Clarion Magazine

Clarion News

- » C7 AppGen Released To CSP Participants
- » Lodestar Software Subscriptions
- » Template News From Huenuleufu
- » EZChangeLog 1.6.0
- » 30% Off WingNut Clarion Utilities
- » DMC 1.5.0.7
- » Noyantis CalendarPro 1.13
- » New Web 2.0 Image Collection Released
- » Dictionary Assistant 2.12.00
- » Data Equity 40% Discount
- » LANSRAD End of Year Specials Available Through SWREG
- » Ingasoftplus Limited Time 25% Relative Discount
- » Noyantis DockingPane ActiveX Control
- » GTL Reaches End Of Line
- » SetupBuilder 7.0 Beta-1 Build 2441
- » Clarion Folk Podcast #4
- » FullRecord 1.90
- » Noyantis CalendarPro 1.12
- » LANSRAD Year End Specials
- » LANSRAD Announces ExChangeLog
- » SetupBuilder 7.0 Beta-1 Build 2434
- » Easing Into C7: Blog Entries
- » Web Site Grid System
- » QuickBooks XML Connect For Clarion
- » Russ Eggen Guest Blog
- » Sage VAT Tip
- » DMC 1.5.0.6
- » ClarionFolks C7 Series Part 4

[More news]

- » Clarion.NET FAQ
- » Clarion# Language Comparison
- » Event Handling In Clarion#

[More Clarion & .NET]

[More Clarion 101]

Latest Free Content

- » C7 AppGen Released To CSP Participants
- » C7 AppGen Release V
- » Source Code Library 2008.11.30 Available

[More free articles]

Clarion Sites

Latest Subscriber Content

C7 AppGen Released To CSP Participants

Just in time for Christmas, SoftVelocity has released Clarion 7 with the AppGen to all CSP participants. A Clarion.NET update is also expected soon.

Posted Wednesday, December 24, 2008

Using HTMLHelp with Clarion

Clarion's default approach to Help has always been via WinHelp, using *.HLP files to store the material. Although WinHelp still works, it's now considered an "unacceptable" method. A better approach is to use compiled HTML help, as Mike Hanson explains.

Posted Tuesday, December 23, 2008

Formatting Names Using Proper Case

ALMOST EVERYONE HATES ALL CAPS! Yet many databases are strewn with upper case data that would be much better presented (and even stored) using proper case. It's easy to fix up your data with Mike Hanson's MHProperClass.

Posted Monday, December 22, 2008

C7 AppGen Release V

There's been another release of C7 AppGen to third party developers.

Posted Friday, December 19, 2008

Creating a Threaded Web Service Client

Although web services are easy to create, the possibility of delays and other errors introduces some potential problems. David Harms shows how to create a background thread to process a web service call and how to manage the resulting cross-thread user interface update.

Posted Tuesday, December 16, 2008

Creating a Real-World Web Service

David Harms sets about creating a web service to make it easier for Stu Andrews to facilitate the Clarion Folk "pondcast".

Posted Tuesday, December 16, 2008

Writing and Using Web Services in Clarion#

Web services are a useful way to expose a programming API across the Internet. And it doesn't hurt at all that they're extremely easy to create and use in Clarion#.

Posted Monday, December 15, 2008

C7 AppGen Beta IV

SoftVelocity has released the fourth AppGen beta to the third party group for compatibility testing. Here's our first look.

Posted Friday, December 12, 2008

Event Handling In Clarion#

There's no ACCEPT statement in Clarion#. So how do Clarion# applications handle events? With event handlers, naturally.

Posted Friday, December 12, 2008

Source Code Library 2008.11.30 Available

The Clarion Magazine Source Code Library has been updated to include the latest source. Source code subscribers can download the November 2008 update from the My ClarionMag page. If you're on Vista please run Lindersoft's Clarion detection patch first.

Posted Thursday, December 04, 2008

[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

Printed Books & E-Books

Clarion Blogs

E-Books

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

Printed Books

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

- » Clarion Tips & Techniques Volume 4 - ISBN 978-0-9784034-09
- » Clarion Tips & Techniques Volume 3 - ISBN: 0-9689553-9-8
- » Clarion 6 Tips & Techniques Volume 1 - ISBN: 0-9689553-8-X
- » Clarion 5.x Tips and Techniques, Volume 1 - ISBN: 0-9689553-5-5
- » Clarion 5.x Tips and Techniques, Volume 2 - ISBN: 0-9689553-6-3
- » Clarion Databases & SQL - ISBN: 0-9689553-3-9

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

From The Publisher

About Clarion Magazine

Clarion Magazine is your premier source for news about, and in-depth articles on Clarion software development. We publish articles by many of the leading developers in the Clarion community, covering subjects from everyday programming tasks to specialized techniques you won't learn anywhere else. Whether you're just getting started with Clarion, or are a seasoned veteran, Clarion Magazine has the information *you* need.

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

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

PDF-XChange Version 4

PDF-XChange Version 4 is now available for download. The template has not been tested against Clarion7. A C6/C7 installer will be released as soon as a definite schema is adopted and all tests are complete.

Posted Wednesday, January 07, 2009

CHT C7 Video Version 2

Gus Creces has provided a re-work of a 20 minute video in which he takes five CHT demo applications, convert the apps to C7 app format, and then compiles in C6 compiler mode and again in C7 compiler mode. The video is approx 80MB (1024x768).

Posted Wednesday, January 07, 2009

Noyantis ShortcutBar 1.14

Version 1.14 of the Noyantis ShortcutBar wrapper template has been released. Modifications include a bug fix to event triggering. The new version can be downloaded from the Members area using the original download and registration details contained in your sales emails.

Posted Wednesday, January 07, 2009

Clarion Third Party Profile Exchange January 2 2009 Release

An update to the Clarion Third Party Profile Exchange is available. This is a maintenance release, both online and data version. C7 Status was updated during this maintenance release - 122 products are marked as C7, 21 vendors included.

Posted Wednesday, January 07, 2009

Seven Icon Collections For $199

1st Logo Design is offering all of its current icon collections for $199 (a $723 value). This is a total of 15.9 Gb of images including: The Mallorca Collection; The Real Estate Collection; The Sterling Collection; The Rio Collection; The LNS Collection; The Webmaster Collection; The Web 2.0 Collection; The Background Collection (bonus set).

Posted Wednesday, January 07, 2009

DMC Updated Installers

This version version of DMC has no new features but does feature rewritten installers for better compliance with Windows Limited Account (non admin) users.

Posted Wednesday, January 07, 2009

Icetips 33% Discount Ending

For the past 17 months Icetips has offered a 33% discount off the regular Gold and Silver subscription prices for customers who had previously purchased Icetips Clarion tools. Until January 23, 2009 existing customers can purchase the Gold subscription for $199.00, $100 off the regular price of $299, and the Silver subscription for $69, $30 off the regular price of $99. This discount ends January 23, 2009. After that, the Gold subscription will cost $299 and the Silver will cost $99.00 When you purchase a Gold subscription you will receive all PowerOffice and Icetips products. You will also receive a single user license for Build Automator ($149 value - see http://www.buildautomator.com) with a one year Maintenance Plan for free.

Posted Wednesday, January 07, 2009

Icetips C7 Installs

Icetips has created fully compatible Clarion 7 installs for XP-Taskpanel, Outlookbar and PowerToolbar. XP-Theme will not be adapted to Clarion 7 as it is not really needed there. A Clarion 7 install is ready for the Icetips Previewer and the install for Magic Locks is being finished. Clarion 7 compatible installs of all products are expected before the end of this week if no major problems come up. All the installs, except XP-Themes, have fully compatible Clarion 7 applications and solutions that have gone through and passed testing with C6 and C7.

Posted Wednesday, January 07, 2009

Icetips Acquires PowerOffice Tools

In early December, Icetips Creative, Inc. took over ownership of PowerXP-Theme, XP-Taskpanel, Outlookbar and PowerToolbar, which were previously owned and developed by PowerOffice AS in Norway. The templates, code and documentation for those four products have been updated. All the tools are included in the Icetips Gold subscription and XP-Theme is included in the Silver subscription.

Posted Wednesday, January 07, 2009

Noyantis ShortcutBar 1.13

Version 1.13 of the Noyantis ShortcutBar wrapper template has been released. Modifications include: Codejock Control Version selection enhanced; User definable actions added to item definitions; Legacy compatibility added. The new version can be downloaded from the Members area using the original download and registration details contained in your sales emails. A new demo example app has been uploaded to the web site.

Posted Wednesday, January 07, 2009

Oldaer

Stu Andrews has released Oldaer into public beta. It's a program that remembers information such as usernames, passwords and software keys for you. Stu would be happy to get feedback on the product.

Posted Wednesday, January 07, 2009

Free VuDeploy

As a token of thanks to everyone in the Clarion Community for all of their help

Posted Wednesday, January 07, 2009

Christmas Podcast

The Clarion Folklore Christmas podcast is available.

Posted Wednesday, January 07, 2009

Lodestar Software Subscription Plan Costs To Increase

Lodestar Software has adopted an annual subscription plan to begin 01-Jan-2009. This move was made to provide for ongoing product improvement and support. Costs will increase slightly sometime in January 2009. Enrollment will provide access to private download areas and, eventually, access to a private news server for sharing information and for product support. Several product enhancements are already in design or will be available shortly for use. Drill down support for RPM is already in use by several developers and will be available for 'RPM Only' and 'Everything' subscribers in January, 2009. There are also numerous handy API functions that were going to be part of the new UI that will be available in 2009 as soon as I get an opportunity to strip them from the current source projects and wrap them in templates for easier use. Like Drill down, these will be available for 'RPM Only' and 'Everything' subscribers. There is also a new UI in the works. Future improvements for AFE include a true server build that runs as a service as well as support for a drop folder for easy TIFF and PDF faxing, and more.

Posted Wednesday, January 07, 2009

Clarion Accessory Deployment Demo

Lindersoft has released an enhanced experimental Clarion Accessory Deployment demo. This demo supports automatic Clarion 7 Template Registration and Unregistration. It also demonstrates how to deploy to Clarion 6 and Clarion 7 from the same setup.exe. The download is login protected. Clarion Template Deployment.sb6 has been renamed to Clarion Accessory Deployment Demo.sb6 in the new #2465 Examples package. To use the new Clarion Accessory Deployment Demo.sb6 you need at least SetupBuilder 6.9 Build 2454.

Posted Wednesday, January 07, 2009

SetupBuilder 6.9 Build 2454 Pre-release

A pre-release of SetupBuilder 6.9 Build 2454 is now available. This release introduces third-party developer support for the new Clarion 7 template registration method and brings SetupBuilder 6 in-sync with the latest SetupBuilder 7 (beta). This release is not available as a web download. This release is available, free of charge, to all SetupBuilder customers who have an active SetupBuilder maintenance subscription plan. If you do not have an active subscription plan, please contact your account manager at sales@lindersoft.com.

Posted Wednesday, January 07, 2009

C7 AppGen Released To CSP Participants

Just in time for Christmas, SoftVelocity has released Clarion 7 with the AppGen to all CSP participants. A Clarion.NET update is also expected soo

Posted Wednesday, December 24, 2008

Lodestar Software Subscriptions

Subscription enrollment will open 20-Dec-09. Enrollment discounts are available through early January, 2009. There are 3 plans available depending on your needs and current licensing. Shortly after the first of the New Year private download areas will be available. Eventually access to a private news server for sharing information and for product support will be added.

Posted Friday, December 19, 2008

Template News From Huenuleufu

Work continues at Huenuleufu on FullRecord and FileTuner. As of January 1 there will be some changes in the product and maintenance plan schema. All FileTuner owners will have their maintenance code extended up to 1/7/2009, free of charge (no matter your plan's current state). Ask for your free code if you need to download the new releases. All PrintWindow owners with their current plan up to date will have it extended one year, free of charge. FullRecord 1.x sales will end as of January

1, but the product will still be supported and fixes will be backported from 2.x as possible. Maintenance plans for NeatMessage and WindowID will no longer be necessary from next year on. You will need to have a current plan to download the newest versions if any; your current plan (if you have it) will be extended free of charge. All Huenuleufu templates (current versions) are fine with Clarion 7 AppGen.
Posted Friday, December 19, 2008

EZChangeLog 1.6.0

Version 1.6.0 of EZChangeLog is available for immediate purchase and download. This is a free update to all registered users. Anyone who has EZChangeLog installed (even in demo mode) can download the update to this new version by using the Check for Updates option on the help menu. New in 1.6.0: Full multi-user capability and discounts for multi-user licenses. Version 1.6.0 also adds basic HTML export capability. The Standard Edition of EZChangeLog is absolutely free.
The Professional edition is $39.95 for a single user license.
Posted Friday, December 19, 2008

30% Off WingNut Clarion Utilities

Until December 31, 2008 all of Wingnut Solutions' Clarion Utilites are 30% off. This applies to LGP, gCalc, and gFF, and to both new licenses and upgrades. All Clarion utilities include full source code. Use the coupon code EOY08 at the checkout and your discount will be automatically applied.
Posted Friday, December 19, 2008

DMC 1.5.0.7

DMC 1.5.0.7 is now available. Changes include: A new feature while cloning or transfering data from CSV to Isam Tables (TPS-DBF-DAT) to enable you to fine tune the DATE and TIME columns contained in TEXT columns by selecting in a drop down list the FORMAT of the source column; A bug fix in EXCEL transfers and cloning tasks. DMC has been tested under the new Clarion 7 IDE and all works perfectly.
Posted Friday, December 19, 2008

Noyantis CalendarPro 1.13

Version 1.13 Beta of the Noyantis CalendarPro wrapper template has been released. Modifications include: No. of Schedules entry for Schedules Created in Embedded Code mode can now be an optional field instead of a fixed value; New method added - GetEventBusyStatus ; Optional HideRefresh parameter added to InsertEvent method - helps speed up mass event loading; BUG FIX - GetViewType method not returning correct value under certain circumstances; BUG FIX - ClearViewSchedules method could GPF when called in WorkingWeek or FullWeek view mode; New demo procedure added - Calendar with Multiple Schedules, each with different colour. The new version can be downloaded from the Members area using the original download and registration details contained in your sales emails. A new demo example application has been uploaded.
Posted Friday, December 19, 2008

New Web 2.0 Image Collection Released

This new collection from 1stLogoDesign includes six colors and nine shapes, in four sizes and four formats. Also included are blank shapes in all colors for you to add your own text. These images can be used in any of your projects where you want to emphasize a special sale, guarantee, purchase, new, etc. Quantity: 1217 Unique Images - 624.5mb. Sizes: 80x80, 96x96, 128x128, 256x256. Formats: PSD, PNG, JPF, GIF. Introductory price: $29.
Posted Friday, December 19, 2008

Dictionary Assistant 2.12.00

Changes in Dictionary Assistant 2.12.00 include Dashboard Views - each tab of the Dashboard represents statistical information regarding your dictionary. Dashboard Views include: General statistical totals; Table record buffer lengths; Column type usage; Index types and properties; Relationship constraints.

Posted Friday, December 19, 2008

Data Equity 40% Discount

Data Equity is offering Clarion developers a limited time 40% discount on all of its products. Use the coupon code HOLIDAY08.

Posted Friday, December 19, 2008

LANSRAD End of Year Specials Available Through SWREG

For those unable to use the PayPal purchase option, LANSRAD has made arrangement to allow purchase of the end of year specials through SWREG. On the specials page click the SWREG link below the PayPal button to use the secure shopping cart at SWREG.

Posted Friday, December 19, 2008

Ingasoftplus Limited Time 25% Relative Discount

Ingasoftplus announces the cutting of prices and a 25% discount on all products (full versions) for the period from 14 Dec till 31 Dec 2008.

Posted Friday, December 19, 2008

Noyantis DockingPane ActiveX Control

Noyantis Software has released its sixth template covering the Codejock controls All users who have pre-purchased the Noyantis Suite will receive their registration and download details shortly. The sixth template covers the very popular and feature rich Codejock DockingPane ActiveX control. The available demo will also automatically install the 30 day evaluation of the DockingPane ActiveX (v12.0.2). The template / control supports: Multiple color themes; Multi languages; Multiple Docking Panes containing various types of contents including Standard Clarion controls, Standard Clarion Window Procedures and External Applications (eg, Outlook, Notepad, Internet Explorer etc, etc).; Real time and Tracker Splitter Styles; Drag-n-Drop Pane Positioning; Sliding AutoHide Windows; Float and Dock Panes plus a lot more... The price of the DockingPane template is $85. Other templates currently available cover the Shortcut Bar, Command Bars (including Ribbon Bars), CalendarPro, Property Grid and Task Panel controls.

Posted Friday, December 19, 2008

GTL Reaches End Of Line

Anticipating that Clarion 7 (with its multi-project solutions) is not far down the road, GTL6.42 is expected to be the end of the GTL line of products (no, Steve doesn't anticipate anyone giving up "L" because of Clarion 7). If you wander away from your work area, GTL will now announce when it successfully completes a run with a "ta da!" sound. Runs where there was a failure emit an "uh oh" sound for each app having a problem.

Posted Monday, December 15, 2008

Clarion Magazine

Event Handling In Clarion#

by Dave Harms

Published 2008-12-12

Last month I discussed two important .NET concepts: delegates, which are a sort of sophisticated callback procedure, and events, which are delegates specifically tailored to event handling needs. Now it's time to look at how delegates and events are used to process window events in Clarion# programs.

Clarion (Win32) applications, one way or another, make events via the ACCEPT statement. But there is no ACCEPT statement in Clarion#. That may seem like a radical departure, but in fact it's not that different from how ABC applications work. So to begin with, here's a brief history of Clarion's Windows event handling.

Legacy ACCEPTs

If you have a good memory, or if you still do Legacy Clarion programming, the following simplified event handling code will look somewhat familiar:

```
    program

    map
    end

Window WINDOW('Caption'),AT(,,159,109),FONT('MS Sans Serif'|
        ,8,,FONT:regular),GRAY
      BUTTON('I''m done'),AT(53,60,45,14),USE(?Close)
    END

    code
    open(window)
    accept
      case event()
        ! Handle window events here
      end
      case field()
      ! Handle field-specific events here
      of ?close
        case event()
        of event:accepted
          break
        end
      end
```

```
end
close(window)
```

The idea is that whenever there's a Windows event associated with this window or one of its controls (or at least an event the Clarion runtime thinks you should know about) the ACCEPT statement will trigger and the code inside the ACCEPT loop will execute. Note the use of FIELD() to determine which control generated the event.

ABC ACCEPTs

In ABC the core code is similar, in that there's still an ACCEPT loop, but that loop is buried inside the WindowManager. Ask method:

```
WindowManager.Ask PROCEDURE
 CODE
 IF SELF.Dead THEN RETURN .
 CLEAR(SELF.LastInsertedPosition)
 ACCEPT
  CASE SELF.TakeEvent()
  OF Level:Fatal
   BREAK
  OF Level:Notify
   CYCLE    ! used for 'short-stopping' certain events
  END
 END
```

The WindowManager.TakeEvent method takes an approach similar to the legacy code above but partitions the various possibilities into different virtual methods such as TakeWindowEvent, TakeAccepted, TakeRejected, TakeSelected, etc. As you embed code into your procedure, derived virtual methods are generated so your code executes when the parent Take*Whatever* method is called.

