

Clarion Magazine

Clarion News

- » Virtual Dedicated Server Sale through July 4th
- » BoxSoft Super CanadaDayWeekend
- » Soft Service Bluetooth Framework
- » Royalty Free PDF Chart Creator Dynamic Link Library (DLL)
- » Clarion Desktop 3.90
- » New Clarion Debugger
- » Firebird Info
- » Ingasoftplus Sixth Anniversary Discounts
- » xReportPreview 3.7
- » Clarion AppStats Beta 1
- » NetTalkCentral.com adds RSS feeds
- » xDataBackup Manager Lite 3.04
- » Bob Foreman Blogs On New Database Browser
- » Window Designer Video
- » PDF-Xchange Build 1.019
- » SuRF Explains New Build System
- » NetTalkCentral.com
- » Help & Manual 4.31
- » CPCS Support Schedule
- » PDF-Xchange Viewer 1.018
- » Seal-Soft Site Back Online
- » Clarion Desktop 3.85
- » J-Skype 1.40
- » Chicago Clarion User Group Meeting June 13
- » Whitmarsh Releases Strategy for Successful Development of Business Information Systems Book
- » InfoZip Wrapper Class Changes
- » A Clarion Love Song
- » Ingasoftplus Anniversary Sale
- » Huenuleufu June Specials
- » J-Cal 1.62
- » Aussie DevCon Saturday Coverage
- » Seal-Soft Domain Problems

[More news]

[More Clarion 101]

Latest Free Content

- » Bob Z's Blog Post On The New Designers, With Full Size Images
- » Source Code Library 2007.05.31 Available
- » Pre-purchase Clarion Tips Vol 4 and save!

[More free articles]

Clarion Sites

- » NetTalk Central

Clarion Blogs

- » Johan van Zyl
- » RADBlog
- » Mark Sarson

Save up to **50% Off ebooks.**
Subscription has its rewards.



Latest Subscriber Content

Pre-purchase Clarion Tips Vol 4 and save!

Clarion Tips & Techniques Volume 4 is now in final editing - **pre-purchase** before the book goes to print and **save up to \$20**. Topics in our latest collection of ClarionMag gems include: browse and form handling, commonly-used Clarion embeds, writing templates, understanding threading issues, controlling printers, using SQL, getting the most out of Vista, managing source code with version control, creating DLLs, calculating dates, using finite state machines, and much much more! **Details...**

Posted Wednesday, June 06, 2007



Rendering Text and Images on Vista Glass

In his previous article on Vista Glass Randy Rogers demonstrated Clarion code to show the background behind the client area. In this installment Randy explains how to render text and images on a Vista Glass surface.

Posted Friday, June 29, 2007

Finding Duplicate DLLs

Different versions of same-named DLLs are the bane of Windows developers. Maarten Veenstra builds on some DLL version code published earlier in Clarion Magazine and creates a utility to find duplicate DLLs.

Posted Thursday, June 28, 2007

The ToClipboard Class

Users commonly interact with data via update forms, and they tend to think of that data the way it is formatted and laid out on the form. And users often want to transfer this information to another person or program. Tim Phillips presents a local class to copy table data to the clipboard.

Posted Tuesday, June 26, 2007

Using A Browse Popup Menu For Lookups

The standard way to handle an update is to select the record, bring up the form, make the changes, and save. But sometimes a faster method is called for: Murray Gillespie shows how to use a browse popup menu for quick lookup-based updates.

Posted Tuesday, June 26, 2007

Bob Z's Blog Post On The New Designers, With Full Size Images

Bob Z has posted a blog entry with a bunch of screen shots showing the new designers for WinForms, ASP.NET, and the Compact Framework. This article is a duplicate of that blog entry with two additions. First, I've inlined the full size images; in Bob's blog entry you have to click on the thumbnails. Second, I've added a few comments to the text. You will need to view the article in printer-friendly mode to see the images in their entirety.

Posted Friday, June 15, 2007

Converting A Process To A Stored Procedure/Trigger

If you're migrating to (or already using) SQL, you may discover that you have some lengthy processes that can be easily converted to much faster and more reliable stored procedures/triggers. Bernie Grosperin shows an example he came across when converting some code from TPS to PostgreSQL.

Posted Friday, June 15, 2007

CapeSoft World Tour Las Vegas

Brian Reid and John Rae attended the CapeSoft World Tour event in Las Vegas, and report back on a week packed with Clarion development help and advice.

Posted Wednesday, June 13, 2007

Kalashnikov Programming

Inspiration for a programming style can come from the oddest of places, notes Tim Phillips.

Posted Monday, June 11, 2007

Source Code Library 2007.05.31 Available

The Clarion Magazine Source Code Library has been updated to include the May source. Source code subscribers can download the January-May 2007 update from the [My ClarionMag](#) page. If you're on Vista please run Lindersoft's [Clarion detection patch](#) first.

Posted Wednesday, June 06, 2007

Pre-purchase Clarion Tips Vol 4 and save!

Clarion Tips & Techniques Volume 4 is now in final editing - **pre-purchase** before the book goes to print and **save up to \$20**. Topics in our latest collection of ClarionMag gems include: browse and form handling, commonly-used Clarion embeds, writing templates, understanding threading issues, controlling printers, using SQL, getting the most out of Vista, managing source code with version control, creating DLLs, calculating dates, using finite state machines, and much much more! **Details...**

Posted Wednesday, June 06, 2007



[\[Last 10 articles\]](#) [\[Last 25 articles\]](#) [\[All content\]](#)

Source Code

The ClarionMag Source Code Library

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

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

[More info](#) • [Subscribe now](#)

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



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

From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

ISSN

Clarion Magazine's ISSN

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

About ISSN

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

Clarion Magazine

Clarion News

[Search the news archive](#)

Clarion Desktop Hits 400 Users

Clarion Desktop now has 400 users and is supported by 148 products from 32 suppliers.

Posted Friday, July 06, 2007

Clarion Desktop 3.95

Clarion Desktop 3.95 has been released. This will most likely be the last version 3.x release.

Posted Friday, July 06, 2007

J-Fax 1.54

J-Fax 1.54 has been released. Changes include: Optional error log file; Fixed problem when faxing single page Clarion reports on Vista; New virtual method called FaxSent, useful for logging etc.;

Optional confirmation message when faxes are sent.

Posted Friday, July 06, 2007

AppStats 1.04 Movie

AppStats 1.04 is now available, along with a movie showing the new user interface. You can now view all controls for a particular error, or all errors for a particular control.

Posted Friday, July 06, 2007

awDebugger 1.0.2

awDebugger v1.0.2 is a maintenance release focused in solving most of the issues reported so far, specially those related to stepping into source code. 30 day free trial available.

Posted Friday, July 06, 2007

AppStats Beta 2

AppStats Beta 2 no longer uses a Clarion template (please delete j-appstats.tpl if you installed it with Beta 1), as it now imports TXAs instead. This is not a runtime spell-checker like J-Spell. This tool is used to spell-check your APP before you release it.

