add a few `using` directives. These aren't strictly necessary, but they'll save you a lot of typing. Without the `using` directive, you'd need to type the fully qualified path name for every method or property you ever access (e.g. `System.Windows.Forms.MessageBox.Show`). With it you need only type `MessageBox.Show`. Only the first three are absolutely necessary for this example. The remaining two are something you'll need about ten minutes after you start writing something real. Over time, you'll probably come up with your own list of `using` directives.

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Collections;
using System.IO;
```

Next, make a method to call for demonstrations. Start with something simple that doesn't do much. For testing purposes, just make a method that shows something with `MessageBox.Show`. My test method looks something like:

```
public void CallDotNet(string message)
{
   MessageBox.Show(String.Format("Message Box in .NET showing {0} from Clarion.",
message));
}
```

If you build the project, you'll have a very simple .NET assembly that you could use with any .NET language. I suggest building it now just to be sure you have everything.

Note that C# doesn't have the concept of raw procedures like Clarion does. Everything has to be a method belonging to a class. This makes some sense, as it helps to eliminate procedure name conflicts. Make a good unique classname, and the chances of duplication are very low. To make the analog of a Clarion procedure that doesn't require an instantiation of an object, you create a *static method*. This produces something just like a Clarion procedure, except you need the classname to reference it.

Make a .NET assembly into a COM component

Now that you have a valid .NET assembly, you need to make it a COM component. It's really easy to do, but Microsoft is trying really hard to kill COM, so they don't give you plain instructions. There are a few steps you'll need to take, and later on a few deployment choices you'll have to make.

The first decision involves where to put your DLLs. Unlike earlier COM components, you can't just put the DLLs anywhere you like. The .NET designers deliberately removed that capability to help eliminate versioning conflicts and

the infamous DLL Hell. Your DLLs have to be either in the same folder as the EXE file that's using them, or they have to be in the *Global Assembly Cache* (GAC). The GAC is a good place for things that you'll use all over the place such as global tools, but I generally put the DLL in the same folder as the Clarion executable. For this example, I'll put the DLLs in the same folder as the Clarion project, but while I'm at it I'll do a couple of things that may be necessary if I decide to move it to the GAC later.

There are two ways to put the DLL in a different location than its default. You can either have .NET make them in your output folder in the first place, or you can copy them after they're built. I generally prefer to just put them in the output folder in the first place. The only time this doesn't work well is if you have a .NET *solution* with multiple *project* files that reference each other. VS 2003 has problems with this, so you have to use the copy method.

For this example, I'll be making the Clarion components in the same folder as the Solution file.

In VS, execute Project | Properties to change the properties for this project.

Under the Build entry, change Output Path to `.\`. That will place it in the same folder as the solution. In practice, you would probably pick a different folder where your Clarion project generates its output. Note that the IDE will always change a hard-coded path to a relative path.

Just below the Output Path, set Register for COM Interop to True. This will register your assembly for COM when you do a build.

This will generate the files directly in the same folder as your solution. If you want to copy the files instead:

Under the Common Properties folder, find Build Events and change it to `copy /y $(OutDir)* $(ProjectPath)`. This does the same thing using a copy. Naturally, you can change the destination path, or the file pattern to copy as appropriate.

Next, change some of the project attributes:

In the Solution Explorer, open the `AssemblyInfo.cs` file. This file contains information that controls how the .NET assembly is built.

Locate this line:

```
[assembly: AssemblyVersion("1.0.*")]
```

and change it to

```
[assembly: AssemblyVersion("1.0.0.0")]
```

At the bottom of the file, locate:

```
[assembly: AssemblyKeyFile("")]
```

and change it to

```
[assembly: AssemblyKeyFile(@"..\..\Interop01.snk")]
```

If you don't change the `AssemblyVersion` entry, .NET will make a new side-by-side version and register that new

version in your registry every time you do a build. Eventually you'll have hundreds of them. Changing it as shown will make just a single version that's updated with each build It's up to you to change the number when you make a version change that matters. You should change the version number any time you make a breaking public interface change after you've deployed your assembly.

The `AssemblyKeyFile` entry makes what's called a *Strong Key* for the assembly. This is something used to for digitally signing the assembly. You don't actually need it unless you deploy to the GAC, but it's always a good idea so I just do it for all projects. Note the `@"` in the string. This is a C# convention that means the string is interpreted like a Visual Basic string. This makes it so you don't have to double up the backslashes. All characters go in the string exactly as entered except double-quotes. To get a double quote, you put in two of them.

The next step is to indicate which classes should be exposed to the COM interface. You do that by *decorating* your declaration with *attributes*. Attributes are special instructions that can attach metadata to any program entity. That metadata can be used for any number of things, including instructing the compiler or linker to generate a COM wrapper for something. In this case, we want to tell .NET to generate a COM wrapper for a particular class:

In your source code, Right above the declaration for any class that you want to expose to COM, add this line:

```
[ClassInterface(ClassInterfaceType.AutoDual)]
```

Once you do that, all public methods, fields or properties for the class are exposed to COM. You can add similar attributes to demote certain of these items so they're public for .NET but private to COM. There's quite a bit of flexibility in what gets exposed to COM and what isn't.

Once you've made these changes, the complete source file looks like:

```
using System;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Collections;
using System.IO;

namespace Interop01
{
   [ClassInterface(ClassInterfaceType.AutoDual)]
   public class Interop01Class
   {
      public void CallDotNet(string message)
      {
         MessageBox.Show("Message Box in .NET showing " + message + " from
Clarion.");
      }
   }
}
```

Make the support files

If you try to build the solution after this change the build will fail because it couldn't find the strong encryption file specified in `AssemblyInfo.cs`. You'll need to make that using some command line tools.

If you have the full retail version of Visual Studio, open a command prompt, using the "Visual Studio xxx Command

Prompt" shortcut, which you'll find in the Start Menu under "Visual Studio Tools". This command executes a batch file that updates your path to find the .NET command line tools, and sets some environment variables for the command line compiler. If you're using the free Visual Studio Express version, download `sn.exe` here, and place it somewhere in your path.

Navigate to your solution directory, and execute this command, to generate the missing Strong Name file:

```
sn -k Interop01.snk
```

If all went well, you should be able to build the project.

You'll notice that I made the `.snk` file the same as my assembly name. Normally, I just make a single `CompanyName.snk` file, put it in a hard-coded location, and always use the same file for all assemblies. I run `sn.exe` one time and I'm done with it.

In my development environment, I'm depending on Visual Studio to register the assembly for COM. When you distribute your app, you'll have to give the users a tool to register the assembly. The distributable will use `regase.exe /i dllname` to register the assembly. This is the .NET version of `RegSvr32` that you're probably used to. You'll find it in your Visual Studio folder, or you can get it here if you're using the Express version.

You should also note that placing the code in the target folder works well for Clarion, but may not work as well for other environments. For example, VB6 would require the DLL to be in one place when you're running in the IDE (the same folder as VB6.EXE) and a different place for a compiled EXE (the same folder as the EXE). If you're trying to use this component from an environment like that, the best thing to do is just put it in the GAC. To do that, you would add these lines to the Post-Build step if you're using a retail version of VS.

```
Call "$(DevEnvDir)..\Tools\VSVars.bat"
gacutil /I $(TargetFilename)
```

If you're using the Express version, get `gacutil.exe` here, place it somewhere in your path, and omit the first line shown above.

This doesn't necessarily mean that you have to put it in the GAC for your users. In this case, the GAC is only necessary for running in the VB6 IDE.

Test with something simple

You now have a public COM component that can be consumed by any COM container you like including Clarion. I don't recommend using this with anything prior to Clarion 5.. The COM-visible name for the control is the *namespace*, followed by a dot and the *classname*. For this example, it's `Interop01.Interop01Class`.

For an example, I'll use a really simple Clarion window. You'll find the simplest of all Clarion projects in the sample code, but you can test it with any Clarion program that has a window. The following code can be added to any window, and it will create an `OLE` control and call the sample method:

```
WorkOpenWin                routine
   data
Interop01                  long
   code
```