So ABC abstracts event handling into method calls, and you place your code in those methods. In this respect, Clarion#'s event handling is much more like ABC than it is like Legacy. But it's still somewhat different from ABC, not least because there's no ACCEPT statement anywhere.

Clarion# event handling

I'll illustrate Clarion#'s event handling with a simple hand coded Clarion# Windows application. As Figure 1 shows, I created this program by selecting a Windows application type from the New Project dialog.

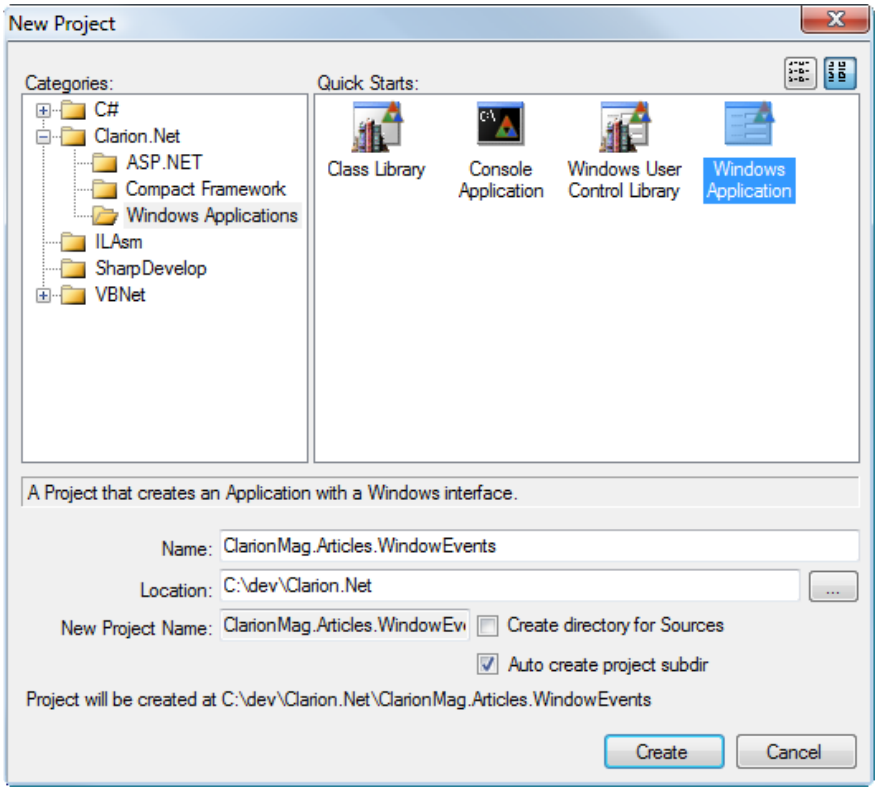


Figure 1. Creating a Windows UI application

Figure 2 shows the Solution Explorer.

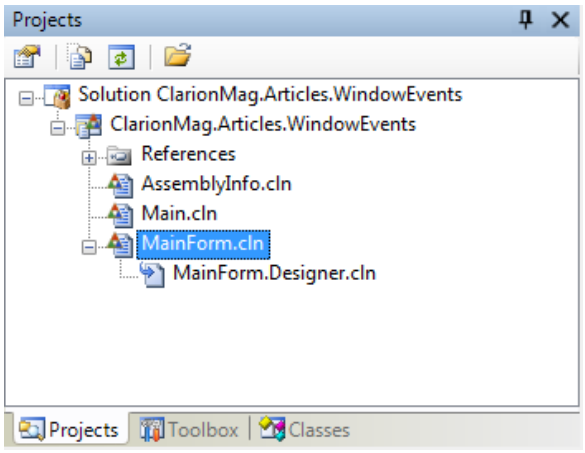


Figure 2. The project files

The Main.cln source file is straightforward:

```
        PROGRAM

        NAMESPACE(ClarionMag.Articles.WindowEvents)
    USING(System)
    USING(System.Drawing)
    USING(System.Windows.Forms)
```

```
    CODE
    !These functions are used to enable XP visual styles for application.
    Application.EnableVisualStyles()
    Application.SetCompatibleTextRenderingDefault(false)
    Application.Run(NEW MainForm())
```

The key here is the last line, which launches a new instance of the MainForm class. In similar Clarion code you'd run a MainForm procedure, but in Clarion# even procedures are object-oriented under the hood, so it makes good sense to take a fully OOP approach here and bypass the conversion of procedural code to OO code.

As Figure 2 indicates, there are two source files that make up MainForm: MainForm.cln and MainForm.Designer.cln. Here's the code for MainForm.cln (minus the MAP, USING statements, etc.):

```
MainForm        CLASS(System.Windows.Forms.Form),TYPE,NETCLASS,PARTIAL
CONSTRUCT           PROCEDURE(),PUBLIC
                END

MainForm.CONSTRUCT  PROCEDURE()
  CODE
  !
  ! The InitializeComponent() call is required
  ! for Windows Forms designer support.
  !
  SELF.InitializeComponent()
```

And here's the similarly abbreviated listing for MainForm.Designer.cln:

```
                MEMBER('')

                NAMESPACE(ClarionMag.Articles.WindowEvents)
  USING(System)
  USING(System.Drawing)
  USING(System.Windows.Forms)

MainForm        CLASS(),TYPE,NETCLASS,PARTIAL
InitializeComponent     PROCEDURE(),PRIVATE
                END

MainForm.InitializeComponent        PROCEDURE()
  CODE
  !
  ! MainForm
  !
  SELF.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font
  SELF.ClientSize = new System.Drawing.Size(292, 266)
  SELF.Text &= 'MainForm'
  SELF.Name &= 'MainForm'
```

Any changes you make will normally be done in MainForm.cln, not MainForm.Designer.cln. When you load up MainForm.cln you have the option (via tabs at the bottom of the pad - see Figure 3) to switch between Source and Design views. When you're in Design view and you move, add or delete a control, those changes are reflected in the Clarion# code in MainForm.Designer.cln. In fact this file contains all the code necessarily to create the window and its controls; it's like Clarion WINDOW structure, only in the form the actual language statements that will be executed.

You don't normally want to update the <formname>.Designer.cln file directly, but I will be coming back to this file periodically to show the event-related code that gets generated by the window designer.

> Note: Yes, you are seeing the MainForm class declared in two different source files. Both declarations have the PARTIAL attribute, which means that the class can be split across multiple source files. This makes it easy to integrate generated code with user-modified code. It can also make things a little confusing, since attributes, parent classes etc. only have to be declared in one of the classes.

Creating an event

I've double-clicked on MainForm.cln in the Solution Explorer to bring up the source file. MainForm.Designer.cln is automatically loaded as well, and by default the design tab is activated, as in Figure 3.

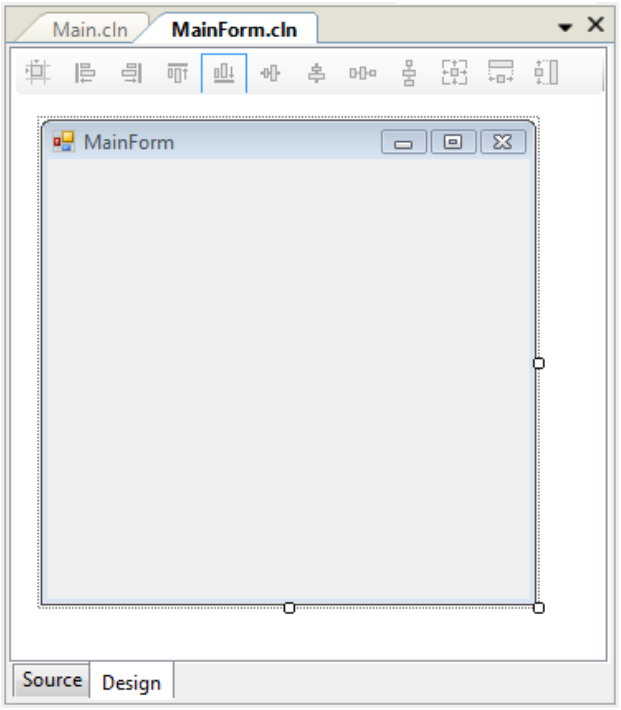


Figure 3. The MainForm class in Design mode

Let's say you want to display a message as the form is closing. Go to the Properties pad and click on the lightning bolt. The event list displays (Figure 4).

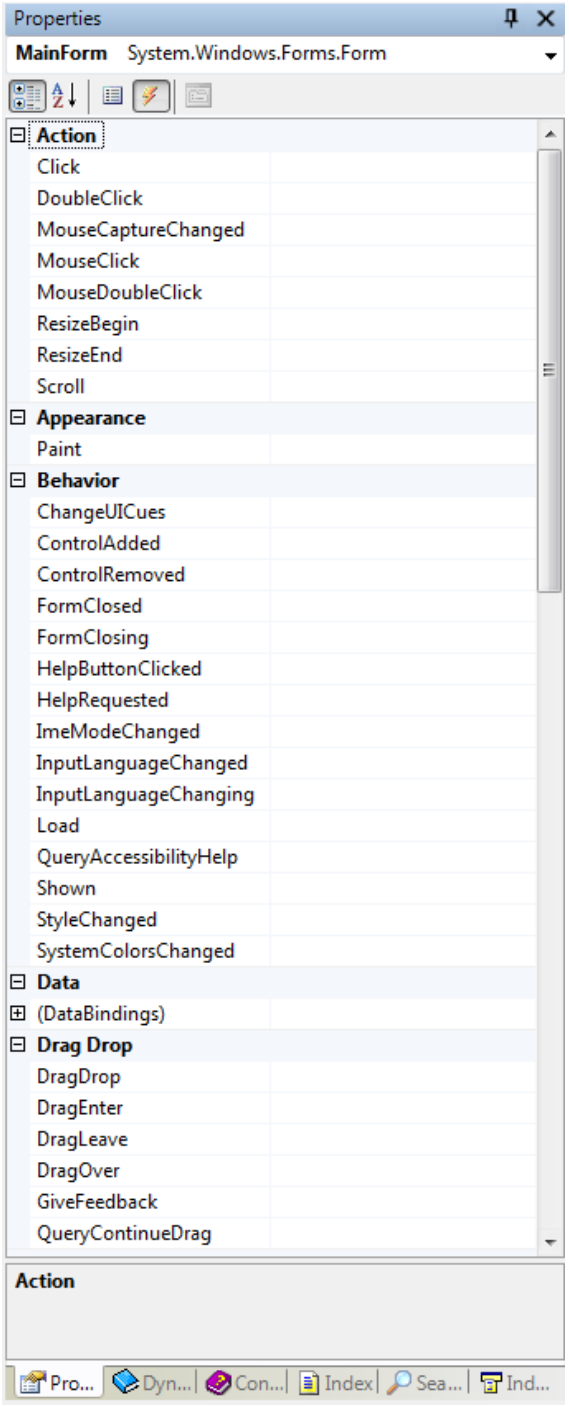


Figure 4. The event list

As the scroll bar indicates, Figure 4 shows around half of the available events. You have the option of looking at these in groups, as shown, or alphabetically.

To execute some code as the window is closing, double-click on the FormClosing item, under Behavior. As you do so, two things happen.

First, you're taken to the source view for MainForm.cln where you see this newly-created code:

```
MainForm.MainForm_FormClosing   PROCEDURE(System.Object sender, |
                                System.Windows.Forms.FormClosingEventArgs e)
```

```
CODE
```

If you've read the previous article on Events you'll recognize this method's signature as the standard signature for an event handler.

Second, the following code is added to MainForm.InitializeComponent in MainForm.Designer.cln:

```
SELF.FormClosing += SELF.MainForm_FormClosing
```

Remember that MainForm is derived from System.Windows.Forms.Form, and the Form class contains numerous public events including one called FormClosing. So if an event handler (like SELF.MainForm_FormClosing) has been registered with FormClosing, that event handler will be called as the form closes.

Custom event handler names

You're not restricted to the standard event handling names. Go back to the event list in Figure 4, type FormIsClosed in the FormClosed entry field and press Enter. You'll be taken back to the source view of MainForm.cln where you'll see this code:

```
MainForm.FormIsClosed  PROCEDURE(System.Object sender, |
                       System.Windows.Forms.FormClosedEventArgs e)
  CODE
```

You've just created an event handler with a non-typical name. It's a good idea, however, to stick with the standard naming convention as the prefix always indicates the control that generates the event.

Let's say you add a control to the window and you give it the name CloseButton. Double-clicking on the Click event will generate a method called CloseButton_Click. That makes it easy to differentiate this event handler from, say, CancelButton_Click.

Mixing things up

When you click on the drop-down list beside any event you're shown a list of all the event handlers currently declared for that window. Specifically, you'll get a list of all methods that have this signature:

```
PROCEDURE(System.Object sender, System.Windows.Forms.FormClosedEventArgs e)
```

That means that you can, if you wish, assign MainForm_FormClosing to CloseButton's Click event. If you do that you'll get a compile error because the method signatures don't match. Button clicks receive a simple System.EventArgs parameter, while FormClosing expects a handler that receives System.Windows.Forms.FormClosingEventArgs, which is a derived class of System.EventArgs. Many of the event handlers take specific EventArgs parameters which makes it less likely you'll inadvertently mix things up, but you could mix up event handlers for same-type controls and never realize it. That's one more reason for keeping to the standard naming convention.

Multicasting

One of the nifty things about events is the ability to attach multiple event handlers. If you had a cancel button and a close button you could assign them each other's handlers this way, typically in the form's Construct method sometime after the call to SELF.InitializeComponent():

```
SELF.CloseButton.Click += SELF.CancelButton_Click

SELF.CancelButton.Click += SELF.CloseButton_Click
```

You have to do this in hand code because there isn't any convenient way to associate more than one handler with an event in the window designer.

You could also exchange the handlers with this code:

```
SELF.CloseButton.Click -= SELF.CloseButton_Click
SELF.CloseButton.Click += SELF.CancelButton_Click
SELF.CancelButton.Click -= SELF.CancelButton_Click
SELF.CancelButton.Click += SELF.CloseButton_Click
```

Summary

Once you have the basic concepts of delegates and events in hand, window and control event handling in Clarion# becomes pretty simple. There's no ACCEPT loop anymore, but just as in ABC you'll find yourself placing code in methods that are called when the corresponding events happen. Once there's a Clarion# AppGen and some templates, adding custom event handling code will no doubt seem even more familiar.

Clarion# also offers some important advantages over Clarion with ABC. You can use multicasting to easily call more than one handler per event, you can programmatically remove handlers, and you can use partial classes to easily add custom code to a class.

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

C7 AppGen Release V

Published 2008-12-19

Blogs on this site

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

Third party developers have an early Christmas present - one more C7 AppGen build. There aren't too many new features in this one - it's mostly bug fixes, including a number of PTSS reports. But there is something many third party vendors have requested: a way to programmatically register templates. You can now use the ClarionCL.exe program to pass instructions to the IDE. So far the only documented options are to register and unregister templates.

Unless something dramatic shows up in testing I won't be reporting further on this build. Hopefully the next release will be the one that goes out to CSP members.

Clarion Magazine

Using HTMLHelp with Clarion

by Mike Hanson

Published 2008-12-23

Clarion's default approach to Help has always been via WinHelp, using *.HLP files to store the material. Although WinHelp still works, it's now considered an "unacceptable" method. A better approach is to use compiled HTML help, which is stored in *.CHM files. I use Help & Manual (www.ec-software.com) to create my help files, and it's a simple choice to create WinHelp or HTML Help.

In this article I'll mention the basic steps for implementing CHM help, but my main focus is to address a stumbling block introduced by Microsoft with one of their many security patches.

WinHelp

With regular WinHelp you add support to your Clarion APP by entering the name of your HLP file in the global properties (Figure 1).

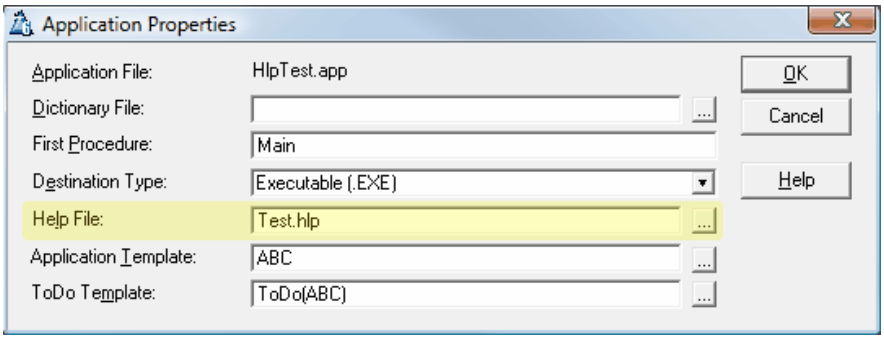


Figure 1. Specifying the help file

Then you enter the topic IDs in your various windows, making sure that there are matching topics in your help file. For example, Figure 2 shows the Help tab of the Frame window. Note the leading tilde (~).

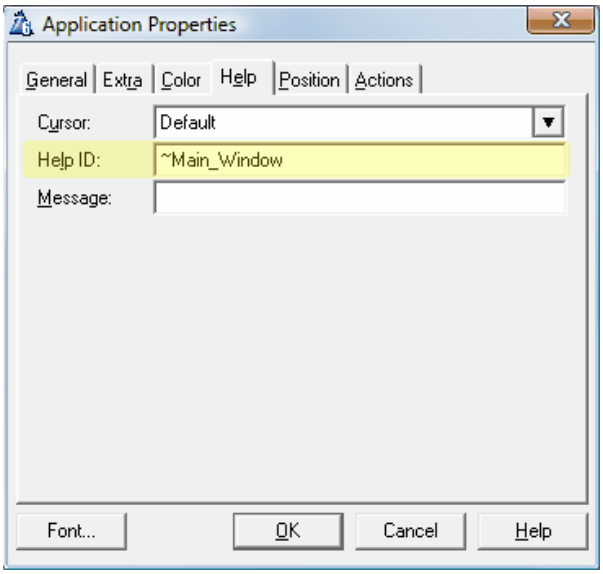


Figure 2. Setting a Help ID

HTML Help

Using HTML Help with Clarion isn't so simple. You don't enter the name of the CHM file into the global properties as above. Instead, Clarion provides a global extension template called wHHGlobal to specify various global options (Figure 3).

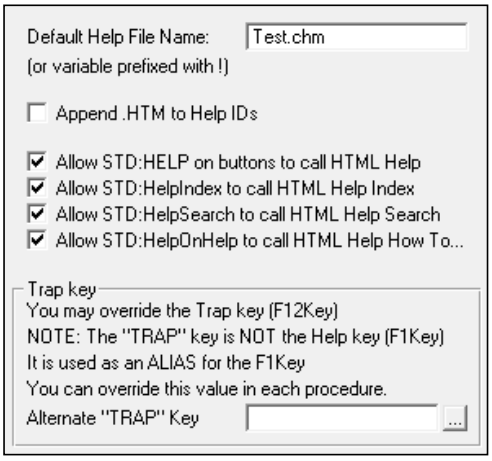


Figure 3. The cwHHGlobal extension template

This global template automatically populates a corresponding procedure template into all (and I mean *all*) of your procedures. Even if you don't specify any additional options in a procedure, and even if there's no WINDOW defined, you'll see a definition like this in the data section:

```
oHH          &tagHTMLHelp
```

I could tweak the template to add `WHERE(%Window)` to the necessary #AT section(s), but it's never concerned me enough to spend the time. Not only that, I would need to remake the change each time a new version of Clarion was released. I try to keep these types of changes to a minimum.