Posted Friday, July 06, 2007

Smart-Type Autocomplete Template

Smart-Type is an autocomplete type template for use in Clarion ABC. It acts on every word typed, not just the first word. The current release works on text boxes and entry fields. An upgrade due shortly will add edit-in-place support. Smart-Type is available on Trial Pay. Trial Pay is a new payment method for online transactions which allows customers to pay for products by trying or buying from well known advertisers. You buy something else you want, and you get Smart-Type free.

Posted Friday, July 06, 2007

SetupBuilder 6.5 Build 1911

Lindersoft has announced SetupBuilder 6.5 Build 1911, a maintenance release. Lindersoft strongly recommends customers upgrade to the latest version of SetupBuilder 6.5 as soon as possible to maintain the highest level of support, performance and reliability. If you have a current SetupBuilder Maintenance and Support Subscription Plan, the update is free of charge. You can get the latest version by selecting "Check for Updates" from within the SetupBuilder 6.5 IDE. You can get the latest documentation by selecting "Check for Documentation Updates" from within the SetupBuilder 6.5 IDE.

Posted Friday, July 06, 2007

Encourager Software C7 Blogs

Encourager Software's Clarion 7 blog section highlights products that compliment C7 and Vista, such as LinderSoft's SetupBuilder 6.5 Developer Edition and the Mallorca Collection by 1st Icon Design.

Posted Friday, July 06, 2007

amazingGUI Web Site Updated

The amazingGUI web site has been updated with images showing the new jelly buttons style. amazingGUI is pure GDIPlus graphics, no pasted images. Available on Clarion Shop for \$119.

Posted Friday, July 06, 2007

Scheduler/Calendar Class Demo

A new demo of Roel Abspoel's scheduler/calendar class is now available for download.

Posted Friday, July 06, 2007

SimPageOfPage IceTips Compatible

The SimPageOfPage Template is now compatible with the IceTips Report Previewer, so people using IceTips Report Previewer can now add page of page, alternate footers and different footers to their reports. This applies to Clarion 5, Clarion 5.5, and Clarion 6.x, but not to Clarion 4. The SimPageOfPage Template is \$19 US, available from ClarionShop

Posted Friday, July 06, 2007

Virtual Dedicated Server Sale through July 4th

For a limited time Oak Park's NetTalk Webserver-ready virtual servers are on sale for \$29.99 per month with no long term commitment. Virtual Dedicated Servers include Admin access - install and run virtually anything on the server. You can manage multiple web sites and host multiple web sites on one server account, and a dedicated server/virtual dedicated server can be used for a wide variety of purposes, including gaming, virtual (i.e., shared) hosting, and hosting of traffic-intensive web sites. For those interested in these servers for Nettalk Webserver applications you will want to select Option 2: The Build Your Own Virtual Dedicated Server and add the Windows Server 2003 Enterprise Edition Operating System in Step 1, along with any other upgrades to your system that you wish.

Posted Wednesday, June 27, 2007

BoxSoft Super CanadaDayWeekend

The BoxSoft office will be closed from Jun 28th through July 2nd.

Posted Wednesday, June 27, 2007

Soft Service Bluetooth Framework

Soft Service Company has launched Bluetooth® Framework - a VCL and ActiveX components library which allows software developers to add Bluetooth®, IrDA®, Serial and ActiveSync® support into their applications. The feature-rich Bluetooth® Framework enables software developers to save up 70% of their working time and create applications mobile devices in the shortest possible time. With SilentAuth™ authentication requests via PIN from Bluetooth®-enabled devices are handled without any user actions on PC side. SmartDetect™ technology, implemented in Bluetooth® Framework, automatically determines the best way to connect a cell phone to deliver GSM modem functionality and manage file transfer. MultiAPIs™ technology enables efficient usage of multiple Bluetooth® adapters installed on a single PC. Demo available.

Posted Wednesday, June 27, 2007

Royalty Free PDF Chart Creator Dynamic Link Library (DLL)

The PDF Chart Creator DLL gives you full control over all aspects of the chart including page size, colors, axes, titles, labels, positioning, legends etc. All you have to do is supply the data and optionally change the default settings and the chart automatically adapts to fit the page based on your settings. Because the charts are produced as PDF they are scalable and resolution independent so they can be viewed and zoomed without any "blockyness" or "jaggies" appearing. They also look great printed even at very high printer resolutions. Each PDF Chart Creator document is typically around 3Kb. Clarion interface included.

Posted Wednesday, June 27, 2007

Clarion Desktop 3.90

Clarion Desktop 3.90 has been released. You can download the update or use the built-in "Check for Updates" option.

Posted Wednesday, June 27, 2007

[New Clarion Debugger](#)

awDebugger is a new Clarion debugger with the following features: Fully customizable Clarion7-like user interface; Standard and Conditional Breakpoints; Standard and Conditional Tracepoints; Immediate Window (a Clarion expressions interpreter); Syntax highlighting on both source and assembler code viewers; More "Step" functions; Breakpoints & Tracepoints lists; Ability of disabling breakpoints & tracepoints; Ability to debug a running process; Bookmarks; Enhanced data viewers. A 30 day trial version is available. awDebugger costs \$299, but if you place your order before July 9th you pay \$225.

Posted Wednesday, June 27, 2007

[Firebird Info](#)

For those who are interested in Firebird, Kelvin Chua suggests this page of summarized links.

Posted Wednesday, June 27, 2007

[Ingasoftplus Sixth Anniversary Discounts](#)

Ingasoftplus celebrates the sixth anniversary of its professional activity with festive prices and discounts to its customers. During June all product prices are discounted by 15%.

Posted Wednesday, June 27, 2007

[xReportPreview 3.7](#)

New in xReportPreview 3.7: Print Settings window: New SetOfficeXPToolbar and GetOfficeXPToolbar for CFCPreviewReport class.

Posted Wednesday, June 27, 2007

[Clarion AppStats Beta 1](#)

Clarion AppStats (I'm still looking for a better name) can spell-check your APP before you release it to your clients. It spell-checks your controls, your menus, your tooltips, etc.

Posted Wednesday, June 27, 2007

[NetTalkCentral.com adds RSS feeds](#)

RSS feeds have been added to all the forum boards on NetTalkCentral.com.

Posted Wednesday, June 27, 2007

[xDataBackup Manager Lite 3.04](#)

xDataBackup Manager Lite v3.04 is now available. Features include: All source, no back box DLLs; Uses standard TRN file for localization; Set/change font, color, height, etc; Assignable hotkeys; New icons; Bug fixes. Updated demo also available. The price has been changed to \$99. Registered users of

xDataBackup Manager Lite can upgrade for \$39.

Posted Wednesday, June 27, 2007

Bob Foreman Blogs On New Database Browser

Bob Foreman has posted a blog entry showing screen shots and features of the new C7 database browser.

Posted Tuesday, June 19, 2007

Window Designer Video

Bob Zaurere has posted a video showing some features of the new Window Designer in C7.

