Clarion Magazine

Clarion News

- » C7 To Support ClearType, Unicode
- » EasyOpenOffice 1.04
- » List Format Manager Video
- » CapeSoft World Tour Early Bird Registration Ends This Week
- » CapeSoft FeMail Client
- » NetTalk 4.22
- » ClassAnatomy Updated
- » BackItUp Updated
- » Clarion Newsgroups On The Web
- » CoolFrames 1.10 Beta
- » Evolution Clarion Manager 1.0 Beta
- » Clarion Desktop 3.02
- » FileTuner 0.15
- » CoolButtons 1.00 Beta Pricing
- » SimTabTree Demo
- » CapeSoft World Tour Johannesburg/Pretoria
- » AmazingGUI Demo
- » FullRecord 1.81
- » Code Commentor 2.0.4
- » NetTalk 4.19
- » Clarion Desktop 3.0
- » Tracker Software Hosting Change
- » AmazingGUI 1.0.5
- » ClarionMag Source Code Library 2007.02.28 Available
- » Dr.Explain 2.5
- » Clarion Desktop 3 Beta 1
- » FullRecord 1.80
- » CapeSoft World Tour: Johannesburg/Pretoria
- » CapeSoft World Tour: Americas & Europe
- » NeatMessage 2.12

[More news]

[More Clarion 101]

Latest Free Content

- » Source Code Library 2007.02.28 Available

[More free articles]

Clarion Sites

Latest Subscriber Content

Vista-Compliant INI Files

The release of Microsoft Windows Vista introduced a new set of challenges for programmers, particularly in the area of Vista's enhanced security. Randy Rogers introduces a derived INIClass that stores INI files in Vista-standard locations.

Posted Thursday, March 29, 2007

Interprocess Communication: Receiving Messages

Larry Sand continues his series on interprocess communication with some code to receive messages.

Posted Wednesday, March 28, 2007

C7 Alpha Bits Part 2

The latest alpha build adds user-defined comments for code completion tooltips.

Posted Tuesday, March 27, 2007

Embed Analysis Part 3

Dave Harms concludes the ABC embed analysis series with a look at browse, process and report embeds, as well as embed usage by procedure type.

Posted Thursday, March 22, 2007

Embed Analysis Part 2

Which embed points do Clarion developers really use? Dave Harms analyzes ABC embed data from hundreds of TXAs sent in by Clarion developers.

Posted Wednesday, March 21, 2007

Using the SQL Advanced Tab

The C6 browse template's SQL Advanced tab introduces a number of new capabilities to SQL browses. Bjarne Havnen explains how to use this tab and elaborates on the related SQL PROP: syntax.

Posted Friday, March 16, 2007

Interprocess Communication: Sending Messages

Larry Sand kicks of a series on inter-process communication with a look at how to send messages using PostMessage and RegisterWindowMessage.

Posted Wednesday, March 14, 2007

C7 Alpha Bits Part 1

Dave Harms tours some of the new editor features, including code folding and code completion.

Posted Saturday, March 10, 2007

Source Code Library 2007.02.28 Available

The Clarion Magazine Source Code Library has been updated to include the February source. Also new in this release: A category/subcategory tree view of all the articles. Locate articles of interest in the tree and click on the link to go to the summary, which has links to the source code (on disk) and the original ClarionMag article. Source code subscribers can download the update from the My ClarionMag page.

Posted Wednesday, March 07, 2007

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

Source Code

The ClarionMag Source Code Library

Clarion Magazine is more than just a great place to learn about Clarion development techniques, it's also home to a massive collection of Clarion source code. Clarion subscribers already know this, but now we've made it easier for subscribers and non-subscribers alike to find the code they need.

The Clarion Magazine Source Library is a single point download of all article source code, complete with an article cross-reference.

More info • Subscribe now

Clarion Blogs

Printed Books & E-Books

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

ISSN

Clarion Magazine's ISSN

Clarion Magazine's International Standard Serial Number (ISSN) is 1718-9942.

About ISSN

The ISSN is the standardized international code which allows the identification of any serial publication, including electronic serials, independently of its country of publication, of its language or alphabet, of its frequency, medium, etc.

Clarion Magazine

Clarion News

Search the news archive

Australian QuickBooks/MYOB Accounting Interface Wanted

Richard Bryce is looking for a software interface to MYOB or Quickbooks, preferably written by someone in Australia or New Zealand. If you happen to have some code that does this then Richard wants to talk to you. The requirement is for full source code, no black boxes. Contact Richard at 0403 892880.
Posted Tuesday, April 03, 2007

C7 To Support ClearType, Unicode

Phase 1 of the C7 alpha test will be wrapping up shortly with a build which includes support for Unicode and ClearType. Also check the blog entry for links to videos on the new project system and visual styles.
Posted Thursday, March 29, 2007

EasyOpenOffice 1.04

EasyOpenOffice 1.04 is now available. EasyOpenOffice is a set of classes and templates allowing you to exchange data between Clarion applications and OpenOffice Calc and Writer.
Posted Thursday, March 29, 2007

List Format Manager Video

Eberto Barrios Romo has posted a YouTube video on using the List Format Manager.
Posted Thursday, March 29, 2007

CapeSoft World Tour Early Bird Registration Ends This Week

Early Bird Registration for the CapeSoft World Tour closes this week. Of course normal registrations remain open until May 10. (Registrations for Johannesburg close on April 15). So if you're planning to attend, save yourself some dollars, euros, pounds and/or rands by registering early. Australians take note: There are only a very limited number of spaces left for the CapeSoft Training / Aussie Devcon combo, although there are a few more spaces for those attending only the CapeSoft Training (in Sydney). So if you plan to attend the Australia leg, register as soon as possible to avoid disappointment. Registrations for the other events have been brisk, so registering earlier rather than later is recommended. (If you intend attending, but cannot yet register, drop CapeSoft a line.)
Posted Thursday, March 29, 2007

CapeSoft FeMail Client

CapeSoft FeMail is a full-featured Email client, also available as source code. Features include: HTML and Text mail support, viewing of HTML and text mail, and GUI HTML editing; Supports attachments, automatic image embedding, saving and opening of attachments etc.; An easy to use address book, or plug in your own using your own data table;Drag and drop support throughout the application; Import mail from Outlook Express (Microsoft Outlook and Windows Mail import/export support coming soon); Flexible mail rules allow you to perform actions on collected emails that match one or more conditions that you specify; And much more.

Posted Thursday, March 29, 2007

NetTalk 4.22

NetTalk 4.22 is now available for download.

Posted Thursday, March 22, 2007

ClassAnatomy Updated

ClassAnatomy changes: The generate export file option would add the methods in alphabetical order instead of logic order.

Posted Thursday, March 22, 2007

BackItUp Updated

New in this release of BackItUp: A new Preview Matches option give syou a preview of files matching a specific profile.

Posted Thursday, March 22, 2007

Clarion Newsgroups On The Web

ClarionDesktop now offers a web interface to some of the Clarion newsgroups on SoftVelocity's server.

Posted Thursday, March 22, 2007

CoolFrames 1.10 Beta

CoolFrames 1.10 Beta is now available for download.

Posted Thursday, March 22, 2007

Evolution Clarion Manager 1.0 Beta

Clarion Manager helps manage multi-DLL/multi-APP development. Features include: Viewing relationships between apps; Embed code manager; test DCT or template changes; Evaluate application design; Document applications; Manage large projects with several applications; Manage multiple projects, with multiple apps; Navigate the map of calls between procedures from different DLLs;

Evaluate dependencies between DLLs. Price is US$139 during the beta, $199 after gold release.
Posted Thursday, March 22, 2007

Clarion Desktop 3.02

Clarion Desktop 3.02 has been released. Changes include: A new menu on the local homepage called "Blogs"; Product "Buy Now" links were not working correctly in the previous release - this has been fixed; Several new features / improvements for suppliers.
Posted Thursday, March 22, 2007

FileTuner 0.15

FileTuner 0.15 is now available. Changes include: Press Enter on the list to select the current file; Tag only button (added to Tag and down); Improved file usage checking to see if file is opened by other users; Copy procedure (to temporal folder) now with progress bar; TPS files without name specified in the DCT now handled properly; Changes to general layout.
Posted Thursday, March 22, 2007

CoolButtons 1.00 Beta Pricing

CoolButtons is now for sale at a reduced price of US$49 during the beta stage and US$99 once it goes gold.
Posted Thursday, March 22, 2007

SimTabTree Demo

A demo of SimTabTree is now available for download. SimTabTree is available from www.clarionshop.com for US $29.
Posted Thursday, March 22, 2007

CapeSoft World Tour Johannesburg/Pretoria

The CapeSoft World Tour will be stopping in Johannesburg/Pretoria.
Posted Thursday, March 22, 2007

AmazingGUI Demo

A 3:30 length video of AmazingGUI is now available for download. This video applies AmazingGUI to the example School application.
Posted Thursday, March 22, 2007

FullRecord 1.81

FullRecord 1.81 divides template prompts into Basic and Advanced interfaces. Demo available. In the demo press Ctrl-Shift-A on any field to see the history log of that particular field.
Posted Thursday, March 22, 2007

Code Commentor 2.0.4

Code Commentor 2.0.4 is now available. This release fixes the "after midnight" bug. When the Code Commentor was started and used and then left running overnight it would no longer be responsive when you came in the next day. You would have to exit and restart the Code Commentor.

Posted Thursday, March 22, 2007

NetTalk 4.19

NetTalk 4.19 is now available. Numerous changes and fixes, including: WYSIWYG HTML editing (in your text boxes on your forms); Multiple form fields on a row; HTML compliant pages; "Wizard" form type; Multiple selects in selection lists, and much more.

Posted Saturday, March 10, 2007

Clarion Desktop 3.0

Clarion Desktop v3.00 Gold has been released. This release is much faster than previous releases. Clarion Desktop now has 400 users and is supported by 136 products from 31 suppliers.

Posted Saturday, March 10, 2007

Tracker Software Hosting Change

Tracker Software is updating its server this weekend and will have a new IP address.

Posted Saturday, March 10, 2007

AmazingGUI 1.0.5

AmazingGUI 1.0.5 is now available. The web site has been reorganized and now you do not need to register in order to download the demo program.

Posted Saturday, March 10, 2007

ClarionMag Source Code Library 2007.02.28 Available

The Clarion Magazine Source Code Library has been updated to include the February source. Also new in this release: A category/subcategory tree view of all the articles. Locate articles of interest in the tree and click on the link to go to the summary, which has links to the source code (on disk) and the original ClarionMag article. Source code subscribers can download the update from the My ClarionMag page.

Posted Wednesday, March 07, 2007

Dr.Explain 2.5

Dr.Explain is a help authoring tool that captures windows, dialogs, forms, and menus from live application, makes screenshots, and automatically adds references to all controls. The program can produce CHM, RTF and HTML help files with interactive screenshots, dynamic menus, cross-references,

and navigation. This release has many new features, including: Menu capturing and auto-capturing: document all your menus with a couple of clicks; Dynamic menu for online manuals: navigate easily through your manual; Validation tool to find all issues in your project in just seconds; Screenshot editor: Add text to your screenshots; Context Help ID support: integrate help files into your applications fast; Command line mode: automate your help compiling processes; Windows Vista support: Run your favorite tool on the new OS; and many more.
Posted Tuesday, March 06, 2007

Clarion Desktop 3 Beta 1

Clarion Desktop 3 (Beta 1) has been released as a free download. This version loads the initial home page almost twice as fast as version 2 did, and subsequent page loads are between 2 and 4 times faster. The Download Manager now also supports installation patches as used by CapeSoft. If you download FM3 for example, Clarion Desktop will download the primary installer, after which it will check what version of Clarion you are running, and it will then automatically download the correct patch installer for you. Both installers are then automatically archived and then launched (in the correct sequence), and the installer passwords are automatically entered for you. There are numerous other improvements.
Posted Tuesday, March 06, 2007

FullRecord 1.80

New in FullRecord 1.80: a Field Inspector window that can be activated on request (with a button), or in the whole system with a global hot-key. Press the hot-key on any field of any window and you will see the history of changes on that particular field for that particular record. As of Clarion 7 alpha 1, FullRecord is fully code-compatible with C7.
Posted Tuesday, March 06, 2007

CapeSoft World Tour: Johannesburg/Pretoria

If you are interested in attending a world tour warm-up event in the Jo'burg/Pretoria area, on May 3,4 and 5 please send Bruce an email. If there is sufficient interest then CapeSoft would be keen to drop in there. Cost would be about R1950 per person for three days, and would include lunch. Current front-runner for the venue would be the St George Hotel in Irene (http://www.stgeorgehotel.co.za).
Posted Tuesday, March 06, 2007

CapeSoft World Tour: Americas & Europe

Registrations are now open. There is early-bird pricing in effect, until March 30. Las Vegas: $750 early bird ($900 after that). On the down side, lunch is not included (the hotel wants $40 per person for lunch) but there are plenty of eateries nearby. Cambridge: $1175 (approximately ~£600 / ~?890) early bird, $1375 (approximately £700 / ?1050) after that. Unfortunately it's a bit more expensive than the US, but that's a side effect of the overall expenses in Europe. On the up side, lunch is included in the price. Five days. Three class rooms. Eighteen topics. Fifty hours. This is the most intensive Clarion event ever.

Posted Tuesday, March 06, 2007

Clarion Magazine

Vista-Compliant INI Files

by Randy Rogers

Published 2007-03-29

The release of Microsoft Windows Vista introduced a new set of challenges for programmers, particularly in the area of Vista's enhanced security. Vista provides some remedies including XP compatibility modes and the use of virtual folders. This may work in the short term, but in the longer term it will be better if your program is Vista compliant. The location of application data, temporary files, and other files that are shared by multiple users of the computer should be reviewed and changed as necessary.

Special folders used frequently by applications, but which may not have the same name or location on any given system, are identified by unique system-independent Constant Special Item ID List, or CSIDL, values. In this article I'll demonstrate a derived INIClass called VistaINIClass that uses the SHGetFolderPath API call to translate the CSIDL constants into folder locations. You simply specify the constant for the desired location, and the class takes care of storing the INI file in that folder. I've also included a small template to make it even easier to implement the class in your applications.

My solution does not deal with the GETINI / PUTINI built in procedures. They could be overridden by modifying the builtins.clw file and providing replacement procedures. I have an aversion to modifying any files shipped by SoftVelocity, so I decided to convert any GETINI / PUTINI procedure calls in my code to INIClass (or, in my case, VistaINIClass) method calls instead.

The class deals strictly with INI file types; INI registry value types are passed through to the PARENT unchanged.

INI folders

INI files provide a particular challenge because they normally reside in the Windows folder. Under Vista the Windows folder is protected so the INI file gets copied to a virtual folder. This means that every user of the computer will have a copy of the INI file located in a virtual folder. In other words, the INI file is no longer shared and changes made by one user are not seen by other users. This behavior may be just what you want; if so I would suggest using a "MyCompanyName" subfolder of the CSIDL_PERSONAL (My Documents) folder to store the INI files.

If you want to share an INI file between multiple users then a "MyCompanyName" subfolder of the CSIDL_COMMON_APPDATA folder seems to be the preferred location. This is a protected folder under Vista, so you need to grant access rights to your subfolder to the appropriate users / group. This could be handled by your install program or by using the cacls.exe command line program that comes with

Windows. A good article on using cacls.exe can be found at TechRepublic.

I have a large number of DLL and EXE applications that require modification to make them Vista compliant. Since the Clarion IDE and ABC Templates do not recognize CSIDL values it seemed, at first, like this was going to be a daunting task. After thinking about the problem for a while I finally decided that deriving the INIClass and providing my own Init method would be a simple, elegant, and easily maintainable solution.

The VistaINIClass

The downloadable source contains the class files and a sample application that uses the class. Here's the declaration from vistaINIClass.inc:

```
!ABCIncludeFile(ABC)
!=================================================================
!Copyright ©2007 Keystone Computer Resources
!Creation Date: 2007/02/20
!=================================================================
 COMPILE('ENDCOMPILE',_ABCLinkMode_)
   PRAGMA ('link (shfolder.lib)')
 !ENDCOMPILE

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

   INCLUDE('ABUTIL.INC'),ONCE

CSIDL_PERSONAL        EQUATE(00005h)  !My Documents
CSIDL_COMMON_APPDATA  EQUATE(00023h)  !All Users\Application Data

vistaINIClass        CLASS(INIClass), TYPE, MODULE('vistaINIClass.clw'), |
                      LINK('vistaINIClass.clw',_ABCLinkMode_), |
                                  DLL(_ABCDllMode_)
Init                 PROCEDURE(STRING FileName, UNSIGNED nvType, |
                      LONG extraData = 0) !,EXTENDS
                    END
!_EndOfInclude_
```

The class needs the SHGetFolderPath API call which is in Windows' shfolder.dll. The PRAGMA statement causes the shfolder.lib file to be linked in. You will need to create the LIB file from the DLL with LibMaker.exe.

There are many CSIDL values; this example only declares the ones I intend to use. The downloadable source contains equates for all documented CSIDL values.

The vistaINIClass is derived from INIClass and only needs to override one of the Init methods:

```
vistaINIClass.clw
  MEMBER
!=========================================================================
!Copyright ©2007 Keystone Computer Resources
!Creation Date: 2007/02/20
!=========================================================================

HANDLE  EQUATE(LONG)
HWND    EQUATE(HANDLE)
HRESULT EQUATE(HANDLE)
DWORD   EQUATE(LONG)
S_OK    EQUATE(0)

  INCLUDE('vistaINIClass.inc'),ONCE

  MAP
    MODULE('kernel32.dll')
      kcr_CreateDirectory(*CSTRING szPath, LONG lpSecurityAttributes)|
            ,BOOL,PASCAL,RAW,NAME('CreateDirectoryA')
      kcr_GetLastError(),LONG,PASCAL,NAME('GetLastError')
    END
    MODULE('Shfolder.dll')
      kcr_SHGetFolderPath(HWND hwnd, LONG csidl, HANDLE hToken, |
            DWORD dwFlags, *CSTRING szPath),HRESULT,RAW,|
                PASCAL,NAME('SHGetFolderPathA')
    END
  END

vistaINIClass.Init  PROCEDURE(STRING FileName, UNSIGNED nvType, |
                      LONG extraData = 0) !,EXTENDS
hr          HRESULT
hFile       HANDLE
```

```
szPath      CSTRING(File:MaxFilePath)
CSIDL       LONG

 CODE
  !check for INI file type
  IF nvType = NVD_INI
    !check for simple filename
    IF ~INSTRING('\',FileName)
      ! Use the following to place ini files in user's
           ! "My Documents\YourCompanyName" folder
      CSIDL = CSIDL_PERSONAL

      ! Use the following to place ini files in the
           ! "All Users\Application Data\YourCompanyName" folder
      !CSIDL = CSIDL_COMMON_APPDATA

      hr = kcr_SHGetFolderPath(0,CSIDL,0,0,szPath)
      IF hr = S_OK
        !TODO: change 'MyCompanyName' to the name
                ! you want to call the subfolder
        szPath = LONGPATH(szPath) & '\MyCompanyName'
        IF kcr_CreateDirectory(szPath,0)
          PARENT.Init(szPath & '\' & FileName, nvType, extraData)
        ELSE
          IF kcr_GetLastError() = ERROR_ALREADY_EXISTS
            PARENT.Init(szPath & '\' & FileName, nvType, extraData)
          ELSE
            PARENT.Init(FileName, nvType, extraData)
          END
        END
      ELSE
        PARENT.Init(FileName, nvType, extraData)
      END
    ELSE
      PARENT.Init(FileName, nvType, extraData)
    END
  ELSE
```

```
    PARENT.Init(FileName, nvType, extraData)
  END
  RETURN
```

The first few lines of code declare some equates, include the class declaration file, and prototype the API calls needed by the class.