Now you've got a few template options to consider.

If your windows already have topic IDs specified on their Help tabs (from an earlier WinHelp implementation), then

you should turn on the Append .HTM to Help IDs setting in the global extension. This tells the template to get the IDs from your windows, remove the leading tilde (~), and add .htm to the end. You'll see code similar to this in your procedure:

```
oHH &= NEW tagHTMLHelp
oHH.Init( 'Test.chm' )
oHH.SetTopic('Main_Window.htm')
```

Be careful, though! If you decide to have one of your windows jump to an anchor further down in a topic (by appending `#AnchorName` to the topic ID), the template isn't smart enough to handle it. You'll get this incorrect code:

```
oHH.SetTopic('Topic#AnchorName.htm')
```

It should look like this:

```
oHH.SetTopic('Topic.htm#AnchorName')
```

Again, I could have edited the template to have it watch for an anchor and insert the `.htm` before it, but I would have been stuck maintaining the template in perpetuity. Instead, I leave the Append .HTM to Help IDs option off, and I just add the .htm extension myself.

If you don't already have the topic IDs in your windows' Help tabs, then I suggest you use the procedure extension options (Figure 4) to specify the Context URL. (This extension setting also overrides anything that you may have specified in the window's Help tab.)

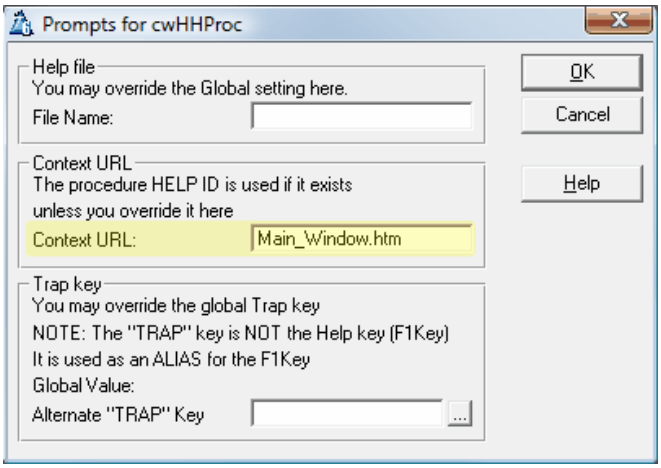


Figure 4. The cwHHProc template prompts

The procedure extension makes it relatively easy to add help IDs to the entire APP. Instead of going into each window structure, navigating to the Help tab, specifying the topic ID, and then backing out and moving on to the next procedure, you can do this:

- Highlight a procedure on the left.
- Expand the "Extension" branch on the right.
- Double-click the Help template on the right.
- Ensure that a suitable topic ID is mentioned.
- Move on to the next procedure and repeat.

Are we done?

If you test your app now you might feel chuffed that you've got HTML Help working with relatively little effort. Don't get

too cocky though, as you might show up at your customer's site (as I did), install the application and its help on a network server and then encounter an error similar to the one shown in Figure 5.

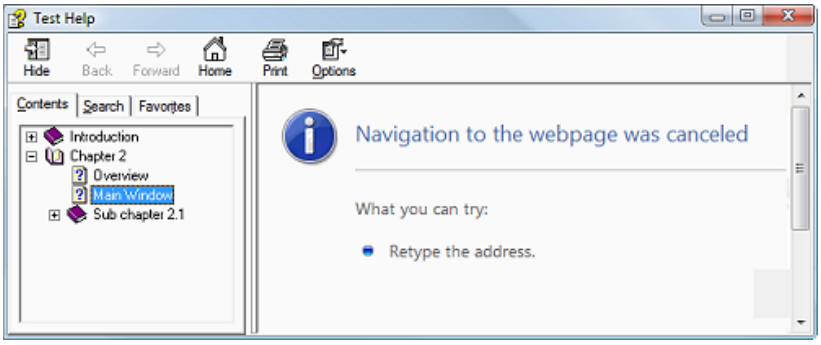


Figure 5. Help file navigation error

What's going on here?!? Well, in their wisdom, the folks at Microsoft (who created and promoted HTML Help) realized that it *could* be dangerous, since HTML pages are subject to various security problems. To reduce the possibility of abuse they decided to make it illegal to use a CHM file hosted on a remote machine (including your own LAN's server). For more information see Microsoft Security Bulletin MS05-026.

Solution #1 - register the CHM file

It turns out that you can tweak a machine's registry to allow a particular CHM file to be used, even if that file is not on the local machine. This can be done manually, but is much easier with EC software's complimentary HHReg HTML Registration Utility. At first this seemed to be a good solution, and I also discovered that Lindersoft's SetupBuilder would run the registration utility it for me.

Unfortunately, it turns out that the ability to update the registry requires administrator access, and this access is sometimes disabled for all users on a network by the site administrator. In this circumstance each user would have to log on as an über-administrator to register the CHM. This didn't seem like a very workable solution to me.

Solution #2 - Copy the CHM File to a Local Directory

Microsoft has suggested an alternative: copy the CHM file to the local machine and then access it from there. That's fine if a user happens to be using the help file directly as a stand-alone file. They can decide where to put it (probably their desktop), and then manually re-open it in the new location. Opening the help file from within the application, however, only works if the application is also in the local directory, but then you no longer have a server-based installation. This approach does show a way forward, however. Here are the solution's criteria:

- A standard location for the help file. A good place to copy the file is the local temporary directory, which can be referenced via the Windows API.
- The file must be copied when the program first starts.
- The file must be copied only if the local copy is missing or different.
- The local file, not the original file, must be referenced by the help system.

I've wrapped up all this functionality into a single procedure called GetHelpFile. Note that Clarion's HTML Help global extension lets you specify a variable name for the help file by prefixing with an exclamation point. Instead of a variable name, use the name of the new procedure as shown in Figure 6.

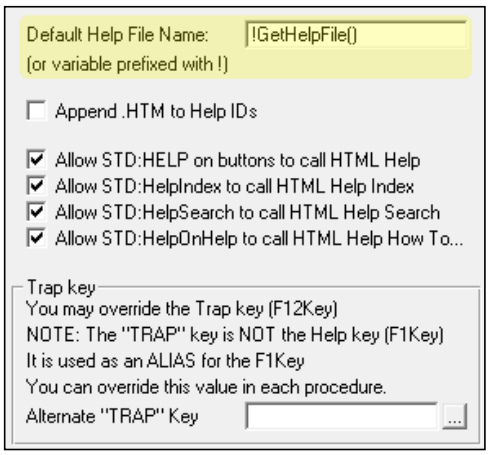


Figure 6. Setting a function as the default help file

This procedure will return the fully qualified path and filename for the help file.

Next, create the new Source procedure and make sure that it's in a module by itself. If you have a multi-APP system then you should define the procedure in the base "dictionary" APP.

In the module settings turn *off* the Allow Repopulate option so that the module embeds don't become disconnected from the procedure. Reference GetTempPathA from the Windows API by adding the following code to the Module Data Section embed:

```
MAP
  MODULE('Windows API')
    GHF::GetTempPath(ULONG,*CSTRING),ULONG,PASCAL,RAW,NAME('GetTempPathA')
  END
END
```

Note the unique name for the Windows API function, which shouldn't conflict with any other definitions of this same procedure elsewhere in the APP. (It doesn't cost anything to have an API call defined multiple times with multiple names as the compiler and linker realize that all the references are to the same resource.)

The procedure's prototype is (),STRING. Here's the pseudo-code:

1. If a local filename has already been determined, then just return that value.
2. Apply default filename, in case of error.
3. Take a "snapshot" of the original remote CHM file. If it's missing, then abort.
4. Determine the directory for local temporary files.
5. Compute the full path (directory\name) for the local file.
6. Remember the current (remote) directory.
7. Change to the local temporary directory.
8. Attempt to take a "snapshot" of the local file.
9. Return to the original remote directory.
10. If the local file is missing or the snapshots are different, then copy the remote file to the local directory.
11. Return the filename to the caller.

After writing an initial version I felt that the single, large procedure was a bit unwieldy, at well over a page of code (with comments, error checks, etc.). I started knocking it down into several local procedures and then realized that it would be a good candidate for *OOPification*.

Data and Class Definitions

Here what goes into GetHelpFile's Data Section embed:

```
HelpFile          EQUATE('Test.chm')  !*** Change to your CHM

SnapshotClass       CLASS,TYPE
Q                  &FILE:Queue,PRIVATE
Construct           PROCEDURE
Destruct            PROCEDURE
Fetch               PROCEDURE(STRING Filename),BYTE
GetSize             PROCEDURE,LONG
GetDate             PROCEDURE,LONG
GetTime             PROCEDURE,LONG
IsEqual             PROCEDURE(SnapshotClass Other),BYTE
                  END
RemoteSnapshot      CLASS(SnapshotClass)
Fetch               PROCEDURE(STRING Filename),BYTE  !Override
                  END
LocalSnapshot       CLASS(SnapshotClass)
Path               CSTRING(FILE:MAXFILENAME+1),PRIVATE
Construct           PROCEDURE
Fetch               PROCEDURE(STRING Filename),BYTE  !Override
Copy                PROCEDURE(STRING Filename),BYTE
GetPath             PROCEDURE,STRING
                  END

LocalHelpFile       CSTRING(FILE:MAXFILENAME+1),STATIC
```

There are several noteworthy items:

- The HelpFile equate must be changed to the name of your own file.
- SnapshotClass contains a FILE:Queue reference, understands how to Fetch a directory entry, has property "getters" for several of its attributes and can compare itself to another snapshot.
- RemoteSnapshot is an object derived from SnapshotClass. It overrides the Fetch method to display an error message if the file is missing.
- LocalSnapshot is also derived from SnapshotClass, although it does much more than RemoteSnapshot. LocalSnapshot is responsible for determining the local temporary directory (along with providing a "getter" to access that value). It also overrides the Fetch method so that it can do a "directory shuffle". Finally, it provides a method to Copy the help file from the current (remote) directory to the local temporary directory.
- LocalHelpFile will remember the location of the local CHM file. It's marked as STATIC so that it retains its value between calls.

Main procedure code

Because so much code will be encapsulated within the class methods, the main procedure code is quite straightforward:

```
IF LocalHelpFile = ''       !Is this the first time?
  LocalHelpFile = HelpFile  !Apply default filename
  IF RemoteSnapshot.Fetch(HelpFile) = LEVEL:Benign
    IF  LocalSnapshot.Fetch(HelpFile) = LEVEL:Benign  |
      AND LocalSnapshot.IsEqual(RemoteSnapshot)
      MESSAGE('CHM already in TEMP')  !*** Remove after testing
      DO CalcLocalHelpFile
    ELSIF LocalSnapshot.Copy(HelpFile) = LEVEL:Benign
      DO CalcLocalHelpFile
    END
  END
END
MESSAGE(LocalHelpFile)  !*** Remove after testing
? ASSERT(LocalHelpFile <> '')
RETURN LocalHelpFile
```

The CalcLocalHelpFile routine is simply:

```
CalcLocalHelpFile ROUTINE
  LocalHelpFile = LocalSnapshot.GetPath() & HelpFile
```

Snapshot Class Methods

SnapshotClass is intended to be a generic base class, providing functionality that any deriver might require. It has the queue reference, which needs to be allocated and disposed in the constructor and destructor:

```
SnapshotClass.Construct PROCEDURE
  CODE
  SELF.Q &= NEW FILE:Queue

SnapshotClass.Destruct PROCEDURE
  CODE
  FREE(SELF.Q)
  DISPOSE(SELF.Q)
```

SnapshotClass.Fetch simply uses the DIRECTORY command to load a single entry into the queue. If no file is found, the code clears the queue and returns LEVEL:Notify. On success it returns LEVEL:Benign.

```
SnapshotClass.Fetch PROCEDURE(STRING Filename)!,BYTE
  CODE
  DIRECTORY(SELF.Q, Filename, ff_:NORMAL)
  GET(SELF.Q, 1)
  IF ERRORCODE() <> 0
    CLEAR(SELF.Q)
    RETURN LEVEL:Notify
  ELSE
    RETURN LEVEL:Benign
  END
```

Consumers of the SnapshotClass shouldn't need to know that it uses a DIRECTORY and a QUEUE to determine the snapshot. As far as the outside world is concerned, the class only has to expose the attributes of Size, Date and Time. I'll treat those as *properties* and provide read-only access to their values via "getters":

```
SnapshotClass.GetSize PROCEDURE!,LONG
  CODE
  RETURN SELF.Q.Size

SnapshotClass.GetDate PROCEDURE!,LONG
  CODE
  RETURN SELF.Q.Date

SnapshotClass.GetTime PROCEDURE!,LONG
  CODE
  RETURN SELF.Q.Time
```

Finally, the SnapshotClass can determine whether it's equal to another snapshot:

```
SnapshotClass.IsEqual PROCEDURE(SnapshotClass Other)!,BYTE
  CODE
  IF SELF.GetDate() <> Other.GetDate()  |
  OR SELF.GetTime() <> Other.GetTime()  |
  OR SELF.GetSize() <> Other.GetSize()
    RETURN False
  ELSE
    RETURN True
  END
```

RemoteSnapshot methods

The only method in the derived RemoteSnapshot class is Fetch. It merely calls the parent method and displays an error if the specified file is missing:

```
RemoteSnapshot.Fetch PROCEDURE(STRING Filename)!,BYTE
ReturnValue BYTE,AUTO
  CODE
  ReturnValue = PARENT.Fetch(Filename)
  IF ReturnValue <> LEVEL:Benign
    MESSAGE('Missing file '& Filename, 'Error!', ICON:Exclamation)
  END
  RETURN ReturnValue
```

LocalSnapshot Methods

Much of the work is done within the following methods. Everything starts in the constructor, which determines the local directory for temporary files:

```
LocalSnapshot.Construct PROCEDURE
  CODE
  IgnoreResult# = GHF::GetTempPath(SIZE(SELF.Path), SELF.Path)
```

The Fetch method is overridden to switch to the temporary directory before calling the parent method, and then to return to the original network directory:

```
LocalSnapshot.Fetch PROCEDURE(STRING Filename)!,BYTE
SaveDir     CSTRING(FILE:MAXFILENAME+1),AUTO
ReturnValue BYTE,AUTO
  CODE
  SaveDir = LONGPATH()
  SETPATH(SELF.Path)
  ReturnValue = PARENT.Fetch(Filename)
  SETPATH(SaveDir)
  RETURN ReturnValue
```

The Copy method is responsible for copying the CHM file from the current (network) directory to the local directory:

```
LocalSnapshot.Copy PROCEDURE(STRING Filename)!,BYTE
  CODE
  MESSAGE('Copying CHM to TEMP')  !*** Remove after testing
  SETCURSOR(CURSOR:Wait)
  COPY(Filename, SELF.Path)
  SETCURSOR
  IF ERRORCODE() <> 0
    MESSAGE('Cannot copy '& HelpFile &' to '& SELF.Path |
          &'||Error #'& ERRORCODE() &' - '& ERROR(),   |
           'Error!', ICON:Exclamation)
    RETURN LEVEL:Notify
  ELSE
    RETURN LEVEL:Benign
  END
```

Finally, the class exposes the value of the temporary directory via a getter so that the main body of code can calculate the full path to the help file:

```
LocalSnapshot.GetPath PROCEDURE!,STRING
  CODE
  RETURN SELF.Path
```

OOP scoping notes

You may have noticed that I'm passing the HelpFile value in as a parameter to some of the class methods, rather than having them access the equate directly. This is just forward-thinking OOP technique. Your classes should represent stand-alone functional units, and they shouldn't make assumptions regarding the existence of variables outside their scope. These methods could be moved into a more generic library with almost no changes.

You may also notice that I've marked all of the class properties as PRIVATE and have provided getters for access. In its

current state these getters are superfluous, since the PRIVATE attribute is overlooked for code within the current module. Again, I've done this in case I move this code to a generic library later.

Source files

The downloadable source zip contains the following files:

- Test.hlp - Sample WinHelp file (1 of 2)
- Test.cnt - Sample WinHelp file (2 of 2)
- Test.chm - Sample HTML Help file
- HlpTest.app - APP to test WinHelp
- ChmTest.app - APP to test HTML Help (including GetHelpFile procedure)
- GetHelpFile.txa - Importable TXA with module + procedure

Conclusion

As you can see, Clarion's HTML Help support is a bit quirky, but basically workable. After implementing it for the first time (but before handling the copy-to-temp issue), I learned that Carl Barnes has a product called CHM4Clarion. Carl's utility includes this feature of copying the CHM file to the temporary directory, along with many other features that improve upon Clarion's built-in solution. I debated whether to change the pre-existing project over to CHM4Clarion, or to implement the above solution. I ended up doing it myself, but Carl's productis an excellent choice if you're looking for a comprehensive, pre-built solution.

Download the source

Mike Hanson is affiliated with BoxSoft, which produces the "Super" series of templates, distributed through Mitten Software. He has been creating add-on products for Clarion since his Public Domain Models for CPD 2.0 back in 1988. He's also written articles for every Clarion-related publication, and has spoken at numerous conferences and training seminars. If you have any questions, you can reach him via www.boxsoft.net.

Reader Comments

Posted on Sunday, December 28, 2008 by John Griffiths

I first tried adding html help as you have clearly described and it does work! Then I tried Carl's CHM4Clarion and found that a much easier solution. That is all I use now. I had some trouble removing all the oHH declarations from all the procedures so I could convert a few Apps to CHM4Clarion.

Add a comment

Clarion Magazine

Formatting Names Using Proper Case

by Mike Hanson

Published 2008-12-22

ALMOST EVERYONE HATES ALL CAPS! Yet many databases are strewn with upper case data that would be much better presented (and even stored) using *proper case*.

To this end, various Clarion data entry controls offer a CAP option:

> The CAP attribute (PROP:CAP) specifies "Proper Name Capitalization," where the first letter of each word is capitalized and all other letters are lower case. The user can override this default behavior by pressing the SHIFT key to allow an upper case letter in the middle of a name (allowing for names such as, "McDowell") or SHIFT while CAPS-LOCK is on, forcing a lower case first letter (allowing for names such as, "von Richtofen").

CAP is a good start, but it's not very *smart*, as these common examples show:

As Entered	**Via ENTRY,CAP**	**Preferred**
mcdowell	Mc**d**owell	McDowell
john smith, md	John Smith, M**d**	John Smith, MD
james o'sullivan, phd	James O'**s**ullivan, Ph**d**	James O'Sullivan, PhD
andrew guidroz ii	Andrew Guidroz I**i**	Andrew Guidroz II
von richtofen	Von Richtofen	von Richtofen
alexander the great	Alexander **T**he Great	Alexander the Great

Also the CAP option works only when you're entering data. There's no facility included with Clarion to capitalize an existing string. (You could probably create a procedure to simulate data entry in a hidden window with PRESS, but that's quite a kludge). Clarion developers routinely ask for a capitalization function, such as a recent request from Alan Cochran in the sv.clarion.clarionbeta newsgroup. That's what spurred me to revisit my solution.

