Clarion Magazine

Clarion News

Hurricane Katrina Relief

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

- » Clarionfoundry Temporarily Offline
- » DigitChanger Updated
- » Free xFunction Library 2.3
- » gReg Half Price Sale
- » EasyCOM2INC 2.07
- » Safe Update Demo Updated
- » Clarion Developer's Challenge Winners
- » CapeSoft Office Messenger 3
- » CapeSoft xFiles Escapes Captivity
- » CapeSoft Safe Update
- » CapeSoft Three Day Spring Cleaning Sale
- » xToolTip 1.9.1
- » xButton 1.6
- » PDF-XChange, PDF-Tools 3.5088
- » Announcing TipLink
- » SealSoft's Free Graphics
- » List Header Customizer 1.2
- » Clarion Developer's Challenge Week #2
- » Clarion Developer's Challenge Week #1 Winner

- » EasyCOMbundle At ClarionShop
- » Buggy 4.1.8
- » xTipHotKey 2.8
- » xToolTip 1.9
- » DigitChanger Final Release
- » Clarion Challenge Returns
- » SetupBuilder Web Site Updated
- » xPrettyAbout 1.0
- » xTimers 1.1
- » SetupBuilder 4 Support Discontinued
- » PrintWindow v1.00 Beta 16
- » EasyCOMCreator 1.02
- » CPCS 6.20 for Clarion 6.2
- » Clarion Magazine's "No DevCon" Sale
- » xDigitalClock 1.9
- » SetupBuilder 5.0 Released
- » EIP Template Version 2
- » xWordCOM 1.5
- » xMisc 1.2

[More news]

Podcast

[Track lists, more podcasts]

Latest Free Content

ClarionMag Blog Launched!

The ClarionMag blog is now online! Rants, raves, and generally unwanted opinions from Clarion Magazine's publisher, Dave Harms.

[More free articles]

Latest Subscriber Content

PDF for August 2005

All Clarion Magazine articles for August 2005 in PDF format.

Posted Friday, September 02, 2005

Mixing Clarion with .NET, Part 2

Now that you know how to call methods in .NET, it's time to put some WinForms objects inside of Clarion windows. It's surprisingly easy once you know a few tricks, as Wade Hatler demonstrates.

Posted Thursday, September 15, 2005

.NET Basics: What Is .NET, And Why Should I Care?

You've heard the pitch before: Microsoft's .NET is the future, and Clarion.NET is the future of Clarion. But really, should you care? And what is .NET anyway? David Harms answers these and other questions.

Posted Thursday, September 15, 2005

Developing .NET Applications With Clarion and Fenix, Part 1

Clarion Programmers have long benefited from Clarion's outstanding capability to integrated data with visual screen design and wizard support to generate code, which can be used as is or tailored as necessary to meet specific needs. RadVenture's Fenix leverages Clarion's technology with a special template set that lets developers create .NET web applications using the Clarion IDE. Ozzie Paez reports. Part 1 of 2.

Posted Wednesday, September 21, 2005

Mixing Clarion with .NET, Part 3

September's focus on .NET continues with Wade Hatler's series on using Clarion and .NET together. In this installment Wade shows how to call Clarion procedures from inside .NET applications.

Posted Wednesday, September 21, 2005

Developing .NET Applications With Clarion and Fenix, Part 2

Clarion Programmers have long benefited from Clarion's outstanding capability to integrated data with visual screen design and wizard support to generate code, which can be used as is or tailored as necessary to meet specific needs. RadVenture's Fenix leverages Clarion's technology with a special template set that lets developers create .NET web applications using the Clarion IDE. Ozzie Paez reports. Part 2 of 2.

Posted Wednesday, September 28, 2005

Mixing Clarion With .NET, Part 4

September's focus on .NET continues with Wade Hatler's series on using Clarion and .NET together. In this installment Wade shows how to how to pass string, date, time and decimal parameters from .NET to Clarion.

Posted Thursday, September 29, 2005

ClarionMag Blog Launched! (free article)

The ClarionMag blog is now online! Rants, raves, and generally unwanted opinions from Clarion Magazine's publisher, Dave Harms.

Posted Friday, September 30, 2005

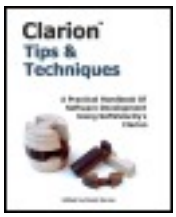
[Last 10 articles] [Last 25 articles] [All content]

Printed Books & E-Books

E-Books

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

Printed Books



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

- Clarion 6 Tips & Techniques Volume 1 - ISBN: 0-9689553-8-X
- Clarion 5.x Tips and Techniques, Volume 1 - ISBN: 0-9689553-5-5

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

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

From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

Clarion Magazine

Clarion News

Search the news archive

CIA Factbook Download

Roberto Artigas has updated the download for the CIA Factbook. The download now includes a test application, dictionary and executable for anyone who wants to play with the data. For those of you that registered and picked up just the data, drop by again and pick up the whole package. You will find it a bit much easier to deal with the data. The TXD that the web scrape program created is also include so you can add it to your own applications.
Posted Tuesday, October 11, 2005

xToolTipPro 1.0

This class and extension template allow you easily change standard tips into native Microsoft tooltip controls with many additional features. Main features: You can set and change in runtime, background and foreground color for tooltips, separately for global and local tooltips; You can have different tooltips on the window; You can make fixed tooltips. And show it automatically over any control of window or in any position of the screen; You can set left, top, right and bottom margins for tooltip text; There are many code templates for ease of use. Support SingleExe, MultiDll (Local Mode, Standalone Mode), 32-bit. Compatible with Clarion 6.2 (build 9047), Clarion 6.1 (build 9034) Compatibility with Clarion 5.5 is possible, but is not guaranteed 100%. xToolTipPro 1.0 already available on ClarionShop for $69.
Posted Tuesday, October 11, 2005

Clarion Developer's Challenge Week #5 Winner

The week #5 Clarion Developer's Challenge winner of XP-TaskPanel ($249) from Gøran Thomassen of PowerOffice is Dean Burgess. Congratulations to Dean and thanks to Gøran for being the vendor of the week.
Posted Tuesday, October 11, 2005

DevDawn Blog

DevCawn is a site which is run by techos who just happen to be Clarion Developers. Their mission for this site is to give a broader picture to what developers do in software development.
Posted Tuesday, October 11, 2005

FinalStep 2.0 Split Up - NeatMessage Emerges

FinalStep has become too complex as a single product, so it is being divided up into subproducts. The first one is a message box replacement called NeatMessage. This new product replaces the message replacement present in FinalStep 1.5 and adds newcharacteristics and better documentation. This is free for current owners of FinalStep and a special low price to new users this month. Some features are pending in release 1.00, but still as it is, it's better than the one included in FinalStep 1.5.
Posted Tuesday, October 11, 2005

TipLink Tool Suite

LANSRAD has released the TipLink Tool Suite, a combination of TipLink Reporter and TipLink TipSynch. These tools let you export and analyze all the tooltip data from your application using an XML file and an analysis program that runs after the export is completed. Price is $24.95 USD.
Posted Tuesday, October 11, 2005

UK Clarion User Group Meeting Nov 21

The next UK Clarion User Group meeting is on Monday Nov 21st at Kings Langley. See the web site for details including travel and accomodation info. New Members are always welcome, and the user group now has several overseas members as well. Book now as over as two thirds of the places were filled on the first day.
Posted Friday, October 07, 2005

Whitemarsh Metabase 6.8

The Metabase Software System's latest version, v6.8, includes big improvements such as an expanded client server layer of human-work saving processes, complete metabase data storage through a SQL engine, and a true multi-user environment for entry, update, and reporting. A item of great interest for Whitemarsh is the start of a Data Management Maturity (DM3) Assessment and Prescription process. The DM3 Assessment data model is complete and the database application that will hold, analyze, and report assessment results has begun. Recently posted to the Website's SQL section is the October 2005 SQL 200n current base documents. Also posted to the web is the SQL/XML document for the Final-Draft International Standard. SQL/XLM will likely be a final standard by Spring 2006.
Posted Friday, October 07, 2005

Three Way Tie In Week Four Of Clarion Developer's Challenge

Week #4 of the Clarion Developer's Challenge has three players tied at ten correct picks: Dean Burgess, Jerry Davis, ands Ed Schneider. Congratulations to this weeks winners and a big thank you to the Clarion Developer's Challenge vendors of the week, Comsoft7, BoxSoft, and SoftVelocity.
Posted Friday, October 07, 2005

xCheckTPS 1.04

xCheckTPS is a free utility for checking TPS files for errors. Features include: You can check both single file and group of files; You have log of checking; Program have some settings for more comfortable using; Program have runtime command line and keys. Works with non-encrypted TPS files with no password. xCheckTPS is based on TPSFile class by Vladimir Yakimchenko.
Posted Friday, October 07, 2005

SysTree C6.2 Compatible

A Clarion 6.2 compatible version of SysTree is now available. Version 1.3.2 contains files for all Clarion 6 versions (6.0, 6.1 AND 6.2) plus the old files for 5.0 and 5.5. This is not a feature update. Registered users can download the update free of charge. The installation password has not changed, so please use the data sent to you upon purchase.
Posted Friday, October 07, 2005

BSPrintList 1.0

BSPrintList was developed primarily as a way to generate WYSIWYG reports for the BST template listboxes.
Posted Friday, October 07, 2005

dpQuery 2.06

dpQuery 2.06 has been released. This version fixes missing mapped fields in saved queries.
Posted Friday, October 07, 2005

Up & Up 1.2

Version 1.2 of Up & Up adds support for recursive (self-related) tables to Up & Up. This feature requires UltraTree version 8, and the UltraTree Recursive Views feature, in addition to Up & Up. The feature has been packaged as a separate add-on to allow Up & Up to be used without this feature by those who don't have UltraTree or the required UltraTree feature. With this capability, Up & Up now provides complete processing and reporting of all file structures supported by UltraTree. Both normal and rollup totaling is fully supported for recursive tables. Normal totaling totals across the records of a single recursive level. Rollup totals accumulate across all levels. Rollups can be accumulated for any particular level using conditional totaling.
Posted Friday, October 07, 2005

iQ-XML 1.10

There is a small update to iQ-XML now available. This version will now handle files picked up from Unix/Linux/MAC/Mainframe files. When parsing, the system will automatically detect EndOfRecord indicators with ASCII 13/10, ASCII 10 or ASCII 13. iQ-XML is a free tool for Clarion developers to add XML functionality to their applications with very little knowledge. iQ-XML comes with both Parser and Writer functions. Quickly generate an XML document from a Clarion Queue, Group structure, or just using the API's. A novice user can also read a complex XML document and fill a Clarion Queue. Navigate easily through-out the document, finding nodes and parsing only what you need. For users who feel comfortable with Templates, there are also templates options available for all the API's. Both PDF, Online HTML documentation, as well as example applications are also included.
Posted Friday, October 07, 2005

solidsoftware Atom Feed

An Atom (1.0) feed with news of solidsoftware product releases is now available.
Posted Monday, October 03, 2005

solidsoftware RSS Feed

An RSS (2.0) feed with news of solidsoftware product releases is now available.
Posted Monday, October 03, 2005

SysList Clarion 6.2 Compatible

A Clarion 6.2 compatible version of SysList is now available. The new version 1.3.3 contains files for all Clarion 6 versions (6.0, 6.1 AND 6.2) plus the old files for 5.0 and 5.5. This is not a feature update. Registered users can download the update free of charge. The installation password has not changed, so please use the data sent to you upon purchase.
Posted Monday, October 03, 2005

xTipHotKey 2.9

New in xTipHotKey 2.9: Modifications in template for compatibility with Data Conversion Templates from SoftCreator.
Posted Monday, October 03, 2005

Oz DevCon Survey

It has been a while since Australian developers have got together for a DevCon type meeting. It used to be called ConVic when it was held in Victoria. The Perth Clarion User group is keen to see a 2006 conference take place, and are even prepared to host it in Victoria. To help the user group assess the viability of such an event, please complete the survey at a special web site. If anyone from overseas is interested in coming then please feel free to express your desire for doing so. Presenters are also wanted.
Posted Monday, October 03, 2005

Data Down Under Distributes SetupBuilder

Data Down Under of Eden, Australia is now the distributor for the SetupBuilder product line throughout Australia and New Zealand.
Posted Monday, October 03, 2005

Clarion Developer's Challenge Week #3 Winners

With 9 correct picks, Bob Foreman Is The Clarion Developer's Challenge Week #3 Winner! Bob, a winner last week, has donated the Softvelocity In Memory Driver prize back into the CDC prize pool and it will be offered as a week #4 prize. The second place winner is Dave Bratovich. the tie breaker was needed to break a 6 way tie! (39). Dave is the winner of Bill Roe's Valutilities!
Posted Monday, October 03, 2005

Free CWPlus Wallpapers

Ingasoftplus has made 14 free CWPlus wallpapers available to download. Sizes are 1024 x 768 and 1600 x 1200 and can be freely used. Kindly designed by Mixer.
Posted Monday, October 03, 2005

Clarionfoundry Temporarily Offline

Due to recent hardware failure, Clarionfoundry will be down for a few days. James will have the site online again as soon as possible.
Posted Friday, September 23, 2005

DigitChanger Updated

DigitChanger now has support for user variables, providing for a real multi-currency system. For clarion 5.5, 6.0, 6.1 or 6.2. ABC and Legacy (Clarion) chains. Multi-DLL support. 100% source code (no black boxes).
Posted Friday, September 23, 2005

Free xFunction Library 2.3

In this version of xFunction: New xGetFileSize PROCEDURE(STRING FileName), LONG; Updated demonstration program and installation kit for Clarion 6.2 (build 9047), Clarion 6.1 (build 9034), Clarion 5.5 and Clarion 5 are available.
Posted Friday, September 23, 2005

gReg Half Price Sale

For a limited time gReg is available at half price.
Posted Friday, September 23, 2005

EasyCOM2INC 2.07

EasyCOM2INC 2.07 is now available. Changes include: Setup program now copies directories \bin, \lib, \libsrc and \template into your Clarion root directory; "Version" was added into the list to avoid redefining system intrinsic; Wrong coclass GUID generation on custom attribute fixed; The code to get IUnknown interface in the QueryInterface method was not generated for eventhandler ([source]); Template improvements. Price: $189. Get a bundle of EasyCOM2INC and EasyCOMCreator for $219.
Posted Friday, September 23, 2005

Safe Update Demo Updated

There's a new Safe Update demo available that demonstrates a simple update of the program from the CapeSoft web site.
Posted Friday, September 23, 2005

Clarion Developer's Challenge Winners

This Week's Winner: Robert Paresi. Robert receives PrintWindow, provided by Jorge Alejandro Lavera of Huenuleufu Development. Bob Foreman and Jerome Atchison tied for second place and each receive TipLink, provided by Charles Edmonds of LANSRAD. Clarion Developer's Challenge hosting is provided by Robby Berthume of Epsilon Hosting.
Posted Friday, September 23, 2005

Clarion Magazine

The ClarionMag Blog

Blog Categories

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

Rate your apps

Direct link

Posted Tuesday, October 11, 2005 by Dave Harms

Ron Childs posted a link in the SoftVelocity chat newsgroup to an online rating calculator which you can use to evaluate your software according to the Tucows Rating Guide. As Ron says, "Prolly a very good check list for any kind of software."

Google Map your tracert

Direct link

Posted Tuesday, October 11, 2005 by Dave Harms

Mark Riffey points out this Italian page that lets you create a Google map of a traceroute. In a DOS window, run tracert (e.g. tracert www.clarionmag.com) and the highlight the resulting trace with the mouse and right-click to copy to the clipboard. Paste into the text box and click the Start button.

The map is pretty small, and it's a world map, so it won't tell you much about traffic around your own city. But if you want the detailed listing of the stops along the way you can look at the page source for the script that plots the map points.

The only problem I found is that the server chokes on Class C private addresses (192.168.0.0 - 192.168.255.255) and displays a number format exception. Not very friendly. So remove those lines, if present, from the head and tail of the trace.