Posted Tuesday, June 19, 2007

PDF-Xchange Build 1.019

PDF-Xchange Build 1.019 adds better CJK font substitution, improved JS support, some features extensions, minor issues corrected and UI improvements.

Posted Tuesday, June 19, 2007

SuRF Explains New Build System

Scott Ferret has posted a blog entry on the new build system in Clarion 7/Clarion.NET. This system is an extension of the XML-based MSBuild system that is part of .NET 2.0.

Posted Tuesday, June 19, 2007

NetTalkCentral.com

John Hickey has set up a web site for users of CapeSoft's NetTalk. Features include a forum and file upload/download areas.

Posted Tuesday, June 19, 2007

Help & Manual 4.31

EC Software GmbH has released Help & Manual 4.31. This is a minor update with a few visible changes, but a number of user-reported issues have been fixed.

Posted Tuesday, June 19, 2007

Clarion Magazine

Bob Z's Blog Post On Designers, With Full Size Images

Published 2007-06-15

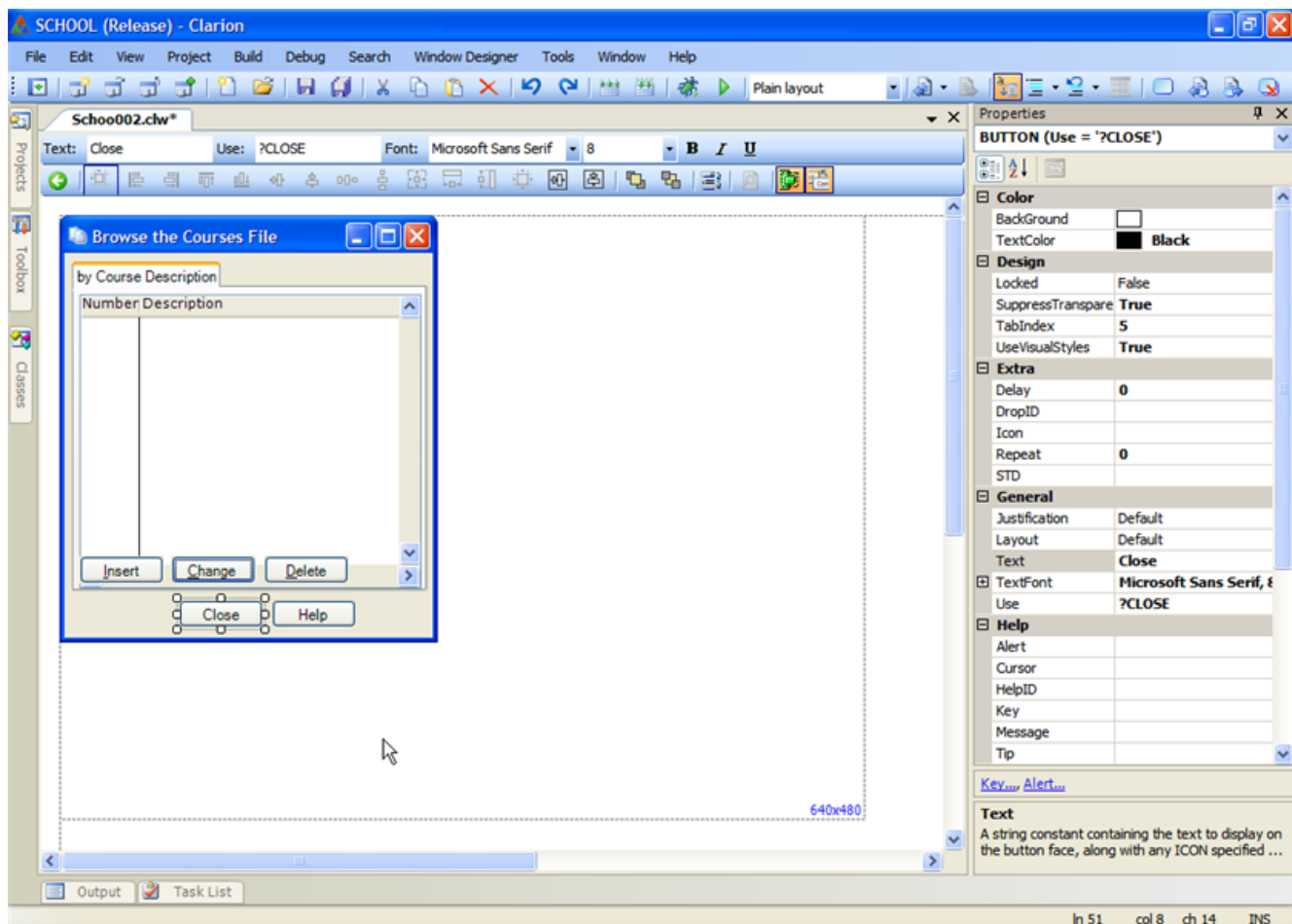
NOTE: The following text is taken verbatim from Bob Z's [blog post](#). I've reproduced it here with the full size images included for easier readability. You will, however, need to view this article in [printer-friendly mode](#) to see the images in their entirety. I've also added a few comments to the text.

Text and images by Bob Zaubere, comments by Dave Harms

Between Clarion 7 and Clarion.Net we have 5 new Designers, there are Designers for Windows, Reports, WinForms, Compact Forms, and Web Forms. Both the Window and Report Designers are new for Clarion 7. The Report Designer is shared by Clarion 7 and Clarion.Net. In other words both platforms; Win32 and .Net can use and share the same Reports.

Early on we made a design decision to ensure that all the Designers would share the same User Interface, so anyone who learns one Designer will have the same experience with any of the other Designers.