This example uses the CSIDL_PERSONAL folder as the location to store the INI files. Comment this line and uncomment the following line to use the CSIDL_COMMON_APPDATA folder instead. Remember to grant users access rights to your subfolder.

The class has the path hard coded so remember to change it in the source before using the class in a production environment.

Next the code creates the path, checks for errors, and calls the PARENT method to complete the work.

All that remains is to replace the INIClass in the application Global Properties|Classes|General|INI Manager with VistaINIClass. That’s pretty easy to do, but takes a lot of clicking if you have many applications to convert. I decided to create a small extension template to do a lot of the work for me.

The template

The VistaINIClass.tpl template file contains one global extension, as follows:

```
#!============================================================
#! Keystone vistaINIClass Template
#! Author:       Randy Rogers (KCR) <rrogers@keystonecr.com>
#! Copyright:    ©2007 Keystone Computer Resources
#!               ALL RIGHTS RESERVED
#!============================================================
#TEMPLATE(vistaINIClass,'vistaINIClass Template'),FAMILY('ABC')
#!
#!
#! ------------------------------------------------------------
#EXTENSION(vistaINIClassGlobal,'Add vistaINIClass to app'),APPLICATION
#! ------------------------------------------------------------
#DISPLAY('This template adds the vistaINIClass.')
#DISPLAY('')
#DISPLAY('There are no prompts for this template')
#AT(%BeforeGenerateApplication)
  #CALL(%SetClassDefaults(ABC), 'INIManager', 'INIMgr', 'vistaINIClass')
#ENDAT
```

The template sets the default value for the ABC INIManager to VistaINIClass. To use the template simply register vistaINIClass.tpl and then add the global extension to each application.

Viewing CSIDL values

The downloadable source includes a small application that displays the paths associated with the various CSIDL values. You'll need to register the vistaINIClass.tpl file first, or else ignore the template warning and change the INI Manager class manually. You may need to click on the Refresh Application Builder Class Information button on the Global Properties window, Classes tab.

Summary

Microsoft Vista introduces a number of security-related problems, and one of these is the need to use virtual folders rather than the Windows folder for INI file locations. With this class and template you can store your INI files in Vista-friendly locations.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Interprocess Communication: Receiving Messages

by Larry Sand

Published 2007-03-28

Larry Sand is an

In the first installment in this series I covered the basics of interprocess communication and looked at how to send messages. Now it's time to write some code to receive messages.

Sending messages is simple using the Windows API, but receiving the message in a Clarion program requires a little more work. Clarion doesn't provide a direct way to "see" the messages being processed by your window. To have a chance to process these messages, you must subclass the window event handler. Subclassing the window procedure involves inserting your own event handling procedure into a chain of event handling procedures called by Windows. Your window procedure must have the prototype as follows (see Part 1 for more information).

```
WinProc Procedure(UNSIGNED hWnd, UNSIGNED uMsg,|
  UNSIGNED wParam, Long lParam),Long,Pascal
```

You may not change this prototype because it's called directly by Windows to process the messages in the message queue.

There are three main steps in subclassing a window procedure.

1. Get and save the address of the existing window procedure
2. Tell your window to use your replacement window procedure to process its messages
3. In the replacement window procedure that you specified in step 2, process the messages that you're interested in and pass all others to the original window procedure that you saved the address of in step one.

The code that accompanies this article is a class called InterprocessComs. The requirement that you not change the prototype of the window procedure presents a special problem when using a class. All methods of a class have an implicit first parameter that's a reference to Self (the instance of the class). Because of this, you cannot use a method for a window procedure. To get around this restriction, the class module contains a window procedure that's shared by all instances of the class. Because this window procedure is shared you have to have a method to allow it access the instance of your class that's attached to your window and its old window procedure address. To do this, you'll use two more Windows API functions, SetProp and GetProp, to set and get a property attached to a window. SetProp and GetProp use the handle to the window to access a window's property list and your replacement window procedure always knows the handle to the window for the current message it's processing.

SetProp allows you to assign a 32 bit value to a hWnd designated by a constant string. Later you'll use GetProp to retrieve the 32 bit value attached to your window. GetProp and SetProp are prototyped in Clarion like this:

```
CMAG_SetProp(UNSIGNED hWnd, *? lpString, UNSIGNED hData),|
  BOOL,Pascal,Raw,Name('SetPropA') ,DLL(1)
CMAG_GetProp(UNSIGNED hWnd, *? lpString),UNSIGNED,Pascal,|
  Raw,Name('GetPropA') ,DLL(1)
```

There's one more property function that's very important; RemoveProp is called when you're done with the property to free the resources it consumed. It's prototyped like this:

```
CMAG_RemoveProp(UNSIGNED hWnd, *? lpString),UNSIGNED,|
  Pascal,Raw,Name('RemovePropA'),DLL(1),PROC
```

In each of these functions the string parameter is prototyped as a void pointer. A *? parameter with the RAW attribute passes the address of the datum to the external function. Normally you prototype a long pointer to a string as *CString with the RAW attribute. However, these three functions accept either a constant null terminated string or an atom. Atoms are created by calling AddAtom or GlobalAddAtom and can be used in place of constant strings. They are slightly more efficient to look up than a constant string using GetProp. Converting the code to use atoms is left to you; as written it will still function perfectly well.

The class declares a window procedure to use in subclassing your window procedure like this:

```
IpcScWndProc Procedure(UNSIGNED hWnd, UNSIGNED uMsg, |
  UNSIGNED wParam, Long lParam)
```

To accomplish the subclassing from the class, an Init method is declared that accepts the Window and a String as parameters. Here's the code for the Init method:

```
InterprocessComs.Init   Procedure(Window W, |
                        String sLinkMessageId)!,BOOL
bResult BOOL,Auto
szLinkMessageId &CString
  Code
  bResult = False
  If W{PROP:Mdi} = '1'
   Self.TakeError(IPC_ERROR_INVALIDUSE, 'Not valid for MDI windows')
   Return bResult
  End

  szLinkMessageId &= New CString(Len(Clip(sLinkMessageId)) +1)
  If szLinkMessageId &= Null
    Self.TakeError(IPC_ERROR_OUTOFMEMORY, 'Not enough memory')
    Return bResult
  End
  szLinkMessageId = Clip(sLinkMessageId)
  Self.LinkMessage = CMAG_RegisterWindowMessage(szLinkMessageId)
```

```
    If self.LinkMessage = 0
      Self.TakeError(CMAG_GetLastError(),'Register link message')
    End
    Dispose(szLinkMessageId)

    Self.hWnd = W{Prop:Handle}
    bResult = Self.SubclassMe(Self.hWnd)
    If bResult = True
      Self.BroadcastLinkMessage()
    End
    Return bResult
```

This form of interprocess communication is not meant to communicate between MDI child windows, so the first thing the code does is to return if you pass in an MDI child window as the Window parameter. You must use caution when sending messages to threaded MDI child windows. Windows is not thread safe when dealing with MDI windows. One problem is that Windows uses SendMessage to communicate and it's possible to have an MDI window's thread execution suspended while it's waiting for SendMessage to return. The Clarion runtime library has implemented code to help prevent this type of problem, but the best way is to use Clarion's POST/EVENT and NOTIFY/NOTIFICATION message processing functions.

Next, you need to register the string with Windows to obtain the registered message identifier that you'll use to communicate with the other process. RegisterWindowMessage requires that you pass a Cstring, and the prototype for the Init method allows you to pass a Clarion string. To convert the message string the code declares a CString reference and uses New to make it one byte larger than the string passed into the method (the extra byte is to accommodate the terminating null (Chr(0))). The String is clipped as it's assigned to the szLinkMessageId CString to remove any trailing spaces. RegisterWindowMessage is then called and if the return value is <> 0 you have successfully registered or retrieved the message id associated with the string. After RegisterWindowMessage is called, you no longer need the szLinkMessageId CString, and the class frees the memory with Dispose.

The handle to the window to be subclassed is stored in a class property. Later, when processing the broadcast message it's necessary to know if the message is from yourself. The class compares the hWnd it receives and the value stored in the property.

Now that the class has the registered message, it can subclass the window procedure. This is done in the SubclassMe method.

```
InterprocessComs.SubclassMe  Procedure(UNSIGNED hWnd)|!,BOOL,Protected
OldWndProc  Long,Auto
bResult     BOOL,Auto
  Code
  bResult = False
  If CMAG_GetProp(hwnd, WPROP_IPC_WINDOWPROC) <> 0
    Self.TakeError(IPC_ERROR_OUTOFMEMORY, 'Window is already subclassed')
    Return bResult
```

```
     End

     OldWndProc = CMAG_GetWindowLong(hwnd, CMAG_GWL_WNDPROC)
     If CMAG_SetProp(hWnd, WPROP_IPC_WINDOWPROC, OldWndProc) <> 0
       If CMAG_SetProp(hWnd, WPROP_IPC_CLASSINST, Address(Self)) <> 0
         CMAG_SetLastError(0)
         If CMAG_SetWindowLong(hwnd, CMAG_GWL_WNDPROC, |
             Address(IpcScWndProc)) <> 0
           bResult = True
         Else
           Self.TakeError(CMAG_GetLastError(), 'SWL Subclass')
         End
       Else
         Self.TakeError(CMAG_GetLastError(), 'SetProp Class Instance')
       End
     Else
       Self.TakeError(CMAG_GetLastError(), 'SetProp Window Proc')
     End
     If bResult = False
       CMAG_RemoveProp(hWnd, WPROP_IPC_WINDOWPROC)
       CMAG_RemoveProp(hWnd, WPROP_IPC_CLASSINST)
     End
     Return bResult
```

This method is responsible for getting the address of the existing window procedure and then setting the window properties necessary for the replacement window procedure to call the correct instance of the class and the original window procedure. The property identifiers are constant CStrings declared in the member module. Notice that they are registry style GUIDs , without the opening and closing "{}", and were generated with guidgen.exe described earlier.

```
   WPROP_IPC_WINDOWPROC    CString('E6B2022E-76B9-44cc-931F-7B14F1299CC9'),Static
   WPROP_IPC_CLASSINST     CString('9D80FCDA-90F3-4c3e-9201-D0F8B48229CE'),Static
```

If the window property WPROP_IPC_WINDOWPROC property is already set, then this window has already been subclassed. It doesn't make sense to have two instances of this class managing one window.

Next the Windows GetWindowLong function is called with the GWL_WNDPROC constant to retrieve the address of the existing window procedure. Remember that the new window procedure requires this address to return control to the original window procedure for messages you don't want to process. SetProp is called to attempt to set the WPROP_IPC_WINDOWPROC property. Then the address of the class is stored in the WPROP_IPC_CLASSINST property. This property is used to access the correct instance of the class from the window procedure. Next, SetWindowLong is called to change the address of the window's window procedure. As soon as this line of code executes Windows will begin passing messages for the window to this procedure. Finally, if any of the subclassing steps fails, the method removes the properties and returns false.

The Init method has one more task to perform if everything up until this point was successful. It will now attempt to broadcast the registered window message to see if a partner is already out there listening. This is done by calling the BroadcastLinkMessage method.

```
InterprocessComs.BroadcastLinkMessage       Procedure()
  Code
  If Self.LinkMessage and Self.hWnd
    CMAG_PostMessage(CMAG_HWND_BROADCAST, Self.LinkMessage, |
          Self.hWnd, WMU_IPC_LINKME)
  End
  Return
```

BroadcastLinkMessage calls PostMessage with the broadcast message window handle HWND_BROADCAST so the message is sent to all top level windows. The message it sends is the registered window message that was saved in the LinkMessage property, and the hWnd of this window is sent as the wParam of the message. Remember that PostMessage does not wait for a response from any other window procedures. It posts the message and returns control immediately. As a result the sender's window procedure will receive the broadcast message within a few milliseconds.

When either the sending or the receiving program receives the broadcast link message it's processed by the window procedure, which receives the hWnd, message, wParam and lParam as parameters. Here's the window procedure that the class implements:

```
IpcScWndProc Procedure(UNSIGNED hWnd, UNSIGNED uMsg, |
               UNSIGNED wParam, Long lParam)!,Long,Pascal

OldWndProc UNSIGNED,AUTO
Me        &InterprocessComs
  Code
  OldWndProc = CMAG_GetProp(hWnd, WPROP_IPC_WINDOWPROC)
  If NOT OldWndProc
    Return 0
  End
  !Get the reference to this instance of the class
  Me &= CMAG_GetProp(hWnd, WPROP_IPC_CLASSINST)
  If Me &= NULL
    Return CMAG_CallWindowProc(OldWndProc, hWnd, |
      uMsg, wParam, lParam)
  End

  Case uMsg
  Of CMAG_WM_COPYDATA
  Of CMAG_WM_DESTROY
    Me.RemoveLink()
  Of CMAG_WM_NCDESTROY
   !This is the last message received when window is destroyed
   !remove the properties added to the window in the
   !subclassme() method
   CMAG_RemoveProp(hWnd, WPROP_IPC_CLASSINST)
   CMAG_RemoveProp(hWnd, WPROP_IPC_WINDOWPROC)
  Of Me.LinkMessage()
    If wParam <> hWnd !don't process broadcast message for self
      Me.OnLinkMessage(wParam, lParam)
    End
  Of Me.UserMessage()
    If Me.UserMessage() <> 0
      Me.OnUserMessage(wParam, lParam)
      Return 0
    End
  Return CMAG_CallWindowProc(OldWndProc, hWnd, uMsg, |
    wParam, lParam)
```

Two variables are declared, one to hold the address of the old window procedure and an InterprocessComs class reference variable. These two variables need their values set to those stored in the window properties before any messages are processed by the window procedure.

GetProp is called to retrieve the OldWndProc variable from the window property list. This was set with the WPROP_IPC_WINDOWPROC constant string in the SubclassMe method. After the call to GetProp, the OldWndProc variable should contain the address of the window procedure that will process all the message not handled by this procedure. You pass control to the original window procedure by calling the Windows CallWindowProc function.

GetProp is called again, this time with the WPROP_IPC_CLASSINST constant string to retrieve the instance of the class. This address is cast to the reference variable allowing you to access all methods and properties of the instance of the class associated with this window. If for some reason the reference is null then the message is passed to the original window procedure by calling CallWindowProc and returning its return value.

After these two properties are retrieved from the window's property list, you can process messages. Some of the messages that this window procedure processes are constants defined in the Windows SDK, and the others are generated by calls to RegisterWindowMessage and are only known at runtime. For a complete list of the constant messages you should download the platform SDK from Microsoft. The most recent one was published in November 2006 and it covers Windows Vista. The .h files contained in the platform SDK are the definitive reference to the Windows SDK.

Five messages are processed by the window procedure. They are: WM_COPYDATA, WM_NCDESTROY, WM_DESTROY, the registered link message, and an optional registered user message. The window procedure doesn't do anything with WM_COPYDATA in the code shown; this message is used to pass more complex data like strings, groups, and record structures between processes. I'll show code for those tasks in a later installment.

WM_DESTROY and WM_NCDESTROY are sent to the window procedure by Windows when the window is being destroyed. WM_DESTROY is sent before the window is destroyed. WM_NCDESTROY is sent to the window after it's destroyed and is the last message your window procedure will receive. In response to WM_NCDESTROY the window procedure removes the two properties, WPROP_IPC_CLASSINST and WPROP_IPC_WINDOWPROC from the window's property list by calling the Windows RemoveProp function. Failure to remove the properties causes a resource leak.

On the other hand, when the window procedure receives the WM_DESTROY message, you know that the window is about to be destroyed and the class uses this opportunity to disassociate itself from its partner process. It does this by calling the RemoveLink method.

```
InterprocessComs.RemoveLink        Procedure()
 Code
 If Self.hWndPartner And Self.LinkMessage And Self.hWnd
  CMAG_PostMessage(Self.hWndPartner, Self.LinkMessage, |
     Self.hWnd, WMU_IPC_REMOVEME)
 End
 Self.hWndPartner = 0
 Return
```

RemoveLink simply posts the registered link message to the other process it's been partnered with in the initial link exchange, with the wParam containing the hWnd of this window and the lParam containing the constant WMU_IPC_REMOVEME.

Processing one of the registered messages requires that the window procedure call a class method to retrieve the value that Windows assigned to your message. When the Case statement executes this line of code:

```
Of Me.LinkMessage()
```

the current instance of class's LinkMessage simply returns the value stored in the LinkMessage property so it may be compared with the current value of uMsg. The window procedure then needs to determine if this window sent the broadcast message. This is true when the hWnd and wParam are equal. Remember that the link message defines the wParam as the hWnd of the window that sent the link message. The window procedure's hWnd parameter is the handle to this window. Only when the two window handles are different is the message processed by calling the OnLinkMessage handler method. This method processes the receipt of all link messages for the window procedure.

```
InterprocessComs.OnLinkMessage  Procedure(UNSIGNED wParam, |
                                Long lParam)
 Code
 Case lParam
 Of WMU_IPC_LINKME
```

```
    If Self.hWndPartner Then Return; End
    Self.hWndPartner = wParam
    CMAG_PostMessage(wParam, Self.LinkMessage, |
      Self.hWnd, WMU_IPC_ACKLINK)
  Of WMU_IPC_ACKLINK
    Self.hWndPartner = wParam
  Of WMU_IPC_REMOVEME
    Self.hWndPartner = 0
  End
  Return
```

OnLinkMessage accepts two parameters, wParam and lParam. These values are forwarded by the window procedure. Remember that the link message defines the lParam as containing the action and the wParam contains the handle of the window that sent the message.