The site name is interesting - wereporters.com. Given my knowledge of Italian, it could be that they're news guys, or maybe just lycanthropes who work for Trenitalia.

Also from Mark - this page on using the Google Maps API.

Canuck Turkey Day

Direct link

Posted Friday, October 07, 2005 by Dave Harms

Up here in the Great White North harvest is already finished. Among other things, that means we celebrate Thanksgiving a month or so before our American cousins. The Clarion Magazine office will be closed on Monday, October 10, 2005. And with three turkey dinners scheduled for the weekend (we have a *lot* to be thankful for), I'll probably be cataleptic from tryptophan poisoning for most of next week. Which is okay - I can use the sleep.

Happy Thanksgiving to all our Canadian friends and Canuck Turkey Day sympathizers!

Experimental RSS Feeds

Direct link

Posted Friday, October 07, 2005 by Dave Harms

I have two new RSS feeds up on the site: a feed for this blog, and another feed for all ClarionMag items. Kick 'em around and let me know how they work. The blog feed was up briefly with full text but no CDATA section so if you got some really ugly HTML you'll need to refresh or recreate the feed in your RSS reader.

For the basics of RSS see this article.

Yo, 3P People, Read This!

Direct link

Posted Friday, October 07, 2005 by Dave Harms

Here's a word for any third party developers who want to get their product announcements up in the ClarionMag news section in the shortest possible time.

First, only a few of you actually email me news items. Mostly I harvest the SV newsgroups, and when I get a free moment I boil them down into a newsy format. This rendering is what takes the time (not that I'm suggesting your news items are dead animals turned into lard - at least not usually). So I'll make you a deal. Send me news items that meet the following requirements, and I'll post them as soon as I get them.

Format - three paragraphs (the first two are single lines), in plain text, as follows:

1. **Title**

   Product version numbers should not include "v" or "Ver" or "Version", just the product number, as in MyGreatProduct 2.1.3.

Do not abbreviate the product name

Use title capitalization (e.g. My New Product Is Now Available).

2. **URL**

   One and only one URL. If you need to present more than one URL, then you need two news items.

   The complete URL, including the http:// (or whatever protocol)

   Use the product URL please, *not* the purchase or download URL.

3. **Description**

   One and only one paragraph, max 250 words.

   Use full the product name at least once.

   Be factual - say what the product does, not how great it is. Do *not* use exclamation marks.

   Plain text only - no formatting or URLs in the body.

   In (North) American English, most nouns are *not* capitalized. Words like template and class, for example, are lower case. Only capitalize these if they are part of the actual product name (e.g. Super Templates).

   Do not use ALL CAPS under *any* circumstances..

Review the current news items for examples of acceptable news content.

If you want your announcement to show up promptly, follow these guidelines, and email me the item. I will continue to harvest news items and massage them into the ClarionMag

format. But that takes time, and occasionally I come across announcements that are so incomplete or incomprehensible that I simply don't bother.

A Spring DevCon?

Direct link

Posted Wednesday, October 05, 2005 by Dave Harms

There's a rumor going around that the next Clarion DevCon will be in March or April. Time to crank up the mill and see what this would mean, if true.

If we do get an announcement shortly, that will tell me that C7 and Clarion.NET both must be close to beta (whether or not they are actually released as beta). Here's why. Z's been burned enough on due date announcements that I can't see him risking a conference without having product pretty much in hand, but a full DevCon needs at least six months lead time to get facilities, speakers, and attendees lined up. Even a smaller tech briefing ought to have a three month runup. You can't just say "Okay, we finally have some good stuff here, let's have a DevCon next month," and neither is it wise to risk announcing an event even six months down the road when there are showstoppers still to be resolved, with no clear (and short) path to resolution.

To me, a DevCon announcement is a clear signal that the new code is rounding into shape. And personally, I don't think a conference just for C7 is adequate - there has to be a significant Clarion.NET product there as well.

On another note, if DevCon does happen in the spring, I'd vote to make the new slot permanent; with the hurricane cycle in its strong phase, and the recent catastrophic events in the gulf, it's time to move out of the cheap season.

What the heck is Web 2.0?

Direct link

Posted Tuesday, October 04, 2005 by Dave Harms

Bill Kinnon points at this article by Tim O'Reilly titled What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software. Web 2.0 is one of those nebulous concepts that seems like it has the potential to completely reshape software development as we know it. Software is increasingly living on the web, and interestingly, databases are a massively important part of that movement.

All of this, I think, only makes Clarion.NET that much more important. Win32 just doesn't offer the same flexibility for web-related development. And clearly Clarion developers have already got the message that the web is massively important to their futures, as indicated by a couple of recent ClarionMag polls.

I'm mulling over an article on Web 2.0 and Clarion...

Clarion and Beyond Compare

Direct link

Posted Monday, October 03, 2005 by Dave Harms

Phil Carroll at Enabling Simplicity (think UltraTree) has created some file comparison rules for those of us using Beyond Compare. Among other things these rules tell BC's compare utility to ignore case and whitespace differences, as well as comment lines (although as comments are defined as anything following the ! character, I suppose there's a slim chance a change with a ! string literal could slip through). Phil sez these rules may be included in release 2.4. Meanwhile you can download them yourself.

I completely agree with Phil on the usefulness of Beyond Compare. Among other things, I use BC to back up my system to removable hard drives, using BC's batch mode.

You need this wheel bad

Direct link

Posted Friday, September 30, 2005 by Dave Harms

Richard Rose posted a message in the SV newsgroups about a new (well, old) mousewheel program that he says works better for him than Flywheel. I've been using Flywheel for years - although the program is free, you do have to register it, but you just use the name "I Am Free" and the code 13601409. Freewheel looks like an even better alternative.

Both these utilities let you use the mouse wheel with applications that don't have any built-in support, like the Clarion IDE. The other app I use on a regular basis that doesn't normally work with the mouse wheel is FrameMaker.

You can download Freewheel from Jim's Software Den, from the Clarion Developer site (download section), and from the UK Clarion user group site.

This time I mean it!

Direct link

Posted Friday, September 30, 2005 by Dave Harms

The ClarionMag blog has officially arrived!

I almost had this blog code ready to go last week, and then I decided to polish things up a little. So here it is Friday again, and hopefully I haven't broken anything on the site. Until the middle of this week I'd only tried the blog stuff on a test server, and when I uploaded some of the changed classes I broke the RSS feeds and downloading. But that's fixed, or should be.

I haven't added reader comments to the blog pages yet - that's going to need a little tweaking, since comments normally require a login, and the blog pages are public access. In the meantime, if you have a question or comment, drop me an email.

[Last 25 entries] [Last 50 entries] [All entries]

Reader Comments

Add a comment

Clarion Magazine

Mixing Clarion with .NET, Part 2

by Wade Hatler

Published 2005-09-15

Now that you know how to call methods in .NET, it's time to put some WinForms objects inside of Clarion windows. It's surprisingly easy once you know a few tricks. In this segment, I'll cover placing .NET controls inside of Clarion windows, and placing .NET forms inside of Clarion Apps, both MDI and non-MDI.

WinForms controls and forms are structured a bit different than other systems you may have seen. Structurally, a WinForms control or window is just an ordinary class that happens to have methods and properties that support window displays. Forms and Controls are very similar, and in fact the Form class is derived from the `Control` class. You can even promote a control to a form by setting a few properties, and demote a form to a control using the same logic. This will come in handy later to build test harnesses. I sometimes create a WinForms form and debug it all in Visual Studio running it as an exe. When I want to plug it into Clarion, I just call a method that creates the form, demotes it to a control and then inserts it into a Clarion form. This gives a lot more flexibility than Clarion's approach. Controls and Windows in Clarion are very specialized language constructs that can't be changed by users. In WinForms, it's easy to override a built in class and make your own specialized class, as I'll demonstrate a bit later. If you're used to Visual Basic or earlier versions of Visual C++, you'll find that WinForms is quite a bit different from both of those as well. VB hid a lot of the form processing behind invisible and unchangeable wrappers, much like Clarion does with the Window structure. Visual C++ split forms and controls up into resource files and a bazillion lines of obscure C++ code.

One thing that WinForms has that should be immediately familiar to you is real-time code generation. The Form Designer works exactly like the Clarion's Window formatter. When you create a form using the form designer, it generates a bunch of perfectly ordinary .NET code that happens to represent that form. Edit the form in the Form Designer, and this code is immediately updated. The converse is also true - edit the code manually, and the changes will show up in the form generator. This is a very powerful way to work... but of course, you already know that since you've been doing it for years.

Microsoft marketing decisions

Regarding COM support in .NET, Microsoft made what is probably a sensible decision from their standpoint, but an annoying one from our standpoint. They'd been singing the COM, COM, COM story for eight years, and wisely decided they couldn't instantly change that to .NET, .NET, .NET without at least throwing a bone to the COM world. So they had developers build all kinds of COM interoperability into .NET, including the ability to make Active-X controls. At the last minute they decided that there were just too many different Active-X containers out there with various levels of compatibility, and they didn't want to have to support them. I don't really blame them. Clarion is just one example of an Active-X container with dodgy support.

How Microsoft implemented this solution is that they left most of the code to make an Active-X control in the codebase, but didn't document it and added big disclaimers that the code is unsupported. The idea is that if it works, good for you, and if it doesn't, it's not their problem.

I think you can decide if something works or not, so I'll describe how to make an Active-X control. There are other ways to put a WinForms control into Clarion, but I don't plan on talking about them.

Create the Active-X control

Microsoft built a very sophisticated wrapper for COM into .NET. I won't explain how it works in detail, because you can find that information easily enough on the net or MSDN (search for "COM Callable Wrapper". All I'm going to do is give you the few magic lines of code you need. You can continue with the example from last time:

1. From Solution Explorer, right-click on your project folder and execute Add | Add User Control. You should end up in the control designer. Change the background color of the control to make it easy to recognize, then open your toolbox and add a button to it.
2. Click on the newly created button, and hit F4 to select Properties.
3. Change the (Name) property to something meaningful. I prefer to match the .NET framework recommendations. With this system all private objects (controls are private by default) and parameters use *camel case*. This means the first letter of the label is lower case, and the first letter of each subsequent word in the label is upper case. The label should only contain real words. I use the default lower-case type of the control as the base, and add at least one word after that. For example, I might named this control `buttonClickTest`. Putting the type as the first word in the label makes it much easier to find your labels later, since all buttons will start with the word "button". It's important to do this right away, because if you change it later in the window designer, the changes don't propagate through all the code. If you want to rename an existing control, you should do it in the source code view.

4. Double click the button to create an event handler. Put something in the event handler to show success. I use this code:

```
private void buttoClickTest1_Click(object sender, System.EventArgs e)
  {
    MessageBox.Show("Button Click handled by .NET");
  }
```

5. Go to the *class* declaration, and add the same COM attribute used before.

```
  [ClassInterface(ClassInterfaceType.AutoDual)]
  public class UserControl1 : System.Windows.Forms.UserControl
```

6. Move the cursor anywhere in the class (I usually go just below the constructor), and add this magic code.

```
[ComRegisterFunction]
static void ComRegister(Type t)
{
   string keyName = @"CLSID\" + t.GUID.ToString("B");
   using (RegistryKey key = Registry.ClassesRoot.OpenSubKey(keyName, true))
   {
      key.CreateSubKey("Control").Close();
      using (RegistryKey subkey = key.CreateSubKey("MiscStatus"))
      {
         subkey.SetValue("", "131457");
      }
      using (RegistryKey subkey = key.CreateSubKey("TypeLib"))
      {
         Guid libid = Marshal.GetTypeLibGuidForAssembly(t.Assembly);
         subkey.SetValue("", libid.ToString("B"));
      }
      using (RegistryKey subkey = key.CreateSubKey("Version"))
      {
         Version ver = t.Assembly.GetName().Version;
         string version = string.Format("{0}.{1}", ver.Major, ver.Minor);
         if (version == "0.0")
            version = "1.0";
         subkey.SetValue("", version);
      }
   }
```

```
}

[ComUnregisterFunction]
static void ComUnregister(Type t)
{
   Registry.ClassesRoot.DeleteSubKeyTree(@"CLSID\" + t.GUID.ToString("B"));
}
```

Explicit GUID

Every COM/Active-X interface needs a GUID (Globally Unique Identifier). In the first simple example I let .NET generate one. This is generally a bad idea, because you lose control of the GUID. .NET generates a GUID based on a bunch of factors including the build number, version and culture for your assembly. This leads to GUIDs that change unexpectedly. Always explicitly create a GUID and specify it with an attribute.

In the retail version of .NET, use Tools | Create GUID and click the Copy button. You'll have to get rid of some curly-braces when you paste it. If you are using the Express edition, you can get a GUID here. Paste an attribute right above your class declaration similar to this:

```
[System.Runtime.InteropServices.Guid("08d855b9-b9b9-4bf5-bad1-3cc109e13f94")]
```

That's all there is to it. Build your project and you have a spanking new Active-X control. In case you're curious, the only difference between an Active-X control and an ordinary Active-X/COM object is a few registry entries. Microsoft simply removed the feature that automatically wrote those entries. The code above does add those entries, but it's not the only way to do it. You could add the registry entries in an installation program or with a utility.

Naturally, I'm showing you the cut-and-paste method because it can be used with any existing control. Since C#, like Clarion doesn't support multiple inheritance, there is no other good way to add new features to a control if it's based on a class you can't change. When it's reasonable to do so, I recommend you do the right OOP thing and derive your own control from `System.Windows.Forms.Control`, add these methods to it, and then create all controls from your new class. That way, all the magic goes in your base class. I'll demonstrate that technique later.

Add the control to Clarion

After you've built the control, you can use it in any Active-X container. In Clarion, you can either declare the control in the Window

structure, or you can create it at runtime.

1. To add it to an existing window, use the Window designer, or paste in this code:

```
OLE,AT(10,10,100,30),USE(?Ole1),CREATE('Interop02.UserControl1'),COMPATIBILITY(020H),TRN
END
```

2. To make it at runtime, use something like this:

```
CreateControl              routine
   data
Interop02                  long
   code

   Interop02 = create(0, CREATE:OLE)
   Interop02{PROP:Compatibility} = 20h                    ! 32 bit
   Interop02{PROP:TRN}           = 1                      ! Transparent
   Interop02{PROP:Create}        = 'Interop02.UserControl1'
   setposition(Interop02, 0, 0, 100, 100)
   unhide(Interop02)
```

You need to make the control transparent for the background color to show through. Build and run the project, and you should see the control, and clicking on the button should produce the message box created above.

In our work at my company, we've found the second method to be more reliable than declaring the control in the window structure. I'm not sure why that is, but it seems to be true with some Active-X controls. There's no penalty for doing it, so that's the method I generally use.

Get rid of cut-and-paste

Before I start on how to make a .NET form in Clarion, let's convert the magic registration code to proper OOP and abandon the cut-and-paste hack.

1. Go back to Visual Studio and create a new class file. Name the file `ActiveXControl.cs`.
2. Rename the class to ActiveXControl, and base it on `UserControl`, e.g:

```
public class ActiveXControl: System.Windows.Forms.UserControl
```

Note: This means the class is derived from the `UserControl` class. Like Clarion, C# and most .NET languages have single-inheritance. Following the control you can list a single class. After the classname, you can list several interfaces if the class will be implementing interfaces.

3. Paste in the same `ComRegister` and `ComUnregister` functions you used last time.
4. Paste in the same `ClassInterface` attribute you used last time.
5. Save that control and you're done with it.

I'm going to work with the simplifying assumption that you just add this .cs file to any project, and you use it manually. You could of course put it in a library project and just reference it if you like.