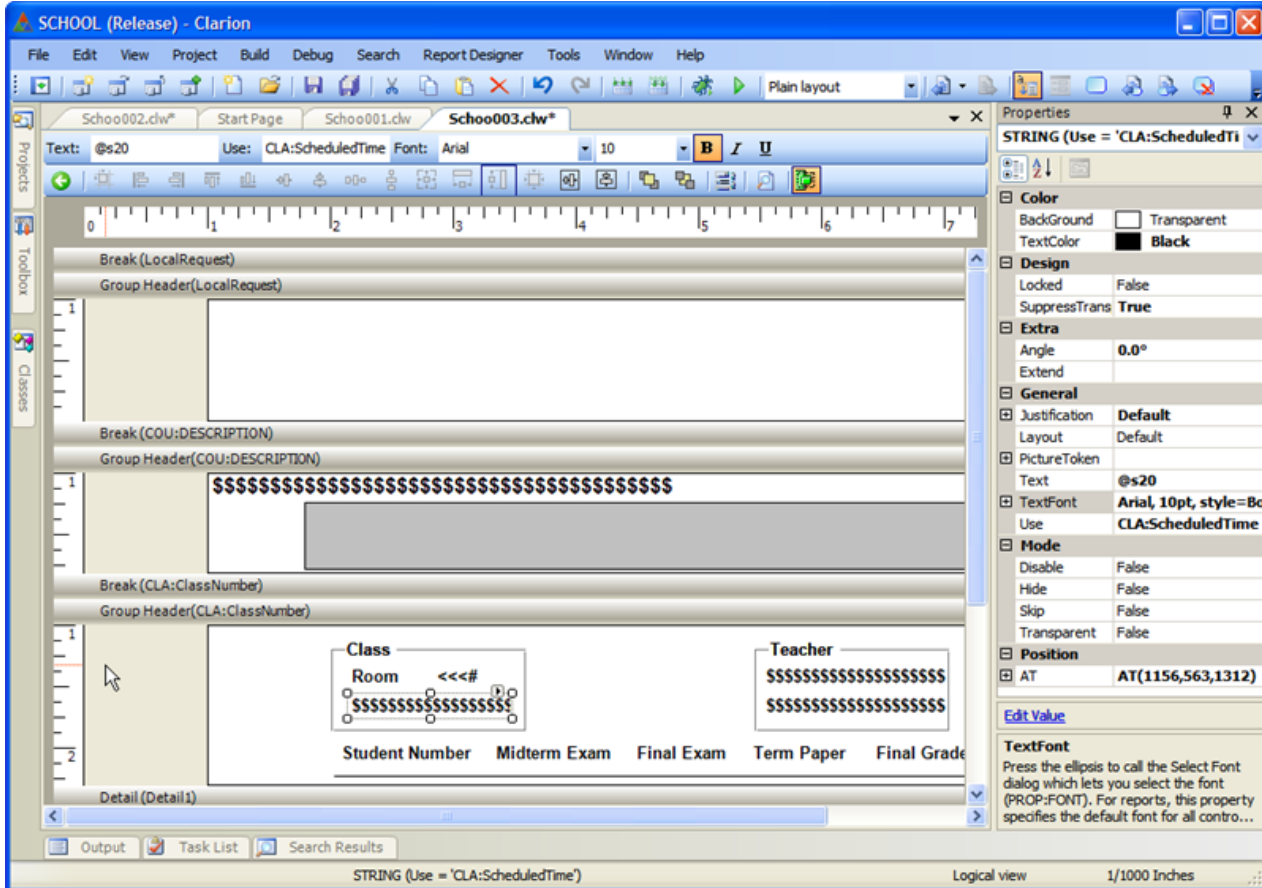
Here is a screen shot of the Window Designer for Clarion 7, and remember that it can also be used to design and modify Windows created for any version of Clarion for Windows.



This is the formatter we've had in the first Alpha, and which has been quite stable.

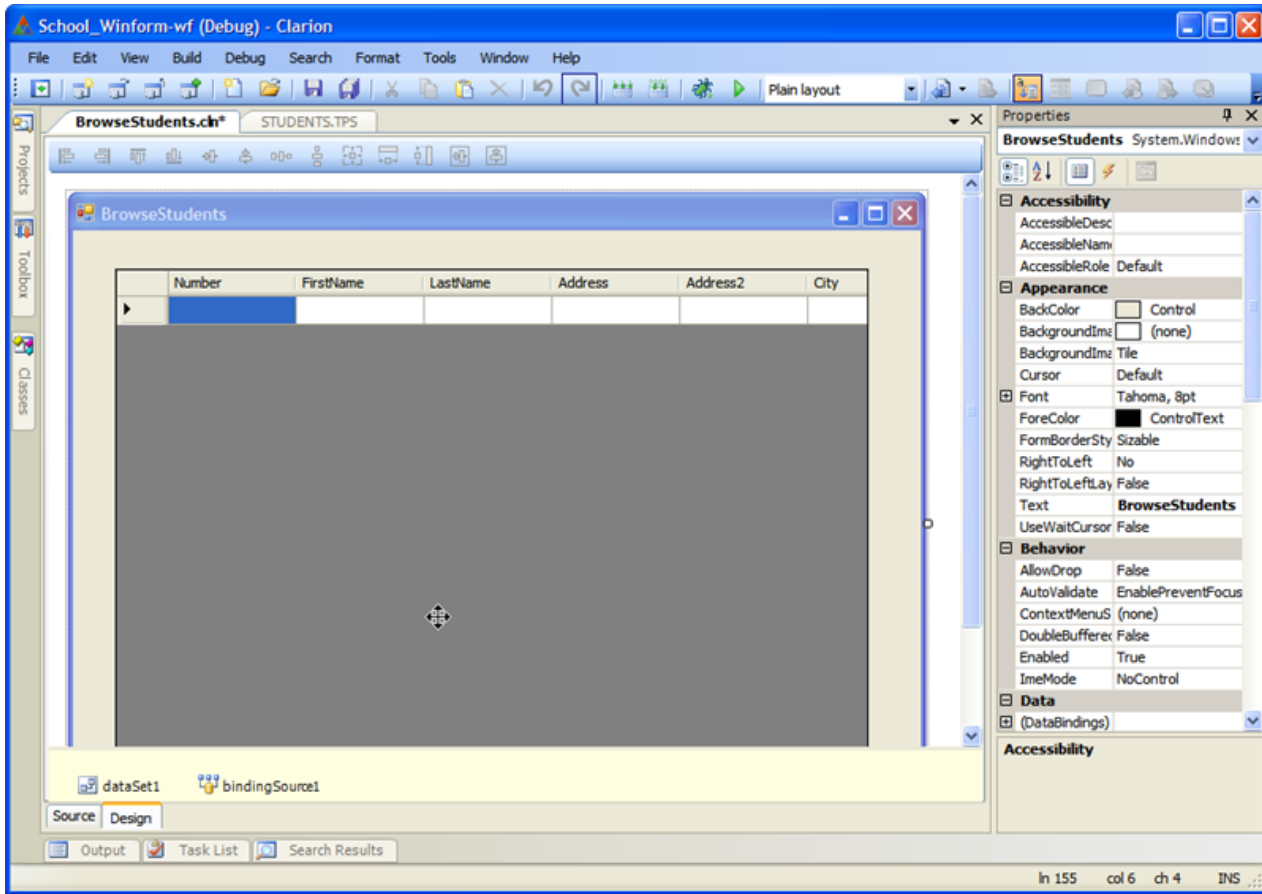
Let's have a look at each of the other Designers, and then we can focus in on the common elements of the Designer UI.

Here's a shot of the Report Designer shared by Clarion 7 and Clarion.Net. Again, take note that the Report Designer can also be used for work on Reports created for any version of Clarion for Windows.