When OnLinkMessage receives an WMU_IPC_LINKME request, it saves the handle to the sending window in the hWndPartner property and then posts the link message back to the sending process with an WMU_IPC_ACKLINK to acknowledge the receipt of the link message. When the initiating process receives the WMU_IPC_ACKLINK request in the link message, it saves the transmitted window handle in its hWndPartner property. After the link and acknowledgement requests are processed, the two processes may send messages to each other via the window handle saved in the hWndPartner property.

When one of the processes destroys the window whose window procedure is processing the Interprocess communication messages, it notifies the other process by sending the link message with the WMU_IPC_REMOVEME action in the lParam. When received, OnLinkMessage sets the hWndPartner to zero to prevent messages from being sent; this also allows the process to establish a new link upon receipt of the link message with the WMU_IPC_LINKME action.

That's the code required to establish the link. Next time I'll show how to exchange messages between processes.

Download the source

independent software developer who began programming with Clarion in 1987. In addition to normal database development, he specializes in connecting Clarion to external devices like SCUBA diving computers, kilns, and satellite transceivers used in medical helicopters. In other lives, he sailed Lake Superior as the owner/operator of shipwreck SCUBA diving tours and later as a Master for the Vista Fleet. When Larry is not programming you'll find him messing about in boats, or with boats.

Reader Comments

Add a comment

Clarion Magazine

C7 Alpha Bits Part 2

by Dave Harms

Published 2007-03-27

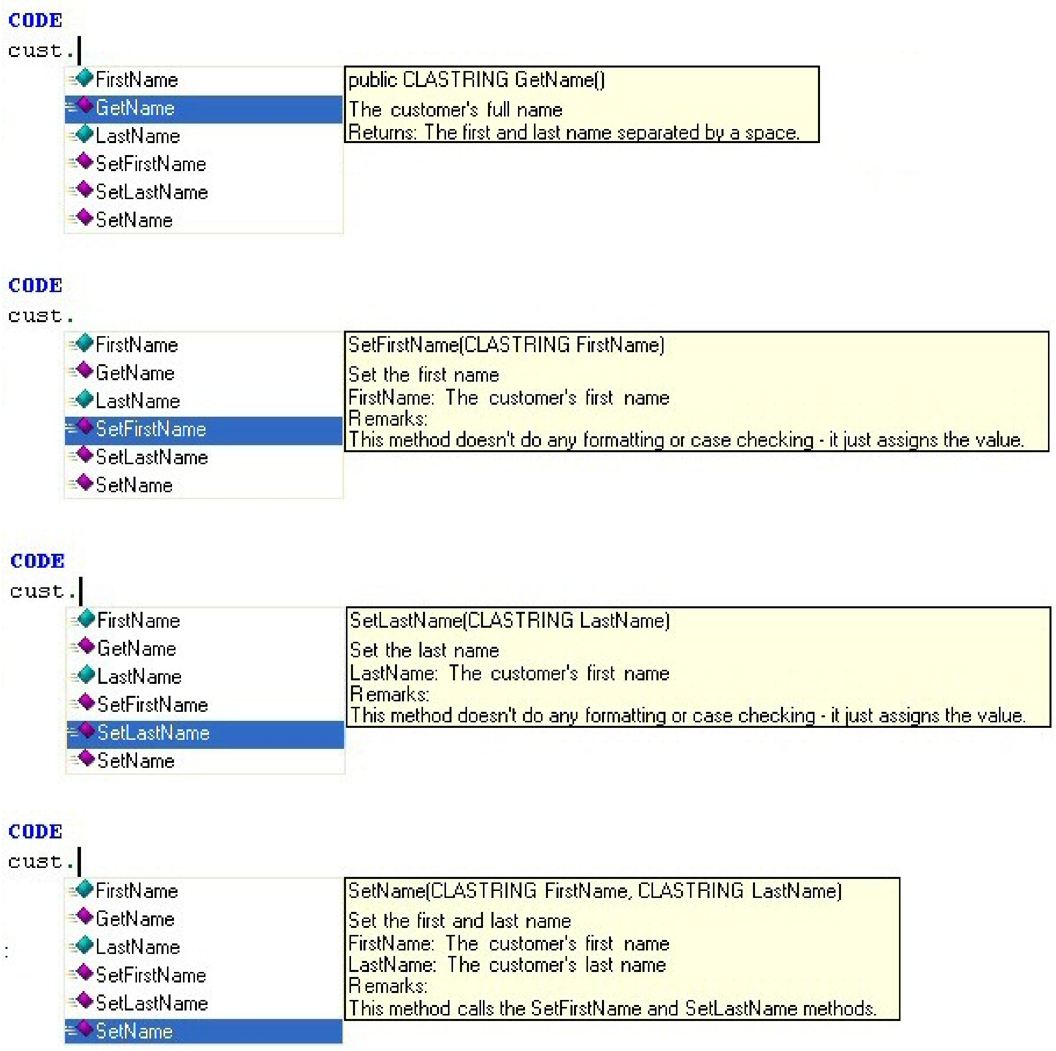
In Alpha Bits Part 1 I mentioned C7's code completion, which pops up information about procedures and classes as you type. The latest Alpha 1 build extends this capability with user-defined text in popups.

Consider the following test program created with C7. It contains a class definition, cust, which has FirstName and LastName properties, and methods to set those values as well as return the combined first and last name:

```
  PROGRAM

OMIT('***')
 * Created by Clarion 7.0
 * User: Dave Harms
 * Date: 23/03/2007
 * Time: 7:12 AM
 *
 * To change this template use Tools | Options | Coding | Edit Standard Headers.
 ***

  MAP
  END


cust           class
FirstName              string(30) !!!Customer's first name
LastName               string(30) !!!Customer's last name
SetFirstName    procedure(string FirstName)
SetLastName     procedure(string LastName)
SetName         procedure(string FirstName,String LastName)
GetName         procedure(),string
```

```
                  end



   CODE
   cust.SetName('Dave','Harms')

   message(cust.GetName())



!!! <summary>The customer's full name</summary>
!!! <returns>The first and last name separated by a space. </returns>
cust.GetName    procedure
        code
        return(clip(self.FirstName) & ' ' & clip(self.LastName))

!!! <summary>Set the first name</summary>
!!! <param name="FirstName">The customer's first name</param>
!!! <remarks>This method doesn't do any formatting or
!!! case checking - it just assigns the value.</remarks>
cust.SetFirstName       procedure(string FirstName)
        code
        self.FirstName = FirstName


!!! <summary>Set the last name</summary>
!!! <param name="LastName">The customer's first name</param>
!!! <remarks>This method doesn't do any formatting or
!!! case checking - it just assigns the value.</remarks>
cust.SetLastName                procedure(string LastName)
        code
        self.LastName = LastName


!!! <summary>Set the first and last name</summary>
!!! <param name="FirstName">The customer's first name</param>
!!! <param name="LastName">The customer's last name</param>
!!! <remarks>This method calls the
!!! SetFirstName and SetLastName methods.</remarks>
cust.SetName            procedure(string FirstName,string LastName)
        code
        self.SetFirstName(FirstName)
        self.SetLastName(LastName)
```

Note the comments preceding each method declaration. Figure 1 shows each of the method code completion popups.



```
CODE
cust.
  FirstName
  GetName
  LastName
  SetFirstName
  SetLastName
  SetName
```

public CLASTRING GetName()
The customer's full name
Returns: The first and last name separated by a space.

SetFirstName(CLASTRING FirstName)
Set the first name
FirstName: The customer's first name
Remarks:
This method doesn't do any formatting or case checking - it just assigns the value.

SetLastName(CLASTRING LastName)
Set the last name
LastName: The customer's first name
Remarks:
This method doesn't do any formatting or case checking - it just assigns the value.

SetName(CLASTRING FirstName, CLASTRING LastName)
Set the first and last name
FirstName: The customer's first name
LastName: The customer's last name
Remarks:
This method calls the SetFirstName and SetLastName methods.

Figure 1. User-defined code completion tips

These popups also appear if you hover the mouse over the any code or declaration with an associated comment block.

The IDE parses all text to the right of the !!! marker as XML, which means that the comment format may use only the specified XML tags. If you use different tags, or if the XML is otherwise badly formed, the comment block will be displayed unformatted.

Comments can be placed before the method declaration (inside the CLASS) or before the implementation. Comments in the declaration take precedence.

Class comments

You can associate comments with classes using the above approach. For example:

```
!!! <summary>Customer class</summary>
!!! <remarks>Models a single customer</remarks>
cust          class
...
end
```

If you hover your mouse over cust anywhere in your code you'll see the tooltip, as in Figure 2.

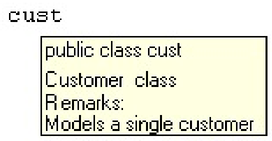

cust
public class cust
Customer class
Remarks:
Models a single customer

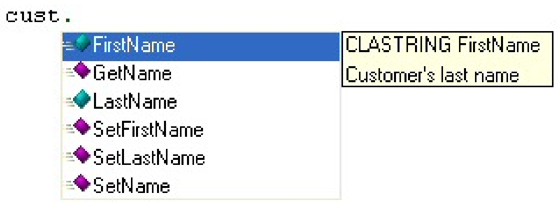
Figure 2. The class comment.

Single line comments

There is an alternate single-line way to add comments: simply follow the declaration with !!! followed by the comment text:

```
FirstName      string(30) !!!Customer's first name
LastName       string(30) !!!Customer's last name
```

Figure 3 shows the comment as displayed for the FirstName variable.


cust.
FirstName
GetName
LastName
SetFirstName
SetLastName
SetName

CLASTRING FirstName
Customer's last name

Figure 3. Single line comments

This is a handy way to document variables. But with methods, be careful: as I mentioned earlier, comments in the prototype supercede any comments in the method implementation. A quick, convenient comment inside the CLASS means more detailed comments in the implementation won't show.

This is a good thing, right?

I realize that not everyone likes code completion. Some think it encourages laziness: programmers rely on the tips instead of on their knowledge of the code base. Others are distracted by the popups (even though you don't have to respond to code completion - you can just keep typing).

If you're not a fan, you can turn code completion off. But I suspect that many Clarion developers will find it enormously useful. By documenting your own code with comments you'll reduce the likelihood of errors, and, I believe, make it easier for new employees to get up to speed on your code base. I also expect that some enterprising third party developer will take the opportunity to create a documentation tool that extracts and formats comments.

One final note: As always, keep in mind that this is alpha code, and there may be changes/enhancements to this functionality before gold release.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Reader Comments

Posted on Sunday, March 25, 2007 by Carl Barnes

XML comments in C# code can be extracted by the compiler into an XML file and "transformed" into HTML documentation pages for the API. Should eventually be possible in Clarion.

Examples:
XML Comments Let You Build Documentation Directly From Your Visual Studio .NET Source Files
http://msdn.microsoft.com/msdnmag/issues/02/06/XMLC/

example: http://msdn.microsoft.com/msdnmag/issues/02/06/XMLC/figures.asp#fig8

How to: Use the XML Documentation Features (C# Programming Guide)
http://msdn2.microsoft.com/en-us/library/z04awywx.aspx

Posted on Wednesday, March 28, 2007 by Ubaidullah Nubar

Is the !!! part configurable, i.e. can it be changed to use !!> as the comment identifier?

Posted on Wednesday, March 28, 2007 by Mark Goldberg

Very nice feature. I'm looking forward to being enticed away from UltraEdit.

This may already be in place, but it seems to me that it would be nice to ignore an entry if it's if empty or all white space
!!! <summary></summary>
!!! <summary> </summary>

This would encourage preset, fill in the blank, comments.

I suppose we could simply alter the !!! symbol for our pre-set comments. Fill in the value, and correct the symbol
for example:
!! <summary></summary>

Will there be any control over formatting of the popups? For example it seems like the remarks tag has an implied CRLF. This is a personal preference, but I found myself wishing that there was a blank line above the "remark:" tag, and no CRLF after it. I also mused over wanting the tag to be highlighted differently from it's content.

Will there be any support for BOLD, ITALIC, UNDERLINE, Color etc.?

There's a conversation on the skype chat regarding the choice of !!! for the magic symbol. A number of us have existing code which uses varying levels of comments. In my case I use it to indicate different sets of remmed source. We're wondering if it would be better to have a different symbol to introduce the magic comments. Brainstorming: !p! for popup, or !d! for documentation. !@! - 'cause it's easy to type (at least it is on a US layout keyboard).

Posted on Wednesday, March 28, 2007 by Viggo Kleven

The !!! may not be very practical, I have that already <g>. Especially when commenting sections of code it tend to increase the number of !!s

!@! is terrible on Norwegian keyboards, requires the AltGr key for the @.

!*! however is very quick and easy.

As Mark Goldberg suggested elsewhere, this could also be !P! for the PopUp and then allow for !D! for lines intended for Documentation parsing. (and !B! for Both).

Posted on Wednesday, March 28, 2007 by Dermot Herron

All I can say is WOW

I have a terrible memory so code completion as powerful as this will be a major godsend for me - I was looking forward to C7 because I could have more than one app open and copy and paste! But this is much better.

Posted on Thursday, March 29, 2007 by Dave Harms

Thanks guys. I've passed along your suggestions to the alpha group for discussion.

Posted on Thursday, March 29, 2007 by Edin Cahtarevic

I think that plain old one "!" should be used.
There is always "<" sign before <summary> or <whatever> and it can be recognized. Like this:

```
! <summary>Set the first name</summary>
! <param name="FirstName">The customer's first name</param>
! <remarks>This method doesn't do any formatting or
! case checking - it just assigns the value.</remarks>
```

So, if there is "! <" sequence, it should be used as start of popup comment, until the closing "</xxxxx>"

It is easier to type and all old comments would be skipped without consequences:

```
! Set the first name
! FirstName: The customer's first name
```

! This method doesn't do any formatting or
! case checking - it just assigns the value.

Regards,
Edin

Posted on Monday, April 09, 2007 by Russell Eggen

There was a confusion about the choice of !!! symbols that started the conversation into this direction. Due to a misduplication of something I said on the Skype chat, more than one person thought the !!! symbols could be changed, they cannot and should not IMNSHO.

This confusion is now cleared up and anyone who uses !!! currently won't affect anything in C7. To make this work you ALSO need the XML tags as Dave described in his article. If those are missing, the !!! is just another comment in your source code. Confirmed by actual tests.

You may now resume your regular coding <bg>.

Add a comment

Clarion Magazine

Embed Analysis Part 3

by Dave Harms

Published 2007-03-22

Last time I showed the basis for my expanded analysis of Clarion embed usage, and I covered some common embed usage including event handling and window initialization. Now it's time to look at browse/process/report embeds and embed usage by procedure type, and I'll wrap up with some conclusions on the use of embeds.

Browse embeds

Here’s the code to extract the browse embed usage data. As with window events, browse events are really places to insert code into class methods.

```
select count(*),param2 from Embed
  where embed = '%BrowserMethodCodeSection'
  group by param2 order by count desc, param2;
```

480	SetQueueRecord
123	TakeNewSelection
92	ValidateRecord
75	ApplyFilter
50	TakeKey
35	PrimeRecord
20	ResetFromView
13	Init
13	UpdateWindow

8	Ask
5	ApplyRange
5	ResetFromAsk
5	ResetQueue
4	TakeEvent
4	UpdateBuffer
3	SetAlerts
2	Open
2	Reset
2	ResetSort
2	ScrollOne
2	SetFilter
1	SetSort

As you probably know, when Clarion displays data in a browse, it's really showing you the data it has in a queue, the structure of which corresponds to the fields you've defined in the browse. The SetQueueRecord method is called whenever the code retrieves a row of data and populates the queue record, so inserting your own code here is a good way to set up calculated fields.

A handful of method calls make up the majority of browse embeds. TakeNewSelection is called whenever the user selects a new record; ValidateRecord is called for each record which is to be loaded into the queue - if you have a filter condition that can't easily be set in the templates you can add it here, and return one of Record:OK, Record:Filtered, or Record:OutOfRange. ApplyFilter is typically used to update the browse filter based on various other conditions such as the value of local variables. TakeKey is used to process alerted keystrokes, including mouse clicks.

Process/Report embeds

There are several ABC classes involved in processes and reports. ProcessClass is derived from ViewManager, as is BrowseClass. And that makes sense since processes, browses and reports all deal in sequential processing of records. The ReportManager class is derived from WindowManager and adds the code needed to run a process and generate a report. Consequently, reports use many of the same embed points as windows and browses.

Embeds by procedure type

I have one last list of embeds, organized by procedure type. Here's the SQL:

```
select ProcFromABC, count(*) as count, embed
  from EmbedProc p left join Embed e
  on (p.embedprocid=e.embedprocid)
  group by ProcFromABC, embed
  order by ProcFromABC, count desc, embed;
```

This list includes embeds generated by third party and custom products, so all of these won't necessarily be available in your application. But if you stick to the embeds with the highest occurrences in each procedure type you're most likely looking at ABC embeds.