Now that you have a good base class, you can create a new control the easy way:

1. Create a new User Control just like last time, and rename the file to ClarionForm.cs, and the classname to `ClarionForm`.
2. Make the control a bit bigger than the default, and change the background color to make it visible.
3. Add a button for testing and add a message box to it just like last time.
4. Give the button a proper name and text, then Double-click the button to create an event handler, and add a *MessageBox*.
5. Change the base class to the one you just created. Change this line:

```
public class UserControl2 : System.Windows.Forms.UserControl
```

to

```
public class UserControl2 : ActiveXControl
```

Now build the project and you'll have a control much like the first example, but without all the cut-and-paste. This won't always work because some controls aren't based on `UserControl`, but it will work in a lot of cases. When it won't work, just use cut-and-paste.

Adding a form to a Clarion project

There are a couple of ways to add a WinForms form to a Clarion project. The best way is to wrap it inside a Clarion window. This makes a lot of things work well that might give you trouble if you use another method.

There are two ways to add the form. I'll demonstrate the simplest, which is to simply extend the Active-X method you already know.

The key to this technique is that you won't ever actually create a WinForms form. You'll simply create a control, and make it fill the entire client area of a Clarion window. You can do it just like before, either by embedding the appropriate code into the window declaration, or by adding it at runtime. With either method, use the `FULL` attribute to make the control fill the entire window, and be sure you don't give it width or position attributes. The code to add it after opening the window would look like this:

```
LoadForm                routine
   data
Interop04               long
   code
   Interop04 = create(0, CREATE:OLE)
   Interop04{PROP:Compatibility} = 20h    ! 32 bit
   Interop04{PROP:TRN}           = 1      ! Transparent
   Interop04{PROP:Full}          = 1      ! Fill the entire client area
   Interop04{PROP:Create}        = 'Interop01.ClarionForm'
   unhide(Interop04)
```

Update your Clarion program and run it, and you'll see this control filling the window. Note that making a window contain a WinForms window doesn't preclude having Clarion controls in the same window, but the Clarion controls won't always work correctly. If you fill the whole client area as I've described here, but leave Clarion controls in the same area, the Clarion controls won't receive mouse clicks or other events properly. If you really want to mix Clarion controls and .NET controls, don't make the OLE control fill the whole window.

Handling other events

When you embed an Active-X control, you can handle a lot of events on either side of the Clarion-.NET fence, and sometimes you might need to handle the same event twice. For an example, here's how to make the resizable test window handle the resize event in .NET.

1. Add some Clarion code to process `EVENT:Sized`. The sample program just shows a message.
2. In Visual Studio, open your form in the form designer (press Shift-F7 if you're in code view), and hit F4 to view the properties. Note that you can't really get to the property sheet without first switching to design view.
3. In the *Property Sheet,* click on the lightning bolt icon to get events, and then double-click on the `SizeChanged` event. You'll get a new delegate to handle size changes.
4. Put some code in just so show that this happened. For my test, I used:

```
MessageBox.Show("Size Changed");
```

Build the project, then build the Clarion project and run it. You'll notice a couple of things:

1. You get an initial size-changed event in .NET the first time you open the window. This is expected, since Clarion filled the client area with the control, which resized the control.
2. Whenever you resize the window, you'll get the .NET event first, and the Clarion event second. That's because .NET hooks into the event loop closer to the metal than Clarion in this particular case. Usually, this is a good thing, but if you need Clarion to get it first you should remove the event entirely in .NET, then expose a public method that you can call from Clarion, once you've done your resizing.

Okay, now what can I do?

Now that you have this basic system, you can take any of the .NET controls (there are thousands), put them inside of a control like the one you made above, and host that control in a Clarion window. For example, the .NET DataGrid is quite a bit easier to use for many things than Clarion's listbox. Both Infragistics and DevExpress have grid and tree controls that are like a dream come true for data display. You can have all of these right now by simply writing a couple of wrappers. You can also get literally thousands pieces of sample code at CodeProject, or GotDotNet, so you might not even have to write the code yourself.

Note that you can also control how you place your new-found Active-X controls in your Clarion window. For example, you can put an Active-X control inside of a region, or group box and have it automatically expand to fill that region or group box. This makes resize logic much simpler, because you can do it all on the Clarion side. You can do this by placing the OLE object inside of the group structure in the window designer, or using the third parameter of the `CREATE` statement.

Flexible forms

WinForms have a lot of flexibility built in. `Forms` in WinForms are derived from `UserControl`, and you can switch a particular object back and forth from a control to a form by changing a few properties. This gives you quite a bit of flexibility. For example, you could make a system where you design a `Form` in Visual Studio, and it works as a form for .NET hosts, and as an Active-X control for non-.NET hosts. You can also make a control and promote it to a form by simply setting a few properties.

Wrapup

I've covered how to stuff WinForms controls into Clarion in several ways. There are a lot of variants on these techniques. The next segment will cover how to call back to Clarion from .NET.

Download the source

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

Reader Comments

Add a comment

- » Thanks Wade, this is a great series of articles, just...

Clarion Magazine

.NET Basics: What Is .NET, And Why Should I Care?

by David Harms

Published 2005-09-15

You've heard the pitch before: Microsoft's .NET is the future blah blah blah, and you've maybe even heard that Clarion.NET is the future blah blah blah. But really, should you care? Is .NET that important? What benefits come with a .NET version of the Clarion language? And is all this important now, or not for years to come?

In this article I'll try to answer some of the questions about the benefits of .NET, and specifically Clarion.NET. I'll also argue (persuasively, I hope) that yes, .NET is the not just the future of Windows development but the present as well, and as soon as a stable version of Clarion.NET is available, Clarion developers will have in hand a pretty stunning combination of code generation and (finally!) cutting edge user interface tools. Clarion.NET also promises full access to a vast world of third party components, and also opens the world to Clarion third party developers.

Heady stuff!

What is .NET?

.NET is a lot of things; a programming specification, a massive class library, a new way of using operating services, a way of sharing code across languages, and so on. It's big, and comprehensive, and sometimes it seems incomprehensible. But at its heart, .NET is all about one simple idea: making software easier to write.

I know, you've heard all this before. Supposedly, all language vendors are working on tools that will make a software developer's job easier. In fact, for a while it was common to hear about vendors who planned to make that job so easy, a machine could do it; we would see software that would write software. This utopian (for the vendor), or dystopian (for the unemployed developer) vision hasn't, for the most part, come to pass. In fact, you could argue that software development hasn't become easier at all; tasks, tools, and languages are all increasingly more powerful and correspondingly more complex.

There are at least three answers to the argument that software development has become more difficult with time. The first is to cave in and agree that yes, writing software has become a real pain in the butt. But that only tells part of the story. Another answer is that software does so much more than in years past, and it is in fact much easier to accomplish any given task with contemporary tools and languages than it would be to reach the same result with, say, assembler. And the third answer is that in fact the language/development tool vendors have learned a few things over the past decade, and have made significant strides forward in some key areas.

The good old days

First, bear with me as I go back just a bit further in history, to a time before graphical user interfaces appeared on personal computers. No, spare me your reminiscences about how little memory your first computer had, or how much it cost; I can't hear you anyway. Let's just take DOS as the typical pre-GUI operating system. In the DOS world, every developer was free (or forced, depending on your perspective) to create his or her own application user interface. There were no standards for menus and keyboard shortcuts, for how you presented data on screen, or for how navigation keys worked (at least until IBM published the Common User Architecture specification). And writing graphics software might mean coding specialized routines for any number of available graphics cards.

The Windows API

Then came Windows (well, first came the Apple Lisa, if we're talking about personal computers of all sorts, but that's another story). Windows 1.0 provided some level of programming standardization, primarily through the Windows Application Programming Interface, or API. In the DOS world, if you wanted to create an on-screen form, you might use a language that had that capability built in, or you might use a commercially available library, or if you were really brave you'd write the field placement code yourself. With

Windows, you called API functions to create the window, to populate it with fields and buttons and other controls, and to handle the user's interaction with that form. A grueling job with C, to be sure, but fairly easy with something like Visual Basic or Clarion.

The not-so-wonderful world of COM

But really the Windows API was pretty low-level stuff, and that meant writing a lot of repetitive code, especially if you were a C coder. It would be much easier if, say, a particular control and its associated code could be bundled up into one easy-to-use unit. And that's how the component revolution happened, initially driven by Visual Basic components, or VBXs, and later by ActiveX components. ActiveX, under the hood, relied on Microsoft's Component Object Model, or COM. (It's important to note that COM is not a programming language; rather, it's a specification for how to create sharable 32 bit Windows objects, along with a set of supporting API functions.)

COM is a significant improvement on the unadorned Windows API, right? Yes and no. COM is an object-oriented specification, whereas the Windows API is procedural, not object-oriented, code. COM does make it possible to package up complex functionality into a single unit. As well, COM is language-neutral - a component written in, say, C++, can be used by Visual Basic. But COM itself is far from easy to use in many languages, and if a task can be done either via the Windows API or .a COM component, less-experienced programmers may well do better with the API. My personal response to COM is usually to run away, very fast.

Just as the Windows API made it actually possible to create Windows applications without writing everything from scratch, so COM made it possible, if not always easy, to reuse components

The power of Windows programming languages had certainly increased, but ease of use was a mixed bag. And then inspiration came from an unexpected source.

Microsoft sees the Sun

In 1995, Sun Microsystems released a product that would have far-reaching implications for Windows programmers. That product was Java. And Java wasn't just another object-oriented programming language, even if that's how it was perceived by the masses. Central to Java was the concept of the Java Virtual Machine (JVM), a software layer between the

Java program and the actual computer. You wrote your program in Java, and the compiler converted it to something called byte code. And byte code was what the virtual machine executed. The Java mantra was "write once, run many", meaning that once you'd written a Java program, you could run it on any hardware for which there was an implementation of the Java virtual machine.

In contrast, Windows compilers created native code; you could run a Windows program written in, say, C, on an x86 processor, but if you wanted to run that same program on Windows running on another processor (say PowerPC) you'd have to recompile your program for that platform (seldom a trivial task, and of course it assumed that Microsoft had ported Windows to that platform already, an even less trivial task).

Java's multi-platform capability was, and is, a mixed success, but Java also boasted some features that really did make programmers' lives easier.

One of these features was the replacement of pointers with references, which sacrificed some power but made programming a whole lot safer; related to this was automatic cleanup of allocated memory (otherwise known as garbage collection) so programmers would no longer have to worry about memory leaks. The JVM managed the programmer's code, making sure that it didn't do things it wasn't supposed to do, and even allowing administrators to create security policies (particularly as of Java 2) governing which applications could make use of which features (e.g. whether an app is allowed to read and/or write files). Java also sported a relatively clean single inheritance design and strong data types, and came with a large class library that simplified many standard tasks.

Microsoft definitely took a few pages from Sun's book when it came time to create the .NET framework. At the same time, there are some key differences between the .NET approach and the Java approach.

Differences

While Java is both a language and a framework, .NET is, in name at least, really only a framework, not any one language. The closest analogue to the Java language plus the Java virtual machine is the C# language plus .NET.

Java code, originally, was never compiled - instead, it was interpreted by the Java Virtual Machine. This resulted in less-than-blistering speed, as noted by many of Java's early detractors. Java performance has since been improved by the creation of Just In Time (JIT)

compilers, which compile Java code to native code on the fly.

.NET also compiles to an intermediate format, called IL (for Intermediate Language) code. Unlike Java byte code, .NET IL code is not designed to be executed on its own, but is always compiled to native code although, like Java JITs, this compilation takes place at runtime.

.NET is more about multiple languages running on Windows than about one language running on multiple hardware platforms. Yes, there is the Mono project, which is a Novell-sponsored open source implementation of .NET for Unix/Linux, but naturally Microsoft is not particularly interested in seeing Mono succeed (unless you subscribe to the theory that Mono is really a back door for Microsoft to take over the *nix world, in which case you really need to get out more). Despite doing a fair amount of research, I really have no good grasp of where Mono is at. Will you one day be able to run a Clarion.NET app on Linux, via Mono? Maybe, but inasmuch as a Clarion.NET app will use WinForms, and WinForms is supposedly still pretty dependent on COM behind the scenes, and Mono doesn't support COM, the whole thing looks a bit iffy. For now, when you think .NET, I suggest you think Windows. But I'm open to faint hope and better information.

Similarities

Both .NET and Java are really complete, object oriented development frameworks. This is clearly the case with .NET, where you can code using any of a number of different .NET languages, and all those languages can make use of the .NET framework class library. Most developers think of Java as simply a language, but in fact there are many languages now that create Java byte code, and which can take advantage of all of the standard classes that are part of the Java framework.

Both .NET and Java are frameworks that "manage" running code. An administrator can decide which kinds of system services an application can use, control its use of memory, etc.

Both .NET and Java come with standard, and truly massive, class libraries that make it relatively easy to do a wide variety of tasks including memory and thread management, data manipulation, security management, system calls, and much more.

Both the .NET and Java class libraries use *namespaces* to make it easier to organize the class library.

Both .NET and Java require a significant download if you don't already have the environment on your computer. The bare bones .NET 1.1 runtime is a 24 meg download, while the .NET 2.0 beta 2 download is over 300 megs; the Java 5 download is around 120 megs.

Namespaces

One of the big problems with the Windows API is it's very difficult to find what you want. There's no consistent organization of the function names, and what's worse, each function prototype must be unique.

The Java and .NET frameworks are not collections of API calls, they're collections of classes which you use to accomplish certain tasks, just as you would use API calls. But unlike the "one big wad of calls" approach used in the Windows API, both Java and .NET divide their class library up into namespaces. The idea is that you can have any number of classes by the same name, as long as each class is in a different namespace (just as you can have any number of identically declared methods, as long as they are all in different classes).

For instance, the Java framework contains two identically named classes called Date, but they are not in conflict because one is in the java.util namespace, and the other is in the java.sql namespace.

When dealing with multiple namespaces, you will typically either need to refer to a class by its fully qualified name (e.g. java.util.Date or java.sql.Date) or tell the compiler that you want to use a list of namespaces, in which case the compiler will automatically try to resolve the class name by looking in that list of namespaces specified for the module.

Namespaces make a tremendous difference when it comes to locating some specific functionality in the library. Take a look at the .NET namespaces at MSDN. There are over a hundred namespaces at present, covering everything from basic programming components such as lists and hashtables to code generation to data access, graphics, and XML. Now, try to find that kind of information (never mind functionality) in the Windows API!

What it all means

From a Clarion perspective, I like to think of the move from Clarion for Windows to Clarion.NET as having some things in common with the move from CPD (or CDD) to Clarion for Windows, including:

- Greater language standardization
- Wider range of third party products
- More functionality with less code

In the next installment I'll talk more about these points, and look at some examples of .NET controls and class libraries.

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 produces the Planet Clarion podcast, which he co-hosts with Andrew Guidroz II.

Reader Comments

Add a comment

Clarion Magazine

Developing .NET Applications With Clarion and Fenix, Part 1

by Ozzie Paez

Published 2005-09-21

Clarion Programmers have long benefited from Clarion's outstanding capability to integrate data with visual screen design and wizard support to generate code, which can be used ‘as is' or tailored as necessary to meet specific needs. It is this capability, combined with the integration of data dictionary directives and the ability to accept a variety of templates, which makes Clarion so attractive to solution providers.

> **Definition:** A *solution provider* delivers a product intended to provide a capability or solution, as opposed to a programmer, whose primary product is the code that he/she writes. Programmers can be solution providers and vice versa, since the classification is based on the nature of the deliverable, as opposed to activity or task being performed.

While Clarion has a long history of delivering client-server applications, ASP.Net introduced new challenges to the Clarion development methodology, which SoftVelocity is currently working to resolve as part of its upcoming Clarion.Net product family. For many of us Clarion programmers who are being pressed by their clients to deliver .Net applications, the future is here, and only one Clarion solution is currently available: RadVenture's Fenix.

Product description and capabilities

Fenix is essentially a new template chain designed to generate VB.Net or C# code to

create .NET web applications. The templates do not generate Clarion code and cannot use templates that are not specifically designed to be Fenix and VB.Net/C# compatible. In other words, you cannot just add Fenix template components to a Clarion application and magically turn it into an ASP.Net application or vise versa; you must choose one or the other, not both.