Rolling My Own

About 15 years ago I created my own Proper function (back when I was still using Clarion for DOS). Over the years Proper has evolved through the many versions of Clarion, and from procedural code to object-oriented. It's also been extended to handle more special case exceptions (some of which were suggested by Skip Williams in Alan's newsgroup thread).

My solution needs to provide a number of facilities:

- A basic Proper procedure to which you pass a string and get a return string with corrected case.
- A ProperControl procedure to which you pass a control's equate. This function converts the field's contents to proper case and DISPLAYs the new value.
- Built-in support for many typical forced-case words.
- The ability to add more forced-case words, or to override/eliminate the built-in list.

The two simple procedures, Proper and ProperControl, are prototyped like this:

```
Proper        PROCEDURE(STRING Name),STRING
ProperControl PROCEDURE(SIGNED Feq),STRING,PROC
```

Note that ProperControl also returns the string result, but the PROC attribute means you can ignore the return value.

The list of built-in forced-case words is not extensive. It *should* handle most of your needs, but you may want to add more, either directly to the class or by varying the list for each implementation. Here's what's included automatically:

- MD, PhD, DDS are formatted appropriately
- NE, NW, SE, SW are capitalized
- Roman numerals from 2 to 49 (II to XLIX) are capitalized, except for the single-character ones that are capitalized automatically.
- Particles: *de, du, la, los, van, von*. These are forced to lowercase. Note that French names capitalize *La, Le* and *Les*, so if you expect to encounter many French names then you might want to override or remove la.
- Articles and conjunctions: *a, an, and, of, the*. These are forced to lowercase, as long as they are not found at the beginning of the name.

Class Declaration

The bulk of the code is in a class called MHProperClass. Here's the declaration, along with the required QUEUE:

```
MHProperForcedWordQueue QUEUE,TYPE
Word                &STRING
PaddedUpper         &STRING
Length              LONG
NotAtStart          BYTE
                END

MHProperClass       CLASS,TYPE
ForcedWord          &MHProperForcedWordQueue,PRIVATE
Name                &STRING
Length              LONG

AcceptControl       PROCEDURE(SIGNED Feq),STRING,PROC
AddForcedWord       PROCEDURE(STRING Word,<BYTE NotAtStart>)
Construct           PROCEDURE
Destruct            PROCEDURE
DisposeForcedWord   PROCEDURE(LONG X),BYTE,PROC,PRIVATE
DisposeName         PROCEDURE
HandleForcedWords   PROCEDURE
ToProper            PROCEDURE(STRING Name),STRING
RemoveForcedWord    PROCEDURE(STRING Word),BYTE,PROC
RemoveForcedWords   PROCEDURE
                END
```

The first thing is the QUEUE,TYPE that's used to remember the forced-case words. Since the length of each forced word could vary significantly, I use a string reference allocated at runtime rather than a fixed-length string.

To speed up processing I also store a PaddedUpper version of the word, again using a string reference. I store the length

since there's no sense in re-determining these values every time I process a name.

Finally, MHProperForcedWordQueue contains NotAtStart, which indicates whether it applies when it appears at the start of the name (e.g. "Alexander the Great" versus "The Great Alexander").

The MHProperClass properties include a queue reference for the ForcedWord list, a reference string for the Name to be processed and returned, and the name's Length. This makes inter-method handling a bit easier and keeps the result handy as long as the class object remains instantiated.

The ProperClass methods are declared in alphabetical order. Here's a functional grouping to clarify their purpose:

Initializating:

- Construct - Allocate ForcedWord queue, and add built-in forced words.
- AddForcedWord(STRING Word,<BYTE NotAtStart>) - Add one forced word.
- RemoveForcedWord(STRING Word),BYTE - Remove one forced word by name.
- RemoveForcedWords - Remove all exceptionsforced words.

Disposing:

- Destruct - Dispose of any allocated memory, empty queues, etc.
- DisposeName - Dispose of Name and set Length to zero.
- DisposeForcedWord(LONG X),BYTE - Fetch a ForcedWord queue record by number, dispose of all references, and delete the record.

Processing:

- ToProper(STRING Name),STRING - Convert passed Name to proper case.
- AcceptControl(SIGNED Feq),STRING - Convert contents of passed control to proper case and update control with new value.
- HandleForcedWords - Scans the Name for all possible force word instances, fixing as necessary.

Class Methods

Let's look at some of the methods, beginning with Construct:

```
MHProperClass.Construct PROCEDURE
  CODE
  SELF.ForcedWord &= NEW ProperForcedWordQueue

  SELF.AddForcedWord('MD')
  !...
  SELF.AddForcedWord('the', True)
```

Construct allocates memory and assigns the reference for a new MHProperForcedWordQueue, then adds the built-in forced-case words. If you want to add forced words, override existing forced words or remove all forced words, then you can call the appropriate methods as soon as your object is instantiated; the constructor will always be called before your code.

The AddForcedWord method populates the ForcedWord queue record and adds the record to the queue:

```
MHProperClass.AddForceWord PROCEDURE(STRING Word,<BYTE NotAtStart>)
  CODE
  SELF.ForcedWord.Length       = LEN(CLIP(Word))
```

```
  SELF.ForcedWord.Word       &= NEW STRING(SELF. ForcedWord.Length)
  SELF.ForcedWord.Word        = Word

  SELF.ForcedWord.PaddedUpper &= NEW STRING(SELF.ForcedWord.Length+2)
  SELF.ForcedWord.PaddedUpper  = ' ' & UPPER(Word) & ' '

  SELF.ForcedWord.NotAtStart   = NotAtStart

  ADD(SELF.ForcedWord)
```

The most noteworthy thing here is the assignment of the PaddedUpper field, which contains the word in upper case with leading and trailing spaces. I use this to search for "[space]WORD[space]" rather than trimmed substrings. Since this value and the Length won't change I may as well calculate them as I populate the queue.

The destructor must ensure that all references are disposed. Usually cleaning up queues is the toughest job, but I've encapsulated that in the RemoveForcedWords method. Here's the simple Destruct method:

```
MHProperClass.Destruct PROCEDURE
  CODE
  SELF.DisposeName
  SELF.RemoveForcedWords
  DISPOSE(SELF.ForcedWord)
```

RemoveForcedWords is also rather concise, as it relies upon the DisposeForcedWord method:

```
MHProperClass.RemoveForcedWords PROCEDURE
  CODE
  LOOP WHILE SELF.DisposeForcedWord(1) = 0.
```

Here's the meat of the matter:

```
MHProperClass.DisposeForcedWord PROCEDURE(LONG X)!,BYTE
  CODE
  GET(SELF.ForcedWord, X)
  IF ERRORCODE() = 0
    DISPOSE(SELF.ForcedWord.Word)
    DISPOSE(SELF.ForcedWord.PaddedUpper)
    DELETE(SELF.ForcedWord)
    RETURN 0
  ELSE
    RETURN 1
  END
```

DisposeForcedWord attempts to fetch the specified queue record and then disposes of the references and deletes the record.

Now that things are being initialized and disposed properly, it's time examine the code that handles the capitalization of a name. This logic is housed completely within the ToProper method:

```
MHProperClass.ToProper PROCEDURE(STRING Name)!,STRING
Mc LONG(-2)
X  LONG,AUTO
```

```
CODE
SELF.DisposeName

SELF.Length = LEN(CLIP(Name))
IF SELF.Length = 0
 RETURN ''

ELSE
 SELF.Name &= NEW STRING(SELF.Length)
 SELF.Name  = Name

 LOOP X = 1 TO SELF.Length
  IF ISALPHA(SELF.Name[X])
   IF X = 1
    SELF.Name[X] = UPPER(SELF.Name[X])
   ELSIF ISALPHA(SELF.Name[X-1])
    IF Mc = X - 2
     SELF.Name[X] = UPPER(SELF.Name[X])
    ELSE
     SELF.Name[X] = LOWER(SELF.Name[X])
     IF SELF.Name[X]='c' AND SELF.Name[X-1]='M'
      Mc = X - 1
     END
    END
   ELSIF UPPER(SELF.Name[X]) = 'S' AND SELF.Name[X-1] = ''''
    !End of the word?
    IF X = SELF.Length OR ~ISALPHA(SELF.Name[X+1])
     SELF.Name[X] = 's'
    ELSE
     SELF.Name[X] = 'S'
    END
   ELSE
    SELF.Name[X] = UPPER(SELF.Name[X])
   END
  END
 END
 SELF.HandleForcedWords
 RETURN SELF.Name
END
```

The local Mc variable holds the starting point of the name like "McDowell". Mc is initialized to -2, which is well before the start of the string.

First of all I dispose of any Name that might be hanging around from earlier calls to ToProper (for this same instantiation). Then I measure the Length of the string, and if it's empty I just return an empty string. Otherwise I allocate a new string to the Name reference and assign the passed value.

Next comes the loop that examines each alpha character in the string. Figure 1 shows a flowchart to illustrate what's going on.

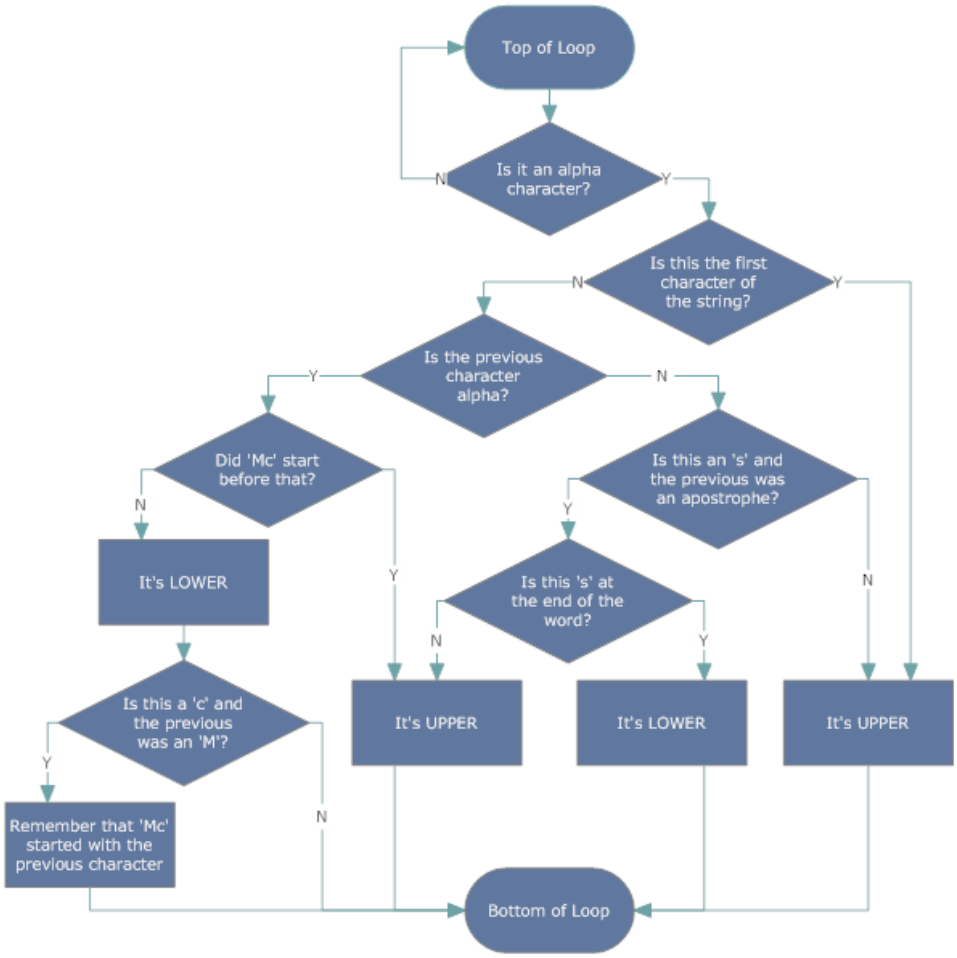


Figure 1. Proper case logic

As you can see, each character's surroundings are checked to determine whether it's at the start of a word. There are also accommodations for apostrophes, "Mc", etc.

Once ToProper applies the standard capitalization rules, it calls the HandleForcedWords method to finish the job:

```
ProperClass.HandleForcedWords PROCEDURE
N &STRING
P LONG,AUTO
X LONG,AUTO
 CODE
 !--- Pad ends and change non-alpha to spaces
 N &= NEW STRING(SELF.Length+2)
 N  = ' ' & UPPER(SELF.Name) & ' '
 LOOP X = SELF.Length + 1 TO 2 BY -1
  IF NOT (ISALPHA(N[X]) OR NUMERIC(N[X]))
    N[X] = ' '
  END
 END

 LOOP X = 1 TO RECORDS(SELF.ForcedWord)
  GET(SELF.ForcedWord, X)
```

```
    P = CHOOSE(~SELF.ForcedWord.NotAtStart, 1, 2)
    LOOP
      P = INSTRING(SELF.ForcedWord.PaddedUpper, N, 1, P)
      IF P = 0 THEN BREAK.
      SELF.Name[P : P+SELF.ForcedWord.Length-1] = SELF.ForcedWord.Word
      P += 1
    END
  END
  DISPOSE(N)
```

The first thing HandleForcedWords does is create an uppercase version of the name with leading and trailing spaces and convert any non-alphanumeric characters to spaces. This makes searching for words easier.

Then the code loops through the ForcedWord queue, searching for matching words. Note that if two versions of the same word (e.g. *la* and *La*) are in the list, then the most recently added will apply.

The local variable P contains the starting position for INSTRING searches. P is initialized to 1 except if the NotAtStart parameter was specified when the forced word was added. Additional searches for the same string are started from the previous instance (+1).

Note that I'm searching for PaddedUpper versions of the forced word within a padded, uppercase version of the name. Therefore, when an instance is found the position P will be the same in the real Name variable (which isn't padded). The code uses string splicing to apply the actual forced word to the name.

Procedural implementation

I called my original procedural function "Proper", and I wanted my OOP version to maintain compatibility with that function. Also, for many situations you may not need to use OOP; regular procedures will work just fine. The following code creates procedure wrappers around local instances of MHProperClass:

```
Proper PROCEDURE(STRING Name)
P MHProperClass
  CODE
  RETURN P.ToProper(Name)

ProperControl PROCEDURE(SIGNED Feq)!,STRING,PROC
P MHProperClass
  CODE
  RETURN P.AcceptControl(Feq)
```

OOP implementation

Now that you've seen the class implementation, you're aware that there is overhead to the instantiation process (especially populating the forced words). If you expect to call ToProper repeatedly you should use the OOP approach. The other benefit of using the class directly is that you can tweak the forced words.

Look to the procedural wrappers for guidance. Just define the MHProperClass object in your data section and then call that object's methods to do your work. If you want to instantiate MHProperClass as a global object then you must either add the THREAD attribute or derive the class to add synchronization support. MHProperClass as written is not thread-safe, and if two threads call methods of a non-threaded global instance of this class at the same time you could get invalid results.

Source code

The downloadable source zip includes the following files:

- Test.app - Test program
- MHProper.ico - Icon for the test program
- MHProper.inc - Class header file
- MHProper.cpy - Map include file (when adding MHProper.clw as an APP module)
- MHProper.clw - Main source module

Conclusion

Perhaps Clarion will eventually get its own Capitalize procedure, which may or may not support as much functionality as MHProperClass. Fortunately you don't have to wait until then, as you can use this solution now and into the future.

I'll admit that there's one glaring omission to my solution: it would be nice if an ENTRY field could support MHProperClass directly (like CAP). This *could* be done by subclassing the control, which I may address in a future article.

Download the source

Mike Hanson is affiliated with BoxSoft, which produces the "Super" series of templates, distributed through Mitten Software. He has been creating add-on products for Clarion since his Public Domain Models for CPD 2.0 back in 1988. He's also written articles for every Clarion-related publication, and has spoken at numerous conferences and training seminars. If you have any questions, you can reach him via www.boxsoft.net.

Reader Comments

Posted on Monday, December 22, 2008 by Terence Davidson

Maybe I'm missing it but there doesn't seem to be a download link

.....

Posted on Wednesday, December 24, 2008 by Dave Harms

Fixed - thanks for pointing that out.

Dave

Add a comment

Clarion Magazine

Creating a Threaded Web Service Client

by Dave Harms

Published 2008-12-16

This is the third article in a series on web services. In Part 1 I described the basic process of creating and using web services; in Part 2 I introduced a web service I created to assist with Stu Andrews' Clarion Folk "pondcasts." In this article I'll explain the WinForms client that uses the web service. For a refresher on what exactly this application is supposed to please read Part 2.

As with the projects discussed Part 2, I created a new WinForms PondCastClient project under the PondCast solution. The default WinForms app has a Main.cln which contains the CODE statement that is the application entry point, and the standard issue code for opening the first window:

```
CODE
!These functions are used to enable XP visual styles for application.
Application.EnableVisualStyles()
Application.SetCompatibleTextRenderingDefault(false)
Application.Run(NEW MainForm())
```

The MainForm class is in MainForm.cln, and at first is just an empty window. In Figure 1 I've populated the necessary controls, including: a Label control; an Entry control; a Listbox control; and five Button controls. The last two buttons are surrounded by a GroupBox control, for visual purposes only.

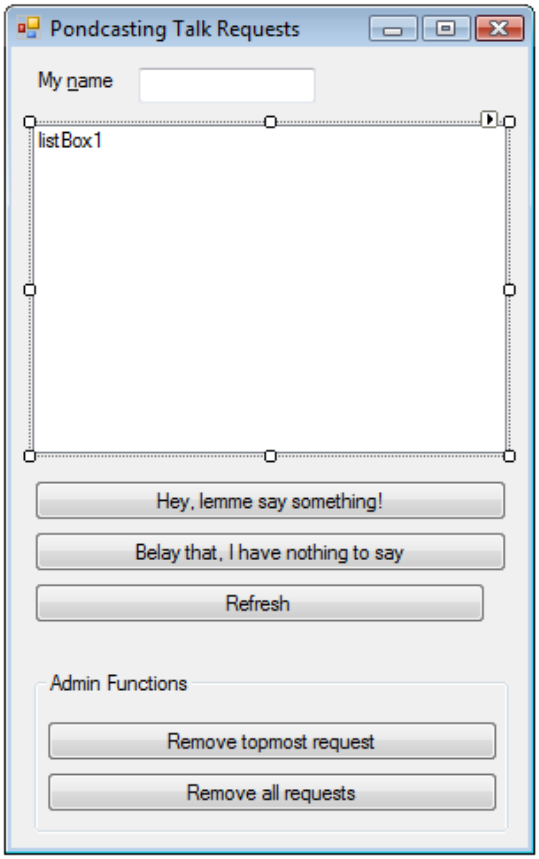


Figure 1. The PondcastClient with with controls populated

The following code executes on the Entry control's Validating event, ensuring that the buttons to add and remove talk requests only work when a value is entered into the Name field:

```
MainForm.UserName_Validating  PROCEDURE(System.Object sender, |
                              System.ComponentModel.CancelEventArgs e)
  CODE
  if self.UserName.Value = ''
     self.TalkRequestButton.Enabled = false
     self.CancelAllButton.Enabled = false
     self.CancelTalkRequestButton.Enabled = false
  else
     self.TalkRequestButton.Enabled = TRUE
     self.CancelAllButton.Enabled = TRUE
     self.CancelTalkRequestButton.Enabled = true
  end
```

The web service

I added the web service as described in Part 1, using the local server URL. I declared the web service object in the class:

```
svc                 localhost.Service
```

and I created an instance of that class in the MainForm.Construct method:

```
self.svc = new localhost.Service()
```

I added event handlers using the approach described in Event Handling In Clarion#.