Procedure Type	Count	Embed
Browse	89	%ControlEventHandling
Browse	49	%WindowManagerMethodCodeSection
Browse	40	%ProcedureRoutines
Browse	14	%TagMethodCode
Browse	10	%BrowserMethodCodeSection
Browse	10	%ControlHandling
Browse	7	%UltraTreeMethodCodeSection
Browse	5	%DataSection
Browse	3	%WindowEventHandling
Browse	1	%EditInPlaceManagerMethodCodeSection
Browse	1	%WindowManagerMethodDataSection
Form	43	%WindowManagerMethodCodeSection

Form	37	%ControlEventHandling
Form	7	%ProcedureRoutines
Form	6	%EditInPlaceManagerMethodCodeSection
Form	4	%BrowserMethodCodeSection
Form	2	%DataSection
Form	1	%BrowserEIPManagerMethodCodeSection
Form	1	%GlobalMap
Form	1	%LocalDataAfterClasses
Form	1	%LocalProcedures
Frame	187	%WindowManagerMethodCodeSection
Frame	134	%ControlEventHandling
Frame	84	%WindowEventHandling
Frame	28	%DataSection
Frame	27	%ProcedureRoutines
Frame	7	%LocalDataAfterClasses
Frame	6	%ToolbarDropItemAction
Frame	4	%GlobalMap
Frame	3	%ControlPostEventCaseHandling
Frame	3	%ControlPostEventHandling
Frame	3	%GlobalData
Frame	3	%ProcedureSetup
Frame	3	%ProgramSetup

Frame	3	%ToolbarAction
Frame	3	%WindowManagerMethodDataSection
Frame	2	%FileDropMethodCodeSection
Frame	2	%ProgramEnd
Frame	1	%AfterFileClose
Frame	1	%AfterFileDeclarations
Frame	1	%AfterFileOpen
Frame	1	%AfterGlobalIncludes
Frame	1	%AfterWindowOpening
Frame	1	%AnyFontABCDisable
Frame	1	%AnyFontABCEnable
Frame	1	%BeforeGlobalIncludes
Frame	1	%BeforeWindowClosing
Frame	1	%BeforeWindowOpening
Frame	1	%BeginningExports
Frame	1	%BrowserMethodCodeSection
Frame	1	%ControlPreEventHandling
Frame	1	%DLLExportList
Frame	1	%DataSetupSection
Frame	1	%FM2Init
Frame	1	%ListboxStyleAfterDefine
Frame	1	%LocalProcedures

Frame	1	%ToolbarInitBeforeCode
Frame	1	%ToolbarMethodCodeSection
Frame	1	%ValidateSelection
Menu	4	%ProcedureRoutines
Menu	3	%WindowEventHandling
Menu	1	%WindowManagerMethodCodeSection
Process	702	%WindowManagerMethodCodeSection
Process	500	%ProcessManagerMethodCodeSection
Process	132	%ProcedureRoutines
Process	86	%ProcessActivity
Process	37	%AfterFileOpen
Process	36	%DataSection
Process	34	%BeforeFileClose
Process	32	
Process	29	%WindowEventHandling
Process	25	%ControlEventHandling
Process	12	%ProcedureSetup
Process	11	%LocalDataAfterClasses
Process	10	%ControlPostEventHandling
Process	9	%ProgramSetup
Process	6	%BeforeFileOpen
Process	5	%AfterFileClose

Process	5	%ControlPreEventHandling
Process	4	%AfterGlobalIncludes
Process	4	%DataSectionBeforeWindow
Process	3	%DataSectionAfterWindow
Process	3	%EndOfProcedure
Process	3	%GlobalData
Process	3	%GlobalMap
Process	3	%ProcedureInitialize
Process	3	%ProcessManagerMethodDataSection
Process	2	%AfterFileDeclarations
Process	2	%AfterProgramCode
Process	2	%BeforeWindowOpening
Process	2	%ProgramEnd
Process	1	%AfterTurnQuickScanOn
Process	1	%BeforeWindowMakeover
Process	1	%LocalProcedures
Process	1	%NetTalkRefreshCode
Process	1	%WindowEventOpenWindowBefore
Report	730	%WindowManagerMethodCodeSection
Report	584	%ProcessManagerMethodCodeSection
Report	126	%BeforePrint
Report	113	%ProcedureRoutines

Report	68	%AfterFileOpen
Report	56	%AfterPrint
Report	45	%WindowEventHandling
Report	42	%ControlEventHandling
Report	40	%PreviewerManagerMethodCodeSection
Report	32	%BeforeFileOpen
Report	31	%BreakManagerManagerMethodCodeSectionLevelAction
Report	26	%AfterInitialGet
Report	21	%DataSection
Report	20	
Report	18	%AfterOpeningReport
Report	16	%LSiBeforeEndpage
Report	12	%ProcedureSetup
Report	12	%ProcessManagerMethodDataSection
Report	9	%BeforeInitialGet
Report	9	%BeforePrintPreview
Report	7	%LSiAfterOpeningFiles
Report	6	%LSiAfterOpeningReport
Report	6	%ProgramSetup
Report	5	%LSiEndOfReport
Report	4	%DataSectionBeforeReport
Report	4	%GlobalMap

Report	3	%AfterOpeningWindow
Report	3	%ControlPostEventHandling
Report	3	%GetNextRecordNextSucceeds
Report	3	%GlobalData
Report	3	%LocalDataAfterClasses
Report	3	%ProgressCancel
Report	2	%AfterFileClose
Report	2	%BeforeKeySet
Report	2	%ChildViewManagerMethodCodeSection
Report	2	%ControlPreEventHandling
Report	2	%HandCodedViewStatements
Report	2	%LSiAfterPrintingDetail
Report	2	%LSiBeforePrintingDetail
Report	2	%NewMethodCodeSection
Report	2	%ReportTargetMethodCodeSection
Report	2	%TargetSelectorManagerMethodCodeSection
Report	1	%AfterFileDeclarations
Report	1	%AfterGlobalIncludes
Report	1	%AfterTurnQuickScanOff
Report	1	%BeforeClosingReport
Report	1	%BeforeLevel1HdrPrt
Report	1	%BeforeOpeningWindow

Report	1	%FileDropComboMethodCodeSection
Report	1	%LSiBeforeOpeningFiles
Report	1	%LSiReportCanceled
Report	1	%NewMethodDataSection
Report	1	%PostPrintFromQueue
Report	1	%ProcedureInitialize
Report	1	%ProgramEnd
Report	1	%ReportAfterLookups
Report	1	%mhViewValidate
Source	1275	%ProcessedCode
Source	829	%DataSection
Source	342	
Source	76	%ProcedureRoutines
Source	26	%ProgramSetup
Source	23	%AfterFileDeclarations
Source	20	%GlobalMap
Source	19	%GlobalData
Source	12	%FM2Init
Source	12	%LocalProcedures
Source	9	%AfterGlobalIncludes
Source	9	%BeforeGlobalIncludes
Source	7	%AfterProgramCode

Source	7	%ProcRoutines
Source	7	%WindowManagerMethodCodeSection
Source	6	%FileManagerCodeSection
Source	5	%ProgramEnd
Source	5	%RelationManagerCodeSection
Source	3	%BeforeFileOpen
Source	3	%EndOfReportGeneration
Source	3	%RecordFilter
Source	3	%RelationManagerDataSection
Source	2	%AdditionalDebugHooks
Source	2	%AfterEntryPointCodeStatement
Source	2	%AfterFileClose
Source	2	%AfterOpeningReport
Source	2	%BeforeFileClose
Source	2	%ProcessActivity
Source	2	%ProgramProcedures
Source	2	%ProgramRoutines
Source	2	%mhViewInit
Source	1	%AfterClosingExports
Source	1	%AfterDctDestruction
Source	1	%AfterDctInitialization
Source	1	%AfterFileOpen

Source	1	%AfterLevel1FtrPrt
Source	1	%AfterPrint
Source	1	%BeforeFileDeclarations
Source	1	%BeforeInitialGet
Source	1	%BeforeLevel1HdrPrt
Source	1	%BeforeWindowOpening
Source	1	%EndOfProcedure
Source	1	%ErrorManagerCodeSection
Source	1	%ErrorManagerDataSection
Source	1	%FieldLevelValidation
Source	1	%FileManagerDataSection
Source	1	%LocalDataAfterClasses
Source	1	%ProcedureSetup
Splash	17	%WindowManagerMethodCodeSection
Splash	4	%ControlEventHandling
Splash	2	%EventCaseBeforeGenerated
Splash	1	%AfterFileOpen
Splash	1	%AfterGlobalIncludes
Splash	1	%BeforeWindowOpening
Splash	1	%GlobalMap
Splash	1	%ProcedureRoutines
Splash	1	%WindowManagerMethodDataSection

Splash	1	
Window	7061	%ControlEventHandling
Window	4557	%WindowManagerMethodCodeSection
Window	1241	%BrowserMethodCodeSection
Window	1160	%ProcedureRoutines
Window	569	%WindowEventHandling
Window	414	%ControlPostEventHandling
Window	273	%LocalDataAfterClasses
Window	262	%DataSection
Window	169	%ControlHandling
Window	144	%TreeSectionMethodCodeSection
Window	128	%UltraTreeMethodCodeSection
Window	127	%ControlPreEventHandling
Window	114	%NewMethodCodeSection
Window	106	%FormatBrowse
Window	95	%WindowManagerMethodDataSection
Window	85	%AfterWindowOpening
Window	77	
Window	68	%LocalProcedures
Window	61	%FileDropMethodCodeSection
Window	61	%UTVMMethodCodeSection
Window	57	%EditInPlaceManagerMethodCodeSection

Window	54	%TagMethodCode
Window	52	%AfterFileOpen
Window	42	%GlobalData
Window	40	%BrowseBoxEmpty
Window	35	%BeforeWindowOpening
Window	35	%BrowseBoxNotEmpty
Window	35	%ProcedureSetup
Window	34	%BrowserEIPManagerMethodCodeSection
Window	30	%FileLookupMethodCodeSection
Window	30	%RecordFilter
Window	28	%ProcedureInitialize
Window	27	%GlobalMap
Window	27	%ProgramSetup
Window	26	%NetTalkMethodCodeSection
Window	23	%XPTaskPanelTaskClickedAfterCode
Window	21	%BrowseBoxDoubleClickHandler
Window	21	%BrowserMethodDataSection
Window	20	%AfterGlobalIncludes
Window	17	%XPTaskPanelTaskLogicAfterCode
Window	16	%DataSectionAfterWindow
Window	16	%FileManagerCodeSection
Window	15	%AfterPrimaryNext

Window	15	%BeforeGlobalIncludes
Window	15	%NewMethodDataSection
Window	13	%AfterFileDeclarations
Window	13	%DataSectionBeforeWindow
Window	13	%EndOfProcedure
Window	12	%BeforeFileClose
Window	12	%BeforeFileOpen
Window	12	%BeforePrimaryNext
Window	12	%OnInsertAfterPriming
Window	12	%PostWindowEventHandling
Window	11	%ListboxStyleAfterDefine
Window	10	%NetTalkMethodRoutineSection
Window	9	%FM2Init
Window	9	%NextTabEmbed
Window	8	%AfterFileClose
Window	8	%BeforeSecondaryDisplay
Window	8	%FileDropComboMethodCodeSection
Window	8	%XPThemeWindowAfterInit
Window	7	%AfterCallingUpdateOnAdd
Window	7	%AlertKeyCaseKEYCODE
Window	6	%BeforeAccept
Window	6	%BeforeCallingUpdateOnRemove

Window	6	%BeforeFileAction
Window	6	%BeforeSecondaryDisplayCreate
Window	6	%BeginAddEntryRoutine
Window	6	%DasTagAfterTagOnOff
Window	6	%FinishWizard
Window	6	%ProcessedCode
Window	5	%AfterProgramCode
Window	5	%BackTabEmbed
Window	5	%BeforeCallingUpdateOnEdit
Window	5	%BrowseBeforeDelete
Window	5	%LocatorMethodCodeSection
Window	5	%ResizerMethodCodeSection
Window	5	%TreeSectionMethodDataSection
Window	4	%AfterEntryPointCodeStatement
Window	4	%BeforePreparingRecordOnAdd
Window	4	%BrowseAfterChange
Window	4	%BrowseAfterInsert
Window	4	%DasTagAfterTagAll
Window	4	%DasTagBeforeKillTaging
Window	4	%FormAllow
Window	4	%UltraTreeMethodDataSection
Window	3	%AfterCallingUpdateOnEdit

Window	3	%AfterCallingUpdateOnRemove
Window	3	%AfterSecondaryNext
Window	3	%AfterTagOp
Window	3	%BCSIfSelect
Window	3	%BeforePrimaryDisplayCreate
Window	3	%BeforeUntagAll
Window	3	%BrowseBeforeChange
Window	3	%BrowseBeforeInsert
Window	3	%ControlOtherEventHandling
Window	3	%CustomAlertEmbed
Window	3	%DasTagBeforeTagAll
Window	3	%FileLookupMethodDataSection
Window	3	%ListboxStyleBeforeDefine
Window	3	%PrimeFields
Window	3	%TEBrowseDropHandlingAfter
Window	3	%TEDropIDOk
Window	3	%XPTaskPanelTaskLogicBeforeCode
Window	2	%AcceptLoopAfterEventHandling
Window	2	%AcceptLoopBeforeEventHandling
Window	2	%AdditionalDebugHooks
Window	2	%AfterImportExcel
Window	2	%BCSLicenseEmbed

Window	2	%BeforeCallingUpdateOnAdd
Window	2	%BeforePrimaryDisplay
Window	2	%BeforeWindowClosing
Window	2	%BreakManagerManagerMethodCodeSectionLevelAction
Window	2	%BrowseAfterDelete
Window	2	%DasTagAfterInitTaging
Window	2	%DasTagBeforeTagOnOff
Window	2	%FieldLevelValidation
Window	2	%LookupRelated
Window	2	%NetTalkAfterInitSection
Window	2	%NetTalkMethodDataSection
Window	2	%OnInsertBeforePriming
Window	2	%ProcessManagerMethodCodeSection
Window	2	%TETreeDropHandlingAfter
Window	2	%VerResourceValueList
Window	2	%WindowInitializationCode
Window	1	%AcceptLoopBeforeFieldHandling
Window	1	%AfterFileNext
Window	1	%AfterInsertRecord
Window	1	%AfterOpeningReport
Window	1	%AfterTotalLoop
Window	1	%AfterTurnQuickScanOff

Window	1	%AfterWindowClosing
Window	1	%AuditData
Window	1	%BeforeAddingStyles
Window	1	%BeforeFlipAll
Window	1	%BeforeFlipOne
Window	1	%BeforeInlineFileAction
Window	1	%BeforePrint
Window	1	%BeforeSecondaryNext
Window	1	%BeforeTagAll
Window	1	%BeforeTagOne
Window	1	%BeforeUntagOne
Window	1	%BrowseBoxAfterUpdate
Window	1	%BrowsePrepNormal
Window	1	%BrowsePrepSelectRecord
Window	1	%DasTagAfterUnTagAll
Window	1	%DasTagBeforeUnTagAll
Window	1	%EIPClickAccepted
Window	1	%EIPEventSelected
Window	1	%EndOfFormatBrowse
Window	1	%FEPreCodeSection
Window	1	%FileDropMethodDataSection
Window	1	%HandyInterNetFtpBeforeInit

Window	1	%HandyInterNetFtpULAborted
Window	1	%HyperActivePostCodeSection
Window	1	%HyperActivePreCodeSection
Window	1	%INIManagerCodeSection
Window	1	%JSTokenTextSelected
Window	1	%MCRTAfterSetQueueRecord
Window	1	%PDFXCDriverDocSaved
Window	1	%PXCDV3PBeforeRunEmbed
Window	1	%ProgramEnd
Window	1	%RefreshWindowBeforeLookup
Window	1	%TaskbarIconEmbed
Window	1	%TaskbarIconMessageProcessing
Window	1	%WinEventTaskBarPopupItems
Window	1	%WindowOtherEventHandling

Conclusions and recommendations

Clarion ABC applications present a bewildering array of embed points, but as this analysis shows, the vast majority of embeds are in just a few areas, including window initialization, event handling, browse display, and source code procedure code and data.

There are still a lot of developers using legacy embed names, which suggests a lack of familiarity with ABC. While legacy embeds have the lure of familiarity they mask the real workings of ABC, and that can make it more difficult to use ABC to its fullest. If you're using ABC but you're not sure what's really happening behind the scenes, it's time to start reading. Resources include the Clarion ABC help, and ClarionMag's ABC Internals and Using ABC topics. You may also want to take a look at a couple of books: Bruce Johnson's Programming in Clarion ABC is available from CapeSoft, and Russ Eggen's Programming Objects in Clarion is available in the ClarionMag store in print and PDF versions.

There is also clearly a lot of code in source procedures. While this is better than having that same code sprinkled throughout embeds, a lot of it can probably be converted to classes (see CLASSy ASCII File Importing for an example), particularly where several source procedures work on common data.

For further information on using embeds see the list of Related Articles on this page, or go to the Embeds topical index page.

I hope this analysis provides some useful insight. If you have a favorite embed point or two which I haven't covered (or haven't adequately explained), let me know.

My thanks again to all who contributed TXAs.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Reader Comments

Posted on Thursday, March 22, 2007 by Tony Tetley

Dave,

This was an interesting and informative series. I wish these articles had been present when I first started into ABC. It takes some time to get over the overwhelming number of embed points and realize that you don't need to use them all.

Thanks,

Tony

Posted on Thursday, March 22, 2007 by Dave Harms

Tony,

Thanks, I'm glad you've found it helpful.

Dave

Posted on Tuesday, April 03, 2007 by Dave Harms

The following people contributed TXAs for analysis. My thanks to all of you!

- Anonymous
- Paul Blais
- Cliff Cady
- Tom Carswell
- Andrew Crockett
- Rick Dafler
- Ben Dell

- Peter Gysegem

- Jim Halpin

- Jorge Herrera

- Terry Hill

- Lynn Howard

- Doug Johnson

- Jeffrey Kolker

- James Lishman

- John Morter

- Geoff Prudames

- Steffen Rasmussen

- Ned Reiter

- Tony Tetley

- Vido Vouk

- Bill Wilson

- Des Yaxley

If I've missed anyone please let me know.

Dave

Add a comment

Clarion Magazine

Embed Analysis Part 2

by Dave Harms

Published 2007-03-21

Embed points are Clarion's great strength: they let you embed custom code within generated code so you get the best of both worlds. But there are a bazillion embed points. How do you know which ones to use? One way is to study the documentation and make educated guesses. Another is to ask experienced Clarion developers which embeds they favor.

A while ago I decided to automate the latter approach. Clarion APPs can be exported as TXAs, which are text files containing all of the APP's information, including embed points. I wrote some code to parse TXAs looking for the embeds, and I published some preliminary results in January. I also asked developers to send me their TXAs for analysis. The response was better than I expected, and I now have over 300 TXAs in hand. In this article I'll go over some of my findings, and I'll show the SQL statements I used to extract the data.

A storage problem

In my original analysis I stored embed data on the Clarion Magazine server, which is physically about 1500 miles (2400km) from my office. As I only had a handful of TXAs to process, the slowdown induced by inserting records across the Internet was only a minor inconvenience. But with over 300 TXAs to process I needed to set up a local database.

This seemed like an excellent opportunity to reacquaint myself with PostgreSQL, which is one of the few truly free open source SQL databases (the other major player being Firebird; MySQL, contrary to popular perception, is not free for commercial use). I'll have more on installing and running PostgreSQL in another article; for now I'll just say that I installed PostgreSQL without difficulty and I've been pleased with its performance in my limited testing to date.