Architecturally, Fenix is built on a three tier Application Model composed of the User Interface, Business Object and Database Layers. The User Interface Layer maps window controls to their associated web controls, the Business Object layer maps tables and relationships defined in the dictionary to an associated Business Object Class, and the Database Layer provides support for the existing .Net Data Providers (SqlProvider, OracleProvider, OleDb Provider and ODBC Provider). Code generated by the User Interface and Business Object Layers are accessible to the programmer and can be edited/customized, although this will require significant knowledge of and experience with VB.Net/C#. It is not very difficult, but it will take time, although the numerous examples included with the product greatly facilitate the transition to .Net development.

The system includes a series of template sets that allow the user to create browses and forms, along with control templates that implement display and entry controls that generally resemble the Clarion client-server paradigm, including tabbed forms, string entries, text boxes, drop-downs and check boxes. Other controls facilitate the integration of external systems and components such as Crystal Reports. Finally, augmenting these capabilities is a set of Wizards that can generate full applications, data browses and forms.

Target system and code generation options

Fenix supports code generation for two types of target systems: Standard and Mobile. The Standard model generates code intended for displaying applications on standard computer screens running a browser such as MS Internet Explorer. The Mobile model generates code that can render screens on either a standard screen or on a mobile device, such as a web-enabled phone. Specifically, the Standard model generates client side, rich interface applications based on the .Net Compact Framework model, while the Mobile model generates server-side Web Applications. A Mobile application will check the display capabilities of the device to which it is connecting and render the display based in part on that information.

The general architecture behind the two models is illustrated in Figure 1, which was obtained from the Fenix manual. While the decision to generate a Standard or Mobile

application is based on a setting during the application configuration, generating Mobile application requires a good understanding of the target systems on which the application will be displayed and the limitations associated with screen controls, particularly as the capabilities of existing devices can vary significantly. This translates into taking additional care with screen size, control selection and performance considerations, which will require careful design and potentially extensive testing.

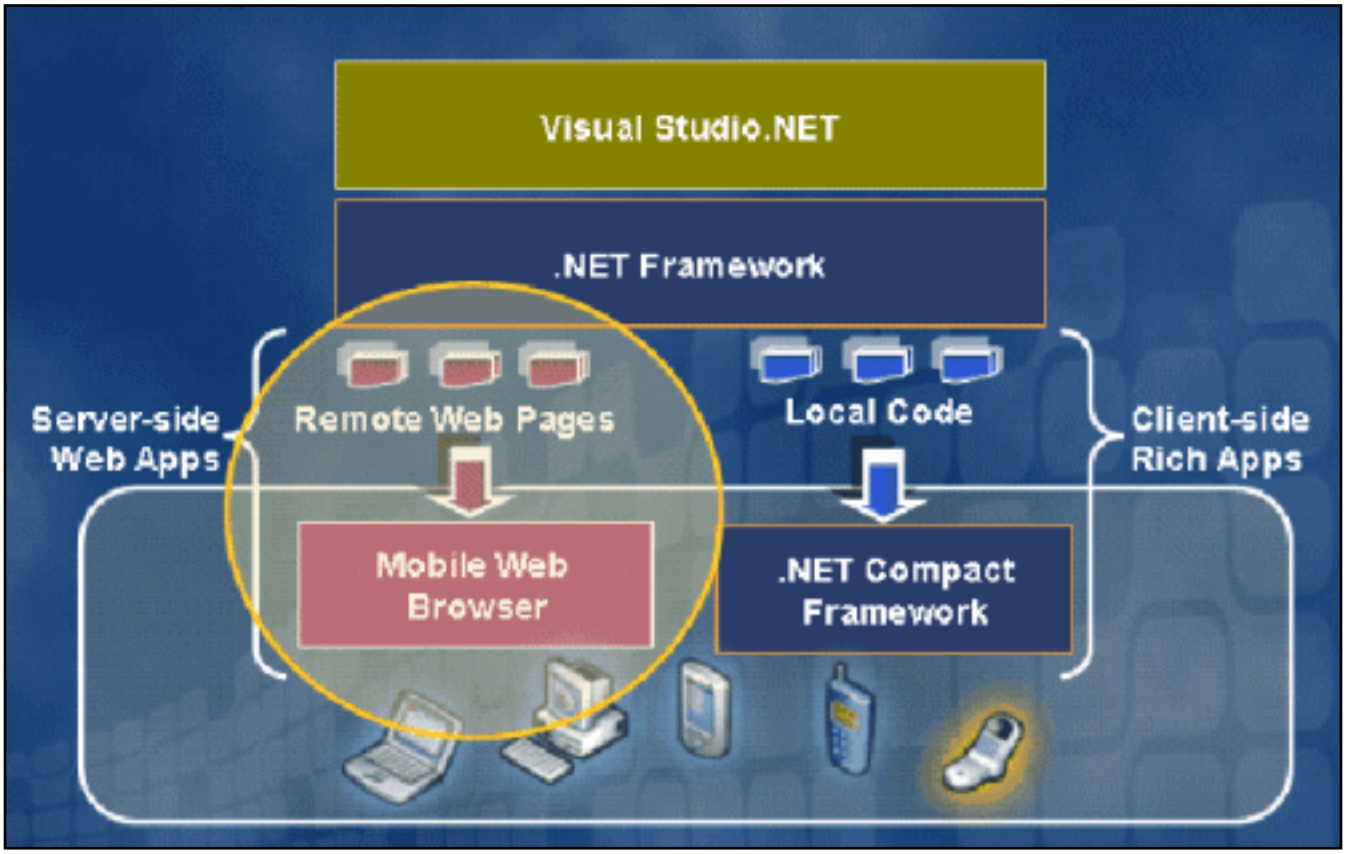


Figure 1. Fenix Development Options

VB.Net versus C#

Once the target system is selected, the next decision applies to the programming language, VB.Net or C#. The choice is entirely up to the developer, although if the application is done for a client, their input should be taken into consideration. If the program will not require significant modifications, i.e., if the Clarion programmer will rely on the Designer for all or most of the functionality, then the programmer's level of comfort with the underlying code isn't that important. On the other hand, most modifications, including the setting of filter parameters and code embeds will have to be done in the target language. Just as importantly, the programmer will need to be aware of which underlying objects have been instantiated, and are visible, when code is inserted. Thus, in all but the simplest

applications, Clarion programmers will need to become familiar with or even learn their preferred .Net programming language.

Documentation

Fenix includes a detailed manual for both the Standard and Mobile target systems. The manuals do a good job in guiding the programmer through the installation of the system and the configuration of the underlying .Net components. In addition, the manuals include a FAQ section that is very helpful in demonstrating how various features can be added, configured and debugged. Finally, there is a separate manual covering the sample applications and the capabilities that they demonstrate. As a set, these provide thorough, although not exhaustive information. You can expect to buy and read several .Net programming books if you plan to extensively modify your application.

Sample applications

Ten separate applications are included with the system, which demonstrate a range of configurations, behaviors and complexities. Five of the applications are loosely based on the Northwind example database, which ships with various MS and other systems, while the rest target specific methodologies for filtering data and accessing databases through ODBC, OleDb and Providers:

- **Northwind Simple** is the simplest application and is particularly useful for testing the initial configuration of the development and deployment environments,
- **Northwind Example** is a full application that demonstrates a range of design approaches, use of controls, embed code techniques and Role Based Security. It includes a help file that explains the techniques used at various points in the system,
- **Northwind Security** illustrates the implementation of Role-Based-Security, one of the primary security models implemented by the .Net environment,
- **Northwind Reports** illustrates the techniques necessary for integrating Crystal Reports into a Fenix application,
- **Northwind Csharp** is a C# implementation of the VB based Northwind Example described above,
- **MyRecord** illustrates the use of Business Object level filtering to provide consistent information filtering based on Role Based Security configurations and user login,

- **Access** implements an application based on OleDb the JET driver to access an MDF File,
- **School** illustrates the use of ODBC to connect to a Clarion database and create a Fenix based ASP.Net front end,
- **Oracle** is an application that illustrates access to Oracle databases through the Oracle Provider,
- **WebServices** provides an example of an application that uses/consumes WebServices.

I recommend that anyone intending to use Fenix to develop their own applications carefully study the sample applications and associated documentation before starting their development.

Installation

Installing Fenix is simple and consistent with the installation of other Clarion templates. RadVenture provides an installation executable that will install and register the templates, with little user intervention. Anyone who has installed Clarion components will recognize the approach. Once the template chain is installed, the sample applications can be opened and studied, although running them will require additional configuration of development system, including the installation of the .Net Development Framework and the configuration of a web site under MS Internet Information Server or a compatible web server.

Configuring the development and deployment systems

.Net development and deployment greatly differs from traditional client server in a variety of ways, including:

- Operating System – to run a .Net application requires a web site on a web server that is properly configured. In the MS paradigm, this means moving to Windows 2000 Professional or Server, Windows 2003 Server or Windows XP Professional, all of which include Internet Information Server,
- The .Net Development Framework is required for developing .Net applications. The framework is free of charge and can be downloaded from Microsoft's web site. It includes various components needed to run .Net applications as well as a .NET compiler. It is a large download so a fast connection is recommended,

- IE WebControls are required if the developer intends to use their behavior, including the Outlook style frame and menu system used in a number of the sample applications. It should be noted that Microsoft does not support these controls and some corporate IT department will refuse to install them. We have successfully installed the navigation control on multiple Windows 2002 Pro, Windows 2000 Server/Enterprise, Windows XP Pro and Windows 2003 server without incident, however, many IT departments are becoming like the credit card commercial with the motto of "Always Say No,"
- Data providers are required for connecting to backend databases like Oracle, MSSQL and MySQL, unless the OleDb/ODBC approach is used,
- External components will be required if the application uses objects such as reports created with Crystal Reports.

Finally, the installation of the environment components and subsystems will need to be accomplished in a specific order because some components register themselves with existing servers and other subsystems. For example, the .Net Development Framework will register itself with MS IIS, which requires that IIS be installed first. The Fenix manual does an excellent job in guiding the user through the process of obtaining, installing and configuring these components, although as new versions come out, problems can arise as a result of behavior changes. In other worlds, there are always nuances that can cause problems.

Validating development and deployment system configuration

Once the development/deployment system has been configured, it should be tested to ensure that the installation and configuration was successful. A good way to accomplish this is by compiling and running the ***NorthWind_Simple*** application. If it compiles and runs, then the base system has been successfully configured.

Creating an application

The process for creating an ASP.Net application with Clarion and Fenix follows the Clarion paradigm closely. The first step is to create a dictionary as you would normally. Once the dictionary has been created, it is used in the creation of the ASP.Net application, just as it would be had the application used the Clarion template chain.

Dictionary considerations

As I mentioned previously, an application created with the Fenix template chain cannot be turned into a standard Clarion (ABC or Legacy) application and vise versa. However, you can use one dictionary with both Fenix and Clarion applications. In the ASP.Net world, most application will have a SQL backend (Oracle, MSSQL, MySQL, etc.), something that will challenge many programmers who are used to stand-alone database systems like TopSpeed, Access and dBase. Given that the dictionary is not affected by the Fenix system, these issues will be common to both ASP.Net and Clarion client server applications. The Clarion manuals include significant information on accessing backend SQL databases through the native SQL drivers (MSSQL and Oracle), while the Fenix manual provides very good guidance and useful information for accessing databases through the MS Data Providers.

Application considerations

Designing a Fenix based ASP.Net application is not significantly different, at least from a process standpoint, than designing a regular Clarion application, as long as the programmer remembers some of the key constraints previously described, i.e., Clarion specific templates will not be available, third party Clarion templates will not work, etc. Other considerations not usually present in the Clarion paradigm include navigation, security and the integration of external components.

Application menu/navigation

Clarion client server applications generally follow the standard Microsoft toolbar approach, although templates are available to give them an Outlook look and feel and programmers are free to use other paradigms, such as a panel with a series of buttons, for example. Fenix is similarly flexible in its approach to navigation, although the more common paradigms are the Outlook and panel-button approaches. For the purpose of illustration, I will reference the on-line example accessible from the Radventure web site to illustrate these approaches.

Figure 2. Fenix Navigation Paradigms (view full sized image)

The Outlook paradigm works by creating three frames or panels on which various areas of the application reside. The top frame provides an area for the name of the application, login request link, logged user and similar global features. This frame can also contain links to a variety of screens and functions, although from a practical perspective, this is usually not a good idea, since it limits the amount of vertical screen space available to other functions. The left frame displays a navigation menu, which uses Microsoft's free IE WebControl to display and configure the links based on an XML file, which Fenix creates from the standard menu design in Clarion. The right lower frame is the location where data browse and form screens are displayed, although as shown on the application screen

above, it can also contain components and links, such as buttons and URLs. The screen capture below shows two screens in a parent-child relationship being displayed in this area.

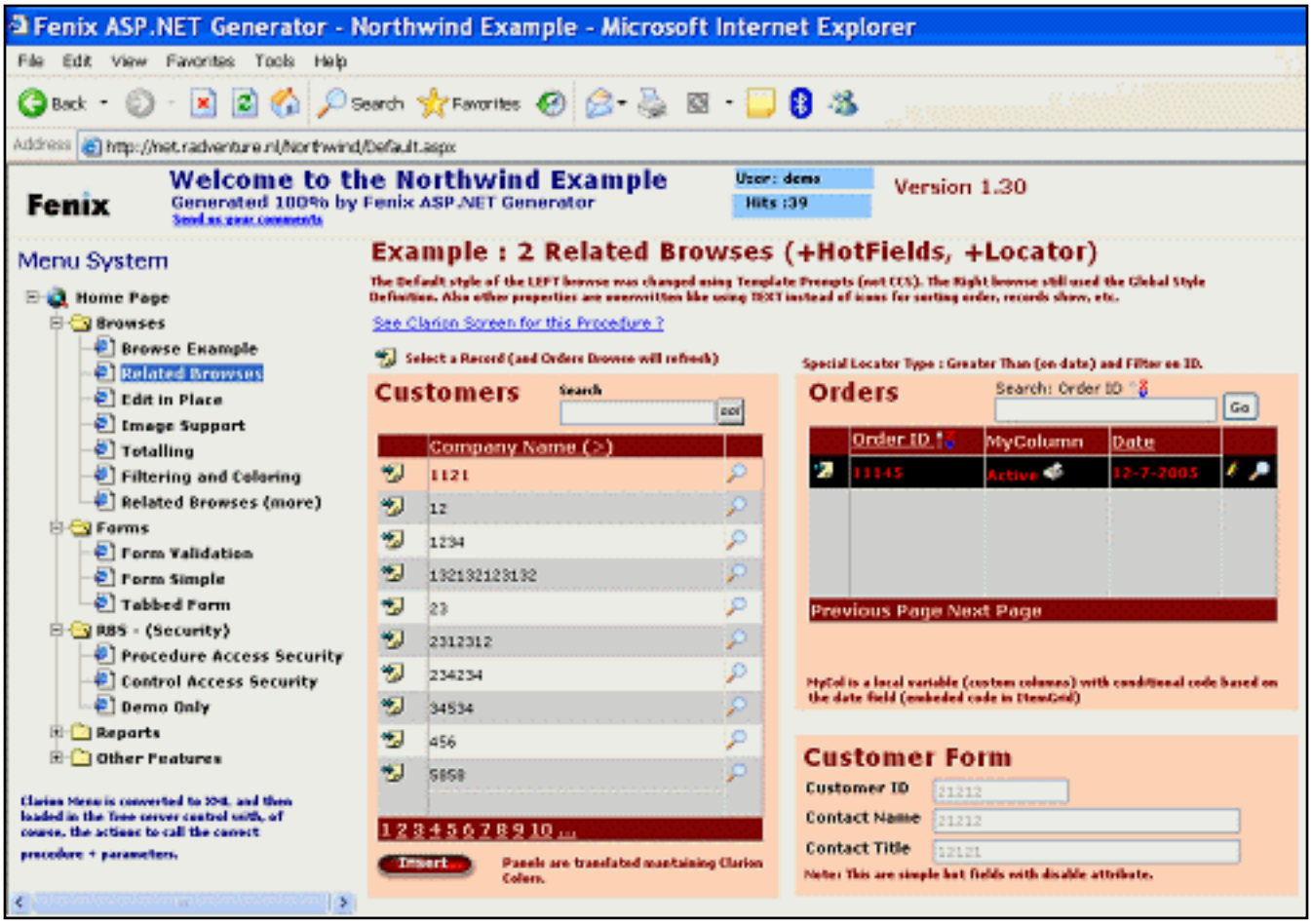


Figure 3. Parent-Child Display (view full sized image)

The alternative paradigm using a panel with buttons within a single frame to provide application navigation simply eliminates the upper and lower left panels on the screen, as shown below. Buttons and URL links are then used in place of the menu to provide access to the application.