If the user presses the button to add a talk request, this code executes:

```
MainForm.TalkRequestButton_Click  PROCEDURE(System.Object sender, |
                          System.EventArgs e)

  CODE
  if self.svc <> null
    Cursor.Current = cursors.WaitCursor
    self.svc.AddTalkRequest(self.UserName.Value)
    Cursor.Current =cursors.Default
    self.Redisplay()
  end
```

The Redisplay method loads up the listbox with whatever information it gets back from the web service:

```
MainForm.Redisplay    PROCEDURE
requests                  string[]
request                   STRING
  code
  if self.svc <> null
    Cursor.Current = cursors.WaitCursor
    requests = self.svc.GetAllRequests()
    self.listBox1.items.Clear()
    foreach request in requests
      self.listbox1.items.Add(request)
    END
    Cursor.Current =cursors.Default
  end
```

I chose a ListBox control because all I really needed was a simple list of text strings. Listbox controls have an Items collection property which in this case contains all the strings to display. I thought at first that clearing the list box and adding the items each time might cause some flicker, but I haven't noticed any display issues so far.

I use a ForEach loop to transfer the strings to the listbox. The local string array requests holds the result of GetAllRequests, and request is a temp variable that holds each string from the array in turn. The whole block of code is wrapped with statements that enable and disable the wait cursor.

The remaining buttons' event handlers are similar to TalkRequestButton_Click but they call different web service methods.

All of the works well enough, except that the display isn't automatically updated. And that raises the whole issue of user interface responsiveness.

Waiting in line

If you're testing this application against a local copy of the web service, you'll probably find the response time to be just about instantaneous. If this kind of responsiveness is guaranteed (not likely) you might be tempted to treat the web service call the same as you would a local method call. And one way to handle regular calls is to use a timer. The way you do this in WinForms is to go to the Toolbox, select Components, and drag a Timer component onto your window.

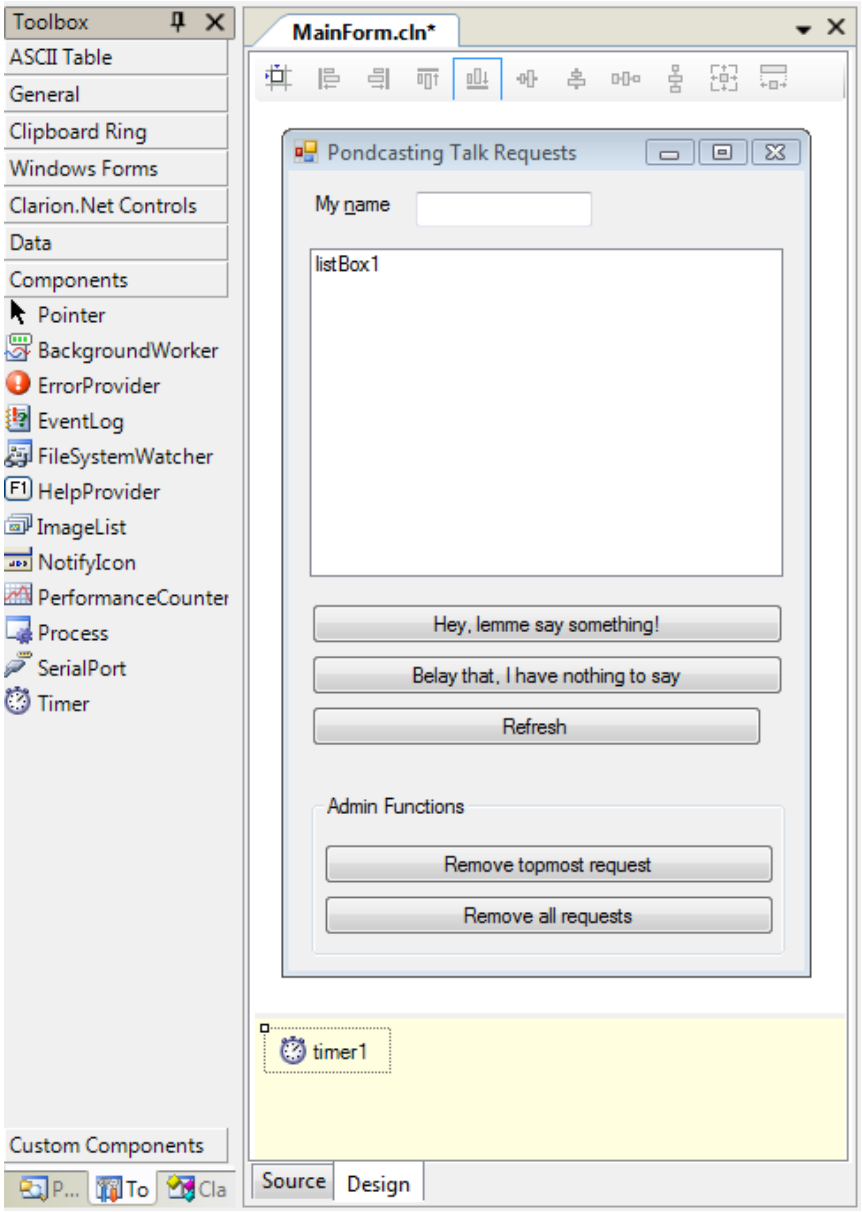


Figure 3. Adding a Timer component

Now go to the Timer component's event list and double-click on the Tick event. In the resulting method source you can add a call to Redisplay:

```
MainForm.Timer1_Tick  PROCEDURE(System.Object sender, System.EventArgs e)
  CODE
  self.ReDisplay()
```

You'll also want to go to the Timer's properties and set the interval to something like 1000 milliseconds.

Problem solved, right? Not so fast.

The first thing you'll notice is that every second you get a flicker of a wait cursor while the list is refreshed. You could change the Redisplay code to remove the wait cursor. That will work fine as long as you're testing on a local web service, but what happens when you move that service to production and it takes longer to get a response from the service because you're having to go across the Internet? That's where you exactly want a wait cursor, so the user knows the interface is going to be unresponsive.

In this case I think it's appropriate to have a wait cursor on the web service calls that are triggered by the user pressing a button. But I would like the automatic refreshing of the listbox happen in the background, without any wait cursor and without any interference in the user's ability to press buttons.

PondCastClientThreaded

The solution to this problem is to start up a background thread and have that thread call the Redisplay method. And this raises some thread safety issues.

If you're following along in the downloaded source code, have a look at the PondCastClientThreaded project. If you want to automatically run this project when you press the Run button, be sure to right-click on the project in the Solution explorer and choose Set as Startup Project.

I've added a delegate to the map:

```
MAP
  StringArrayDelegate(String[] sa),DELEGATE
end
```

(Incidentally, adding this delegate causes the structure designer to burp and issue a warning about a class being declared before the window class. You can safely ignore the warning.)

I've added a USING directive:

```
using System.Threading
```

And I've added a new method to the MainForm class and a new object declaration called StateLock:

```
RequestMonitorThread    PROCEDURE
...
StateLock               object
```

My constructor has a new local variable:

```
t                   THREAD
```

and the constructor code, following the InitializeComponent call, now looks like this:

```
self.StateLock = new Object()
self.svc = new services.clarionmag.com.PondcastingService()
t = new System.threading.Thread(self.RequestMonitorThread)
t.IsBackground = TRUE
t.Start()
```

My StateLock object is nothing more than a synchronization token; I'll explain in a moment how it's used.

The t variable is an instance of the Thread class, which manages the thread on which RequestMonitorThread runs. Most important, the thread's IsBackground property is set to True. This ensures that the thread will terminate when MainForm terminates; if IsBackground is set to False then you have a daemon thread that will continue to run even after your application terminates (and you'll have to kill the process in TaskManager).

Finally, the thread is started with the Start method.

Here's the code for RequestMonitorThread. Remember that although this method is running in its own thread, that thread only exists as long as the method is doing something. For that reason the code sits in a loop, sleeping for a set period of time and then getting the current talk requests from the service and redisplaying them. The initial Sleep statement, before the loop begins, is there to make sure that all the window's objects have had time to get fully instantiated. I've lost the reference to the document that prompted me to put this code in, but apparently there can be issues with cross-thread communication happening before a window is quite ready. The code may in fact not be needed, but it isn't going to hurt either. Belt and suspenders, that's me.

```
MainForm.RequestMonitorThread       PROCEDURE
  code
  System.Threading.Thread.Sleep(500)
  LOOP
    System.Threading.Thread.Sleep(2000)
    self.ReDisplay(self.svc.GetAllRequests())
  END
```

In fact, two seconds may be too long a time to wait. If the web service takes two seconds to execute, then the delay is four seconds. Salt to taste.

Things get interesting when this background thread calls Redisplay. You may have heard that WinForms is not thread safe. If you attempt to update a control on one thread from a different thread, as is happening here, bad things can happen. There are a variety of strategies for managing thread safety; what follows is one of the most common approaches.

Here's the updated code for the Redisplay method. There's one minor change is that this method no longer sets the cursor. When called from the background thread you don't want the cursor changed; when called via the Force Refresh button, that code will have to set the cursor. And there are two major changes:

```
MainForm.Redisplay  PROCEDURE(String[] requests)
request               STRING
  code
  if self.InvokeRequired
    self.Invoke(new StringArrayDelegate(self.ReDisplay), requests)
```

```
    return
  end
  SyncLock(self.statelock)
    self.listBox1.items.Clear()
    foreach request in requests
      self.listbox1.items.Add(request)
    END
  END
```

There are two things going on here: there's some "invoke" stuff, and there's a SyncLock statement. Let's consider them one at a time.

The mysteries of Invoke

MainForm is derived from System.Windows.Forms.Form, and as such it has some properties and methods that are useful for managing cross-thread updating. One of these properties is InvokeRequired, which is true whenever MainForm's code executes on a thread other than the thread on which the class was created. That is going to be the case whenever RequestMonitorThread calls Redisplay. Now, if InvokeRequired is true the code calls the Invoke method, which is also part of System.Windows.Forms.Form. This method essentially looks around for the thread on which MainForm was created and switches context to that thread. The code then creates a new delegate (think callback, think function pointer) based on Redisplay and calls that delegate. Essentially this is a thread switch and a single recursive call. The return statement immediately following ensures that no other Redisplay code executes on the background thread.

This bit of code guarantees that everything after the InvokeRequired test will be running on the UI thread. And that's where the SyncLock statement comes in. SyncLock is new to Clarion# and seems to be analogous to the C# lock statement. The parameter to SyncLock can, I assume, be any object; in this case StateLock serves simply as a token. As long as one thread has hold of the object via SyncLock no other thread can execute that same block of code. The chances of contention are low in this case, since the user would have to trigger a Redisplay at the same time as the background thread triggers a Redisplay, but it could happen.

Invoke is a synchronous method; that is, the method doesn't return until the method it calls finishes executing. You can also use BeginInvoke, which is asynchronous; in this case the time it takes to actually update the listbox is very short, so either Invoke or BeginInvoke will work just fine.

When the service won't serve

There are a great many ways to improve this class, but one thing that definitely needs to be handled is a failure of the web service. As currently written any failure of the service, such as an invalid method call or a complete failure to connect with the service will result in an unhandled exception being thrown and the client application crashing. To take care of this I need to handle the exception in code. Here's an updated snippet from the RequestMonitorThread method; see the source code for all the places where Try/Catch was implemented:

```
TRY
  requests = self.svc.GetAllRequests()
catch (Exception er)
  self.listbox1.items.Add('Autorefresh error: ' & er.ToString)
```

```
END
```

Now if there's a problem with the service all that happens is the error message gets appended to the list box. That's not a perfect solution, but it will at least give some indication that something's gone wrong.

Bug fixes and random thoughts

As I was wrapping up the code I did some intensive testing which basically involved putting up two copies of the client app side by side, with different names, and madly, randomly clicking buttons on both as quickly as possible. And I induced a crash more than once. Usually these were on the Invoke statement, but eventually (after changing back to BeginInvoke) I got a clear error message: I was trying to add a null string to the list box. So I added a null test and the error went away.

But I still wish I knew just how I was getting that null string in the first place. If you see the reason, let me know.

The other thing that struck me was the oddity of having the web service source code on the web server where it is automatically recompiled any time that source changes. I manage my web server (a Windows 2008 Server box located somewhere in the U.S. - honestly I forget where) via Remote Desktop Connection. So there were times when I had RDC running on one monitor, and the podcast client app running on another, and I'd make a change in the server code that would have a visible effect on the client app as picked up the latest data via its background thread.

Weird.

Summary

It seems like a long way from that first idea of a "pondcasting" application to the web service now running on the ClarionMag backup server and the multi-threaded client application I described in this article. But in fact it's a short trip: creating and consuming web services can be pretty simple stuff. And the client app I described here isn't really unique, either. With a little work it (and the service) could be adapted to use as a chat client. Any number of improvements come to mind, such as putting all service calls on their own threads, and restricting or queuing button presses.

I hope you find web services as interesting and useful as I have.

Download the source

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

Creating a Real-World Web Service

by Dave Harms

Published 2008-12-16

In Part 1 of this series I demonstrated the basics of creating and using web services, and I alluded to my inspiration for a web service and accompanying client application to help with the Clarion Folk podcast:

> I started thinking about web services after the first Clarion Folk podcast. Stu Andrews, who hosts and records the podcast, is in Australia, as are Kim Davies and Ken Wragg (although they're not all in the same part of the country). But Bruce Johnson is in South Africa and I'm in Canada, and at those distances the lag time can be significant. Ken may be saying something to which I want to respond, and Bruce may be thinking the same thing. I hear Ken stop, I start talking, Bruce starts talking, and a couple of seconds later I hear Bruce talking. And Bruce hears me, and we both stop, and then there's a long pause while we wait to see who's going to talk next. And then we both start talking again at the same time, and so on.
>
> I had this bright idea that if we could all somehow see who wanted to talk next we'd be less likely to talk over each other. I imagined each of us having a small desktop application with which we could log our "talk requests" and see everyone else's talk requests.

Figure 1 shows the Clarion# desktop application that resulted from this project (and which I'll describe in Part 3). As you can see, this app has the following functionality:

- Register a request to talk (repeatedly, if you wish, hence the ! marks)
- Cancel a request to talk
- Force a refresh of the display (but this happens automatically as well)
- Remove the topmost request (admin function)
- Remove all requests (admin function)

One caveat: we haven't had a chance to use it in a podcast yet so I don't know how well it will really work, but it was a fun project nonetheless.

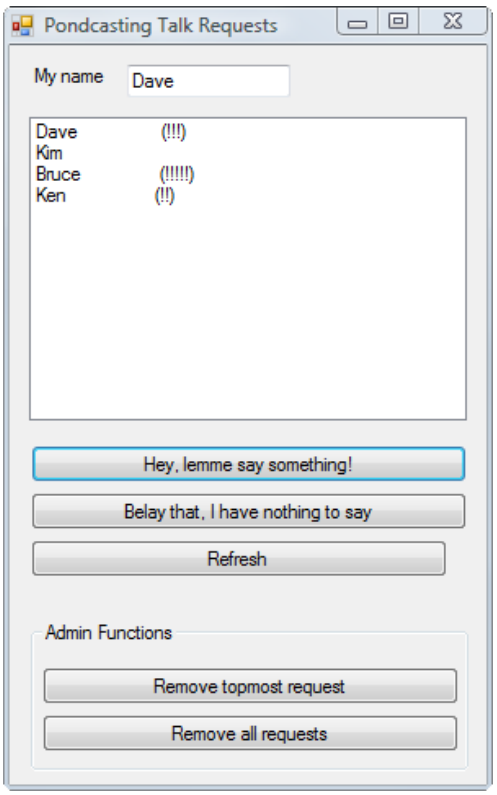


Figure 1. The Pondcast client app

Design choices

I can think of two ways to make this kind of application work. One is to set up a peer to peer network, and I don't know anything about peer to peer networks, so I'm not sure how I'd approach that (and in any case I expect I'd need some sort of organizing server to look up one peer with another).

The other approach is to have a central repository of information, and each of the applications acting as a client. Client-server sounds a lot more familiar to a database guy like me. In fact, one way would be to set up a database and give the client apps direct access to that database. But that means configuring database access on all the client machines, and dealing with potential firewall issues, all of which sounds like a huge hassle.

A web service seemed like a much better solution. I wouldn't have to worry about opening up access to the database, configuring client file drivers, or have the hassle of client-side firewall issues since just about everyone allows HTTP. And I have a publicly-accessible Windows 2008 server (it hosts the ClarionMag backup site) so I set to work creating a web service and a client application.

Solutions and projects

Both Clarion 7 and Clarion# make it easy to have one solution containing a number of projects. I knew I would want at least three projects in my solution: one for the web service, one for some unit tests (so I could have reasonable confidence in the web service code), and one for the WinForms client application. Actually I ended up with four, as I created two versions of the client.

The kind of podcasting Stu does has been dubbed "pondcasting" since it involves making calls "over the pond." So Pondcast seemed like a suitable name for the overall solution. Figure 2 shows the directory tree.

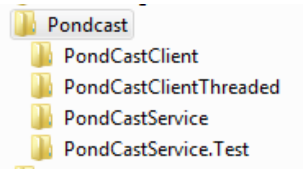


Figure 2. The Pondcast directories

Here's a little trick. I wanted an empty solution to contain all of the sub-projects, but Clarion# doesn't have a way of directly creating empty solutions. So I simply created a Pondcast solution and deleted the resulting Pondcast project from that solution. I then right-clicked on the solution in the Solution explorer and chose Add | New Project for each of the sub-projects. I won't go into the details of creating each of the projects, however. The basic concept is the same as in Part 1.

Here's another trick: In build 4125 if you right-click on a project in the Solution explorer and choose Run Project, you'll run the project under the debugger, which may not be what you want. Instead, right-click and choose Set as Startup Project. That way when you press the Run button this project will be started (after any needed compile takes place).