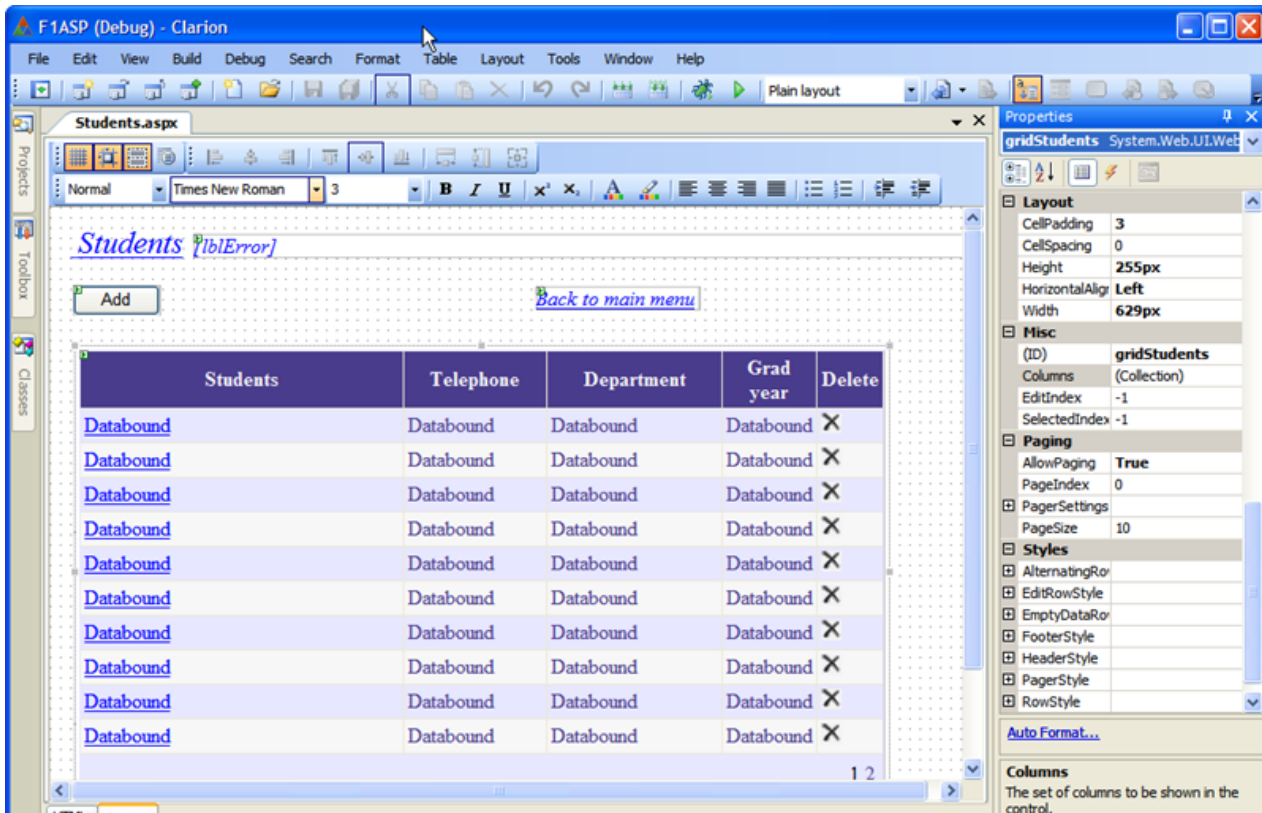
The report designer is also in the first alpha - nothing new here.

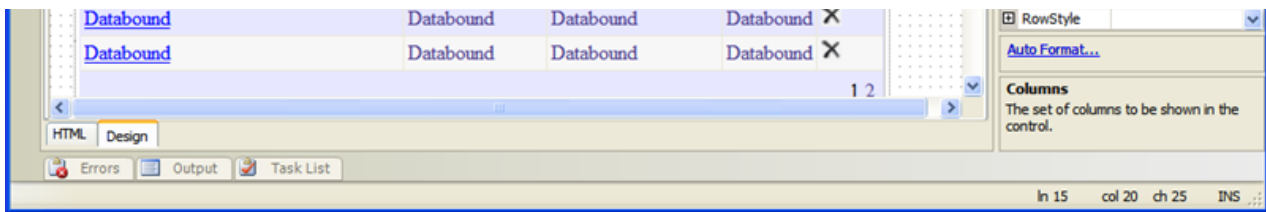
Now have a look at the WinForm for Designer.



This is now new territory. The browse control looks different from what we're used to presumably because it's a native .NET control, although as Z indicated at the Oz DevCon SV has added the kind of functionality Clarion developers have come to expect in a browse, including page loading.

And here is a shot of the ASP.Net Web Form Designer.