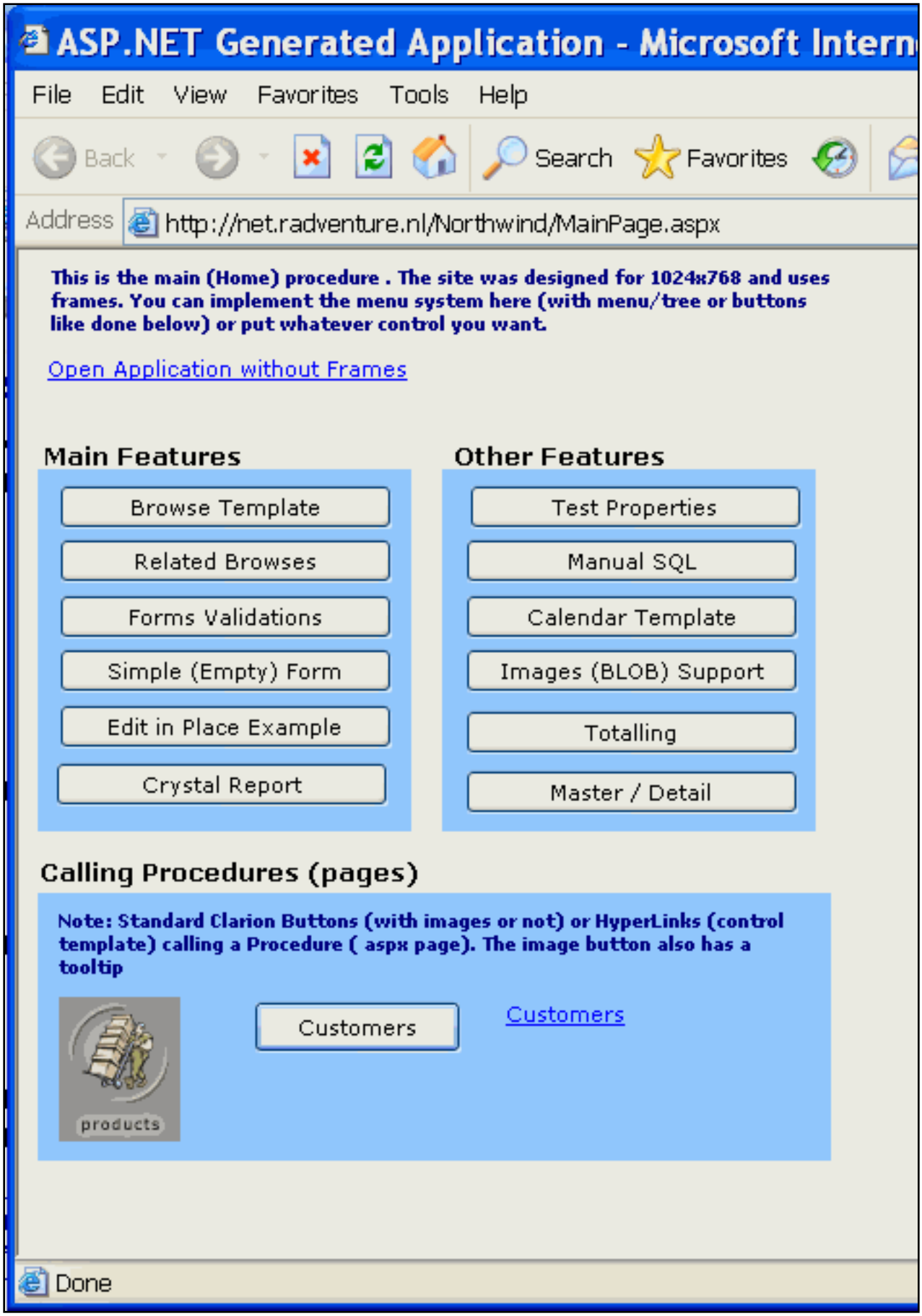


Figure 4. Application without Frames (view full sized image)

Clicking on a button will display the associated function screen, as shown below. A consideration for this type of design is the inclusion of at least some limited navigation controls at each screen level to allow the user to jump from screen to screen, or back to the main page, without having to back-step through the application. Thus, while this approach maximizes vertical and horizontal space, it does so at the expense of ease of navigation.

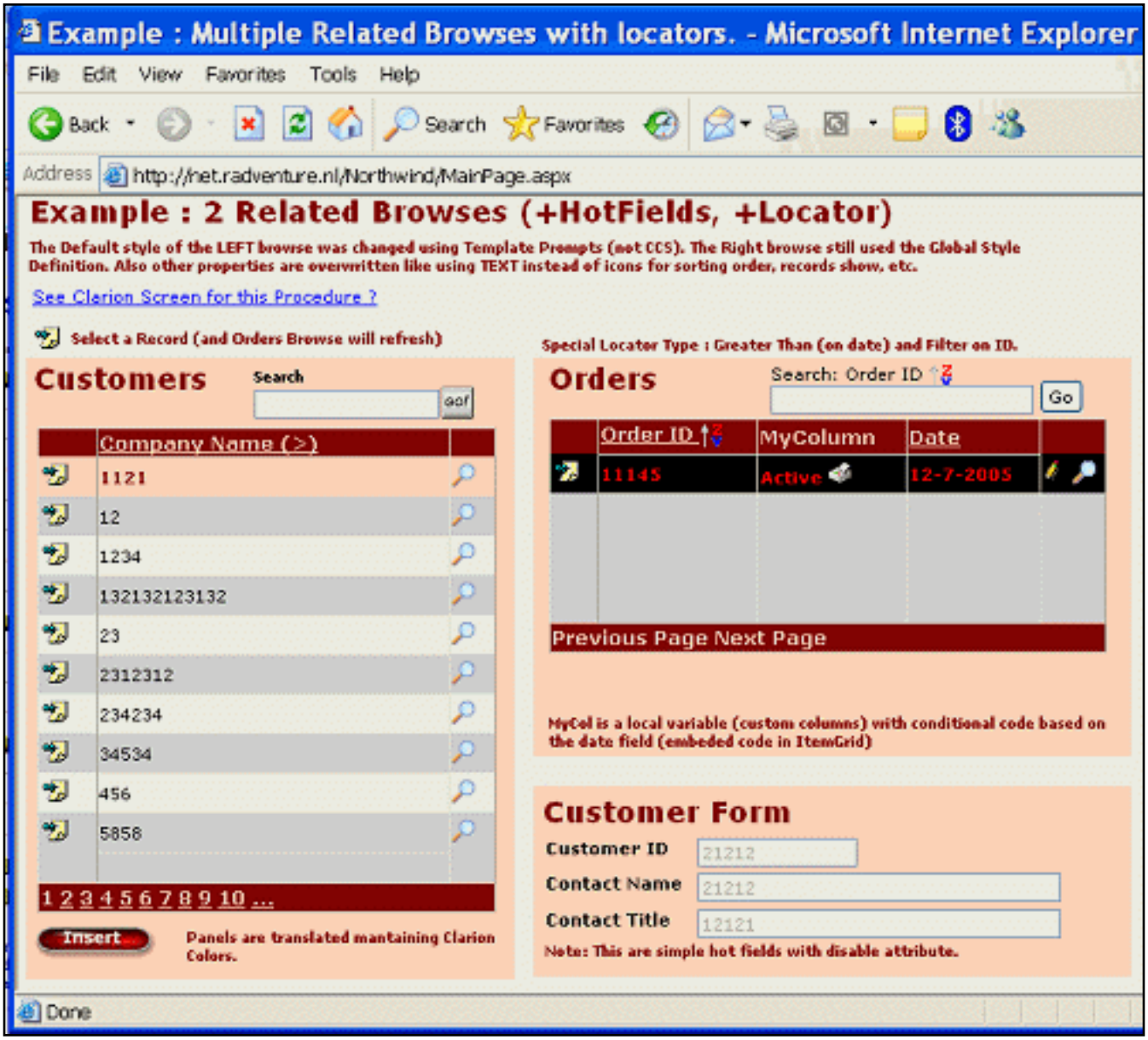


Figure 5. Single Panel Application Display (view full sized image)

Next week I'll continue with a look at security and reporting, and then show how to create a simple application with Fenix.

Ozzie Paez is a systems engineer with over 25 years experience as a programmer and

database architect using a variety of languages including Fortran, Modula 2, Pascal, Basic, Assembly and Clarion. He has been using Clarion since 1987 and became a Certified Clarion Developer in 1999. Mr. Paez's work history includes the US Military, Civil Nuclear industry, Department of Energy's Weapons Complex, Telecommunications, Technical Support and Security Center Management. He currently works as a systems engineer, solutions provider, software quality assurance lead and security researcher for a variety of clients, including the US Air Force. He has published a variety of articles in the US and Europe on issues ranging from business management, corporate leadership, regulatory compliance and quality, to his current focus developing models for conducting threat assessments in the war on terrorism.

Reader Comments

Add a comment

Clarion Magazine

Mixing Clarion with .NET, Part 3

by Wade Hatler

Published 2005-09-21

So far, everything I've shown has had Clarion in the driver's seat talking to .NET. Now it's time to turn the tables and talk back to Clarion from .NET. Calling Clarion 5 and earlier from external procedures of any kind was always pretty dodgy, so I always avoided it in the past and sent all external communications through the `SendMessage` API. With Clarion 6, the problems seem to be resolved (cross your fingers) so it's now easy and seemingly reliable to call Clarion from an external application. All you need to know is how.

You can call a Clarion procedure from .NET or any other language that supports a C or WINAPI style interface pretty easily. You just have to change your Clarion slightly.

The first requirement is that any procedure you call must be in a DLL, and it must be exported just like you would need if you wanted to share it between Clarion projects. I'm not going to detail how to make a DLL project, but if there's a lot of demand I could do it in a follow-up article. For the moment, I've added a project `SharedDll.prj` and associated files that creates `SharedDll.dll` which can be called from any appropriate client. Assuming you know how to make a DLL, all you have to do is export the procedures and make some changes to the prototype in the `map`.

Calling convention

By default, Clarion uses a register calling convention that's very fast and efficient, but non-standard. There are a few other similar techniques out there, but none match Clarion's. To fix this, you'll have to replace Clarion's normal calling convention with the Pascal calling convention.

Most of the time, you'll only be exporting procedures that don't have any overloads. In other words, there's only one procedure with a given procedure name. If this is the case, then it's pretty easy to fix the problem. Assuming you're generating all of your code using some template based paradigm, you just need to add `,PASCAL` to the declaration. For example, you might change:

```
InteropProc procedure()
```

To

```
InteropProc procedure(),pascal
```

If you've created the procedure in AppGen, just put the PASCAL attribute at the end of the prototype entry. In hand code, add it to the MAP.

Most templates should handle everything correctly with just that change. If they don't, you'll have to figure out the problem. Most of the time, if there's a problem it will be because you have a disallowed parameter type, the .EXP file has the wrong declaration, or the declaration in the .CLW file doesn't match the .INC file. Here are a few hints.

1. All exported procedures must be listed in AppName.exp, which is usually generated by a template. This lists all the publicly exported procedures. Yours should be in there.
2. The exported name of the procedure will change when you add the PASCAL attribute. Before doing that, the procedure name will be mangled to include the return type and parameter types. After changing the calling convention to Pascal, the name mangling will disappear and all you're left with is the procedure name in upper case.
   Note: This is the reason you can't use overloaded procedures with the simple version of this technique. If you do need to call overloaded procedures you'll need to use another technique, which I'll discuss that later.
3. The PASCAL entry might have to change in both an .INC and a .CLW file, depending on how your templates generate code. If one or the other is missing or they don't match, then you'll have problems.

For the moment, I've created the simplest of all exports in my SharedDLL.dll project. It has a few public procedures of escalating complexity that I'll use as examples through the rest of this segment.

Plug into .NET

Assuming you now have a DLL with a properly exported procedure, (meaning you can call it from Clarion in another DLL), you need to prototype it in .NET. The system used to do this is called pinvoke, which is short for *Platform Invoke*. It's also called p/invoke. You'll also hear the term *Interop* quite a bit, which is a generic term for interoperating between .NET and the rest of the world. P/Invoke is one subsection of the Interop world. The COM wrapper we used to call into .NET is also a form of interop.

To move data between .NET and the world you have to *marshal* the data. This term means more or

less "convert the data to a form someone else can understand and put it somewhere that they can read it". You've probably already done some marshaling without knowing it. COM uses marshaling to move data around between components, and .NET is just an extended version of the same idea. In the case of .NET, all the native data is managed and garbage collected. When passed to other platforms using p/invoke, it makes a publicly available copy, passes that copy to the external routine (unless you get tricky), and when necessary for call-by-reference semantics, copies the modified data back to .NET.

To make a simple function call, all that's necessary is a simple declaration in one of your .NET source files, and .NET will take care of all the details of marshalling its managed data over to your procedure in a form that it can understand, and taking the return value and any passed-by-reference parameters back.

To call your DLL from any .NET procedure, you just decorate a corresponding method declaration with the proper info. You'll do this by prototyping a method declaration in .NET syntax that uses the `DllImport` attribute. This has the effect of telling .NET where to find the method, what parameters and return types it accepts, and how data should be marshaled. It's very similar to prototyping Windows API calls in your `MAP`, which you're probably familiar with. The `DllImport` attribute combines the two steps you would have in Clarion, namely updating the `MAP` and adding a `LIB` to the project or app file. The `LIB` step that we use in Clarion isn't necessary.

Invoke this method

To make your procedure visible to .NET:

1. Create a *public class* to prototype your method declarations. Everything in C# and most .NET languages has to be in a class. You can declare interop procedures in any class you like, but I think it's good form to have all of your external interface declarations in one place. I even go to the step of putting all Clarion interface logic in its own folder, using one or more source files depending on how many there are. Of course, you don't have to be fanatical about it. If you only need one or two, just prototype them in your main file.
2. Add the following `using` directive to the top of the file. This isn't absolutely necessary, but it will reduce typing later.

```
using System.Runtime.InteropServices;
```

3. Create a `DllImport` attribute that includes the name of the DLL. Within the attribute, set the `EntryPoint` to the name of the procedure in all upper case (the Pascal convention). After the attribute, prototype the procedure using `public static extern`, followed by the C# equivalent of the procedure. A procedure that takes nothing and returns nothing would look like this:

```
        [DllImport("SharedDLL", EntryPoint="PUBLICPROCEDURE")]
        public static extern void PublicProcedure();
```

All of this comes together in a file that looks something like this:

```
using System;
using System.Runtime.InteropServices;

namespace Interop03
{
   public class CWLink
   {
      [DllImport("SharedDLL", EntryPoint="PUBLICPROCEDURE")]
      public static extern void PublicProcedure();
   }
}
```

Calling the procedure

Now you're ready to call this procedure just as if it were a .NET method. For an example, open `ClarionForm.cs`. Go to the `button1_Click` method and add this line after the message box:

```
CWLink.PublicProcedure();
```

If you run the sample and click the button, you'll see a message box delivered from .NET, and a Clarion window opened using `PublicProcedure`. That's all there is to it.

As you can see, it's easy to call simple Clarion procedures from .NET. Next time I'll look at passing parameters, returning values, and overloaded procedures.