The schema

Here's my database schema as described by psql, the PostgreSQL command line interpreter. First, the list of tables and sequences (a.k.a. autonumbering keys):

```
embeds=# \d
                List of relations
```

```
 Schema |            Name             |   Type   |  Owner
--------+-----------------------------+----------+----------
 public | embed                       | table    | postgres
 public | embed_embedid_seq           | sequence | postgres
 public | embedapp                    | table    | postgres
 public | embedapp_embedappid_seq     | sequence | postgres
public | embedproc                   | table    | postgres
 public | embedproc_embedprocid_seq   | sequence | postgres
```

And here are the details for each of the three tables: embedapp, embedproc, and embed:

```
embeds=# \d embedapp;
                          Table "public.embedapp"
   Column    |     Type      |             Modifiers
-------------+---------------+--------------------------------------
 embedappid   | integer       | not null default nextval
                             ('embedapp_embedappid_seq'::regclass)
 txa          | character(60) |
 embedchainid | integer       | not null
Indexes:
    "embedapp_pkey" PRIMARY KEY, btree (embedappid)


embeds=# \d embedproc;
                          Table "public.embedproc"
   Column    |     Type      |             Modifiers
-------------+---------------+--------------------------------------
 embedprocid  | integer       | not null default nextval
                             ('embedproc_embedprocid_seq'::regclass)
 embedappid   | integer       | not null
 embedchainid | integer       |
 procname     | character(60) |
 procfromabc  | character(60) |
 proccategory | character(60) |
Indexes:
    "embedproc_pkey" PRIMARY KEY, btree (embedprocid)


embeds=# \d embed
```

```
                      Table "public.embed"
   Column    |     Type      |             Modifiers
-------------+---------------+--------------------------------------
 embedid     | integer       | not null default nextval
                              ('embed_embedid_seq'::regclass)
 embedchainid | integer       |
 embedprocid  | integer       | not null
 embed        | character(100) |
 param1       | character(60)  |
 param2       | character(60)  |
 param3       | character(60)  |
 priority     | integer        |
 linesofcode  | integer        |
Indexes:
   "embed_pkey" PRIMARY KEY, btree (embedid)
```

The embedapp table corresponds to the TXA; the embedproc table stores procedure embed information; and the embed table holds the data for each embed, including the embed name, the parameters, the priority number, and the number of lines of code (although at present I'm not tracking lines).

Parsing TXAs

I very briefly covered the process of parsing TXAs in the first embeds analysis article, so I won't go into that here except to note that I'm now also filtering based on template type. For this article I'm only looking at TXAs from ABC applications.

And once again I feel compelled to note that this task would be a heck of a lot easier of TXAs were XML files because of what I perceive as inconsistent tag usage. That's just one more example of the virtues of XML over ad-hoc text file formats.

Procedure types

The first useful bit of information I extract after processing the TXAs is a summary of the procedure types represented in the database. Here's the SQL statement:

```
select count(*) as count, ProcFromABC,ProcCategory
  from EmbedProc group by ProcFromABC,ProcCategory
  order by count desc, ProcFromABC, ProcCategory;
```

And here are the first fifty records:

Count	**Procedure**	**Template**
1457	Source	
1272	Window	
1174	Window	Browse
932	Window	Form
538	Report	
430	Process	
101	Process	Process
92	Window	Window
85	Frame	
72	Source	Source Window
71	Report	Report
25	Browse	
20	Window	Assign
18	Splash	
17	Window	ReverseEngineer
17	Source	Library
15	Window	Browse Tree
14	Window	SFR
14	Source	Global
13	Window	assign
13	Window	UT BOM - Select
13	Window	UT BOM
13	Report	Invoices

11	Form	
10	Process	MapMaker
10	Source	Source
9	Source	Source Report
8	Window	Select
7	Browse	Reallocate
7	Window	ReverseEngineering
6	Window	Menu
6	Window	re-assign
6	Window	Analyses
5	Window	UTBOM
5	Window	Actions
5	Window	Import
5	Window	SQL
4	Source	Function
4	Source	Generic Function
4	Window	New
4	Window	Generic Window Dialog
4	Source	Holder
4	Window	Import-Export
3	Form	Form
3	Window	Library - Window
3	Process	Actions
3	Window	Viewer

3	GENERATED	

This list can be a bit confusing to read at first. The Procedure column is the procedure type, and for the most part this is limited to one of Frame, Window, Report, Process and Source. Generally speaking only the first few rows are of much significance, as most procedures with 25 or fewer instances are custom templates.

I was a bit surprised to see Source procedures in the number one spot. These are hand coded procedures, but created within the AppGen. It's a good thing to see; almost any application will have some code that lends itself to a custom function, and placing this code in a source procedure makes it more maintainable.

Window procedures are a bit of a catch-all, since browses and forms are both built on top of generic windows. This statement retrieves all the Window procedure subtypes:

```
select count(*) as count, ProcCategory from EmbedProc
  where ProcFromABC = 'Window' group by ProcCategory
  order by count desc, ProcCategory;
```

And here are the results:

Count	**Category**
1272	*not specified – generic window*
1174	Browse
932	Form
92	Window
20	Assign
17	ReverseEngineer
15	Browse Tree
14	SFR
13	UT BOM
13	UT BOM - Select
13	assign
8	Select
7	ReverseEngineering
6	Analyses
6	Menu

6	re-assign
5	Actions
5	Import
5	SQL
5	UTBOM
4	New
4	Generic Window Dialog
4	Import-Export
3	Library - Window
3	Promote
3	Viewer
2	Assign UT - H
2	BrowseTree
2	CODE window
2	Calendar
2	Fields
2	Generate
2	Global
2	Graphs
2	H-T and H-T
2	Invoices
2	Materials
2	Moisture
2	Ruddscale
2	Tag
2	UT-H Tag
2	Wizard
1	Diary
1	FTP

1	Groups
1	MapMaker
1	Reallocate
1	Remove/Reassign
1	ReportManager
1	SFR-T and H-T
1	SFRTag
1	SFRTag and H-T
1	Selects
1	Setup - E
1	Solace VariView
1	Source
1	UT Tag

Again, only the first few rows are really valuable for analysis as the custom templates make their appearance soon afterward. Window procedures without a specific subtype are generic windows, on which developers populate their own controls. Browse, Form and Window subtypes are procedures created with their respective wizards.

The embeds

Now on to the embeds. In ABC, embed points are for the most part locations within virtual methods, which is itself a topic well beyond the scope of this article. But in short, ABC applications rely heavily on the ABC class library, and most of the code that does the work is contained in those classes. Whereas legacy applications generate all the code your application needs, ABC applications generate derived classes that add the functionality you specify in the templates, along with any code you place in embed points.

This is the statement I use to retrieve the list of most-used embed points:

```
select count(*),embed from embed group by embed order by count desc,embed;
```

And here are the results:

Count	Embed	Description
7392	%ControlEventHandling	Events for controls on the window

6293	%WindowManagerMethodCodeSection	WindowManager methods, code section – one for each method
1560	%ProcedureRoutines	Local routines
1281	%ProcessedCode	After the CODE statement in a Source procedure
1256	%BrowserMethodCodeSection	BrowseManager methods
1183	%DataSection	Before the CODE statement in a Source procedure
1086	%ProcessManagerMethodCodeSection	Process/Report methods
733	%WindowEventHandling	General window events
430	%ControlPostEventHandling	Code to execute after window control event handling
296	%LocalDataAfterClasses	Procedure data, after class declarations
179	%ControlHandling	General control handling
160	%AfterFileOpen	After files are opened
144	%TreeSectionMethodCodeSection	Third party
135	%ControlPreEventHandling	Code to execute before window control event handling
135	%UltraTreeMethodCodeSection	Third party
127	%BeforePrint	Legacy embed – same as TakeRecord, before printing
116	%NewMethodCodeSection	At end of procedure, before routines

106	%FormatBrowse	Legacy embed, same as SetQueueRecord
100	%WindowManagerMethodDataSection	Window manager methods, data section – one for each method
88	%ProcessActivity	Legacy embed, same as process/ report TakeRecord method
86	%AfterWindowOpening	Legacy embed, same as WindowManager.Init priority 8100
83	%LocalProcedures	After %LocalRoutines embed
71	%GlobalData	For global data declarations
71	%ProgramSetup	Program setup, after dictionary is initialized
The following embeds are not annotated		
68	%TagMethodCode	
64	%EditInPlaceManagerMethodCodeSection	
63	%FileDropMethodCodeSection	
63	%ProcedureSetup	
61	%UTVMMethodCodeSection	
60	%GlobalMap	
57	%AfterPrint	
53	%BeforeFileOpen	
48	%BeforeFileClose	
41	%ModuleDataSection	

40	%AfterFileDeclarations	
40	%BeforeWindowOpening	
40	%BrowseBoxEmpty	
40	%PreviewerManagerMethodCodeSection	
36	%AfterGlobalIncludes	
35	%BrowseBoxNotEmpty	
35	%BrowserEIPManagerMethodCodeSection	
33	%BreakManagerManagerMethod CodeSectionLevelAction	
33	%RecordFilter	
32	%ProcedureInitialize	
30	%FileLookupMethodCodeSection	
26	%AfterInitialGet	
26	%NetTalkMethodCodeSection	
25	%BeforeGlobalIncludes	
23	%XPTaskPanelTaskClickedAfterCode	
22	%FM2Init	
22	%FileManagerCodeSection	
21	%AfterOpeningReport	
21	%BrowseBoxDoubleClickHandler	
21	%BrowserMethodDataSection	
19	%DataSectionAfterWindow	
18	%AfterFileClose	

17	%DataSectionBeforeWindow	
17	%EndOfProcedure	
17	%XPTaskPanelTaskLogicAfterCode	
16	%LSiBeforeEndpage	
16	%NewMethodDataSection	
15	%AfterPrimaryNext	
15	%ProcessManagerMethodDataSection	
14	%AfterProgramCode	
12	%BeforePrimaryNext	
12	%ListboxStyleAfterDefine	
12	%OnInsertAfterPriming	
12	%PostWindowEventHandling	
11	%ProgramEnd	
10	%BeforeInitialGet	
10	%NetTalkMethodRoutineSection	
9	%BeforePrintPreview	
9	%FileDropComboMethodCodeSection	
9	%NextTabEmbed	
8	%BeforeSecondaryDisplay	
8	%XPThemeWindowAfterInit	
7	%AfterCallingUpdateOnAdd\	
7	%AlertKeyCaseKEYCODE	

7	%LSiAfterOpeningFiles	
7	%ProcRoutines	
6	%AfterEntryPointCodeStatement	
6	%BeforeAccept	
6	%BeforeCallingUpdateOnRemove	
6	%BeforeFileAction	
6	%BeforeSecondaryDisplayCreate	
6	%BeginAddEntryRoutine	
6	%DasTagAfterTagOnOff	
6	%FinishWizard	
6	%LSiAfterOpeningReport	
6	%ToolbarDropItemAction	
5	%BackTabEmbed	
5	%BeforeCallingUpdateOnEdit	
5	%BrowseBeforeDelete	
5	%LSiEndOfReport	
5	%LocatorMethodCodeSection	
5	%RelationManagerCodeSection	
5	%ResizerMethodCodeSection	
5	%TreeSectionMethodDataSection	
and a bunch more…		

There are obviously some third party embed points in this list, such as those for NetTalk, DAS Tools, and Clarion Handy Tools. Some of these embeds (such as %ProcedureRoutines and %BeforePrint are single embeds; others represent multiple embeds, such as %ControlEventHandling. Also note that there are still quite a few developers using the Legacy embed view – this view in the embed list shows the familiar Legacy names which eases the transition to the ABC classes.

Event embeds

The first parameter to the %ControlEventHandling embed is the field equate, and the second is the event, so to get the list of commonly used event equates I use this statement:

```
select count(*),param2 from Embed
  where embed = '%ControlEventHandling'
  group by param2 order by count desc, param2;
```

And here is the data:

Priority	Event
2777	Accepted
90	AlertKey
88	Selected
72	NewSelection
15	PreAlertKey
10	Drop
9	Drag
7	TabChanging
3	MouseIn
2	Expanded
1	MouseUp

No big surprises on the event handling, except maybe that so few coders use drag/drop or other mouse movement events.

WindowManager method embeds

How about the WindowManager method embeds? Here’s the SQL (note that I’m back to param1 again):

```
select count(*),param1 from Embed
  where embed = '%WindowManagerMethodCodeSection'
  group by param1 order by count desc, param1;
```

And the data:

Count	**Method**
4944	Init
160	Kill
91	Reset
87	Open
83	AskPreview
78	Update
74	TakeCompleted
66	Run
56	PrimeFields
50	OpenReport
35	TakeNoRecords
28	TakeFieldEvent
26	TakeWindowEvent
17	SetControlProperties
14	Ask
14	PrimeUpdate
13	InitControlProperties

12	TakeEvent
11	TakeAccepted
5	InsertAction
5	TakeSelected
4	EndReport
4	TakeRecord
3	TakeCloseEvent
3	TakeNewSelection
2	SetControlValues
1	SetAlerts

Clarion programmers love that WindowManager.Init method. And why not? It’s a great place to prime variables, create controls, and do all sorts of other setup tasks. Here’s a closer look the preferred priorities:

```
select count(*),priority from Embed
  where embed = '%WindowManagerMethodCodeSection'
  and param1 = 'Init' group by priority order by priority;
```

Count	Priority
6	1
1	2
1	5
1	10
35	50
35	300

34	450
45	500
234	501
1	1000
4	1300
8	1500
11	1700
7	1800
1	2000
4	2001
2	2250
3	2300
89	2500
7	2501
	hang on a sec…

Okay, I could go on with that list for a few more pages. Here's something more sensible – priorities grouped by thousands:

```
select count(*),floor(priority/1000)*1000 as floor from Embed
  where embed = '%WindowManagerMethodCodeSection'
  and param1 = 'Init'
  group by floor order by floor;
```

Count	Priority
392	0

31	1000
133	2000
50	3000
440	4000
615	5000
325	6000
1158	7000
1238	8000
554	9000
8	10000

Ah, that's a bit more useful. The data now shows the number of embed points at priority 0-999, 1000-1999, and so forth. (As an aside, this is one of the reasons I love working in SQL. I didn't have to write any fancy code to process the list of embeds and do sums – I just employed a server side function (`floor`, in this case) and let the server do the work. I'm far from an SQL expert but I often use statements like these to generate ad-hoc reports.)

But what do these priorities mean? Embed priorities simply let you assign code at various places in the Init code. Here's an example of Init code taken from the Embeditor:

```
ThisWindow.Init PROCEDURE

ReturnValue          BYTE,AUTO

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

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 300]

  ! Enter procedure scope
```

```
 GlobalErrors.SetProcedureName('ImportTXAs')
 ! [Priority 2700]

 ! Snap-shot GlobalRequest
 SELF.Request = GlobalRequest
 ! [Priority 4950]

 ! Parent Call
 ReturnValue = PARENT.Init()
 ! [Priority 5050]

 ! Set options from global values
 IF ReturnValue THEN RETURN ReturnValue.
 SELF.FirstField = ?List1
 SELF.VCRRequest &= VCRRequest
 SELF.Errors &= GlobalErrors
 ! [Priority 5300]

 ! BIND variables
 ! [Priority 5800]

 ! Setup Toolbar Object
 SELF.AddItem(Toolbar)
 ! Initialize the procedure
 CLEAR(GlobalRequest)
 CLEAR(GlobalResponse)
 ! [Priority 6500]

 ! Procedure setup standard formulas
 IF SELF.Request = SelectRecord
   SELF.AddItem(?Close,RequestCancelled)
 ELSE
   SELF.AddItem(?Close,RequestCompleted)
 END
 ! [Priority 7300]

! Open Files
```

```
Relate:TextFile.Open
Relate:embed.Open
Relate:embedapp.Open
Relate:embedchain.Open
Relate:embedproc.Open
SELF.FilesOpened = True
! [Priority 7800]

! Open the window
SELF.Open(Window)
! [Priority 8005]

! Call ListBoxStyle Define Routine
Do DefineListboxStyle
! [Priority 8080]

! Restore from INI file
INIMgr.Fetch('ImportTXAs',Window)
! [Priority 8400]

! Process field templates
! [Priority 8800]

! Prepare Alert Keys
SELF.SetAlerts()
! [Priority 9500]
 DIRECTORY(txaq,'*.TXA',ff_:NORMAL)   !Get all files and directories
! End of "WindowManager Method Executable Code Section"
RETURN ReturnValue
```

Files are opened later on in the process; since embed code often uses file data, it’s no surprise the majority of the embed points in use are later on in the Init method.

In the next installment I'll continue with a look at browse/process/report embeds and embed usage by procedure type, and I'll wrap up with some conclusions on the use of embeds.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Reader Comments

Add a comment

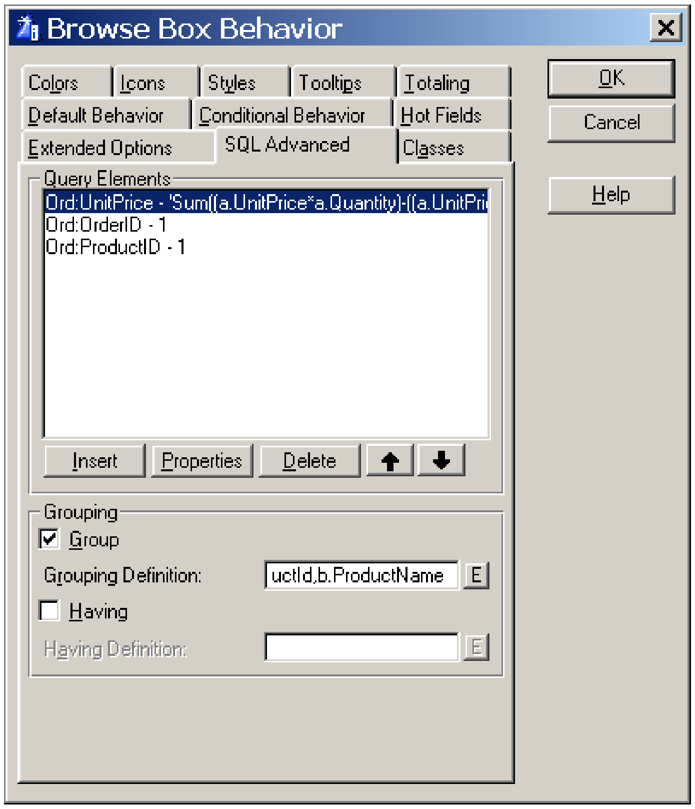
Clarion Magazine

Using the SQL Advanced Tab

by Bjarne Havnen

Published 2007-03-16

In Clarion 6 a SQL Advanced tab was introduced to the Browse Box Behaviour window. The tab's primary purpose appears to be to display calculated fields, for example SUM, AVG, COUNT, but these fields can display any kind of information, such as from stored procedures and lookups in unrelated tables. In this article I'll explain the usage of this tab and the related SQL properties (PROP:Name, PROP: Where, PROP:Order ,and PROP:GroupBy), and I'll give a demonstration of highly effective totalling on a regular Clarion ABC browse.