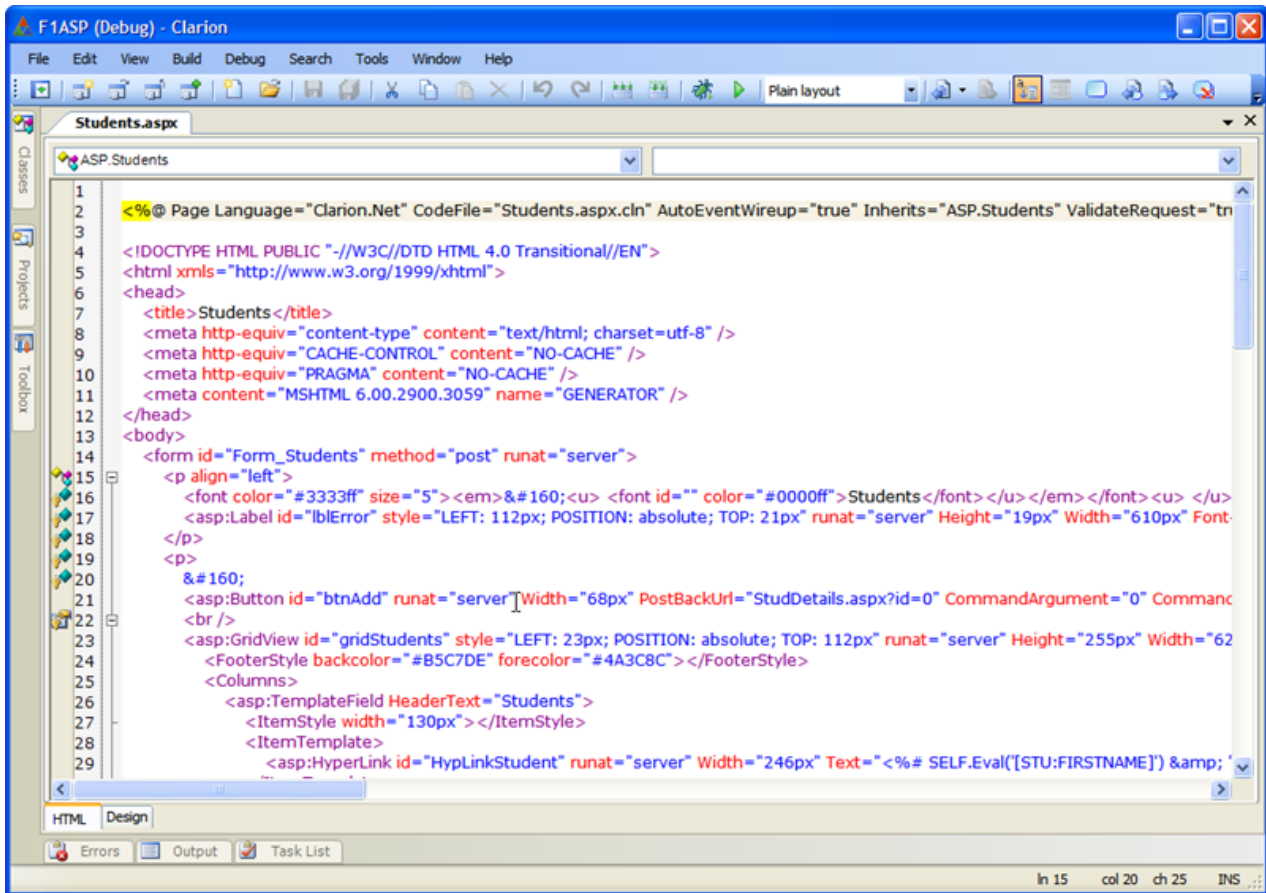




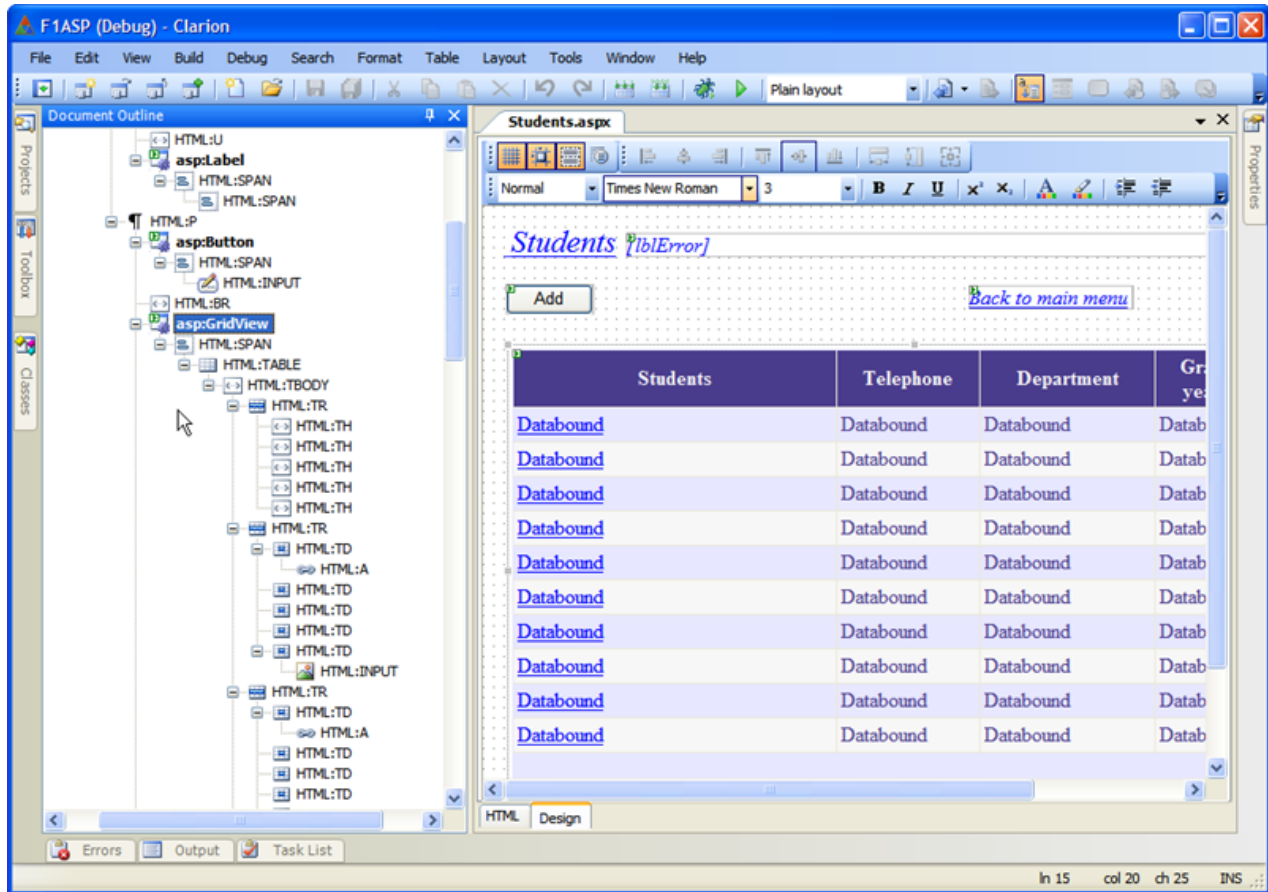
As Russ indicated in his *DevCon* coverage, the WebForm designer lets you see the page as it will appear in the browser. The code below shows the use of absolute positioning to achieve this.

The ASP.Net Designer has three possible Views, note the Tab text; HTM and Design. The screen shot above shows the Design view, but you can also work directly with the HTML that the Designer creates for you.

Here is the ASP.Net HTML view



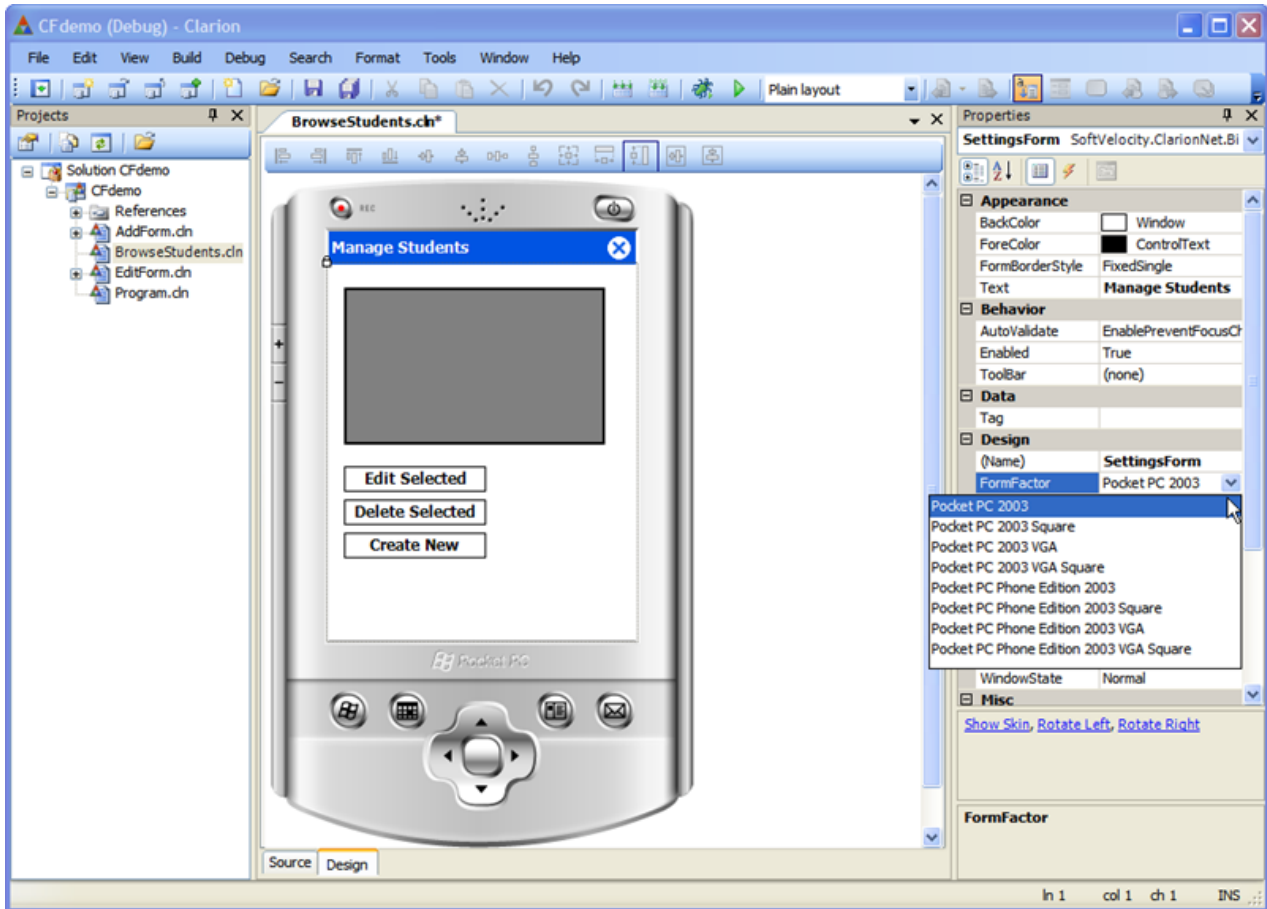
And finally we can also look at an ASP.Net WebForm in Outline view, as shown here