The pondcast service

I declared my Service class in App_Code\service.cln as follows:

```
Service          CLASS(System.Web.Services.WebService),TYPE,NETCLASS,PARTIAL
AddTalkRequest       procedure(string name),public
GetAllRequests       procedure,String[],PUBLIC
PopTalkRequest       procedure,public
RemoveTalkRequest    procedure(string name),public
Reset                PROCEDURE,public
                 END
```

Of course, that class won't fly as a web service; it needs attributes! Here's the same class decorated with WebService and WebMethod attributes, and with a few additional support methods:

```
   [WebService(Description='Support for pondcasting conversations.')]
Service          CLASS(System.Web.Services.WebService),TYPE,NETCLASS,PARTIAL
CloseFile          PROCEDURE
Construct          PROCEDURE


                 [WebMethod(Description='Add a talk request')]
AddTalkRequest       procedure(string name),public
                 [WebMethod(Description='Get all talk requests')]
GetAllRequests       procedure,String[],PUBLIC
GetRecordByName      procedure(string name),boolean
OpenFile           PROCEDURE
                 [WebMethod(Description='Remove the topmost talk request')]
PopTalkRequest       procedure,public
                 [WebMethod(Description='Remove a talk request')]
RemoveTalkRequest    procedure(string name),public
                 [WebMethod(Description='Clear all talk requests')]
Reset                PROCEDURE,public
LastError          STRING
```

```
END
```

Storage

Clearly I needed some way to store talk requests so they could be distributed back to the client applications. My first thought was to use a queue, and I did some tests. But I discovered that while I could add a record to the queue, that data was never there when I asked for it.

My problem was that I was thinking in terms of server lifetime, not web service lifetime. As I said in Part 1, web services run in the context of a web server, typically IIS, or perhaps Apache, or during development the web server that comes with Clarion#. And while the web server stays running between requests, the same is not true of the web service. Each time you call the web *service* the web *server* creates a new instance of that service. So each time I asked for the list of talk requests I was talking to a brand new instance of the service with a newly-create queue.

After I understood what was happening I decided it made a lot more sense to go with a database, and the simplest solution there was a TPS file:

```
FName           clastring(500)


TalkRequest     file,driver('TOPSPEED'),PRE(TKR),name(FName),create
IDKey           key(tkr:ID),PRIMARY
NameKey         key(tkr:Name),nocase
record          RECORD
ID              signed
Name            CLASTRING(20)
Requests        signed
                end
              END
```

This file has a CREATE attribute so the service can make it as necessary, but that's one thing I may yet change. It seems less risky to provide an already-created file than to give the app the chance of wiping out existing data.

The Construct method sets up the file name. If you don't specify an explicit path for the file, the web server will attempt to create it in the default path. In the case of IIS, this is the going to be something like \Windows\System32\inetsrv and you won't have permission to create the file, and you wouldn't want to create the file there even if the web service did have permission.

The constructor code builds the file name based on the physical path of the web service itself. Context is the application context, and Request is the object representing the request made by the client application (which could be a browser, as in the initial test in Part 1, or more likely a custom application).

```
Service.Construct       PROCEDURE
  code
  FName = self.Context.Request.PhysicalApplicationPath |
   & 'TalkRequest.tps'
```

The CloseFile method is straightforward, and is only there to provide a parallel for the OpenFile method:

```
Service.CloseFile       PROCEDURE
  code
  close(TalkRequest)
```

The AddTalkRequest method opens the file, attempts to get the user's record, and if found updates the request count. If no record is found then AddTalkRequest adds a new record. I'm a little rusty on my autoincrement code but this implementation seems to work.

```
Service.AddTalkRequest   procedure(string name)
id                       long
   code
   self.OpenFile
   if self.GetRecordByName(name)
      TKR:requests += 1
      put(TalkRequest)
   ELSE
      loop 5 times ! max number of tries to add
         clear(TKR:ID,1)
         set(TKR:IDKey,TKR:IDKey)
         previous(TalkRequest)
         if errorcode()
            id = 0
         ELSE
            id = tkr:id
         end
         clear(tkr:record)
         tkr:Name = name
         tkr:Requests = 1
         tkr:ID = id + 1
         add(TalkRequest)
         if ~errorcode()
            break
         ELSE
            self.LastError = error() & ' ' & errorcode()
         END
      END
   END
   self.CloseFile
```

GetAllRequests loops through the file and builds a string array with one element for each record. Since this is .NET code, the arrays are all zero based (that change was made to the language to prevent confusion, since any non-Clarion .NET objects are going to expect zero-based arrays). If a user has more than one request, an exclamation mark is added to the text for each additional request. This lets users emphasize how badly they want to talk.

Finally, if no requests are found the method returns the message 'No talk requests at this time'. There's also a little bit of error reporting going on.

```
Service.GetAllRequests   procedure
s                        String[]
x                        LONG
```

```
    code
    self.OpenFile
    s = null
    if records(TalkRequest)
       s = new String[records(TalkRequest)] ! + 1] ! +1 debug
       x = 0 ! 1 debug
       clear(TKR:record)
       set(TKR:IDKey,TKR:idkey)
       LOOP
          next(TalkRequest)
          if errorcode() then break.
          if TKR:requests > 1
             s[x] = TKR:name & ' (' & all('!',TKR:Requests-1) & ')'
          ELSE
             s[x] = TKR:name
          END
          x += 1
       END
    end
    self.CloseFile
    if s = null
       s = new String[1]
       if self.LastError <> null and self.lasterror <> ''
          s[0] = self.LastError
       ELSE
          s[0] = 'No talk requests at this time'
       END
    end
    return s
```

GetRecordByName is a utility method that attempts to retrieve the record corresponding to the user's name:

```
Service.GetRecordByName  procedure(string name)
    code
    if records(TalkRequest)
       clear(TKR:record)
       TKR:name = name
       set(TKR:namekey,TKR:NameKey)
       next(TalkRequest)
       if ~errorcode()  and TKR:name = name
          return true
       END
    END
    return FALSE
```

PopTalkRequest simply gets the oldest record in the table and deletes it:

```
Service.PopTalkRequest   procedure
  code
  self.OpenFile
  if records(TalkRequest)
    clear(TKR:record)
    set(TKR:IDKey,TKR:idkey)
    next(TalkRequest)
    delete(TalkRequest)
  END
  self.CloseFile
```

OpenFile is a utility method that attempts to open the file in shared mode. If the file cannot be opened it is recreated, but I'm having second thoughts about this as I've had a few situations where it appears the file has been recreated due to some unknown error on the SHARE attempt. At the very least the file should probably only be created if it is missing or if the format is bad.

```
Service.OpenFile PROCEDURE
  code
  close(TalkRequest)
  loop 5 times
    share(TalkRequest)
    if errorcode()
      close(TalkRequest)
      create(TalkRequest)
      if ERRORCODE()
        self.LastError = 'Openfile error: ' & error() & ' ' & errorcode()
      END
      share(TalkRequest)
    ELSE
      break
    END
    System.Threading.thread.Sleep(100)
  END
```

RemoveTalkRequests tries to get the user's record; if found the record is deleted.

```
Service.RemoveTalkRequest       procedure(string name)

  code
  self.OpenFile
  if self.GetRecordByName(name)
    delete(TalkRequest)
  END
  self.CloseFile
```

Reset's only job is to close the file, open it in exclusive access mode and empty all records:

```
Service.Reset    PROCEDURE
  code
  close(TalkRequest)
  open(TalkRequest,18)
  empty(TalkRequest)
  close(TalkRequest)
```

Now how each web service method has to open and close the file, since the Service object is recreated for each request. Creating the object anew each time may seem like an expensive proposition, but in fact it's not that big a deal.

Running the service

If the web service is the only project in the solution, just press the Run button; if it's one of several projects make sure it's the startup project. Right-clicked on the project in the Solution explorer and set that option if necessary. Then press Run.

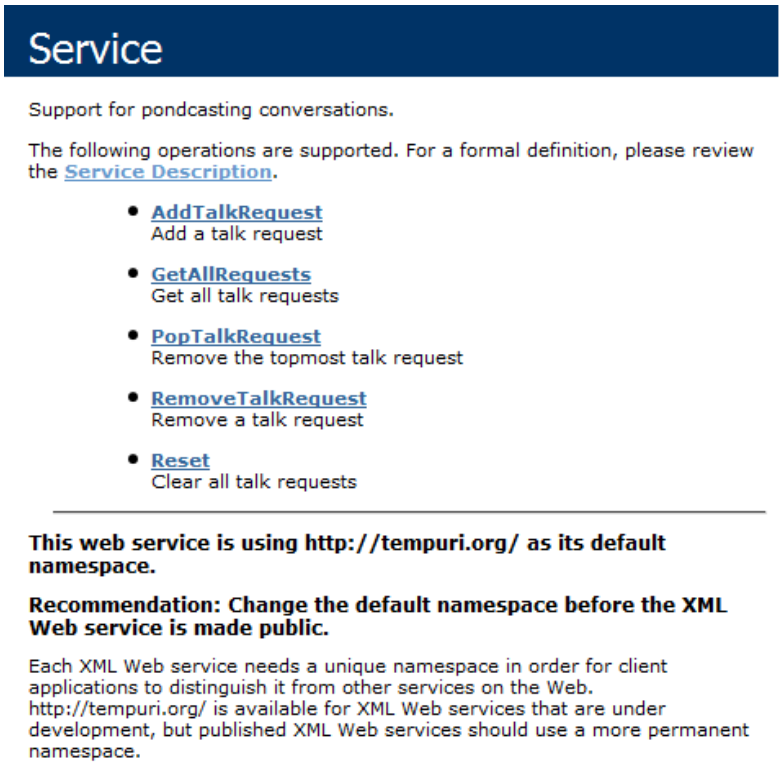


Figure 3. The web service as seen in the browser

Note the recommendation in Figure 3 about changing the web service's default namespace. The place you do this is in the service class's WebService attribute. The Description property is already assigned:

```
[WebService(Description='Support for pondcasting conversations.')]
```

Going by the C# samples out there, the WebService attribute should have both Description and Namespace properties, like this:

```
[WebService(Description='Support for pondcasting conversations.',|
  Namespace='http://services.clarionmag.com/pondcastdemo')]
```

Only that code won't compile, because the compiler sees Namespace as a reserved word. You need to escape Namespace with a pair of ampersands, like this:

```
[WebService(Description='Support for pondcasting conversations.',|
  @@Namespace='http://services.clarionmag.com/pondcastdemo')]
```

Compile and run the web service again, and the default namespace warning goes away.

The unit tests

I created another project under the Pondcast solution, this one for unit testing. I've described unit testing in some detail in previous articles. The following tests give me some baseline assurance that the web service is working as expected:

```
                MEMBER('')

                NAMESPACE(PondCastService.Test)
                MAP
                END

  using NUnit.Framework

  [TestFixture(Description='Test PondCastService')]
TestService     class,public
                [Test]
TestAdd         procedure,public
                [Test]
TestPop         procedure,public
                [Test]
TestPopThenAdd  procedure,public
                [Test]
TestReset       procedure,public
                [Test]
TestSortOrder   procedure,public
              end

TestService.TestAdd PROCEDURE
svc             PondCastSvc.Service
expected        STRING[]
result          STRING[]
  code
  expected = new String[5]
  svc = new PondCastSvc.Service
  svc.Reset()
  svc.AddTalkRequest('Dave')
  result = svc.GetAllRequests()
  expected[0] = 'Dave             '
  NUnit.Framework.Assert.AreEqual(expected[0],result[0],|
```

```
      'Failure to get expected string back')
   svc.AddTalkRequest('Dave')
   result = svc.GetAllRequests()
   expected[0] = 'Dave              (!)'
   NUnit.Framework.Assert.AreEqual(expected[0],result[0],|
      'Failure to get expected string back')
   svc.AddTalkRequest('Stu')
   result = svc.GetAllRequests()
   expected[0] = 'Dave              (!)'
   expected[1] = 'Stu               '
   NUnit.Framework.Assert.AreEqual(expected[0],result[0],|
      'Failure to get expected string back')
   NUnit.Framework.Assert.AreEqual(expected[1],result[1],|
      'Failure to get expected string back')


TestService.TestPop PROCEDURE
svc                 PondCastSvc.Service
expected              STRING[]
result              STRING[]
   code
   expected = new String[5]
   svc = new PondCastSvc.Service
   svc.Reset()
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Stu')
   svc.PopTalkRequest()
   result = svc.GetAllRequests()
   expected[0] = 'Stu               '
   NUnit.Framework.Assert.AreEqual(expected[0],result[0],|
      'Failure to get expected string back')
   NUnit.Framework.Assert.AreEqual(1,result.Length,|
      'wrong size array returned')

TestService.TestPopThenAdd PROCEDURE
svc                 PondCastSvc.Service
expected              STRING[]
result              STRING[]
   code
   expected = new String[5]
   svc = new PondCastSvc.Service
   svc.Reset()
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Stu')
```

```
    svc.AddTalkRequest('Stu')
    svc.AddTalkRequest('Stu')
    svc.PopTalkRequest()
    svc.AddTalkRequest('Dave')
    result = svc.GetAllRequests()
    expected[0] = 'Stu               (!!)'
    expected[1] = 'Dave               '
    NUnit.Framework.Assert.AreEqual(2,result.Length,|
       'wrong size array returned')
    NUnit.Framework.Assert.AreEqual(expected[0],result[0],|
       'Failure to get expected string back - array index 0')
    NUnit.Framework.Assert.AreEqual(expected[1],result[1],|
       'Failure to get expected string back - array index 1')

TestService.TestSortOrder PROCEDURE
svc                 PondCastSvc.Service
expected              STRING[]
result                  STRING[]
x                       LONG

    code
    expected = new String[5]
    svc = new PondCastSvc.Service
    svc.Reset()
    svc.AddTalkRequest('Dave1')
    svc.AddTalkRequest('Dave2')
    svc.AddTalkRequest('Dave3')
    svc.AddTalkRequest('Stu')
    svc.AddTalkRequest('Stu')
    svc.PopTalkRequest()
    svc.AddTalkRequest('Dave')
    result = svc.GetAllRequests()
    expected[0] = 'Dave2              '
    expected[1] = 'Dave3              '
    expected[2] = 'Stu               (!)'
    expected[3] = 'Dave               '
    NUnit.Framework.Assert.AreEqual(4,result.Length,|
       'wrong size array returned')
    loop x = 0 to 3
       NUnit.Framework.Assert.AreEqual(expected[x],result[x],|
       'Failure to get expected string back - array index ' & x)
    END



TestService.TestReset       PROCEDURE
```

```
svc                PondCastSvc.Service
result             STRING[]
   code
   svc = new PondCastSvc.Service
   svc.Reset()
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Dave')
   svc.AddTalkRequest('Stu')
   svc.Reset
   result = svc.GetAllRequests()
   NUnit.Framework.Assert.AreEqual(1,result.Length,|
      'wrong size array returned')
   NUnit.Framework.Assert.AreEqual('No talk requests at this time',result[0],|
      'Failure to get expected string back')
```

I've become a big fan of unit testing and test-driven development. Although I'm not employing it as extensively as I'd like to (at least not yet), there have been a number of occasions where I've had to go back to a project after some time has passed. It's a great comfort to be able to run a suite of tests after making modifications, particularly when the code is no longer fresh in my mind.

Deploying the service

I've spent most of the last ten years running Linux web servers, not Windows web servers, so I'm still getting up to speed on IIS. Nonetheless I found the web service easy to deploy under IIS 7. I copied the entire project to a directory on the server and I created a new web site for the service. I set the web service directory as the web site directory, configured the bindings (IP address and port number), made sure I had a corresponding DNS entry, and I was off to the races. Attempting to browse the service from within the IIS Manager didn't work, because that URL pointed to the root directory. But all I had to do was append Service.asmx?wsdl to the URL and all was well.

Once I had the actual URL to use I deleted the old web service reference and created a new web service reference using that URL. I kept the reference name the same so none of my source code needed changing.

If you'd like to try connecting to the service with your browser, with the client app (described in Part 3) or with your own app, use the following URL:

http://services.clarionmag.com:4820/pondcastdemo/service.asmx?wsdl

Summary

Web services are a popular and easy way to expose programming APIs to the Internet. Clarion# hides much of the complexity of web services; you don't have to mess around with SOAP or otherwise get your hands dirty with HTML or XML. You simply create a web service project and define the service as a regular Clarion# class with a few attributes thrown in to let the web server know how to handle the class. Using the web service is even easier; just add a web reference to your application, create an instance of the service, and call its methods.

Now it's time to develop a more complex client application.

Download the source

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

Writing and Using Web Services in Clarion#

by Dave Harms

Published 2008-12-15

Clarion# has expanded Clarion development beyond the desktop, into mobile and web development. Clarion# web development is further subdividable into ASP.NET web sites and web services.

In this series of articles I'll cover both creating a web service and using that web service in a WinForms application. If you don't have a need to create a web service you may want to skip straight to the third installment, but I encourage you to read all the material so you have a solid grounding in the technology. And you may realize, as you read, that you do in fact want to create your own web service(s).

But what's *is* a web service, and why would you want to use or create one?

In short, web services are procedure calls that operate across the Internet, via HTTP. The information that's sent back and forth is packaged up as XML, but thankfully in Clarion# that's all abstracted and you don't have to deal with XML documents directly. Web services make it possible to get information from remote servers into a .NET application (and vice versa) without having to set up a database connection or parse HTML pages.

Web services are available for just about any kind of information you can imagine, including weather, foreign exchange, movie databases, travel information, geolocation, stock quotes, mathematics, country code lookups, web search, translation... well, you get the idea. For a fairly long list of available web services have a look at the seekda! web services portal.

The inspiration

I started thinking about web services after the first Clarion Folk podcast. Stu Andrews, who hosts and records the podcast, is in Australia, as are Kim Davies and Ken Wragg (although they're not all in the same part of the country). But Bruce Johnson is in South Africa and I'm in Canada, and at those distances the lag time can be significant. Ken may be saying something to which I want to respond, and Bruce may be thinking the same thing. I hear Ken stop, I start talking, Bruce starts talking, and a couple of seconds later I hear Bruce talking. And Bruce hears me, and we both stop, and then there's a long pause while we wait to see who's going to talk next. And then we both start talking again at the same time, and so on.

I had this bright idea that if we could all somehow see who wanted to talk next we'd be less likely to talk over each other. I imagined each of us having a small desktop application with which we could log our "talk requests" and see everyone else's talk requests (Figure 1).

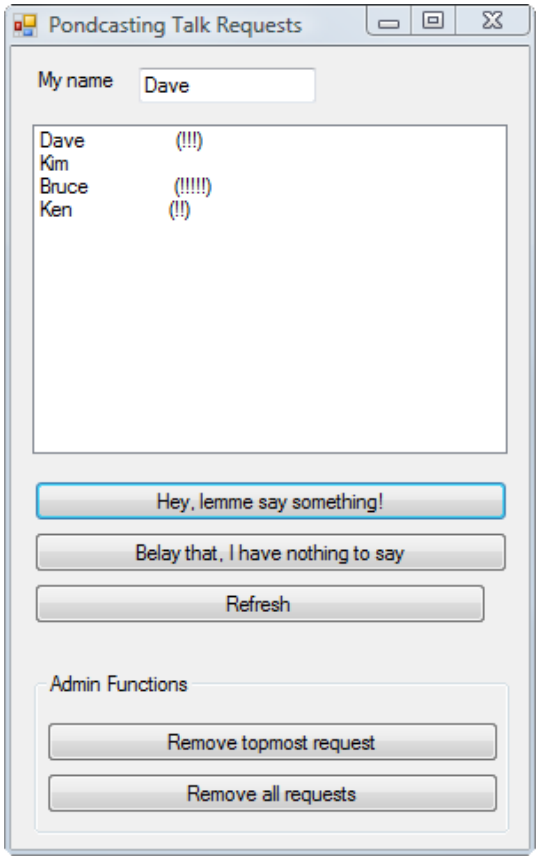


Figure 1. The "talk requests" application

But how to implement something like this? I could make this a database application and give everyone access to a database on my server. But that seemed like overkill, and would require everyone to set up a database connection and possibly open some firewall ports to allow that connection.