Ord:UnitPrice - 'Sum((a.UnitPrice*a.Quantity)-((a.UnitPri
Ord:OrderID - 1
Ord:ProductID - 1

[✓] Group
Grouping Definition: uctId,b.ProductName
[ ] Having

Figure 1. The SQL Advanced tab

The basic idea of these template changes is rather simple: You set up a list of fields for the view engine and you specify the field values.

Field	SQL statement	Meaning

Ord:Field1	Sum(b.Total)	Sum itemlines
Ord:Field2	(Select CustNo from Customers Where CustomerId = a.CustomerId)	Display non linked field instead of linked field
Ord:Field3	1	1

If you have a VIEW with the above fields (for now ignoring the Value column), the runtime library will create a SELECT statement like this:

Select a.Field1,A.Field2, A.Field3 From Orders A Join,Where etc….

With the values specified above, however, the templates will generate some code to assign the SQL statements (where specified) to each field. I'll explain the syntax of that code a little later, but the basic idea is to use the new PROP:Name syntax to replace a field in the SELECT statement with a custom SQL statement. In this way you can embed sub-selects and aggregate functions in the SQL statement that's sent to the back end.

Here's a simple example. My challenge is a quite common one: I like to display the total sales per product. For this example I'll use the Order_details table from the Northwind sample database. This table does not store the totals, so I can't just do a SUM(TOTAL) which I otherwise would have done.

The syntax to get the sum per product is:

```
SELECT a.ProductID, b.ProductName, SUM(a.UnitPrice * a.Quantity -
  a.UnitPrice * a.Quantity * a.Discount) AS Total
  FROM [Order Details] a INNER JOIN
  Products b ON b.ProductID = a.ProductID
  GROUP BY a.ProductID, b.ProductName
```

Figure 1 shows Clarion 6's SQL Advanced tab with several custom field assignments. The reason for assigning a value of 1 to ORDERID and PRODUCTID is the way Clarion projects fields when linking other tables - the linking fields in both tables are projected and the SQL engine won't allow this. Assigning a value of 1 to the field makes the query look like:

```
Select Sum etc, 1,1, b.productId, b.productname group by ...
```

which is legal. More on this later.

The example in Figure 1 won't work, however. Most queries are illegal because the primary key field is projected into the view structure. This has since been fixed with a checkbox to suppress the primary key field (look on the Extended options tab), but there are still quite a few pitfalls. The primary key field is also the preferred ORDER BY field, unless PROP:ORDER is overridden in the Additional Sort field. Also, when browsing the result of a total query not related to a particular record you will need to set the browse to File Loaded since there is no means for Clarion to refresh without refreshing the entire browse.

What

properties?

MS SQL Server and Sample Database

If you don't already have SQL Server and the Northwind database, you may wish to install Microsoft's freely available SQL Server 2005 Express Edition.

The Northwind example database is not included with SQL Sever Express but is available as a separate download.

The SQL Advanced tab involves PROP:NAME, PROP:GROUPBY and PROP:HAVING, which all require correct SQL syntax just like PROP:SQL. Initially I didn't understand the importance of these properties, as I'd been using PROP:SQL for all my custom SQL code. My enlightenment came later – and I'll get back these properties in a bit. First, here's a real world example.

Regular Totaling - retiring ResetFromView

In Clarion's standard (non-SQL Advanced) approach getting a total means scanning the records in the view. This is okay for most parent-child relations, but it is a performance killer in unfiltered browses and likewise when a filter is applied that cannot be evaluated on the server. I can make the totalling conditional, but unfortunately I can't change the filter from, say, *this month's orders* to *all orders* without toggling the totalling variable. I doubt I have to demonstrate that scanning a million records on every reset results in a big performance hit. One day I made that mistake with my customer's data, which set me on a quest to solve the problem.

What I wanted to do was to make a VIEW similar to the browse view but with a minimum of fields, using the browse's filter and retrieving the total on the filtered view. In order for the filters to be valid I also had to join some related tables, which immediately caused a failure due to Clarion's JOIN implementation.

When you use JOIN Clarion projects all fields in a table, unless you specify individual fields with PROJECT(field). When combined with PROP:NAME using an aggregate function, the SELECT statement is invalid. Consider this view:

```
View:Orders View(Orders)
  Project(Ord:Total)
  JOIN(Cus:K_CustomerId,Ord:CustomerId)
    Project(Cus:Name)
  End
End
```

Say I use PROP:Name to replace Ord:Total with a SUM statement:

```
View:Orders{'Ord:Total',Prop:Name}='Sum(Total)'
```

This code will result in a select statement such as:

```
Select Sum(Total),b.Name from orders ...
```

This SQL is invalid because it combines an aggregate function with a non-aggregate field (b.Name) and there is no Group By clause. If, however, I add this statement:

```
View:Orders{'Cus:Name',Prop:Name}=1
```

then the SQL sent to the backend will be:

```
Select Sum(Total),1 From Orders Join Customers on ...
```

which turns out to be perfectly valid SQL.

Below is some sample code that overrides the ResetFromView method to total some fields in a related table. I also do a count on the resultset in order to display the number to the user. Invalid filters will report the totals on the entire table, and this way I can take appropriate action if the number of records totalled is equal to the total number of records in the table. This is based on my own tables; I'll leave it up to you to make it work in yours.

```
BRWOrders.ResetFromView PROCEDURE



!variable to hold file status
LStat Ushort



!Declare view, same joins as the browse view, but less fields

View:Orders View(Orders)
  Project(Ord:OrderId)
  Project(Ord:Total)
  JOIN(Cus:K_CustomerId,Ord:CustomerId)
    Project(Cus:Name)
  End
End



  CODE
  LStat = Self.Primary.me.SaveFile() !save pointer View:Orders{Prop:Filter}=Self.View{Prop:Filter} !
Copy Filter
  View:Orders{Prop:Order}='SQL(1)' !order by first field to suppress the Primary field
  Open(View:Orders) !open view
  View:Orders{'Ord:Total',Prop:Name}='Sum(Total)'
  View:Orders{'Ord:OrderId',Prop:Name}='Count(OrderId)'
  View:Orders{'Cus:Name',Prop:Name}=1
  Set(View:Orders)
  Next(View:Orders)
  Recs = Ord:OrderId
  Total = Ord:Total
  Close(View:Orders)
  Self.Primary.Me.RestoreFile(lStat)
  Return
```

This override gives me the total opposite of an application hang – it results in an immediate response. The one thing it does not do is handle the conditional totalling. This can be solved by repeating the sequence per condition and appending the filter to the regular filter.

PROP:Name

PROP:Name is not a new property, but in good Clarion spirit it is extended to the VIEW structure.

Consider this VIEW:

```
View:Orders VIEW(Orders)
  PROJECT(Ord:Total)
End
```

As I indicated earlier, with PROP:NAME I can change the value retrieved from the backend:

```
View:Orders{‘Ord:Total',PROP:NAME} = ‘Sum(Total)'
```

This will transform the query from:

```
Select Total From Orders
```

to

```
Select Sum(Total) From Orders
```

PROP:Name can be used with any value, as long as that value is valid SQL.

PROP:GROUPBY

This property sets the SQL Group By clause. If I extend the view from above to include a customer number the query will fail with the error "column is invalid in the select list because it is not contained in a aggregate function and there is no GROUP BY clause".

I can use PROP:GroupBy to show totals per customer:

```
View::Orders{Prop:GROUPBY}='customerid'
```

PROP:HAVING

HAVING is a filtering query element; it can be used to narrow the result set to match any criteria. For example:

```
! show only customers having at least
! 10000 dollars worth of orders.
View:Orders{PROP:HAVING}='Sum(Total)>10000'
```

This is a basic example, and it might not work. SQL is a beast, and it is not tamed by Clarion alone – a developer needs to know both his SQL and his Clarion to have any fun. When using PROP:HAVING you have to evaluate one of the fields in the query list, and sometimes you have to remember to prefix the fields with A, B,C, D etc based on their position in the file (this can be changed with PROP: ALIAS). Recently I have made an habit of prefixing every field regardless of the number of tables in the view; that way the code won't break if I join more tables.

SQL() and PROP:ORDER

SQL() is a new function in Clarion 6. It is a replacement for PROP:SQLFILTER and PROP:SQLORDER that can be used to concatenate a regular Clarion expression with a SQL expression. Thus, it can be used in the filter and order field of any Clarion VIEW. I've used it extensively to do table lookups with EXISTS or IN instead of linking all possible tables. Typically, when I needed to email my Clarion customers I used a query to runtime filter the customer list to only those who had purchased some of my Clarion-related products.

PROP:ORDER has existed for ever, right? Yes it has, but if you try to use PROP:NAME without a valid order part, the query will fail. The reason is that Clarion expects an ORDER clause as part of any view processing statement. If you don't provide one, Clarion will by adding the primary key field as the ORDER BY part. This will cause a similar "column is invalid" error. One solution is to set the additional sort order to the same as the PROP:GROUPBY columns, where that applies. Another solution that I stumbled over is to use the SQL() function and set:

```
View:Orders{Prop:Order}='SQL(1)'
```

With MSSQL the number refers to the select list, so the server will group by the first query element, in this case SUM(Total), ordering with the lowest value first. This simplifies the coding in some circumstances. It is also considered by SoftVelocity development team to be the best workaround for a missing PROP:ORDER.

Why not just use PROP:SQL then?

Now, what's the point of using these properties instead of the more common PROP:SQL approach? The answer is, believe it or not, simplicity. These properties can be used together with all the different VIEW properties, so I can combine a Sum(), AVG(), COUNT() with, for example, PROP:FILTER, using the best of both worlds.

I tend to make errors with date filtering in my PROP:SQL statements. If I choose to filter on the Clarion date using PROP:FILTER, the RTL will translate for me. My code looks like this:

```
View:Orders{Prop:Filter}='Ord:CustomerId=1 And '|
 & 'Ord:OrderDate_Date>=Date(1,1,Year(Today()))'
View:Orders{‘Ord:Total',Prop:Name}='Sum(Total)'
```

What the two approaches have in common is that the filter that is sent to the backend is valid SQL. Clarion doesn't translate everything, but the view engine will handle invalid filters on the client. This can't work with aggregate functions, as they are evaluated at the server. Recently, one of my customers made a filter using MONTH() and YEAR() and TODAY(). It was rather clever, but the combination resulted in a client side filter, effectively reporting the grand total of the entire table instead of the filtered result. I changed the filter to something like

```
Ord:OrderDate>=Date(Month(Today()),1,Year(Today()))
```

and the Clarion RTL translated it correctly.

Another important difference is that when using PROP:SQL I can't use SET(view) as I would otherwise. This difference is one more thing to think of and I like my code to follow certain conventions. The old PROP:SQL approach and the new properties share one common problem: because you're using string constants for the field names, changes in the dictionary aren't automatically carried forward to your hand-coded SQL statements.

PROP:WHERE

PROP:WHERE is actually a file property, not a VIEW property. It works just the way PROP:SQLFilter does on a view. The practical use of this can best be seen in conjunction with a LOOP NEXT (FILE) structure. Everywhere you would use a CYCLE within such a loop, you can use a PROP:WHERE instead; the difference is that you leave the evaluation on the server thus reducing network traffic. PROP:WHERE can also be set in the dictionary as a driver string, but since it does not affect the VIEW engine it doesn't seems to be very useful.

```
Clear(File)
Ord:CustomerId = Cus:CustomerId
Set(Ord:K_CustomerId,Ord:K_CustomerId)
!get this year's orders
Orders{Prop:Where}='Year(OrderDate)=Year(GetDate())'
Loop Until Access:Orders.Next()
  !obsolete code block
  If Year(Ord:OrderDate_Date)<>Year(Today())
    Cycle
  End
  !End obsolete
  Do ActionPerRecord
End
```

Summary

The SQL Advanced tab is one of the new features in Clarion that seemed unnecessarily complicated when it was first introduced, in part because the template implementation was not complete, and because it looked like it just provide another way to solve old problems. However, the properties involved provide a common interface to the SQL engine. Since they can all be combined, the developer can easily expand existing code without a complete rewrite. When used in handcode, the code is easy to follow and as a consequence, easy to alter when needed. My conclusion after exploring the SQL Advanced tab is that it represents yet another Clarion feature which is commonly underestimated.

Download the source

Reader Comments

Add a comment

Clarion Magazine

Interprocess Communication: Sending Messages

by Larry Sand

Published 2007-03-14

Interprocess communication, or IPC, is the communication between two or more programs or processes. There are a variety of ways to accomplish this, including the use of a shared INI or data file, information passed on the command line, named pipes, mail slots, TCP/IP, RPC, or memory mapped files. They all have their place; some are best suited for communication between computers, and others for communication within the local machine.

This article series discusses interprocess communication on the local machine using Windows messaging and the "copy data" technique. You'll learn how to register messages with windows and use them to set up communication between two processes. Then you'll see how to use this messaging protocol to pass data between the processes. I'll begin with a simple message with a 32 bit value and progress to passing data in a user defined structure.

Please note that this type of protocol is not suitable for communication between a service and a program with a user interface in Windows Vista. Windows Vista isolates session zero, which is the session that Vista uses to run services, and you cannot send messages to a process with a user interface running in another session. For complete information on the session zero isolation, see this article on Microsoft's web site.

Windows Messages

As you may know, Windows uses messages to communicate events and data to the processes that it manages. For example, every time you move the mouse, Windows generates a WM_MOUSEMOVE message. This message is sent to the window that the mouse cursor is moving over. The message contains the handle to the window, a constant message identifier (WM_MOUSEMOVE) , the keys and mouse buttons pressed, and the x, y coordinates of the mouse pointer. Windows messages are received by a procedure attached to your window class called a "window procedure ". You can prototype the window procedure like this in Clarion:

```
WinProc Procedure(UNSIGNED hWnd, UNSIGNED uMsg,|
        UNSIGNED wParam, Long lParam),Long,Pascal
```

In this prototype hWnd is the handle to the window receiving the message, uMsg is the message, and wParam and lParam are optional parameters (in the case of the WM_MOUSEMOVE message they are used to pass the keys and pointer position). The WM_MOUSEMOVE message belongs to a class of messages reserved for use by Windows. That is, you only receive these messages with your window procedure, you usually don't send them to a window.

Sending messages

In Clarion you can use the POST() and NOTIFY() functions to send a message (event) to a window, and you read the message with the ACCEPT loop. The ACCEPT loop is analogous to the Windows window procedure; that is, it's used to read and dispatch messages (events). This is a useful similarity, as I'll show later.

In Clarion you can declare a user defined event (a message) in the range 1024 (EVENT:User, or WM_User in Windows parlance) to 32767 decimal, or 0400 to 7FFF hex. After defining an event (message) in this range you can POST() it to your ACCEPT loop or use the NOTIFY/NOTIFICATION functions to send and receive the message in a thread safe manner within your process. POST only allows you to send a message, while NOTIFY/NOTIFICATION allow you to send a message and a 32 bit parameter. These user messages are designed for you to send within your application to a known window, not between processes. They're also the cause of some contention because window controls also use messages in this range. For example, a textbox with the RTF attribute, also known as a Richedit control in the Windows SDK, uses several messages in this range. One of these is WM_CANPASTE which is defined as WM_USER + 50.

Microsoft advises that you use the base WM_APP (32768 decimal or 8000 hex) for your user defined application level messages. The following table lists the message ranges and uses:

Range	**Description of message range**
From 0 through WM_USER –1	Messages reserved for use by the system.
From WM_USER (0400 hex) through 07FFF hex	Integer messages for use by private window classes.
WM_APP (8000 hex) through 0BFFF hex	Messages available for use by applications.
0C000 hex through 0FFFF hex	String messages for use by applications.
Greater than 0FFFF hex	Reserved by the system for future use.

While the Clarion documentation notes that messages sent with POST and NOTIFY/NOTIFICATION should be in the range EVENT:User to 07FFFh, there's no problem using them to send messages in the application range (8000h to 0BFFFh).

Messages in the fourth range (0C000h to 0FFFFh) are not defined by the programmer. Instead, you pass a string to the RegisterWindowMessage Windows API function which returns an integer in that 0C000h to 0FFFFh range. RegisterWindowMessage is prototyped in Clarion as follows (note that Windows API functions in this article series are prototyped with a CMAG_ prefix to prevent name collisions in your application):

```
CMAG_RegisterWindowMessage(*CSTRING lpString),UNSIGNED, |
  Raw,Pascal,Name('RegisterWindowMessageA'),DLL(1)
```

All applications that call RegisterWindowMessage with an identical string receive the same message identifier. There's nothing to clean up when you're done because there is no way to unregister the window message. This is how you'll communicate between your applications. Both applications call RegisterWindowMessage and they'll use the returned message to communicate with each other.

Sending Messages

Windows provides synchronous and asynchronous messaging functions for sending the message between applications. This article series will discuss three of the message functions: SendMessage, and SendMessageTimeout are the synchronous functions that may return a value; PostMessage is the asynchronous function.