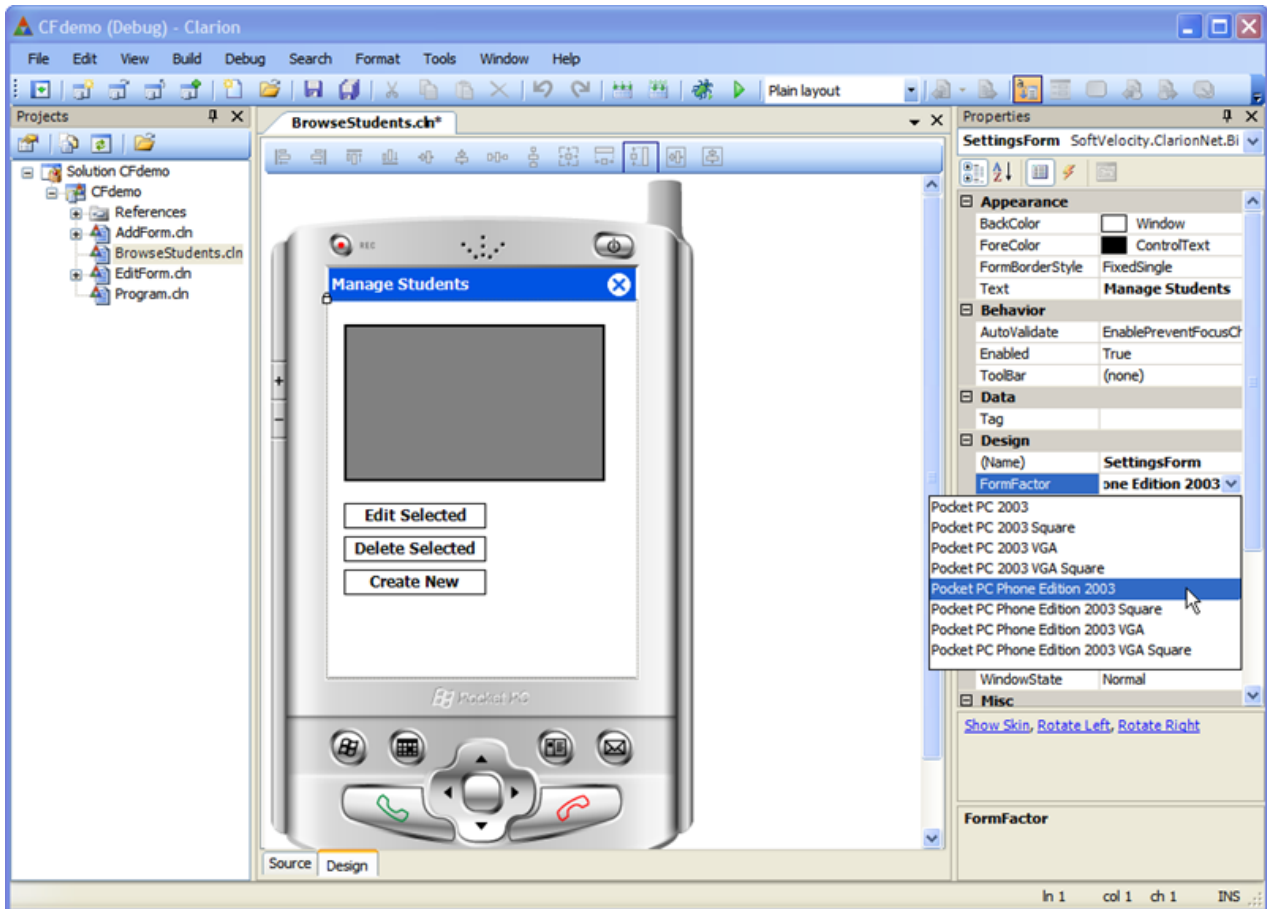
Now I'm curious to see what appears when I click on one of the sub-nodes. If you've ever used Firefox's DOM Inspector you'll immediately recognize the usefulness of this view.

Now we'll take a look at a few screen shots of the Compact Forms Designer. As you can see the Designer can display the form sized for the target device and with a skinned representation of the device. Just select the desired FormFactor from the dropdown list.