But a web service would solve the problem nicely. No need to set up database connections, and no need to worry about firewalls since the web services use HTTP which is allowed just about anywhere.

I'll get to that particular web service and the client app in parts two and three of this series; first I want to show how easy it is to create and consume web services

A simple example

Creating a web service in Clarion# is trivial. Choose File | New Solution. As in Figure 2, choose ASP.NET and then ASP.NET Web Service.

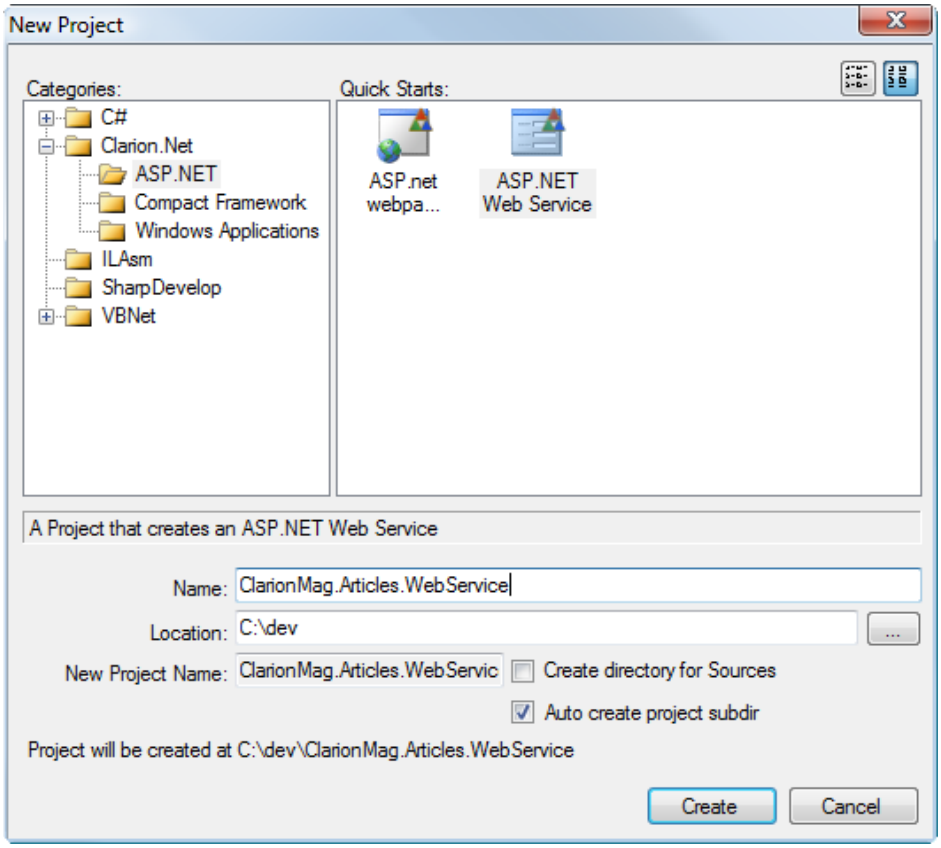


Figure 2. Creating a web service

After you complete the dialog you'll see another dialog, shown in Figure 3. You can leave all the settings at the defaults. If you choose IIS you won't be able to debug your web service as Clarion# is not able to attach to the IIS process.

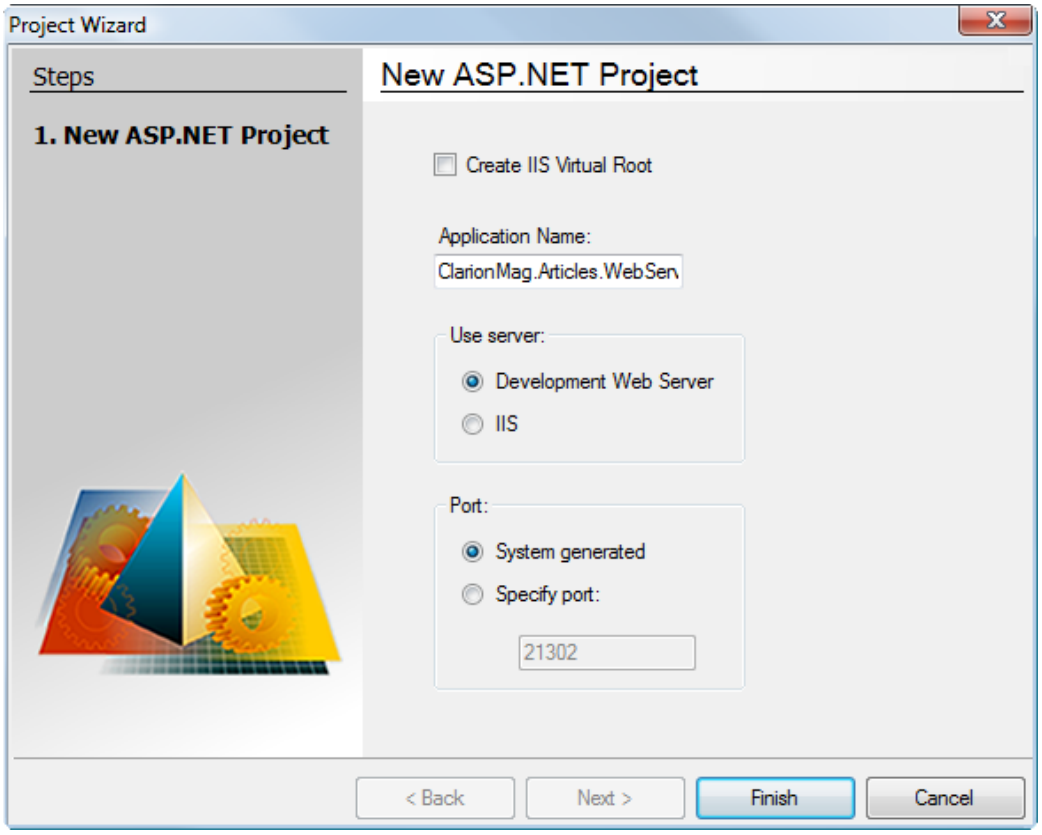
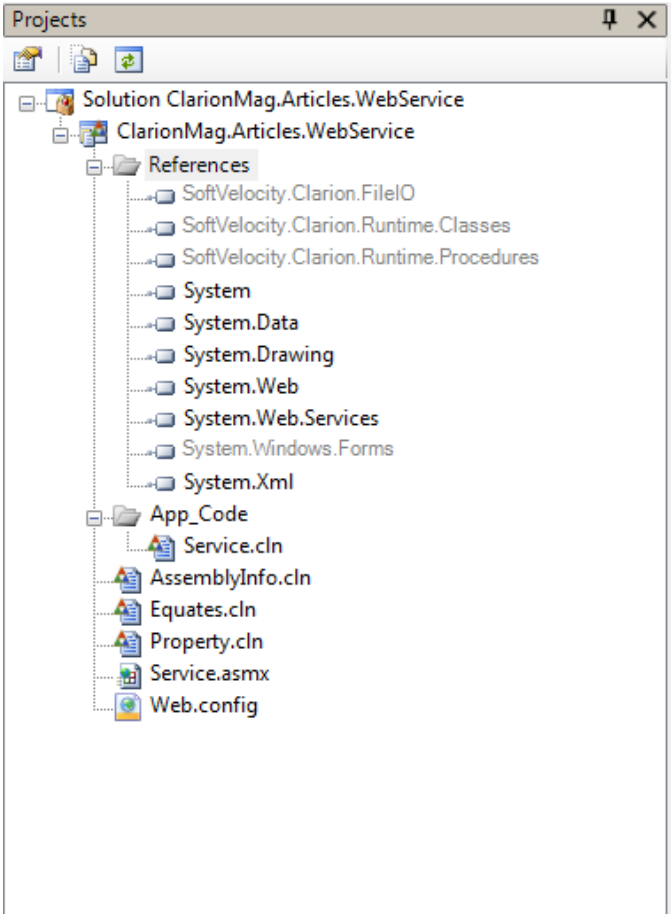


Figure 3. Setting the web service options

Figure 4 shows the web service's files.



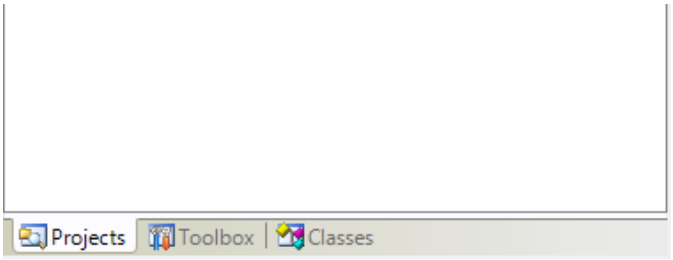


Figure 4. The web service solution

All of the files in Figure 4 are pretty much standard issue, with the exception of App_Code\service.cln. This file contains the actual web service source code:

```
[WebService(Description='This is my WebService description.')]
MyService CLASS(System.Web.Services.WebService),TYPE,NETCLASS,PARTIAL
[WebMethod(Description='Return the result of a + b')]
Add          PROCEDURE(SIGNED a, SIGNED b),SIGNED,PUBLIC
[WebMethod(Description='Return a Hello world string.')]
SayHello     PROCEDURE(),System.String,PUBLIC
[WebMethod(Description='Return Hello name.')]
SayHelloName  PROCEDURE(System.String name),System.String,PUBLIC
 END

MyService.Add PROCEDURE(SIGNED a, SIGNED b)
 CODE
   RETURN a + b

MyService.SayHello PROCEDURE()
 CODE
   RETURN 'Hello World'

MyService.SayHelloName PROCEDURE(System.String name)
 CODE
   RETURN 'Hello '& name
```

MyService contains three methods: Add, SayHello and SayHelloName. I'll show you how to use these methods later, but for now just keep in mind that this is only an example. You can change these methods, delete them, or add new methods. There's nothing magic about these three.

There is a bit of magic, however, in the attributes, those funny looking things in square brackets. The [WebService] attribute indicates the following class is a web service, and the [WebMethod] attribute indicates the following method is a method within that web service.

But just who or what is reading these attributes?

Services and servers

Web services don't execute all on their own; instead, they're sort of like a mini web site. And just like web sites, web services run in the context of a web server. In a production system that server will probably be IIS; in development you're

probably going to use the SV development web server included with Clarion#. In any case it's the server that loads up the web service class, reads the attributes and knows how to handle the various methods. So let's see how that actually works.

To test your new web service simply press the green "compile and run" button in the IDE or press Ctrl-F5. When you do this for the first time you'll probably get this error (at least as of build 4125):

> The CodeDom provider type "ClarionCompiler.CodeProvider, SoftVelocity.Clarion.ClarionCodeProvider, PublicKeyToken=4aca31c24e330653" could not be located. (ASPCONFIG) - C:\dev\ClarionMag.Articles.WebService\web.config:45

If you look in web.config for the source of the error you'll see this (line breaks added):

```
<system.codedom>
 <compilers>
 <compiler language="Clarion.Net;Clarion#;Cla#;Clarion" extension=".cln;
   .inn;.inc;.int" type="ClarionCompiler.CodeProvider,
   SoftVelocity.Clarion.ClarionCodeProvider,
   PublicKeyToken=4aca31c24e330653" />
 </compilers>
 </system.codedom>
 </configuration>
```

The IDE compiles your code using Microsoft's aspnet_compiler.exe. Microsoft doesn't ship a Clarion# compiler, so the above entry defines the assembly containing the necessary compiler. And the assembly containing that compiler should be in the bin subdirectory. But the IDE (as of build 4125) doesn't create the bin subdirectory, so you have to do that. Right-click on the Project, choose Add | New Folder and create the bin subdirectory. Actually you'll need two files in your bin subdirectory: copy the following from the Program Files\Clarion.Net\bin directory

- SoftVelocity.Clarion.ClarionCodeProvider.dll
- SoftVelocity.Clarion.CompilerMessages.dll

Press the run button again. This time you should see the page shown in Figure 5.

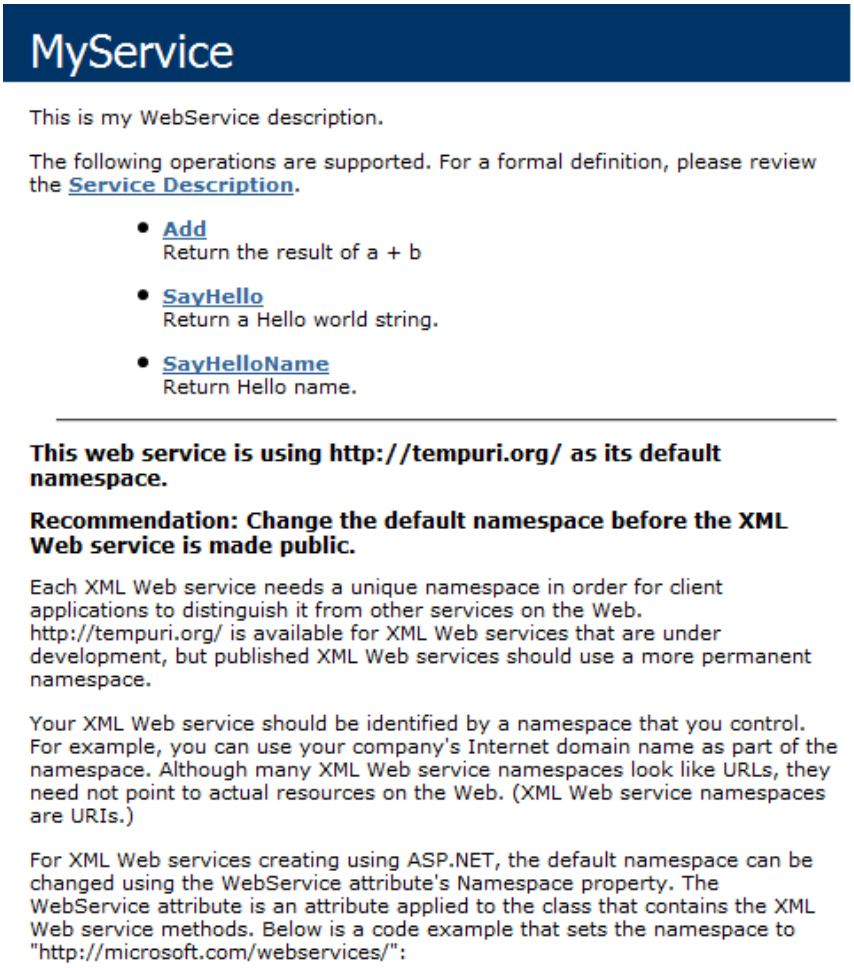


Figure 5. The web service information page

There's an important note on the need to change the namespace; I'll show you how to do that in the next installment.

Click on the link for the Add method. You'll see the page in Figure 6.

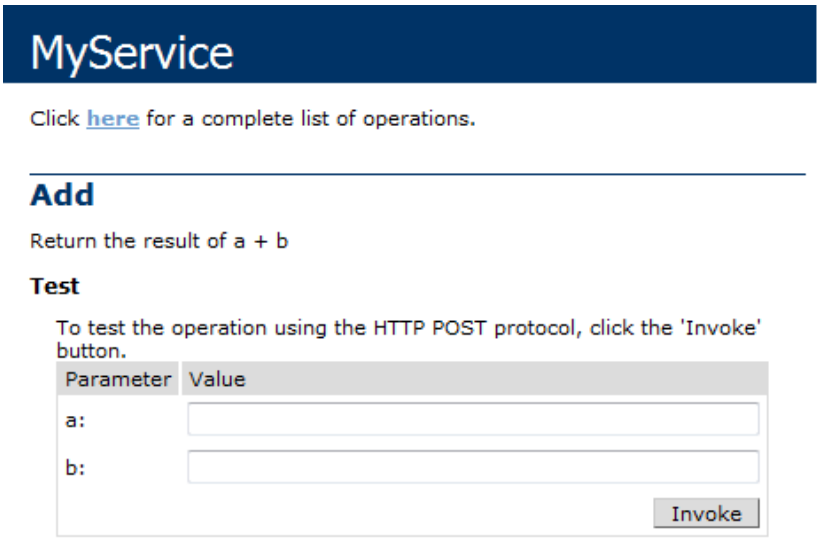


Figure 6. The Add method

Enter two numbers, say 45 and 123. Press Invoke to send the procedure request to the server and you'll get back a page that looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<int xmlns="http://tempuri.org/">168</int>
```

Make a note of the URL you used to get to the main web service page. In my case it's

```
http://localhost:21738/ClarionMag.Articles.WebService/Service.asmx
```

That's how easy it is to use a web service from a web page. But most likely you want to use it from within a Clarion# application.

Using a web service

For the sake of simplicity I'm going to create a console application to test my web service. This is just a simple command-line application; in the third installment in this series I'll provide an example of a Clarion# WinForms application that uses a web service.

To create the console app go to File | New Solution. Select Clarion.NET | Windows Applications | Console Application. I called mine ClarionMag.Articles.WebService.TestWithConsole. This application doesn't contain much, just the standard AssemblyInfo.cln file and a Program.cln file with the following code:

```
CODE
System.Console.WriteLine('Hello World')
System.Console.ReadKey()
```

That code will get changed to some code that calls the web service and gets back a result.

It's easy to add a web service to an app. Just right-click on References in the Solution explorer and choose Add Web Reference. You can see the dialog in Figure 7.

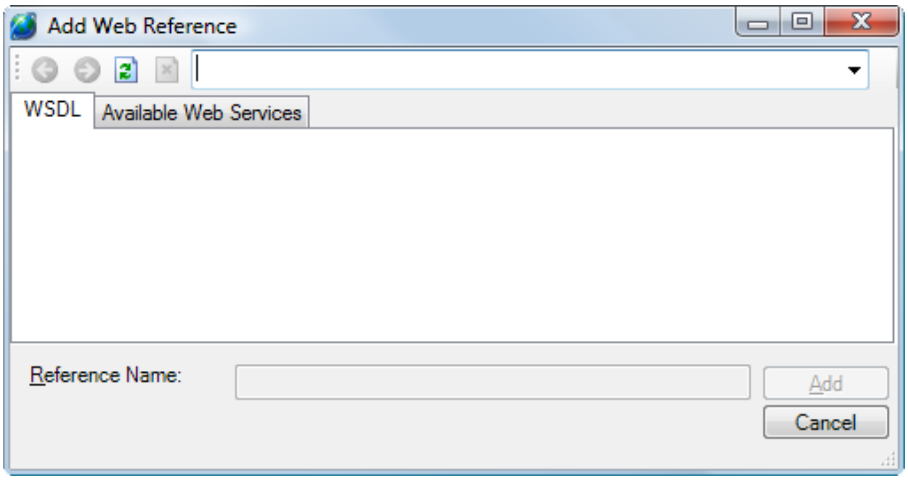


Figure 7. The web references dialog

Remember that URL you made a note of? In Figure 8 I've pasted the URL into the entry field and, most importantly, I pressed Enter when I was done; if there's another way to trigger the actual lookup of the web service I haven't found it.

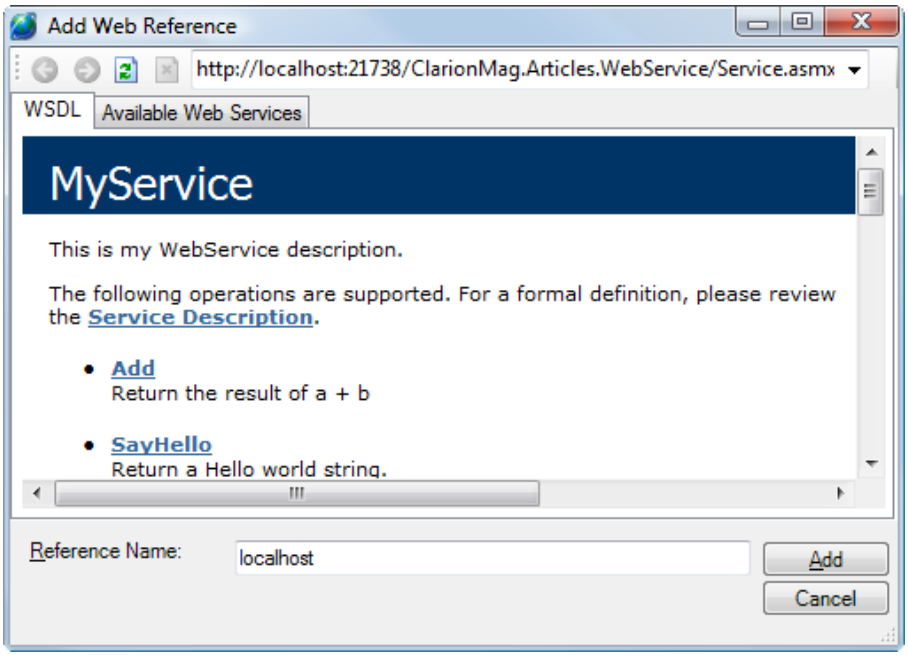


Figure 8. The web reference after loading the URL

Figure 8 shows the same page as in Figure 5. Click on the Available Web Services tab (Figure 9). For some reason two different bindings show up for the service; I expect that's a bug. In any case, it doesn't seem to be one that matters. Just click on the service name, as shown in Figure 9. If you wish you can change the reference name, which is the top level name space. If you leave it as localhost then you'll call the Add method using the code localhost.MyService.Add(3,4), and so forth.

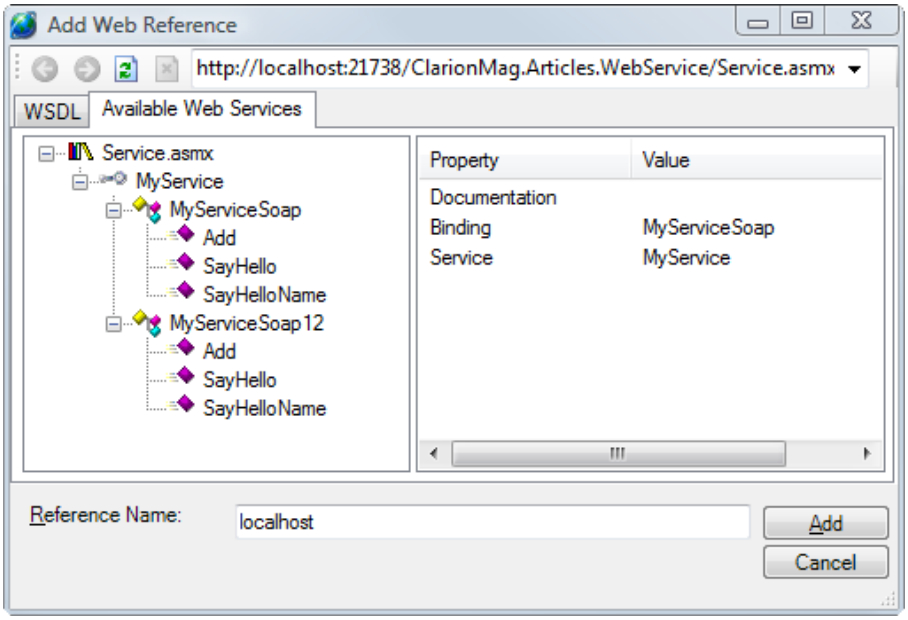


Figure 9. Adding the web reference

Figure 10 shows the web reference files added to the project. Feel free to explore these, but I don't advise you try changing them. If something changes in the web service simply right-click on localhost (or whatever you've named the web reference) and choose Refresh Web Reference.

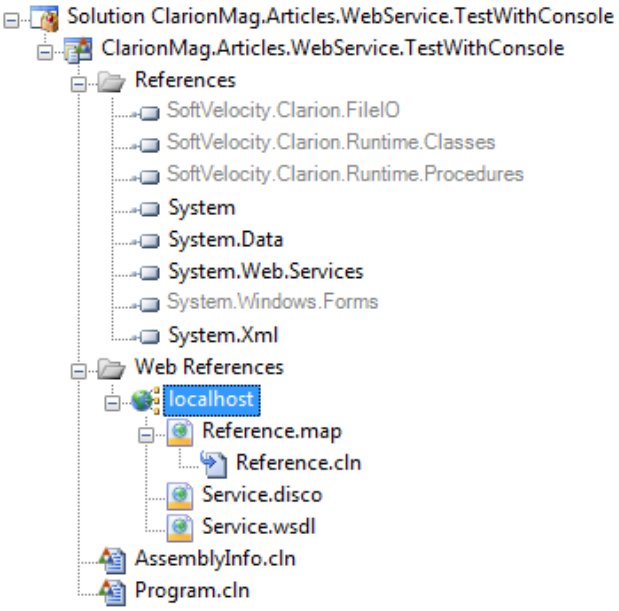


Figure 10. The web reference in the project

Now you're ready to actually execute some web service code. Go to Program.cln and, immediately after the MAP/ END statement, change the code as follows:

```
svc             localhost.MyService

  CODE
  svc = new localhost.MyService()
  System.Console.WriteLine('5 + 7 = ' & svc.Add(5,7))
  System.Console.WriteLine('Press any key...')
  System.Console.ReadKey()
```

The svc object is an instance of the web service, and once you create that instance you call it just as you would any other class. All the code needed to support calling the service and getting the result back is automatically supplied and executed as needed. Here's the output from that class

```
5 + 7 = 12
Press any key...
```

And that's it! All of this takes far longer to write about than to do. Just for fun I went through the process again, creating a new web service and a console to test the service. At a fairly leisurely pace I was done in under three minutes.

Now that you have the idea of how web services work, it's time to look at a real-world web service.

Download the source

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 Tuesday, December 16, 2008 by Leo Lakota

Yes.....!

Add a comment

Clarion Magazine

C7 AppGen Beta IV

by Dave Harms

Published 2008-12-12

SoftVelocity has released a fourth C7 AppGen build to third party vendors for compatibility testing. There are a number of bug fixes, as usual, as well as several important usability improvements. As with the previous release the initial response to the build has been a little on the quiet side, although some new bugs have been reported.

One of the annoyances noted in previous builds was that you couldn't drag column widths in the listbox formatter. The release notes say you can now do this, but at first I couldn't get it to work. I tried a window from the DLLTutor application and no matter what I tried, I couldn't seem to get a resize cursor. I even closed down C7 and navigated to C:\Users\<MyUserName>\AppData\Roaming\SoftVelocity\Clarion\7.0\ (on Vista, that is) where I deleted the contents of the temp and preferences directories. After restarting C7 I tried a different window. This one (Figure 1) showed vertical lines at the end of each column, and I found I could resize these columns with no difficulty.

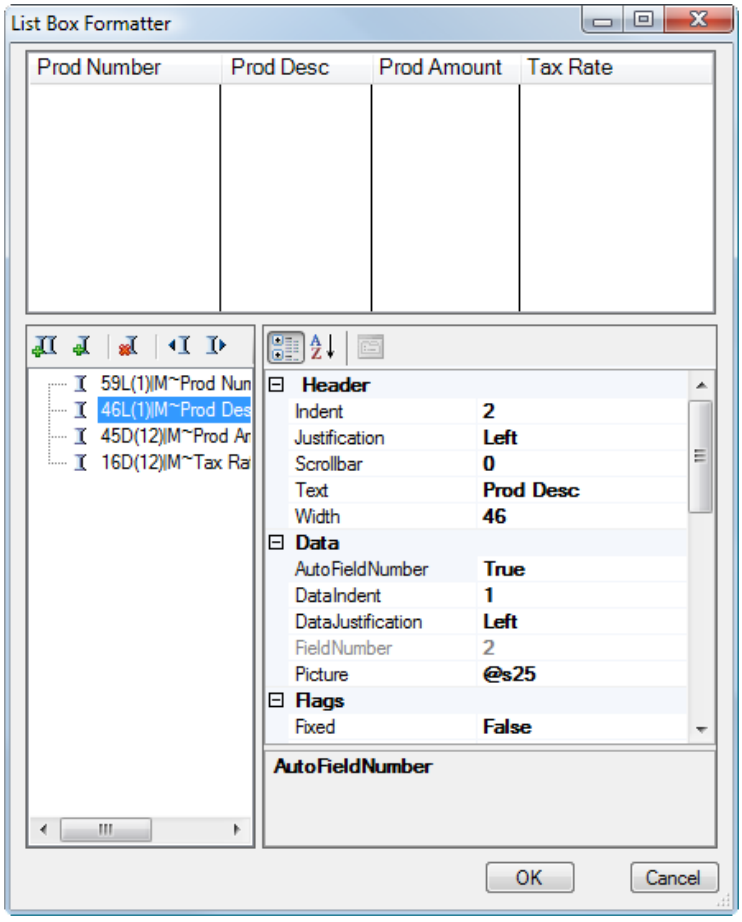


Figure 1. Resizing columns

I assumed the fix had something to do with my deleting the temp and preference files, but then I happened upon the

first window and I still couldn't drag those columns. It appears I needn't have cleared out those directories.

It turns out that if Prop:RightBorder is true resizing works just fine. But if Prop:RightBorder is false before you bring up the listbox formatter you won't be able to resize at all until you set Prop:RightBorder to true. And if you turn Prop:RightBorder off and try to resize the column, things get weirder. PTSS 31540.

That's not a big bug, and I expect it'll be squashed in the next release. I mention it because it's illustrative of the kinds of issues that are cropping up at this stage in the beta. Most of the big problems have been taken care of, and more time and effort is now going into usability issues.

Also on Figure 1 note that the Header category has been tweaked with a leading space to put it at the top of the list. It's a hack, and not a pretty one. I found it a bit stunning that Microsoft's PropertyGrid control, which is used for the property pads in the IDE, all through Microsoft's products and in countless other apps, doesn't provide an obvious way to do custom category sorting. So I did a bit of research and discovered several references to using tab characters to fudge the sort order. Apparently the tabs are stripped off when the category names are displayed. I found a C# example, converted it to Clarion#, verified that it worked and submitted it with a PTSS (#31545). Figure 2 shows the resulting demo app. Under normal circumstances the order of the categories would be OptionA, OptionB, OptionC.

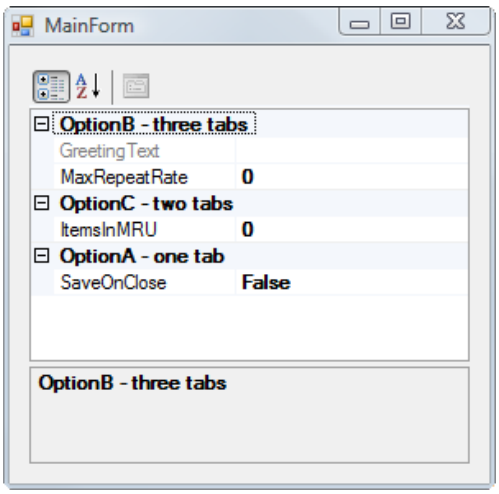


Figure 2. A demo of category sorting

The new sort order makes sense, given that the tab character is ASCII 9, which is less than any printable character. If you have two strings beginning with 'A', which is ASCII 65, and one is prefixed by two tabs and the other by one, clearly 0909 is going to come before 0965.

PropertyGrid controls are pretty interesting. Look for an article on this subject in the new year.

Another helpful change to the Property list is that all the categories are now automatically expanded.

The change list says some accelerator keys have been added to the AppGen buttons, but I don't see what those keys are. I must've missed that memo. There is a new option to generate and build just the currently selected application, which is a big help if you have a large multi-DLL solution.

The image used in the menu editor for a separator line has been changed from a vertical line (which didn't make a lot of sense) to a horizontal line.

There have been quite a few template-related fixes. That's right, this whole beta process was intended to shake out template issues, wasn't it? Feedback on template issues seems to be slowing down, and that tells me that it's probably time to let the CSP folks take a kick at the AppGen.

There have been various other bug fixes, including some random exceptions, synchronizer lockups, problems with field numbers in listboxes, and so forth.

Like others, I've noticed problems with menus staying open after they're no longer needed. That only seems to happen if

you have an application open and you click somewhere in the application pad. Clicking in any other part of the IDE closes the menu.

There's a Dictionary Changes pad; was that there before? Apparently I'm the last person to notice it. The help says:

> The Dictionary Changes pad shows all automatic changes that have been applied to the dictionary. You can double click on any item in the pad and that item will be opened in the dictionary editor.

The Dictionary Changes pad shows information about the conversion of a dictionary from an earlier version of Clarion. One of the things that can trigger a change is the use of a label that is no longer allowed. Figure 3 shows the Dictionary Options for the IDE, General tab - note the Enforce Clarion# name restrictions option.

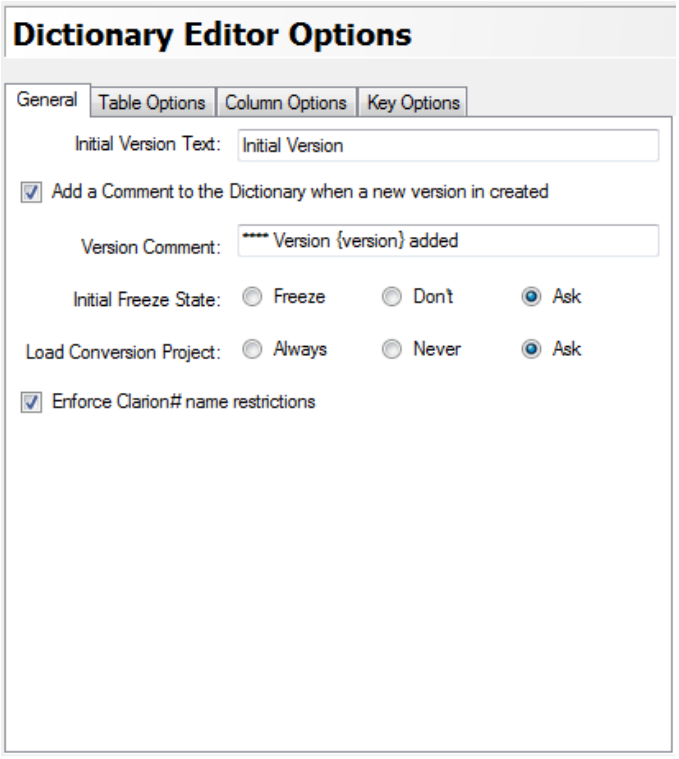


Figure 3. The Dictionary options

Although this is a C7 release, Clarion# and C7 share the same IDE (even though they are current distributed as entirely separate builds), and Clarion#, as a .NET language, has a larger set of reserved words. For instance, you can no longer use CHECKED or AS for labels; if you have a column called As it will be changed to column_As; a table called As would become table_As. These changes will be shown in the Dictionary Changes pad.

Derived fields can also trigger changes that are logged in the Dictionary Changes pad. From the Help:

> In previous versions of Clarion when you changed a field that had other fields derived from it you had to manually apply the changes by either clicking on the Refresh button in the derived field or by selecting Edit/Distribute File. In Clarion 7 changes are automatically applied to derived fields that do not have freeze checked as soon as you save changes to a derived from field.
>
> Because of this behavioral change, when a dictionary is converted from C6 to C7, all fields that are derived from other fields have freeze checked. A message appears in the Dictionary Changes Pad to this effect, but each field is not listed separately.
>
> As the propagation is now automatic, the Freeze check box on Tables and Globals no longer has any meaning. So it has been removed.
>
> You can no longer derive a field from itself. Any dictionaries that have this will have the derived from attribute removed. This

change appears in the Dictionary Changes Pad.

You can no longer have a field derived from another field with a different type. Any fields that have this will have their derived from attribute removed. This change appears in the Dictionary Changes Pad. Once a field is derived from another field, its data type fields are disabled.

The moral of this story: When converting dictionaries, always check the Dictionary Changes pad.

Summary

There's still at least one issue outstanding that I consider major, and I've been harping on it for a while. When the compiler encounters an error caused by embedded code, clicking on that error takes you to the generated source file when it should take you to the embeditor, as C6 does. I've also mentioned there are workarounds, but this has to be fixed in the beta. It doesn't, I think, have to be fixed in the third party beta (although that would be nice).

After the previous beta release I suggested one more build ought to be enough for the third party phase. And hopefully this is it.

In the event that the next release does go out to CSP folks, here's a word of caution: If the beta blows up on you in what seems to be a disastrous way, take a deep breath. I've seen lots of newsgroup postings and Skype messages and whatnot proclaiming this beta or that beta to be a load of crap because it won't do X. Sometimes that's the case; it really won't do X, and there's a major bug that needs to be fixed. Other times what seems like a disaster is really just an inconvenience and the bug has a workaround. And still other times what seems like a bug isn't a bug; it's just that there's a new way of accomplishing a certain task.

As with the last beta, major bug reports seem to be down and usability issues are front and center. That's a good sign; it means the product is becoming stable, and now the focus is on efficiency. By that I don't mean raw speed, since C7 is already a rocket at generating and compiling. The important thing now is to make sure that C7 allows users to make the most of their keystrokes and mouse clicks. And that process will benefit greatly from a wider beta.

Here's to a little something under the tree. No pressure, Bob, no pressure.

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 Friday, December 12, 2008 by Michael Gorman

Good software and good wine should never be served before their time....

So, I'm always for latter right than earlier wrong...

Take your time. From what I've seen via the movies, blogs and other reviews, this is going to be a fantastic new version.

Regards,
Mike Gorman

Posted on Friday, December 12, 2008 by Lee White

For the curious, the temp and preferences folders use this default path in XP:

Documents and Settings\<UserName>\Application Data\SoftVelocity\Clarion7.0

Posted on Monday, December 15, 2008 by Ken Stone

Thanks for a great and informative article.

Add a comment