Last time I showed how to use .NET's interop services to call simple Clarion procedures from a .NET application. Now it's time to look at overloaded procedures, passing parameters, and returning values.

Overloaded procedures

In the example in the previous installment, I used the `PASCAL` calling convention to change both the name and the calling convention at the same time. If you want to call a procedure that has overloads (meaning multiple procedures with the same name, but different parameters), you'll need to keep the original mangled name, but change the calling convention. You will use Clarion's `NAME` attribute to change the name back to the mangled name. To do that you need the mangled name, and there are a couple of ways to get it. If you have a reliable template that generates an EXP file, you can get it

from the EXP file *before* you add the PASCAL attribute. If not, you can hack it out of the OBJ file. For example, start with this Clarion procedure:

```
TakesLongReturnsLong procedure(long pValue), long
```

If you have a template generated EXP file, open it and look for TakesLongReturnsLong. If you don't have the EXP file, open the OBJ file in a hex editor and search. Either way, you'll find that the public name for that procedure is really TAKESLONGRETURNSLONG@Fl. When you add the PASCAL attribute, it changes to TAKESLONGRETURNSLONG, and the mangling is lost. To get the PASCAL calling convention with the original mangled name, just add a NAME attribute with the original name.

```
TakesLongReturnsLong procedure(long pValue), |
  long, name('TAKESLONGRETURNSLONG@Fl')
```

Naturally, you'll have to paste the same name in the EntryPoint attribute in .NET if you do this.

You can also eliminate the EntryPoint attribute in .NET altogether by changing the Clarion name to what .NET thinks it should be. You could do that with this code:

```
TakesLongReturnsLong procedure(long pValue), |
  long, name('TakesLongReturnsLong')
```

The one thing to watch out for with using the NAME attribute is that your template may or may not generate the EXP file correctly using this approach. Most templates should, but I've seen some that don't, so be warned. If the EXP isn't generated right then you'll get a linker error.

If you get into desperate measures because you can't control how Clarion generates its output and you can't change the template, you can always make a tiny little hand-coded wrapper DLL that does nothing but expose entry points that .NET can consume that calls your Clarion functions correctly.

Passing parameters

Now that you know you can call a procedure, it's time to pass some parameters and return a value. In the sample you'll see a procedure called TakesLongReturnsLong:

```
TakesLongReturnsLong procedure(long pLongVal),long,pascal
```

This is pretty straightforward, except for one thing. A long in Clarion is *not* the same thing as a long in .NET. In Clarion, it's a 32 bit signed integer, and in .NET it's a 64 bit signed integer. A 32 bit integer in the .NET world is either an int, or a Int32 or in some cases an IntPtr. IntPtr

means a 32 bit integer that's a handle, such as hWnd or a pointer like a C++ `*void`. Most interop that uses anything remotely resembling a handle or a pointer uses `IntPtr`.

.NET has data types matching most of the Clarion data types. The chart below shows the conversions between basic Clarion data types.

Clarion	**.NET**	
BYTE	byte or Byte	
SHORT	short or Int16	
USHORT	ushort or UInt16	
LONG, SIGNED	int, Int32 or IntPtr (depends on context)	
ULONG, UNSIGNED	uint or UInt32	
SREAL	float, or Single	
REAL	double or Double	
BFLOAT4, BFLOAT4, BFLOAT8, PDECIMAL, PSTRING	Not supported. Convert them to strings or write a custom marshaler.	
STRING, CSTRING, BSTRING	Use BSTRING. See below	
DATE, TIME	Use custom marshalers described below.	

Note that there are two entries for most basic data types because the first is a keyword that's specific to C#, while the second is the official CLR classname for data type. `Int32` is part of the Compact Language Framework and will work with any .NET language, while `int` is a C# keyword that happens to mean the same thing, but might not be used in another .NET language.

You don't actually have to use this chart in C# if you set up some `using` statements at the top of the interface file. These work much like Clarion's `EQUATE` statement except they only apply to the current file and you can't use anything like an `INCLUDE` to share them. Here are the using statements for the basic data types:

```
using CWByte     = System.Byte;
using CWShort    = System.Int16;
using CWUShort   = System.UInt16;
using CWLong     = System.Int32;
using CWULong    = System.UInt32;
using CWSigned   = System.Int32;
using CWUnsigned = System.UInt32;
using CWSReal    = System.Single;
using CWReal     = System.Double;
```

Once you have this at the top of the file, just declare the Clarion parameters using CW followed by the Clarion data type. Note that C# is a case-sensitive language, so you have to get the case of every letter right.

Once this structure is in place, then the .NET prototype for this function is:

```
[DllImport("SharedDLL", EntryPoint="TAKESLONGRETURNSLONG")]
public static extern CWLong TakesLongReturnsLong(CWLong pLongval);
```

To try it out, just pass a value to the procedure and display what comes back. I added a second button to the sample that looks like this:

```
private void button2_Click(object sender, System.EventArgs e)
{
   MessageBox.Show(String.Format("Received {0} from Clarion",
                                 CWLink.TakesLongReturnsLong(1234)));
}
```

This procedure will pass the value 1234 to Clarion. Clarion displays the value and returns 5678, which .NET proudly displays.

Next week I'll continue with a discussion of string, date, time, and decimal parameters.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Developing .NET Applications With Clarion and Fenix, Part 2

by Ozzie Paez

Published 2005-09-28

Last week I covered most of the basic features of the Fenix .NET development environment for Clarion. I have a few more points to cover and then I'll show how to create a simple Fenix application.

Security

Unlike many client-server applications, ASP.Net applications almost always include some form of security, since they are typically hosted on the Internet or an Intranet. Security can be implemented by using Windows Security to generate a challenge-response request, i.e. the same type used to log on to computers and networks. This paradigm can also be implemented using a variety of credential methodologies from security cards to biometrics, or a combination of these.

At the application level, Fenix uses a simple model that is tightly integrated into ASP.Net, Role Based Security or RBS. RBS defines access to the application and its function in terms of a role, which is assigned to a user. When the user logs on, the application captures his assigned role, which in turn can be used to control everything from access to screens to rights at the individual table level. RBS can be implemented globally at the table/object level, at each individual screen or both. It can also be used to dynamically hide components, such as buttons and menu items. Within this paradigm, roles need to be defined and mapped up front as part of the application requirements. Typical roles include Guest, User, Manager and Administrator, with each being assigned specific privileges by the application. Fenix includes sample applications and detailed descriptions of RBS and its implementation.

The log-in screen included with Fenix (see Figure 6) should feel familiar to anyone who uses

computers. Once the user logs on, the application menu displays, and the user is free to access authorized functions. If a user attempts to access a function to which he does not have rights, a security message is displayed, as shown in Figure 7. Both the log-in and security messages can be fully customized by the client, including changing icons to match client requirements.

Figure 6. Typical Fenix Authentication Login Screen (view full sized image)

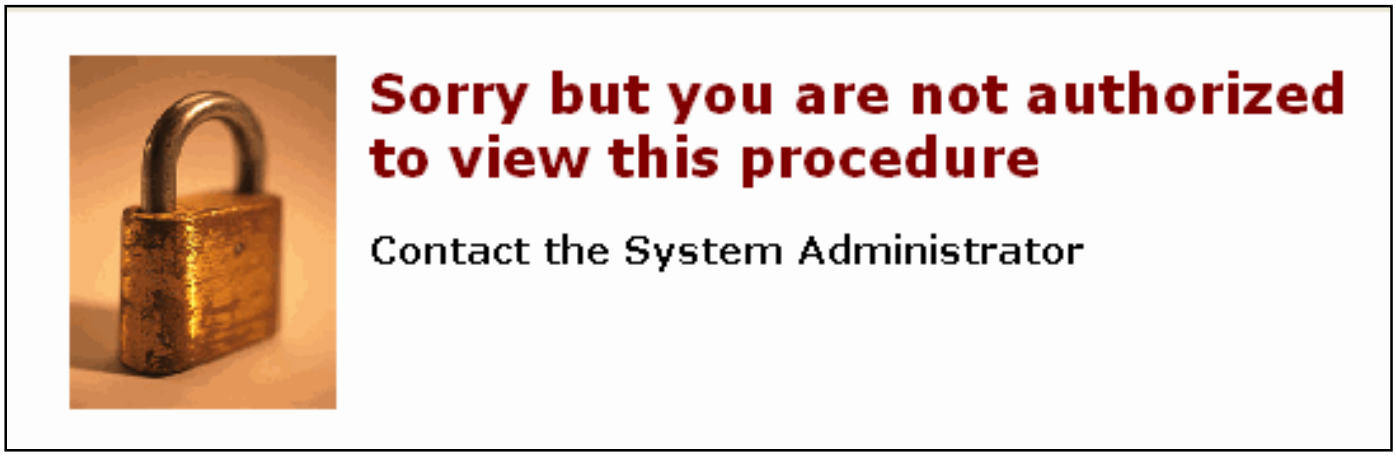


Figure 7. RBS Security Message

External components

Fenix includes templates that support the integration of Crystal Reports, with additional templates being in the works. Using the Crystal Reports templates is straightforward, provided the programmer knows how to install, use and configure CR. Once the specific idiosyncrasies of CR are addressed, developing and deploying reports is fairly simple. This is typical of the ASP.Net environment, where integration becomes key to success. An example of a CR report called from within Fenix ASP.Net application is shown in Figure 8.

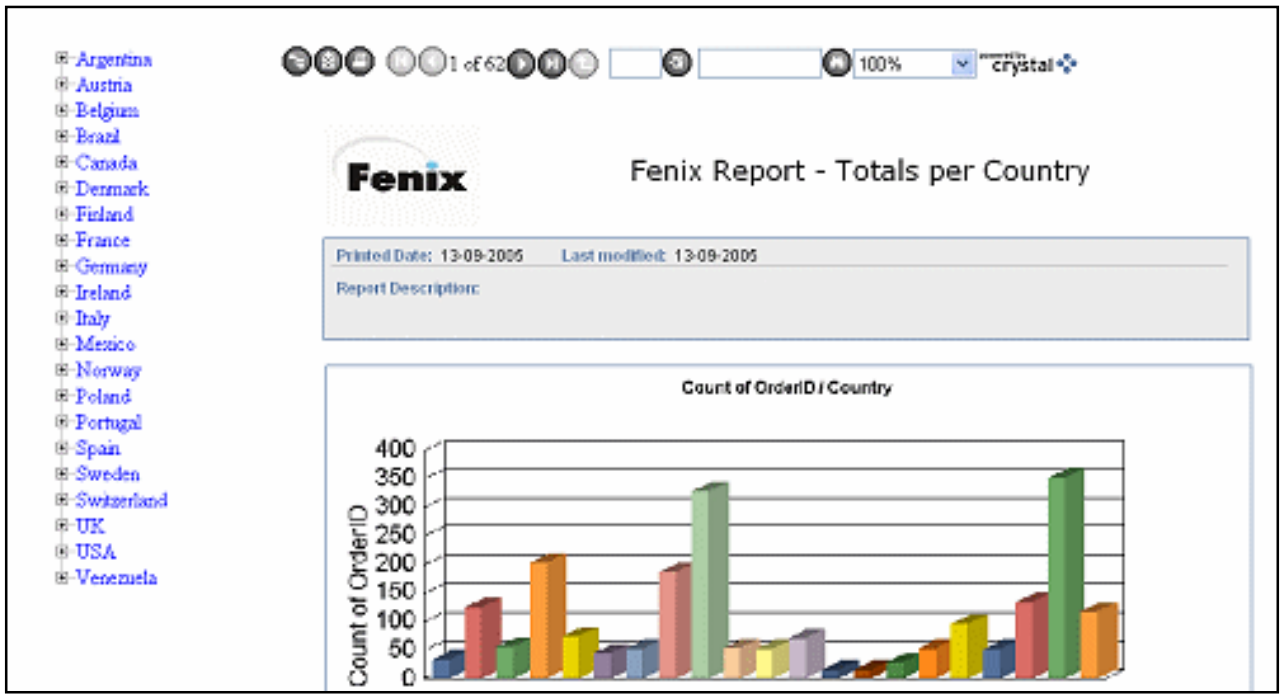


Figure 8. CR Report Called From Fenix Generated ASP.Net Application (view full sized image)

Creating a simple application using Fenix

Creating an application with Fenix actually begins by configuring the development folder and environment. By this time, the .Net Development Framework should have been installed along with the IE Web Controls, so all that remains is to setup the location of the application as follows:

1. Create the development folder for the application.
2. Create a bin subdirectory and install the Fenix and IE WebControl DLLs and components.
3. Create an image directory and copy the images used by the application and controls.
4. Create a web site or virtual directory from within MS Internet Information Server that will point to the development folder.
5. If you are using a backend database such as MSSQL, you will also need to

configure the appropriate users and permissions to support the connection.

The Fenix manual has detailed explanations for items one through four. If you are not used to connecting .Net applications to backend databases, then you will need to do a bit of reading to get item five done. None of this is difficult and is simply part of the price we pay for developing modern applications for the types of clients that are most likely to use our services in the future.

Once the above has been accomplished and a dictionary is available, launch Clarion and create the new application as follows:

1. From the Clarion toolbar select New|Application to bring up the new application dialogue. From this dialogue, select the dictionary and then the ASPX template chain, which is the Fenix template chain, as shown in Figure 9.

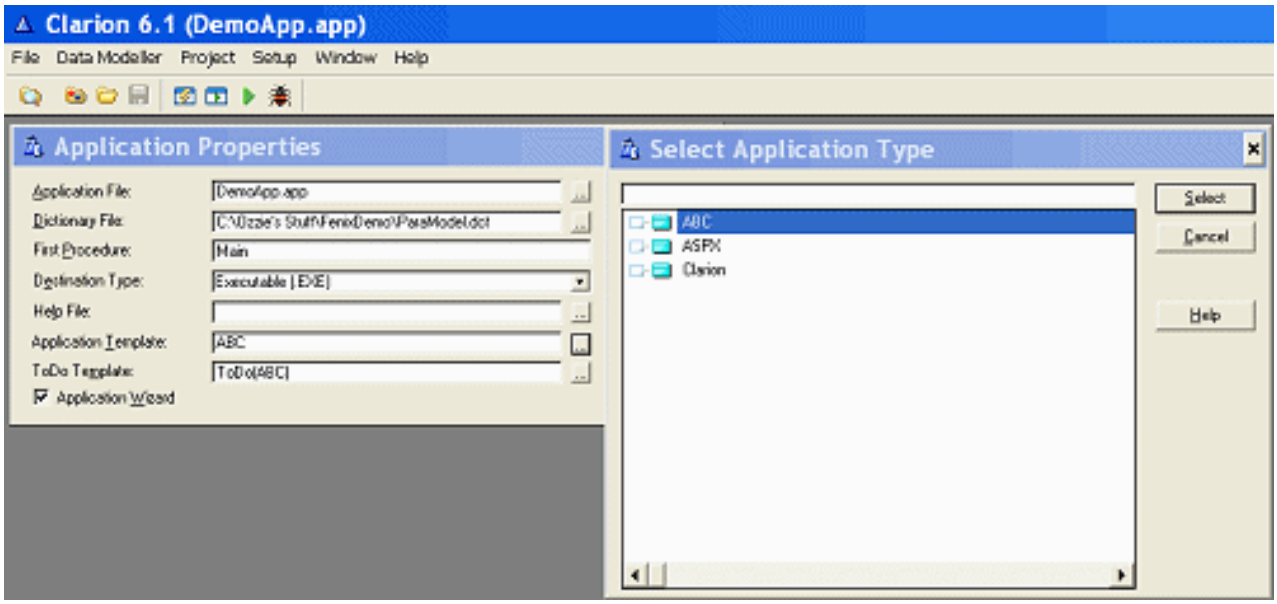


Figure9. Setting Initial Application Properties – Select ASPX as the Application Template Chain (view full sized image)

2. On the next Application Wizard screen, select the target language (VB or C#) and the project type (Figure 10). You will need to supply the wizard with the Project's Virtual Directory Name, which is the Virtual Directory that was created within IIS.