The first messaging function I'll cover is PostMessage and it is prototyped like this:

```
CMAG_PostMessage(UNSIGNED hWnd, |
  UNSIGNED nMsg, |
  UNSIGNED wParam, |
  Long lParam |
  ),BOOL,Pascal,Proc,Name('PostMessageA') ,DLL(1)
```

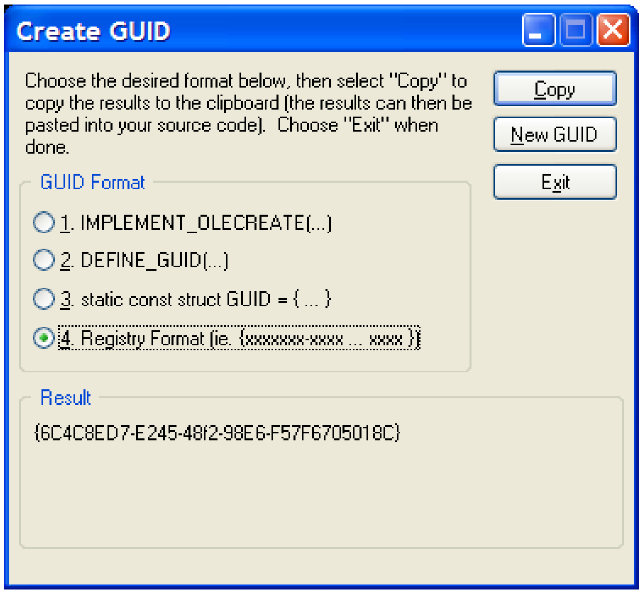
If you compare this prototype to the prototype of the window procedure above you'll notice that they have identical parameters. The reason for this is that the window procedure receives an exact copy of the message sent by PostMessage. PostMessage does this by placing your message on the end of the message queue for the destination window and returning immediately. It does not wait for the window procedure to process the message, and the window procedure processes the messages out of its message queue in the order they were received.

You now have a method to create a message that's safe to pass across process boundaries, and a way to send that message. The one thing that you still need is the handle of the window (hWnd) you want to send the message to. A window handle uniquely identifies every window in every process on the local machine. The problem is that with interprocess communication you don't necessarily have a way to know the window handle in your other process. To help solve this, Windows has a broadcast handle that instructs PostMessage to send your message to all top level windows. The constant is HWND_BROADCAST and has a value of 0FFFFFFFFh.

This is where the registered window message comes into use. If you were to broadcast a message in the WM_USER or WM_APP range there's a distinct possibility that another application would respond to that message in an unpredictable way, such as hanging or crashing. With a registered window message there's no chance that the message can be misinterpreted by another process provided that you used a completely unique (to your application) string to register the message.

You can use a program called guidgen.exe to create a **G**lobally **U**nique **ID**entifier (GUID) which will function nicely as a unique string. Guidgen.exe is provided with Visual Studio and other development tools, so you may have it on your computer. If not, you can download it from Microsoft. If that page has moved just search for "guidgen download" and you should find a new link to the file.

Once you have Guidgen.exe on your computer, run the program and you'll see something like Figure 1.



Choose the desired format below, then select "Copy" to copy the results to the clipboard (the results can then be pasted into your source code). Choose "Exit" when done.

GUID Format

1. IMPLEMENT_OLECREATE(...)
2. DEFINE_GUID(...)
3. static const struct GUID = { ... }
[✓ 4. Registry Format (ie. {xxxxxxxx-xxxx ... xxxx })]

Result

{6C4C8ED7-E245-48f2-98E6-F57F6705018C}

Figure 1. Creating a GUID with Guidgen.exe

Select the Registry Format option and click the Copy button to copy the string to the clipboard. Every time you run the program or click the New GUID button, you'll get a different GUID.

Now you have a way to create a unique message that's safe to send across process boundaries with the PostMessage function.

Next, you'll need a framework for establishing communication. When your application wants to establish a link with a partner application it will send a registered message using the HWND_BROADCAST constant for the handle to the window. If the partner application is already running it will receive the broadcast message and then respond by posting the same registered message back to the initiating application. In the process, each application will learn the handle to the other's window. If no other application responds to the broadcast message, you just wait and listen for another application to broadcast the message.

When the partner application is already running, the link conversation is something like this:

Request:
Hello is anyone there? …. (sent to all top level windows)

Response:
Yes, I'm here, and here's my hWnd.

The link message is defined as:

hWnd The window handle of the window that receives the message (may be HWND_BROADCAST)

uMsg Self.LinkMessage , the return value from the RegisterWindowMessage for the GUID received in the Init method.

wParam The window handle of the window sending the message

lParam The action to take on receipt of message, this may be one of the following:
WMU_IPC_LINKME Setup the link between the two processes

WMU_IPC_ACKLINK Acknowledge the receipt of the link message

WMU_IPC_REMOVEME Disassociate the link between the processes

Returns Zero

That's how you send messages. Next time I'll look at receiving messages sent by a Clarion program.

Larry Sand is an independent software developer who began programming with Clarion in 1987. In addition to normal database development, he specializes in connecting Clarion to external devices like SCUBA diving computers, kilns, and satellite transceivers used in medical helicopters. In other lives, he sailed Lake Superior as the owner/operator of shipwreck SCUBA diving tours and later as a Master for the Vista Fleet. When Larry is not programming you'll find him messing about in boats, or with boats.

Reader Comments

Add a comment

Clarion Magazine

C7 Alpha Bits Part 1

by Dave Harms

Published 2007-03-10

This article begins a new series in Clarion Magazine. The "Alpha Bits" articles will be shorter pieces covering one or more interesting aspects of the new IDE. The breadth of the new IDE, and the fact that it's still in alpha, presents some special reporting problems. First, some things you see here may change before final release. Second, both the size of the IDE and the continuing improvements make any exhaustive treatment almost impossible. Once C7 goes gold Clarion Magazine will compile and update this information into a comprehensive reference.

One of the pillars of the new C7 IDE is the source editor. Essentially this is the SharpDevelop code editor with some Clarion-specific functionality added, such as code folding, code completion/intellisense and the structure formatters (I'll have more about the formatters in another article).

The editor comes with a some standard functionality that's been sorely lacking in the 16 bit Clarion IDE, such as multi-level undo, drag/drop, code folding, navigation options, code formatting, and configuration options, most of which I'll touch on in this article. But as nice as these features are, there's much, much more to the editor; in future articles in this series I'll look at the structure designers, the context menu, code macros, and extending the editor with add-ins, to name a few topics.

Undo

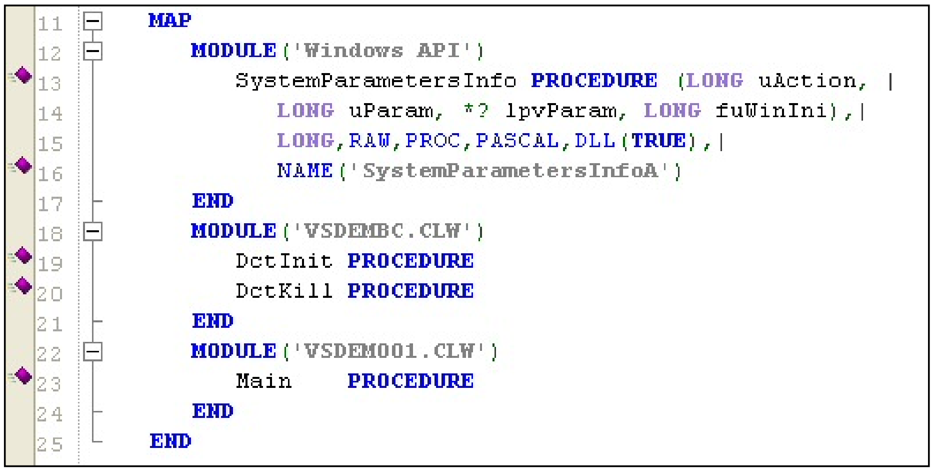
It is so nice to have an editor with real multi-level undo capability. The old editor has an undo capability all right – it will completely undo your code, given half a chance, especially if you move the cursor to a new location before pressing Ctrl-Z. I haven't yet encountered any limitations or bugs with undo or redo in the new editor – it's worked flawlessly.

Drag/drop

You can drag and drop text in the new editor, something that isn't supported at all in the old editor. That may not be a huge factor for most developers but it's definitely nice to have standard functionality.

Code folding

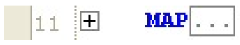
One of the very first features of C7 ever announced is code folding in the editor. Figure 1 shows a MAP structure with markers indicating which areas can be folded.



```
MAP
  MODULE('Windows API')
    SystemParametersInfo PROCEDURE (LONG uAction, |
      LONG uParam, *? lpvParam, LONG fuWinIni),|
      LONG,RAW,PROC,PASCAL,DLL(TRUE),|
      NAME('SystemParametersInfoA')
  END
  MODULE('VSDEMBC.CLW')
    DctInit PROCEDURE
    DctKill PROCEDURE
  END
  MODULE('VSDEM001.CLW')
    Main    PROCEDURE
  END
END
```

Figure 1. A MAP structure in C7

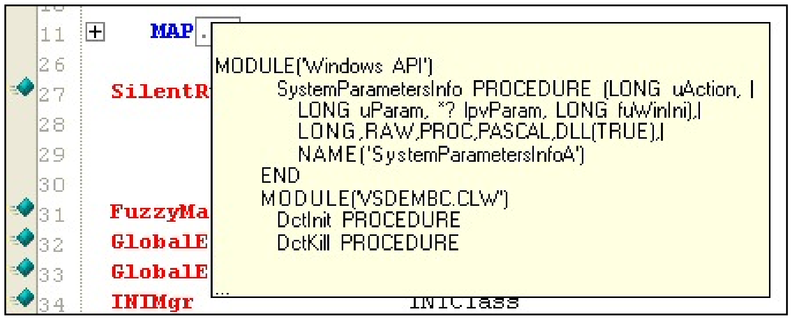
Click on the top level box and the map collapses, as shown in Figure 2.



```
MAP...
```

Figure 2. The folded MAP

Hover your mouse over the ellipsis box and you'll see a popup showing the folded code (Figure 3). If there's a lot of hidden code then the contents will be truncated for display.



```
MAP...
SilentR
FuzzyMa
GlobalE
GlobalE
INIMgr                  INIClass

MODULE('Windows API')
    SystemParametersInfo PROCEDURE (LONG uAction, |
      LONG uParam, *? lpvParam, LONG fuWinIni),|
      LONG,RAW,PROC,PASCAL,DLL(TRUE),|
      NAME('SystemParametersInfoA')
  END
  MODULE('VSDEMBC.CLW')
    DctInit PROCEDURE
    DctKill PROCEDURE
...
```

Figure 3. The popup showing folded code

Code completion

Last year word came down that code completion would not be in C7, but it is there in the alpha release after all. Consider the following class declaration:

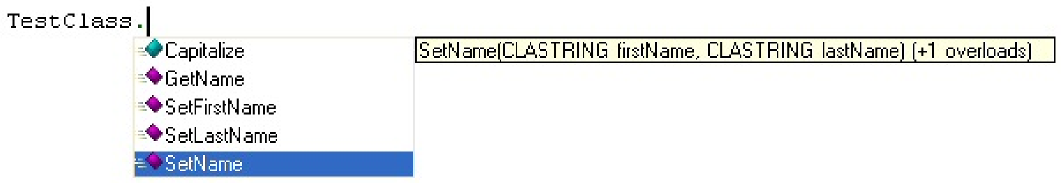
```
TestClass     class
Capitalize       byte(0)
FirstName        string(30),private
MiddleName       string(30),private
LastName         string(30),private
```

```
SetFirstName      procedure(string firstName)
SetLastName       procedure(string lastName)
SetName          procedure(string firstName,string lastName)
SetName          procedure(string firstName,string middleName,string lastName)
GetName          procedure,string
                end
```

Let's say you want to use this class in your code. Type

TestClass.

and after you type the period, code completion looks up the available properties and methods for TestClass (see Figure 4).



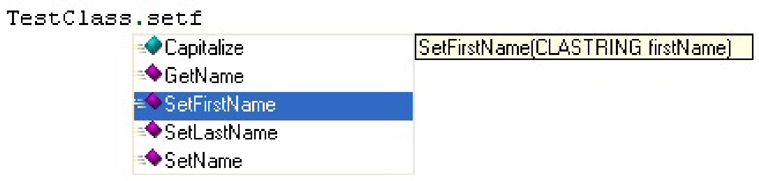
```
TestClass.
Capitalize
GetName
SetFirstName
SetLastName
SetName
```

SetName(CLASTRING firstName, CLASTRING lastName) (+1 overloads)

Figure 4. Code completion

In Figure 4 I've clicked onthe SetName method, which causes the prototype to appear in a box to the right of the method list. This method is overloaded; that is, there are two methods by the same name, and this is noted in the prototype by the text (+1 overloads).

There are several ways I can use code completion to select a method or property. In Figure 4 I selected SetName with the mouse, but I can also simply type the method (or property) name. The list functions like an incremental locator. In Figure 5 I've typed TestClass.setf and SetFirstName is selected.



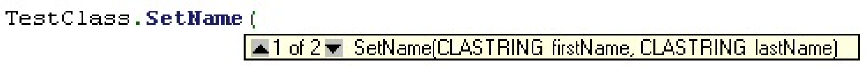
```
TestClass.setf
Capitalize
GetName
SetFirstName
SetLastName
SetName
```

SetFirstName(CLASTRING firstName)

Figure 5. Typing until the desired item is selected.

Once I have the name I want I simply press Enter. Or I can finish typing the method name and the popup will disappear when I'm done.

If you're typing a method that takes parameters, as soon as you press the opening parenthesis the method prototype appears, as in Figure 6.

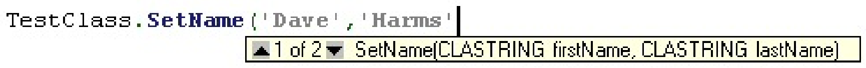


```
TestClass.SetName(
```

1 of 2 SetName(CLASTRING firstName, CLASTRING lastName)

Figure 6. The method prototype

In the case of Figure 6 there are two possible prototypes, and you can use the mouse or the arrow keys to scroll through the available options. This popup will stay active while you fill in the parameters, as in Figure 7. As soon as you type the closing parenthesis, the popup disappears.

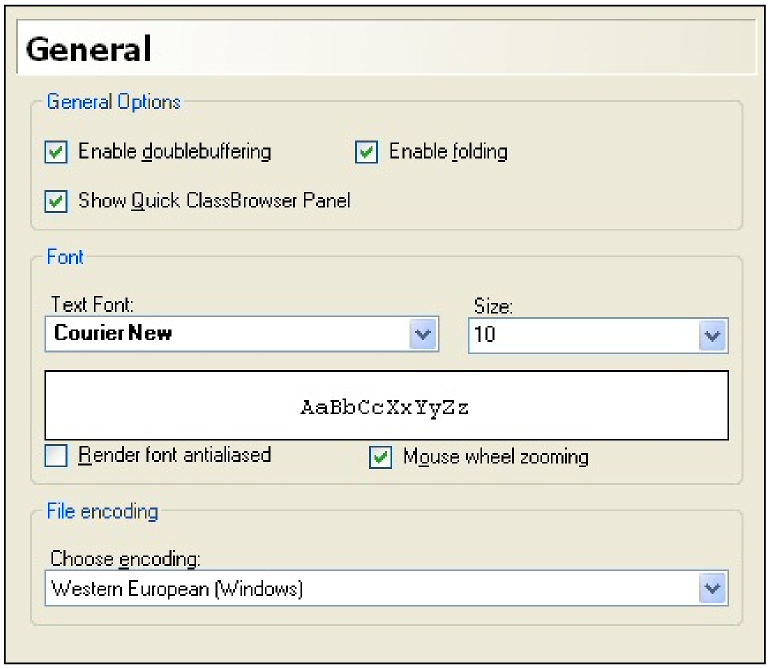


```
TestClass.SetName('Dave','Harms'
▲1 of 2▼ SetName(CLASTRING firstName, CLASTRING lastName)
```

Figure 7. Typing the parameters

Configuring the editor

Without going into a lot of detail, here are the various editor configuration windows, some of which have further options.



General

General Options

- [✓] Enable doublebuffering
- [✓] Enable folding
- [✓] Show Quick ClassBrowser Panel

Font

Text Font: Courier New
Size: 10

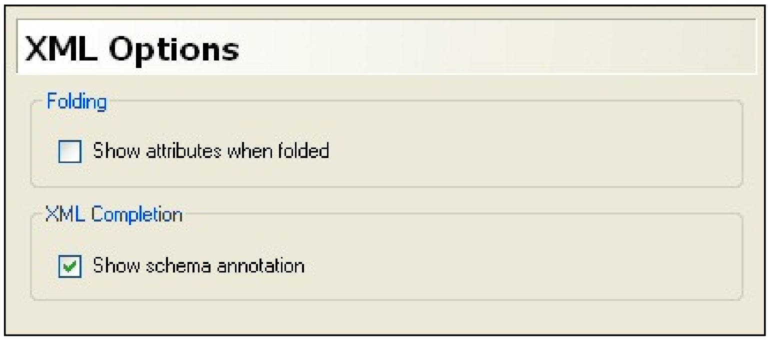
AaBbCcXxYyZz

- [ ] Render font antialiased
- [✓] Mouse wheel zooming

File encoding

Choose encoding: Western European (Windows)

Figure 8. General editor options



XML Options

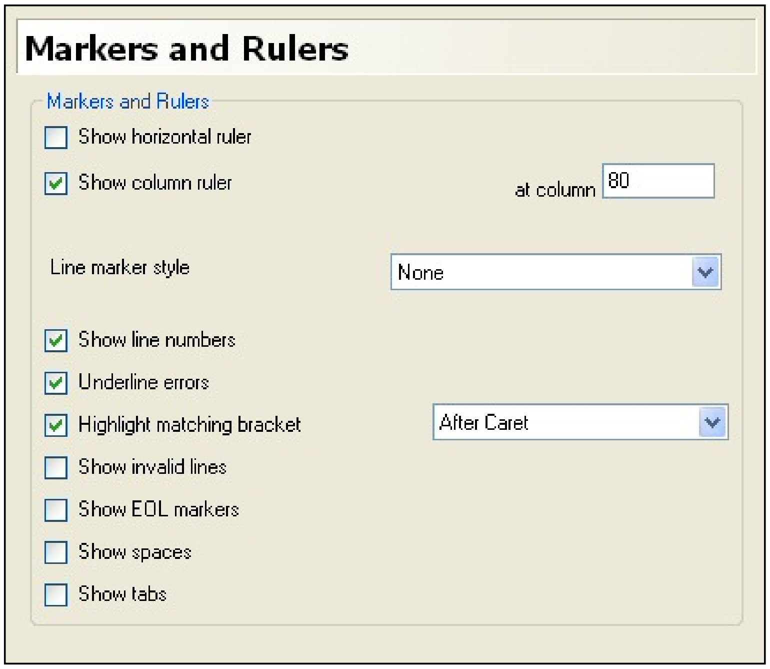
Folding

- [ ] Show attributes when folded

XML Completion

- [✓] Show schema annotation

Figure 9. XML Options



Markers and Rulers

Markers and Rulers

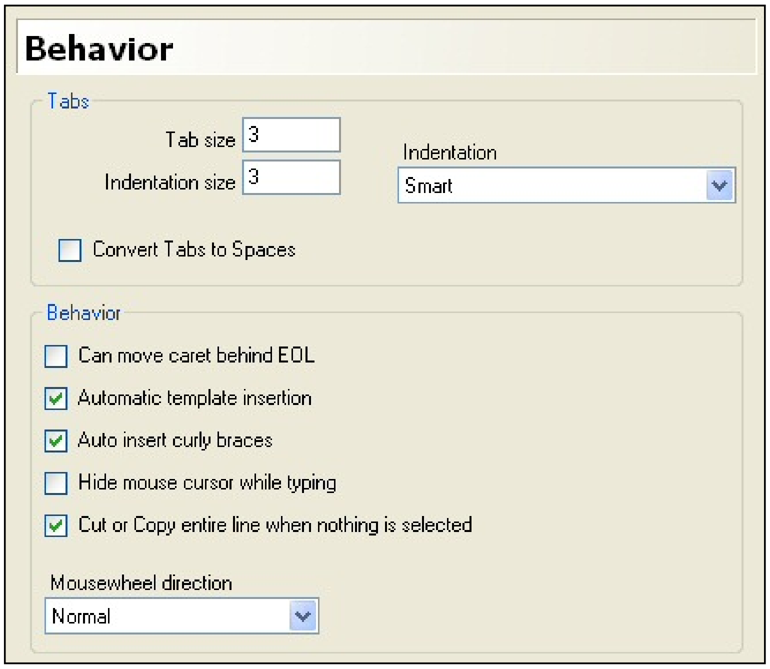
- [ ] Show horizontal ruler
- [x] Show column ruler — at column: 80

Line marker style: None

- [x] Show line numbers
- [x] Underline errors
- [x] Highlight matching bracket — After Caret
- [ ] Show invalid lines
- [ ] Show EOL markers
- [ ] Show spaces
- [ ] Show tabs

Figure 10. Markers and Rulers

One option I particularly like here is Show column ruler. This places a faint gray vertical line at the specified column, which is a handy indicator for those of us who tend to write overly long lines of code. This doesn't affect your code formatting at all; it's simply a reminder of where you might want to put a line break.