```
Interop01 = CREATE(0, CREATE:OLE)
Interop01{PROP:Create} = 'Interop01.Interop01Class'
Interop01{'CallDotnet("Message from Clarion")'}
```

Compile and run the Clarion program, and you should see a message box with the descriptive text. You'll notice that I created a new OLE control instead of adding one to the window. You can do either, but in practice creating a new control seems to be more stable with certain COM components.

Now return something

Returning something isn't much harder than that what I've just shown. Change the `CallDotNet` method to look like this:

```
public string CallDotNet(string message)
{
   MessageBox.Show(String.Format("Message Box in .NET showing {0} from Clarion.",
message));
   return "This text returned from .NET";
}
```

Change the Clarion code to look like:

```
Interop01 = CREATE(0, CREATE:OLE)
Interop01{PROP:Create} = 'Interop01.Interop01Class'
Message(Interop01{'CallDotnet("Message from Clarion")'})
```

Compile and run this, and you'll see one message box displayed by .NET and a second one displayed by Clarion.

Wrapup

I hope this will give you a bit to whet your appetite. For *a lot* of the projects I've done over the years, this capability would have made huge differences. I wish I'd had .NET a lot sooner, because the .NET framework supplies a lot of extended functionality that just doesn't come with Clarion. In the next segment, I'll show how to add WinForms controls to Clarion windows, or WinForms forms to Clarion apps.

Download the source

Wade Hatler has been around Clarion so long he got the very first patch release for Clarion 1.0 for DOS. That was back when Clarion had a dongle, no compiler and no templates. Wade co-owned IntelliScan, a company producing third-party Clarion products for several years. For the last 10 years he has worked for IMPAC Medical Systems creating software for the management of Cancer Therapy. He recently completed a 3-year, 12,000 mile, 12 country bicycle tour of the world, during which he carried a laptop and worked about half-time. He lives with his wife and daughter near Seattle. You can reach Wade at *WadeHatler@wademan.com*.

Reader Comments

Add a comment

- » Great article Wade, looking forward to the rest! Thanks...

Clarion Magazine

Querying SQL Data

by Thomas Ruby

Published 2005-08-31

It sure would be handy if Clarion had a way to run an SQL query and just get back a result. Say I want the total of a bunch of records. In SQL I can just issue a query like:

```
SELECT SUM(Amount) FROM tblSomething WHERE SomeField = SomeValue
```

The problem is that in Clarion I need a view to project just the amount field and…

But wait! Without telling us, SoftVelocity has slipped something really useful into Clarion. They gave us a class they called `cCwADO` which handles this kind of situation perfectly, and using it is almost as easy as falling off your chair.

Prerequisites

If you just try using `cCwADO` in any old ABC program yourself, you'll get a GPF every time, but there are just three simple things you need to do to make it work.

1. Include the class: You'll need to include the class, and this is done on the module level. Switch your main application window to the "Module View" and open the module that contains the procedure in which you want to use `cCwADO`. Go to the embeds and add one line of code to the start of the module:

```
INCLUDE('cwado.inc'),ONCE
```

2. Get the COM system initialized. To do this, SoftVelocity included a little class called cCOMINITER, and the needed code is built into the construction method. This class got automagically included with the cCwADO class, so all you need is an instance of the right class. I put an embed in the procedure data that looks like this:

```
ComIniter CCOMIniter
```

Now, as far as I can tell, this needs to be done only once per application, and I don't think it will hurt anything if it's done again. Just don't quote me on that. You can see if it worked by checking:

```
IF ComIniter.IsInitialised()
```

3. Add some Defines to your project: Select Project – Properties to get the project editor, click the Properties button and select the Defines tab. You'll see two defines for the `ABCDllMode` and the `ABCLinkMode`. Add some more:

```
_COMLinkMode_
_COMDLLMode_=>off
_ADOLinkMode_
_ADODLLMode_=>off
_svLinkMode_
_svDLLMode_=>off
_ADOMPRLinkMode_
_ADOMPRDLLMode_=>off
_SVDllMode_=>0
_SVLinkMode_=>1
```

I'm not at all sure what all of these are, or what their settings need to be for various memory models, but these work fine for an application using the CxxRUN.DLL link mode or the Local mode.

The class

Now you're ready to use the class. You can declare an instance of the `cCWAdo` class in a procedure data embed, or even in the data area of a method. Just type the following:

```
DataSet cCWAdo
```

You can also declare the class this way:

```
DataSet CLASS(cCWAdo)
End
```

While you're there, make a connection string variable:

```
ConnectionString CSTRING(256)
```

To use the class, you first need to get connected to your database. Luckily, there's a simple method to do that:

```
DataSet.SETConnection(ConnectionString)
```

This connection string is not the same as the owner string in the dictionary, but is formed like this:

```
Provider=SQLOLEDB.1;Persist Security Info=False;User ID=sa;PASSWORD=sa;Initial
  Catalog=book;Data Source=MAX\S2000
```

The server name is the Data Source parameter. You can also use a DSN name here if you have one. In this example, Initial Catalogue is the database, and Password and User ID are self-evident. You can build this `CString` with code if you like, but for tinkering, I just assigned it to `ConnectionString` all at once. Of course, now you all know how to hack into my test database, assuming you're in my shop.

Next, tell the object where you want the data to land. Any variable in memory will work:

```
DataSet.AddFieldsInfo('', 'Counter', RecordCount, 0 )
```

There are two string parameters that indicate which column from the result set you want. First is the table name, then the column name. You see I've left the table name blank, because I intend to use a statement like:

```
select count(*) as counter from tblDetail
```

and the aggregate counter value isn't associated with a table.

The third parameter is the variable where you want the data to go. In this case, I just have a local variable called `RecordCount` defined as a `LONG`.

The last parameter is almost always zero. If the column you're retrieving is a date, you need to use the `adDBDate` constant. This value is from an enum in adoint.inc and comes free when you include the class. There are other values for different data types.

You call the `AddFieldsInfo` once for each column you're getting back, or you can use the `addFieldsInfo` method:

```
DataSet.addFieldsInfo('tablename', DisplayQueue)
```

Internally, this handy method uses `WHO` and `WHAT` to add the field info for all the parts of a group, queue or even file record. It will match the name of the queue field with the column in the dataset for you.

Now you're ready to execute the query:

```
B = DataSet.ExecuteQuery('select count(*) as counter from tblDetail')
```

The `ExecuteQuery` method returns 1 if it retrieved a row and 0 if it did not. For a simple example like this, it placed the count into the recordcount variable for me.

If you are expecting more than one row, you can just loop through the returned rows this way:

```
LOOP
   IF DataSet.Next() = 0 THEN BREAK END
END
```

Here's a scrap of code showing how I retrieved the dataset into a queue:

```
B = DataSet.ExecuteQuery('SELECT CustomerID, CompanyName FROM Customers')
LOOP
   ADD(DisplayQueue)
   IF NOT  DataSet.Next() THEN BREAK END
END
```

Close the data set

Once you're done with the data set, you should close it:

```
DataSet.Close()
```

And this actually leaves it in a condition to execute another query if you want.

Conclusion

Retrieving data directly from an SQL database has always been a challenge with Clarion. Or so it would seem. Without telling us, SoftVelocity has given us all the tools we need to execute any SQL query and retrieve the results into a group, queue or even separate variables.

You need to remember:

1. Include the Class in the module section.
2. Create the cComIniter instance.
3. Add the text to the application defines.
4. Create an instance of the `cCWADO` class.
5. Use the `AddFieldsInfo` method(s) to tell the class instance where to put the data.
6. Execute your query.
7. Loop through the result set.

Tom Ruby, who is no relation to the man who shot Lee Harvey Oswald, is an independent contractor living in the middle of a hayfield in Central Illinois with his wife Susan and two red-headed sons, Caleb and Ethan. He has been using Clarion for Windows since the summer of '95. Before that, he was a "TopSpeeder" using Modula II, so he has never used the DOS versions of Clarion.

Reader Comments

Add a comment