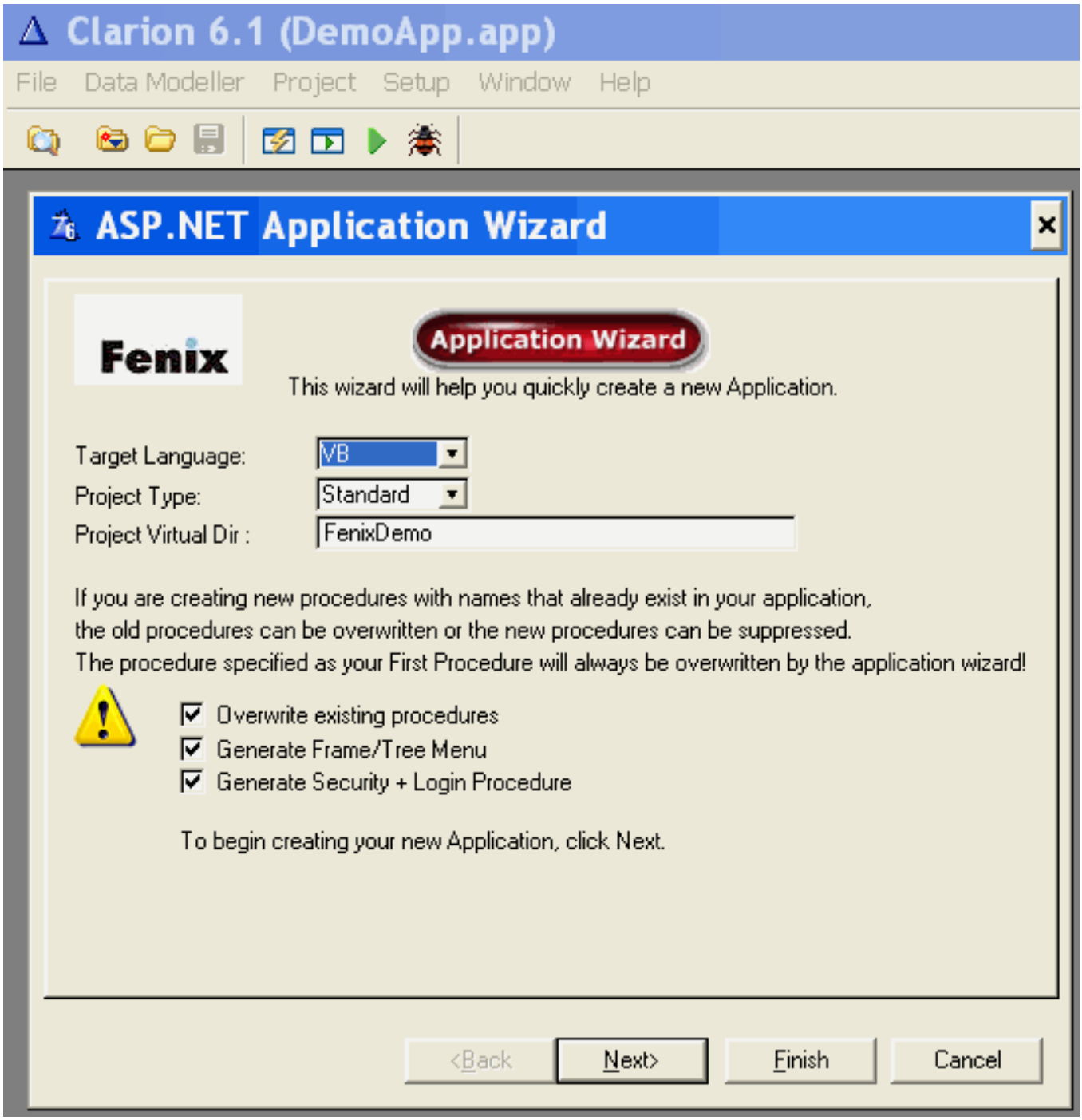


Figure10. Setting Language, Project Type and Virtual Directory

3. Create the connection to the backend database, which must be accessible for the Connection Builder to work. Provide a connection name, which is essentially a string variable containing connection information that will be used when the application connects to the backend database. Pick the target database and version (in Figure 11, Microsoft SQL Server 2000). Finally you can create your own connection string, but the connection builder makes this much easier and it validates that the connection and its availability (don't forget to have the SQL backend running if you are connecting to it). Figure 11 shows the results of running the connection builder and the connection information that it creates for you.

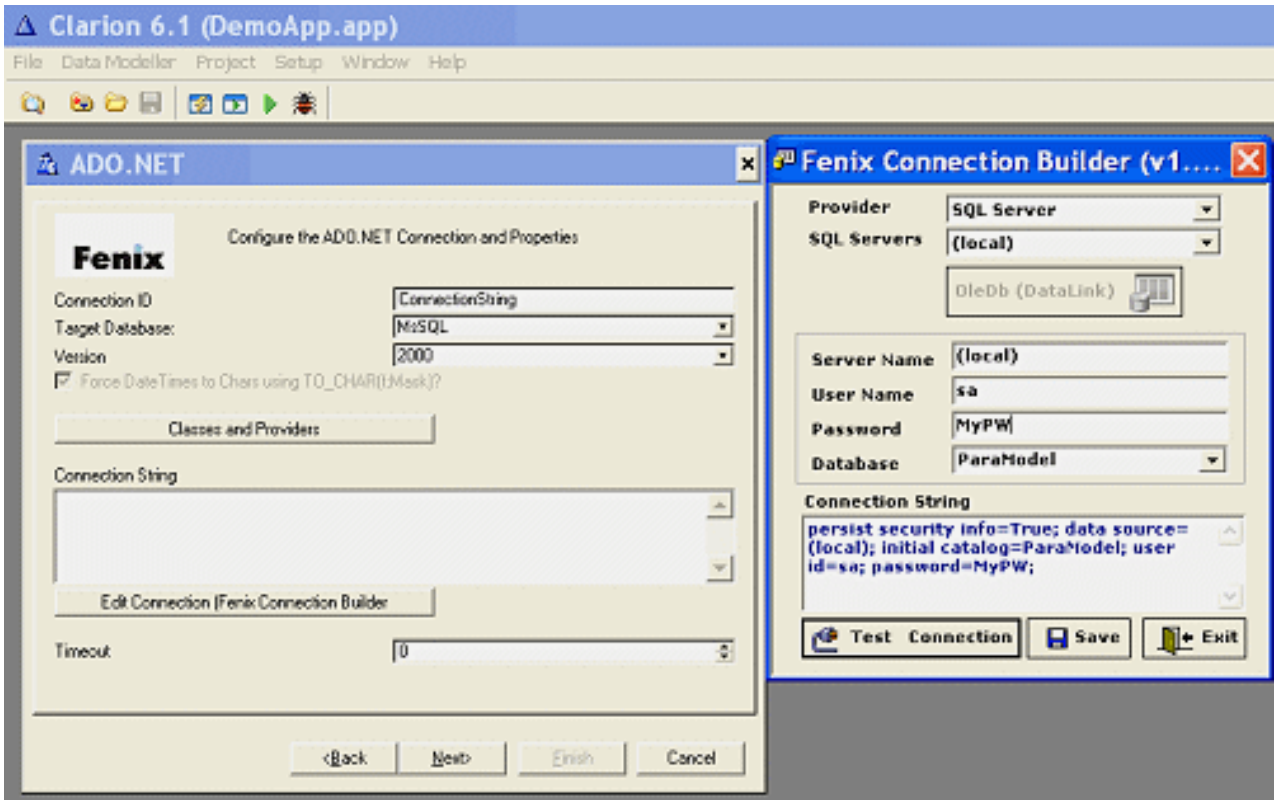


Figure 11. Defining the Database Connection (view full sized image)

4. The final Wizard screen (Figure 12) allows the designer to define which actions will be allowed from a database access and information modification perspective, and will be used to configure the initial screens and controls for the browse-form combination. Just take the default the first time to see what is created.

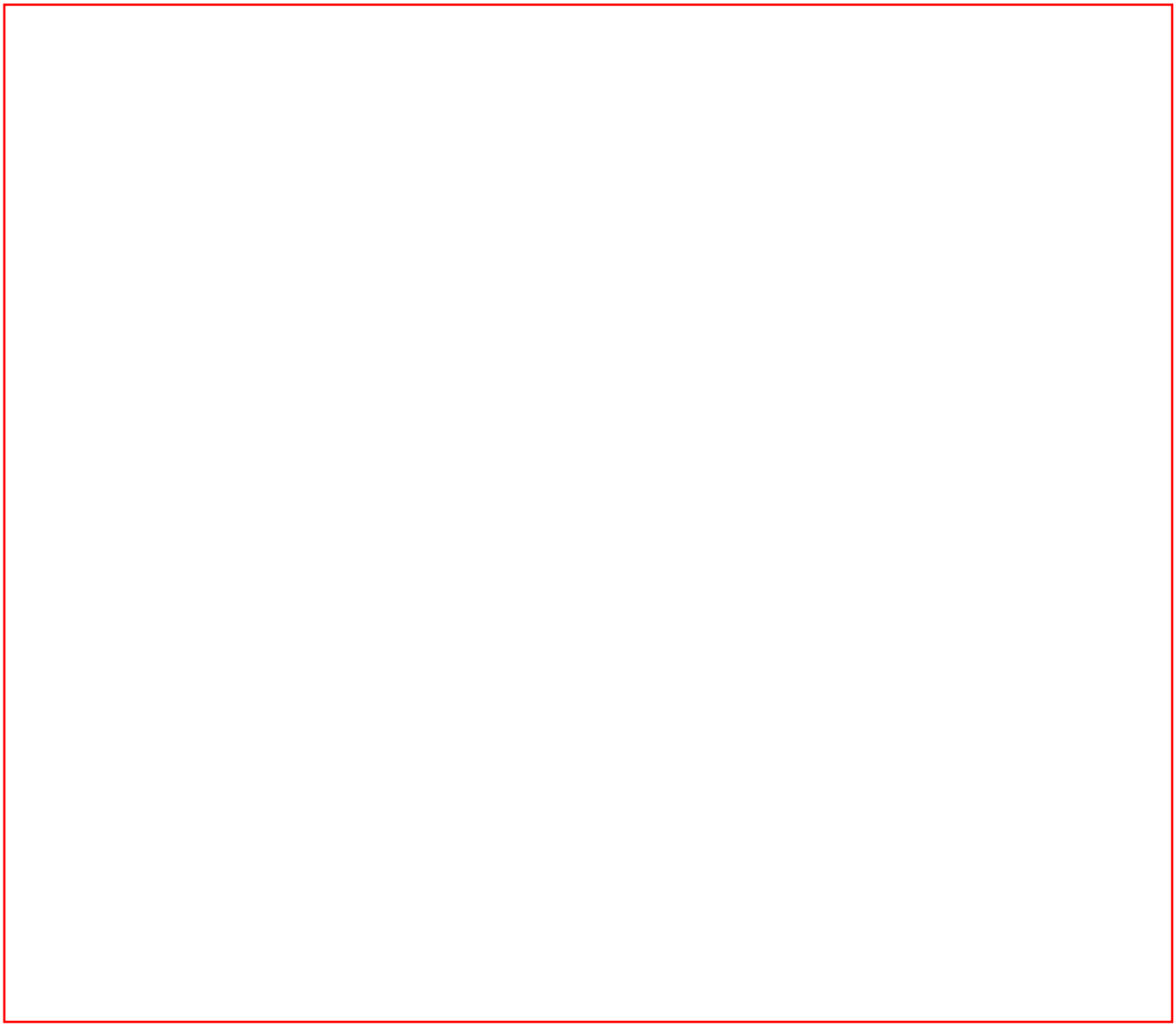


Figure 12. Record Access Configuration

5. Click on Finish to have Clarion generate and compile the application.

Results

The above steps would normally take just a few minutes, once the environment has been configured. The results are not what would be called finished and ready to deliver, but they are certainly good enough to demo flows, controls and have a first, early session with a reasonable client. After that, it will take some work to give the application its own personality and to customize it as necessary, but it is a great, fast and productive start, of the type Clarion programmers are used to. The three screens below show the login, navigation and screens produced by the previous steps.

Figure 13. Fenix Default Login Screen

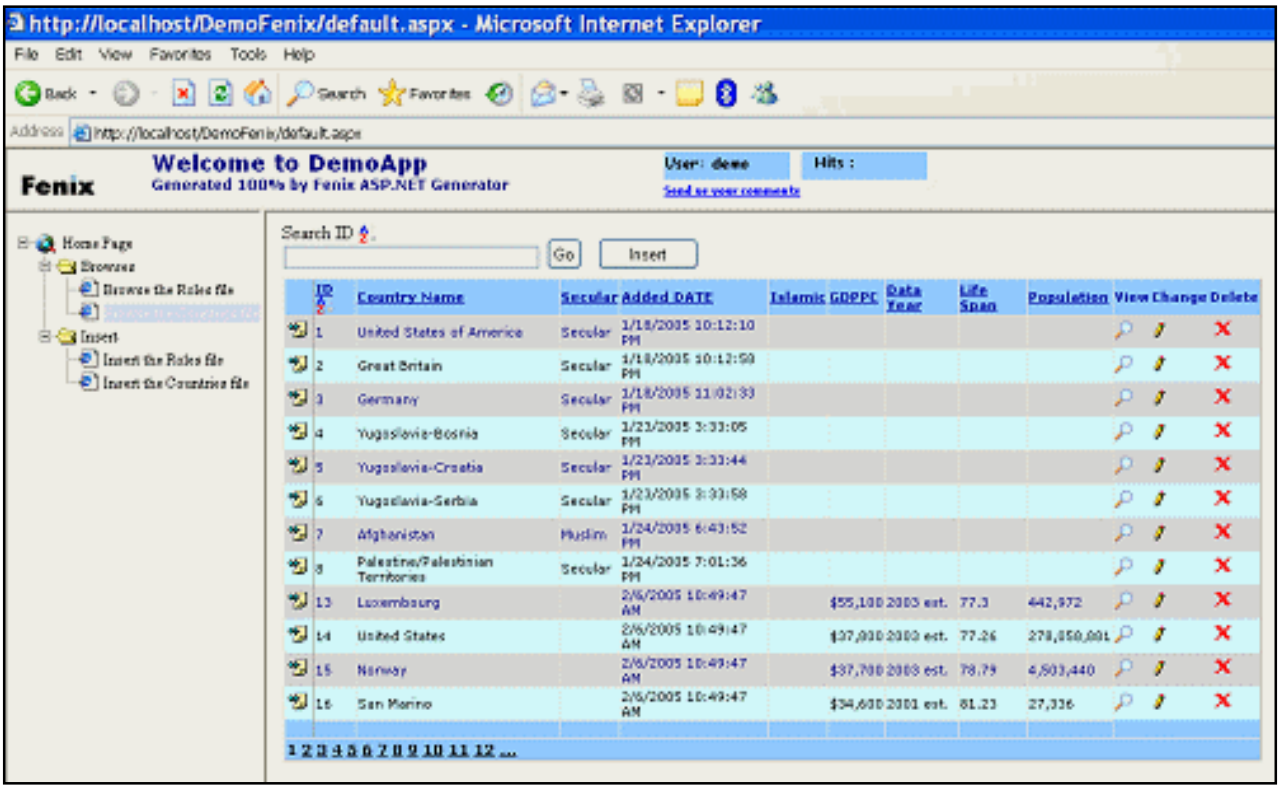


Figure 14. Fenix Default Browse (view full sized image)