Behavior

Tabs

Tab size: 3
Indentation size: 3
Indentation: Smart

- [ ] Convert Tabs to Spaces

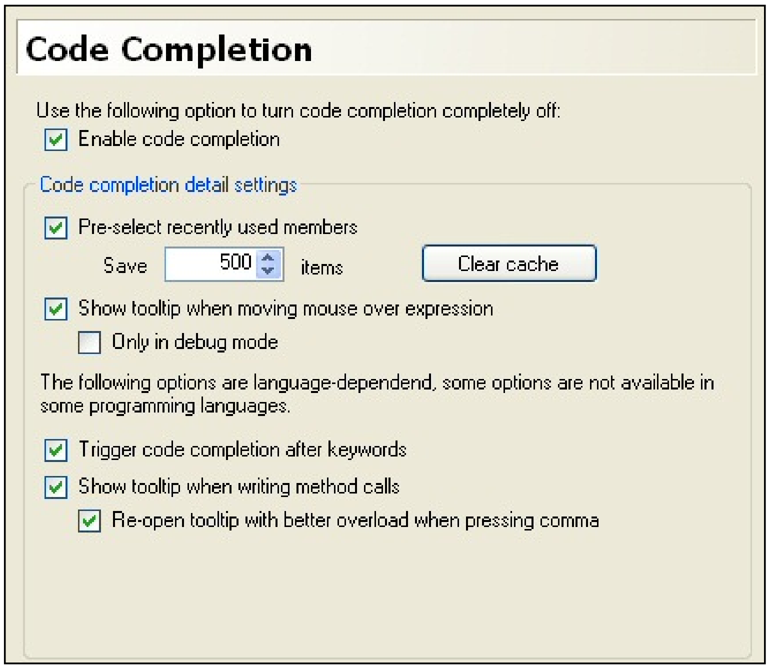
Behavior

- [ ] Can move caret behind EOL
- [x] Automatic template insertion
- [x] Auto insert curly braces
- [ ] Hide mouse cursor while typing
- [x] Cut or Copy entire line when nothing is selected

Mousewheel direction: Normal

Figure 11. Editor behavior options

The option to move the caret, or cursor, behind EOL makes the editor behave like the Clarion editor, where you can put the cursor anywhere on the line, not just in used space.



Code Completion

Use the following option to turn code completion completely off:

[✓] Enable code completion

Code completion detail settings

[✓] Pre-select recently used members

Save 500 items

[✓] Show tooltip when moving mouse over expression

[ ] Only in debug mode

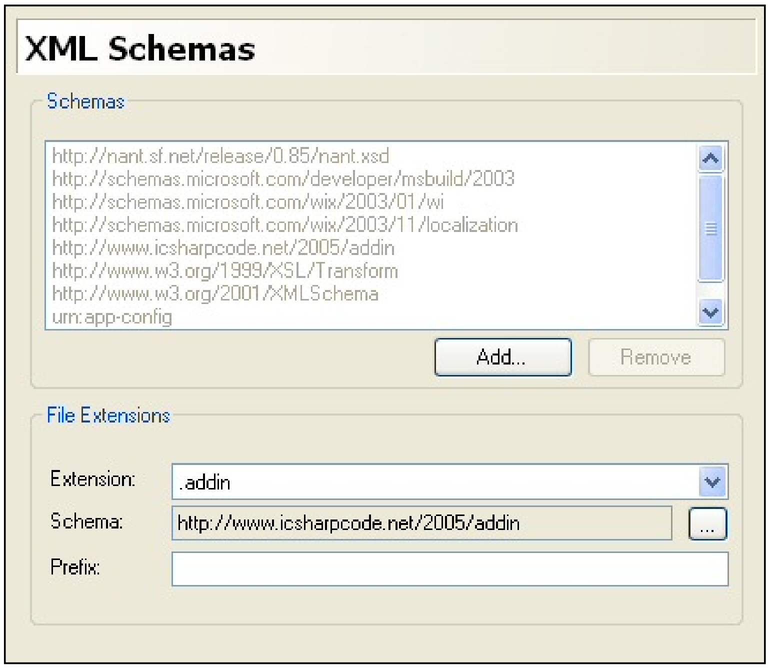
The following options are language-dependend, some options are not available in some programming languages.

[✓] Trigger code completion after keywords

[✓] Show tooltip when writing method calls

[✓] Re-open tooltip with better overload when pressing comma

Figure 12. Code completion options



XML Schemas

Schemas

```
http://nant.sf.net/release/0.85/nant.xsd
http://schemas.microsoft.com/developer/msbuild/2003
http://schemas.microsoft.com/wix/2003/01/wi
http://schemas.microsoft.com/wix/2003/11/localization
http://www.icsharpcode.net/2005/addin
http://www.w3.org/1999/XSL/Transform
http://www.w3.org/2001/XMLSchema
urn:app-config
```

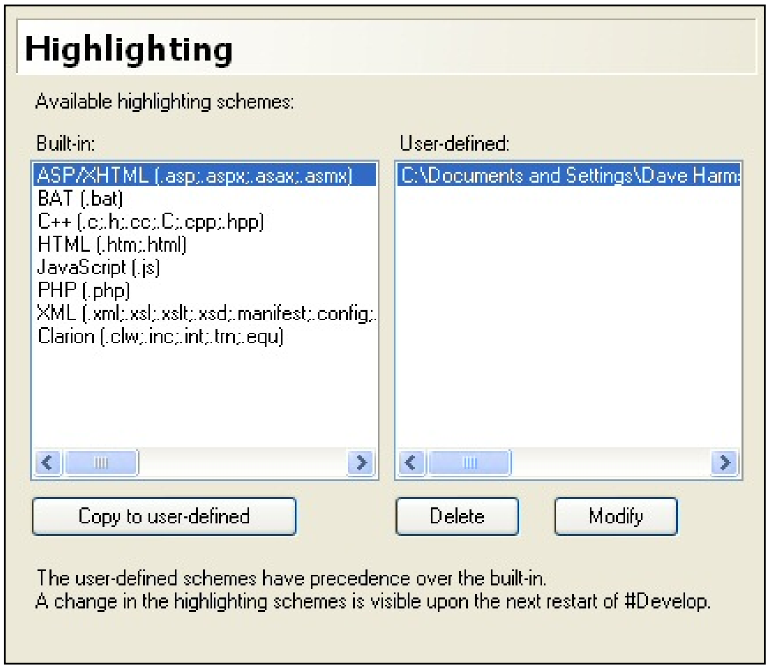
File Extensions

Extension: .addin

Schema: http://www.icsharpcode.net/2005/addin

Prefix:

Figure 13. XML schemas associated with file extensions



Highlighting

Available highlighting schemes:

Built-in:

ASP/XHTML (.asp;.aspx;.asax;.asmx)
BAT (.bat)
C++ (.c;.h;.cc;.C;.cpp;.hpp)
HTML (.htm;.html)
JavaScript (.js)
PHP (.php)
XML (.xml;.xsl;.xslt;.xsd;.manifest;.config;.
Clarion (.clw;.inc;.int;.trn;.equ)

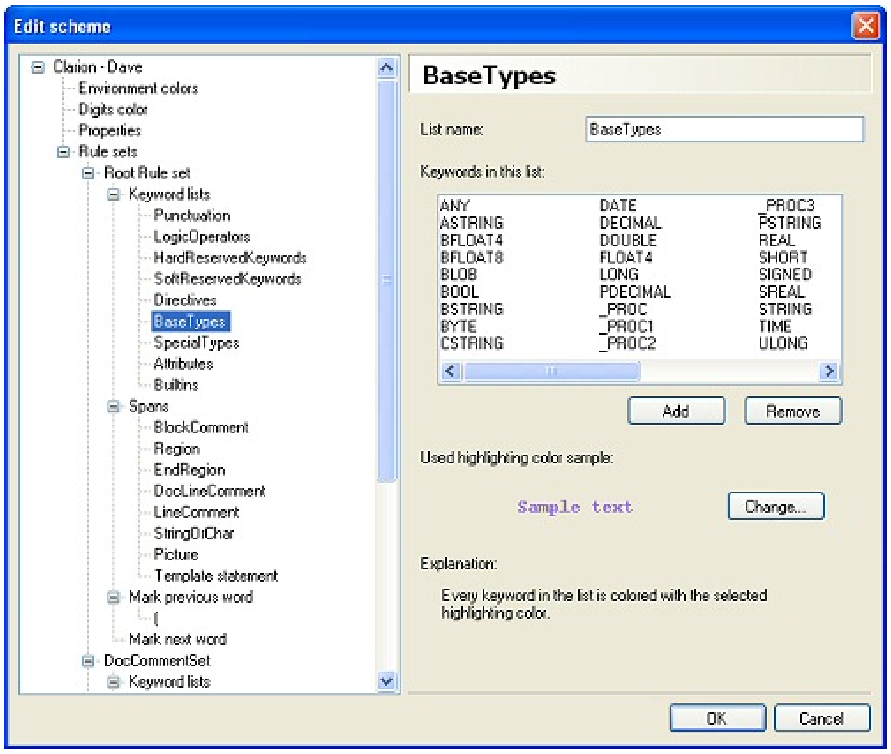
User-defined:

C:\Documents and Settings\Dave Harm

The user-defined schemes have precedence over the built-in.
A change in the highlighting schemes is visible upon the next restart of #Develop.

Figure 14. Highlighting options

As Figure 14 indicates, C7 is capable of displaying a variety of programming languages (you should also be able to use the gold release to compile code in a number of languages, but I haven't begun to look at that yet). You can modify the highlighting to suit your own tastes. Figure 15 shows a partial view of the available options.



Edit scheme

Clarion - Dave
Environment colors
Digits color
Properties
Rule sets
Root Rule set
Keyword lists
Punctuation
LogicOperators
HardReservedKeywords
SoftReservedKeywords
Directives
BaseTypes
SpecialTypes
Attributes
Builtins
Spans
BlockComment
Region
EndRegion
DocLineComment
LineComment
StringOrChar
Picture
Template statement
Mark previous word
(
Mark next word
DocCommentSet
Keyword lists

BaseTypes

List name: BaseTypes

Keywords in this list:

```
ANY        DATE       _PROC3
ASTRING    DECIMAL    PSTRING
BFLOAT4    DOUBLE     REAL
BFLOAT8    FLOAT4     SHORT
BLOB       LONG       SIGNED
BOOL       PDECIMAL   SREAL
BSTRING    _PROC      STRING
BYTE       _PROC1     TIME
CSTRING    _PROC2     ULONG
```

Used highlighting color sample:

Sample text

Explanation:

Every keyword in the list is colored with the selected highlighting color.

Figure 15. Code highlighting options (view full size image)

As Figure 15 suggests the rule set for syntax highlighting can get pretty complex. I've played a bit and have added some rudimentary template language highlighting, but I'm sure much more can be done. Further details on the syntax highlighting rules can be had from the e-book Dissecting a C# Application: Inside SharpDevelop, available as a free download from APress.

Once again, I feel as though I've barely scratched the surface of the new IDE. In upcoming Alpha Bits installments I'll look at the new screen and report structure designers (some very nice stuff there), the context menu, and of course extending the editor with addins.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Reader Comments

Posted on Sunday, March 11, 2007 by John Morter

Dave, One feature of the old Editor I really appreciated was its configurability (via C60EDT.INI) 'cos it enabled me to assign most of the main edit commands to same keystrokes as my favourite TextEditor of many, many years. Do you know whether this feature will be available with the new editor too? Rgds, JohnM

Posted on Monday, March 12, 2007 by Dave Harms

John,

Yes, you can assign hot keys by editing the binaddinsclarion.addin file. But these are single hotkey combinations only - the IDE doesn't support multiple key combinations (a hot key followed by another command keystroke). That isn't to say that it can't be done - seems to me you could create an add-in that would respond to a hotkey and wait for an additional command string.

It also seems to me that it would be possible to create an add-in to more easily customize hotkeys...

Dave

Posted on Monday, March 12, 2007 by Loren Gregg LaBaw

Off the subject, I see BString in the base data type. Do you know if it will be selectable data type in C7?

Thanks

Loren Gregg LaBaw

Posted on Tuesday, March 13, 2007 by Dave Harms

Loren,

Do you mean in AppGen? If so I'll be able to answer more intelligently when it's out<g> (in phase four).

Dave

Posted on Wednesday, March 14, 2007 by Rick Martin

Hi Dave,

Does Phase I have the ability to do command line compilation? Either for a project or solution?

Thanks,
Rick

Posted on Wednesday, March 14, 2007 by Dave Harms

Rick,

Yes, you can do command line compiles with Alpha 1. The project system is built on MSBuild and retains that product's command line capability.

Dave

Posted on Wednesday, March 14, 2007 by Rick Martin

Thanks, Dave.

If I had thought about it I would have realized that <g>.

I had originally started to ask about using a command line to build an APP from TXA (or XML?). But it dawned on me you don't have AppGen yet so it would be pretty hard to build an APP file in any way.

Rick

Add a comment

Clarion Magazine

The ClarionMag Blog

Get automatic notification of new items! RSS feeds are available for:

XML All blog entries
XML All new items, including blogs

Blog Categories

- »All Blog Entries
- »Clarion 7 Clarion.NET
- »Future Articles
- »News flashes
- »Nifty Stuff

Greenbar tutorial on YouTube

Direct link

Posted Tuesday, March 20, 2007 by Dave Harms

Eberto Barrios Romo posted a link in the SV newsgroups to this Clarion greenbar tutorial on YouTube.

Search engine maintenance

Direct link

Posted Monday, March 12, 2007 by Dave Harms

I'm doing some maintenance on the ClarionMag search engine so results may be erratic for a while.

WinHlp32.exe for Windows Vista

Direct link

Posted Monday, March 12, 2007 by Dave Harms

If you or your customers are running Vista you've probably noticed that WinHlp32.EXE, the program that displays .HLP files, is not included with the operating system. A download of WinHlp32.exe is now available from Microsoft:

> Windows Help (WinHlp32.exe) is a Help program that has been included with Microsoft Windows versions starting with the Microsoft Windows 3.1 operating system. However, the Windows Help program has not had a major update for many releases and no longer meets Microsoft's standards. Therefore, starting with the release of Windows Vista, the Windows Help program will not ship as a feature of Windows. If you want to view 32-bit .hlp files, you must download and install the program (WinHlp32.exe) from the Microsoft Download Center.

Unfortunately, this KB article indicates that you amy not distribute WinHlp32.exe to your customers; they'll have to download it themselves:

> Also, third-party programs that include .hlp files are prohibited from redistributing the Windows Help program together with their products. Users who want to view 32-bit .hlp files must download the program from the Microsoft Download Center, and then install it on their computers. The download for Windows Help is still in development and will be available in early 2007.

Thanks to Dave Troxell and Jim Kane for the info.

First C7 IDE add-in created

Direct link

Posted Monday, March 05, 2007 by Dave Harms

One of the more intriguing aspects of the new Clarion IDE (which is a superset of the SharpDevelop IDE) is its extensibility. This raises the possibility of third party vendors (and individual developers) not just adding features to the application generator via templates, but actually modifying the IDE's internal functionality. And now it's been done. Last week there was some discussion in the alpha newsgroup about the Clarion editor's Ctrl-\ feature, which toggles the case of text. While the SharpDevelop IDE already has this capability built in, it only operates on selected text or, if no text is selected, on the entire file! What to do? If you're Carlos Gutierrez, you get your hands on SharpDevelop and you create an add-in to supply the missing functionality.

I'll have more on how Carlos created the add-in (and how you can do likewise) in an upcoming article. Meanwhile, think of all the things that could be done with the text editor (like correcting keyword case, indenting code, etc). And then remind yourself that this isn't *just* about the editor.

The possibilities are mind-boggling.

Changes to comments

Direct link

Posted Thursday, March 01, 2007 by Dave Harms

I don't usually enable comments on public-access articles because the comment system requires a login, i.e. it does not allow anonymous comments. I finally got around to making a couple of changes to the code, and as a result the Add a Comment link only appears if you are logged in. I've also changed the display to show all comment text on each article. This means that comments will also appear in the monthly PDF (at least as of the date of PDF creation).