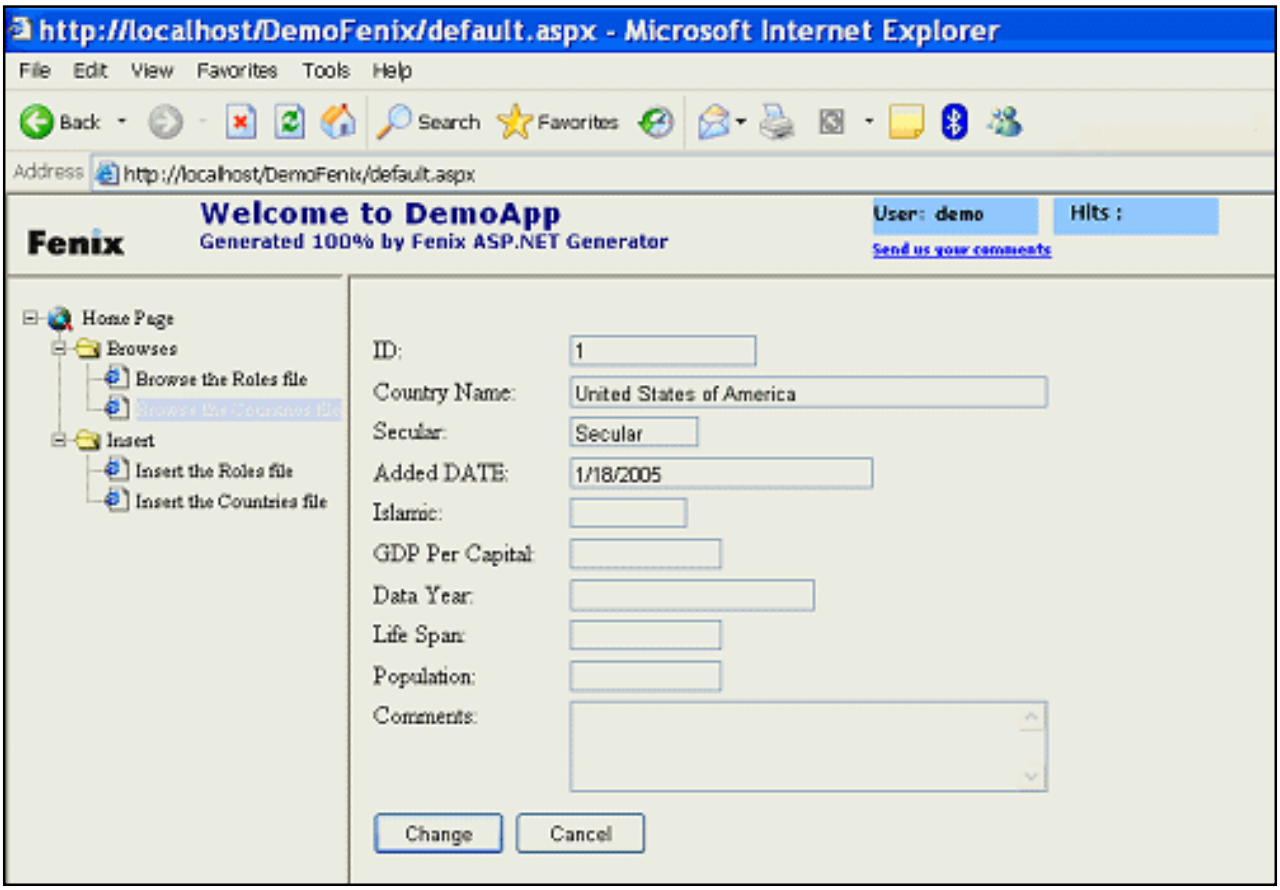


Figure 15. Fenix Default Form (view full sized image)

Support

I cannot stress enough the importance of paying for premium support, particularly if you are new to ASP.Net, SQL databases and using third party .Net controls. Not that RadVenture will train you over email, but they do provide good, targeted support. There are also a number of forums, which provide assistance and an opportunity to collaborate with your fellow Clarion/Fenix colleagues. Based on our experiences, the premium support paid for itself many times over.

Conclusion

This article was intended to provide a starting point for those interested in ASP.Net development using Clarion. So, how does Fenix stack up against other environments, such as Macromedia's DreamWeaver? Quite well for its intended purposes, i.e. process defined, data driven applications. I have used DreamWeaver MX to develop for the US Military and it is a good system, however, it is a web site development system extended to provide data access. Clarion/Fenix is a data access and process architecture system that works within a web site

paradigm. How well it stacks up to Microsoft's own development suite is not something that I can personally answer authoritatively, but based on limited experience, for the purpose intended, Clarion/Fenix is much more productive. More importantly, you can design the application using Clarion/Fenix and maintain it using Microsoft's suite by using a template provided by RadVenture. The RadVenture VsNetProject template essentially creates the project file for either VB or C# that allows the Fenix project to be opened within the Visual Studio (2002 or 2003) IDE. It should be noted that this is a one-way process. In other words, if Visual Studio is used to make changes and the application is later regenerated from Clarion, those changes will be lost.

ASP.Net development is not as simple as the client-server model of old, and not always the best option for many clients; however, the world is going to web-enabled applications, even when they are not the best solution. Many Clarion programmers have been able to get by with desktop applications using TopSpeed and similar file drivers, but those opportunities have diminished significantly, at least for our group. The move to backend databases and web-enabled applications is unstoppable at this point and for Clarion programmers, Clarion/Fenix provides an effective and mature solution that is available right now.

Ozzie Paez is a systems engineer with over 25 years experience as a programmer and database architect using a variety of languages including Fortran, Modula 2, Pascal, Basic, Assembly and Clarion. He has been using Clarion since 1987 and became a Certified Clarion Developer in 1999. Mr. Paez's work history includes the US Military, Civil Nuclear industry, Department of Energy's Weapons Complex, Telecommunications, Technical Support and Security Center Management. He currently works as a systems engineer, solutions provider, software quality assurance lead and security researcher for a variety of clients, including the US Air Force. He has published a variety of articles in the US and Europe on issues ranging from business management, corporate leadership, regulatory compliance and quality, to his current focus developing models for conducting threat assessments in the war on terrorism.

Reader Comments

Add a comment

Clarion Magazine

Mixing Clarion With .NET, Part 4

by Wade Hatler

Published 2005-09-29

Last week I introduced the subject of calling Clarion procedures from .NET. Now it's time to look specifically at how to pass string, date, time and decimal parameters.

String parameters

There are a couple of ways to handle strings in Clarion. I originally worked out a whole complicated method for building a class to convert things back and forth between Clarion `STRING` and `CSTRING` parameters, but then I discovered the undocumented `BSTRING` type. `BSTRING` is the Clarion equivalent of the COM standard `BSTR`, which is what Visual Basic uses natively. Using `BSTRING` for passing and returning values is quick and painless. Here's all you have to do:

1. Change `STRING` or `CSTRING` to `BSTRING` in your `MAP` and in your procedure declaration.
2. Recompile any apps that make use of the procedures you changed.
3. Add a couple of attributes to your declarations in .NET and you're done.

The whole key to passing string as `BSTRING` is the `[MarshalAs(UnmanagedType.BStr)]` attribute. You can apply it to parameters, or return values. Unfortunately, you have to type or paste it in manually for every string declaration, and use one form for parameters and a different form for return values, but it's not the end of the world since it's cut-and-paste of attributes that never change.

The sample has a string procedure that looks like this in Clarion:

```
TakesAndReturnsString procedure(bstring pValue),
    bstring, pascal
```

In C#, the equivalent looks like:

```
[DllImport("SharedDLL", EntryPoint="TAKESANDRETURNSSTRING")]
[return: MarshalAs(UnmanagedType.BStr)]
public static extern string TakesAndReturnsString([MarshalAs(UnmanagedType.BStr)]
string  value);
```

Notice that you need the `return:` marshalling attribute before the function declaration to handle a returned string, and a slightly shorter version for parameters. Attributes are a .NET convention that can either attach metadata to some entity (e.g. add the developer's dog's name to a class), or instructs the compiler to do something. In this case, they're telling the compiler how to marshal the data to the DLL.

You can make it slightly easier to type and read by declaring a constant to shorten up that type declarations. The constant has to be declared within your marshaling class.

```
const UnmanagedType CWBString = UnmanagedType.BStr;
```

Then your declaration becomes the more attractive:

```
[DllImport("SharedDLL", EntryPoint="TAKESANDRETURNSSTRING")]
[return: MarshalAs(CWBString)]
public static extern string TakesAndReturnsString([MarshalAs(CWBString)] string
value);
```

If you find yourself in a situation where you just can't change the Clarion `STRING` parameters to `BSTRING` for some reason, there is a way to make a `STRING` parameter and pass it to Clarion. I'm not documenting it here because I eventually abandoned it for most work, and you can always write a small Clarion wrapper class to act as the external interface, which is much simpler in the end.

Dates and times

.NET handles dates and times differently than Clarion. In fact, .NET handle these types differently from VB and other MS languages as well. .NET doesn't really have a concept of date and time as separate values. Time values are measured in 100-nanosecond units called ticks, and a particular `DateTime` is the number of ticks since midnight on 1/1/0001. This resolution is quite a bit finer than Clarion's (100,000 times the resolution to be exact), although the precision still depends on the source of the time value.

To work with a date as an entity you would just use midnight at the start of that date. To work with time individually, you would work with a `DateTime` object that has a fixed and ignored date.

Clarion can store dates in the `DATE` format, which matches the date format for the Btrieve database, or as a `LONG`. Internally, Clarion always works with dates as a `long`, so if you create a function with a `DATE` parameter, and then use that date parameter for anything, Clarion will convert it to a `LONG`. The only time it's actually consumed in the native `DATE` format is when it's inside of a record buffer in a Btrieve file.

It's quite possible, and not all that difficult to make a converter to work directly with the `DATE` data type, but I didn't bother. It's much easier to just change any date parameters to `LONG` parameters and recompile your application. (I'll leave it at that for now, but I may whip up a native `DATE` processor later).

To send dates back and forth, the numeric representation has to be changed. The change is relatively simple math, but I wanted an easy and invisible way to do the processing. There are three ways to solve the problem, each of which has advantages and disadvantages.

1. Create a wrapper object for every procedure and put the code in the wrapper. You can use ordinary Interop to call Clarion, and then use methods in the wrapper to do the conversion. This is easy to do, but requires quite a bit of untidy cut-and-paste action, which is both annoying and error prone.
2. Create a custom marshaler to do the date conversion. This is an extremely flexible way to move data back and forth between .NET and any other consumer, but they're fairly complicated to write, and using them requires a really long `MarshalAs` attribute. I may need them sooner or later, but that process is overkill for this use.
3. Create a custom `struct` to do the conversion, and use *implicit conversions* to do the heavy lifting.

I generally use option three, but it's a bit of a close call. The first thing to do is create a `struct` to do the heavy lifting. In

C#, a `struct` is similar to a `class`, but it has a few differences. The main difference is that it's a value type, and it can't have a parameterless constructor or destructor. There are several other arcane differences, but the main difference is that the data is created on the stack, and the struct itself is passed on the stack by value, where a class is always a reference type and passed by reference. In Clarion terms, it's sort of a cross between a `group` and a `class`.

This `struct` will use *implicit operators* to do the conversions. The complete `struct` looks like this:

```
public struct CWDateLong
{
   public Int32 _cwDateAsLong;
   const long DATEOFFSET = 657432;

   public static implicit operator DateTime(CWDateLong obj)
   {
      return new DateTime((obj._cwDateAsLong + DATEOFFSET) * TimeSpan.TicksPerDay);
   }
   public static implicit operator CWDateLong(DateTime value)
   {
      CWDateLong result = new CWDateLong();
      result._cwDateAsLong = Convert.ToInt32((value.Ticks / TimeSpan.TicksPerDay) -
DATEOFFSET);
      return result;
   }
}
```

The actual conversions are just math to convert the different resolutions and offsets of the values. The `operator` methods simply define an automatic mechanism for converting data types. To use this class, you just use it in your pinvoke signature like this:

```
[DllImport("SharedDLL", EntryPoint="TAKESANDRETURNSDATE")]
public static extern CWDateLong TakesAndReturnsDate(CWDateLong value);
```

Once this is in place, you can just use the method. It only has one real annoyance. The method returns a `CWDateLong` instead of a `DateTime`. There's an implicit conversion to convert it to a `DateTime`, so assigning it to a `DateTime` like this works fine:

```
DateTime tomorrow = CWLink.TakesAndReturnsDate(DateTime.Today);
```

However, if you want to use it somewhere that doesn't explicitly need a `DateTime` you'll have to cast it to a `DateTime`. For example, you might be tempted to display the return value from Clarion like this:

```
MessageBox.Show(CWLink.TakesAndReturnsDate(time).ToShortDateString());
```

This will produce a compiler error because the method is returning a `CWDateLong` instead of a `DateTime`. There is an implicit conversion to convert it to a `DateTime`, but in this case the compiler doesn't know that you want a `DateTime`. This means you have to cast the result to a `DateTime` to use it. The above statement needs to be changed to this to work:

```
MessageBox.Show(((DateTime) CWLink.TakesAndReturnsDate(time)).ToLongDateString());
```

Casting is very common in C# because of the strongly typed nature of the language (whereas Clarion does a lot of automatic type conversion), but it's annoying. You can fix it one time when creating the interop by writing a small wrapper method that returns what you really want. The code looks like this:

```
[DllImport("SharedDLL", EntryPoint="TAKESANDRETURNSDATE")]
public static extern CWDateLong TakesAndReturnsDate_External(CWDateLong value);
public static DateTime TakesAndReturnsDate(DateTime value)
{
   return (DateTime) TakesAndReturnsDate_External(value);
}
```

This is a bit of a hack, but I haven't figured out a better way. If you do, let me know.

Times

.NET doesn't really have anything that works like Clarion's `TIME`, so there are a bunch of ways to skin the cat. The method I chose is to just use a `DateTime` and ignore the Date portion. Unfortunately, you can't have an invalid date portion so I arbitrarily chose 1/1/0001 as the date I'll use. When sending a `DateTime` to a Clarion `TIME` parameter, the date is discarded anyway. When I get a return value from Clarion, I add that date to the front of it. The only complicated part is the infamous "Clarion Time Off By 1" issue, which means you need to add one to a Clarion time or it doesn't display right. I'm not even going to bother showing that class here, because you can get it from the source code.

Decimals

Clarion and .NET both have decimal data types, but they're completely different animals. A Clarion decimal is a BCD encoded group of digits of a fixed size and decimal placement. .NET decimals are a really long binary number, plus an offset to indicate where the decimal is and another byte for the sign. They both serve the same purpose, but since they don't necessarily have the same resolution, you have to be a bit careful to be sure the variables on each side are compatible in terms of range and precision. It is possible to pass a .NET decimal as a decimal, and then decode it on the Clarion side, but I chose not to do so. A much easier, and more generally applicable method with only slightly worse performance is to simply marshal the data as a string, and that's generally what I do.

Once you've decided to marshal as string, there are two ways to do it. My first approach was to write a *struct* with an implicit conversion, as with dates. In the end, I didn't really like that because, again, it wasn't returning what I really wanted, so I changed my approach completely. My new favorite is to just bite the bullet and write a wrapper class that does what I want in the first place.

The first step is to change your Clarion class to use the old standby, the *bstring*. If you have a procedure that's called infrequently from Clarion, or where performance is never an issue, you can just change the parameter type to string and leave it at that. It seems like a hack, but Clarion will convert your data to a string on the way in, and then convert it back on the way out. The only penalty is a slight drop in performance. Alternatively, you can leave your original procedure alone, and write a Clarion wrapper procedure that does nothing but provide the public interface to .NET.

Once you've converted your Clarion procedure to use *bstrings*, create one *pinvoke* signature for calling Clarion, and another to make your public .NET interface. This pair will look something like this:

```
[DllImport("SharedDLL", EntryPoint = "TAKESDECIMAL")]
[return: MarshalAs(CWBString)]
private static extern string TakesDecimal_Wrapper([MarshalAs(CWBString)] string
value);
public static decimal TakesDecimal(decimal value)
{
   return Decimal.Parse(TakesDecimal_Wrapper(value.ToString()));
}
```

Next time I'll look at passing groups.

Download the source

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

Reader Comments

Add a comment

- » Wade, I am really enjoying this serie, it's really...