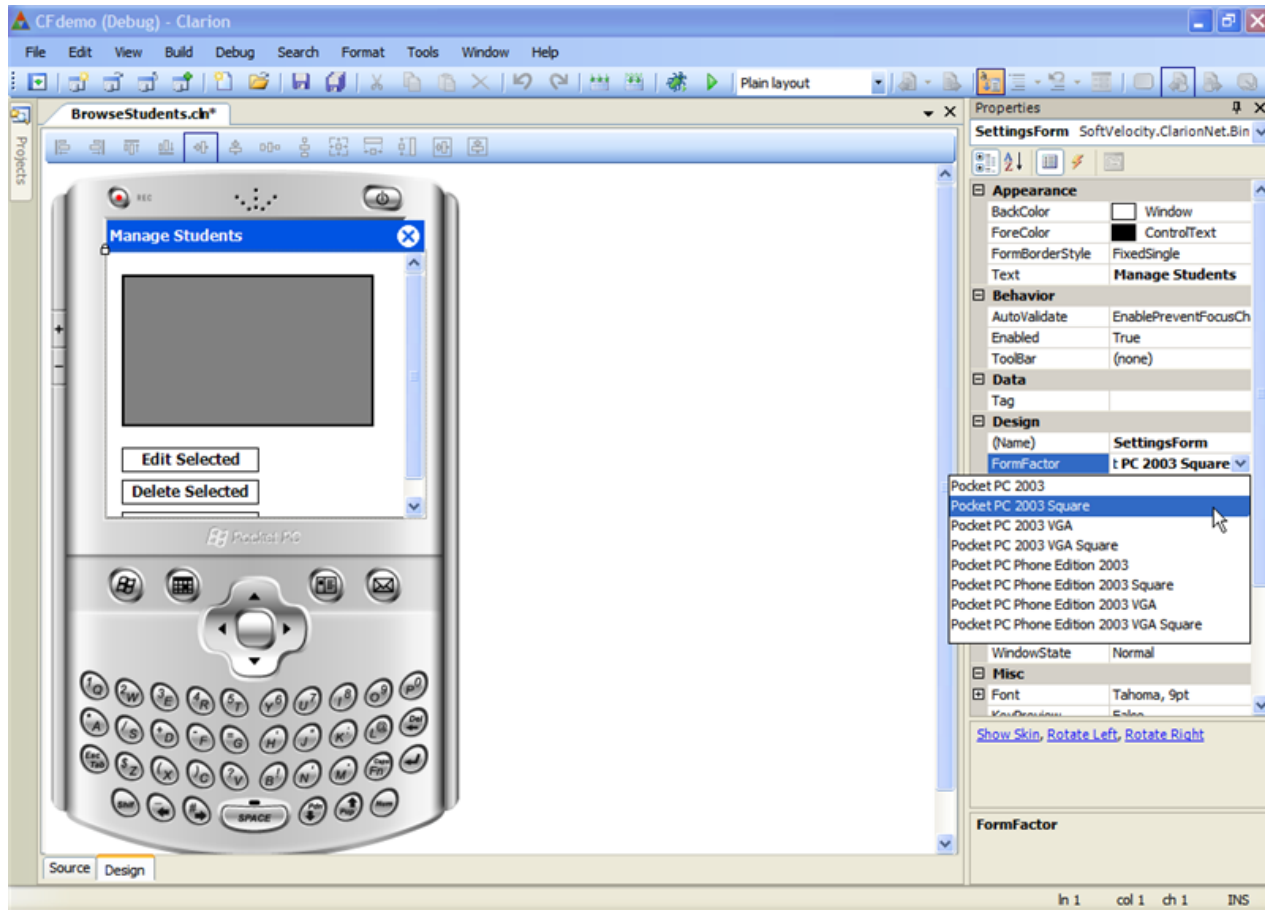
PocketPC 2003



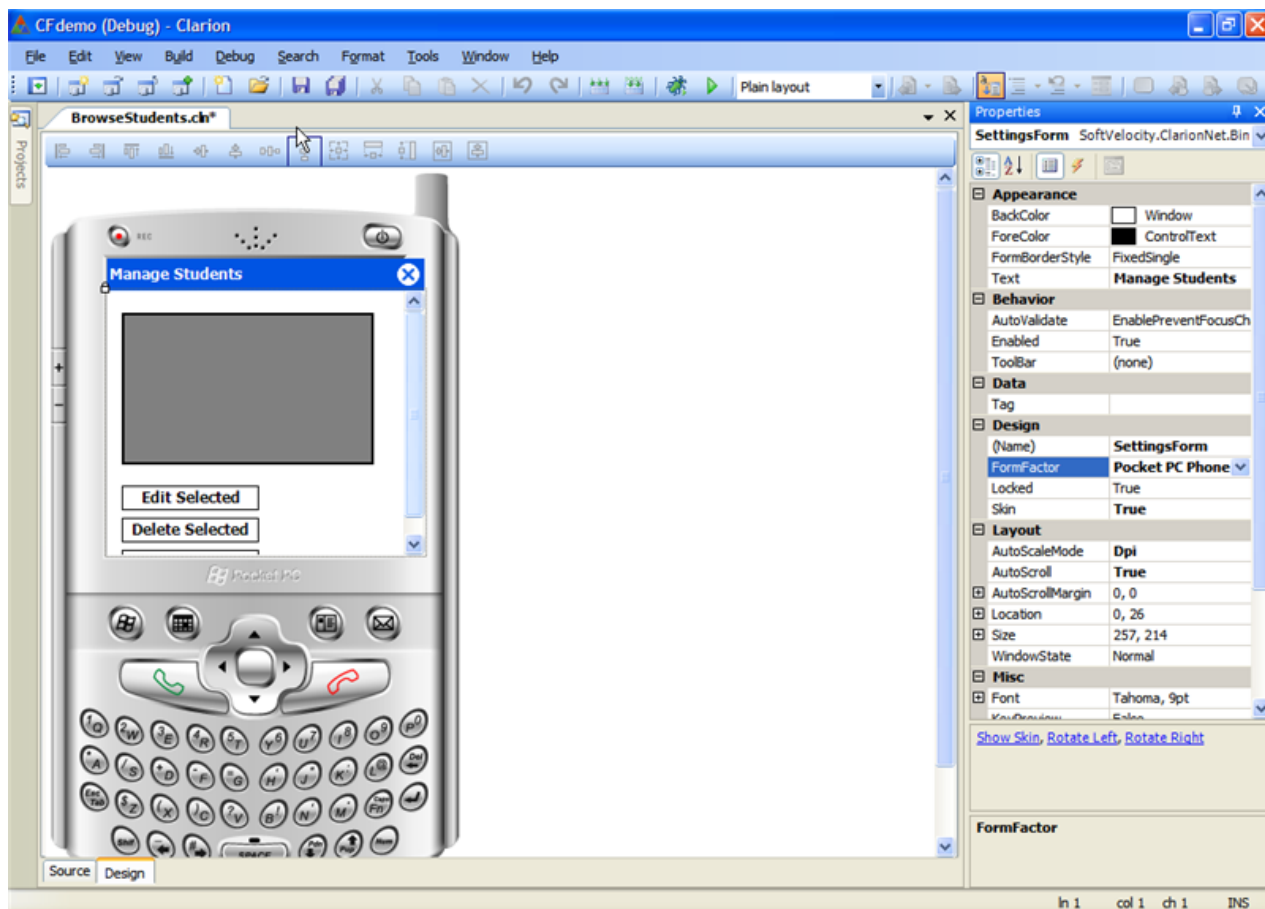
Pocket PC Phone Edition 2003



Pocket PC 2003 Square



Pocket PC Phone Edition 2003 Square





The compact framework support in the first release of Clarion.NET is terrific news, and I know a lot of Clarion developers are anxious to take this product out for a spin.

Now that we've seen all five Designers let's review the common UI they all share. The most obvious piece is the Properties Grid which is common to all the Designers, shown here side-by-side

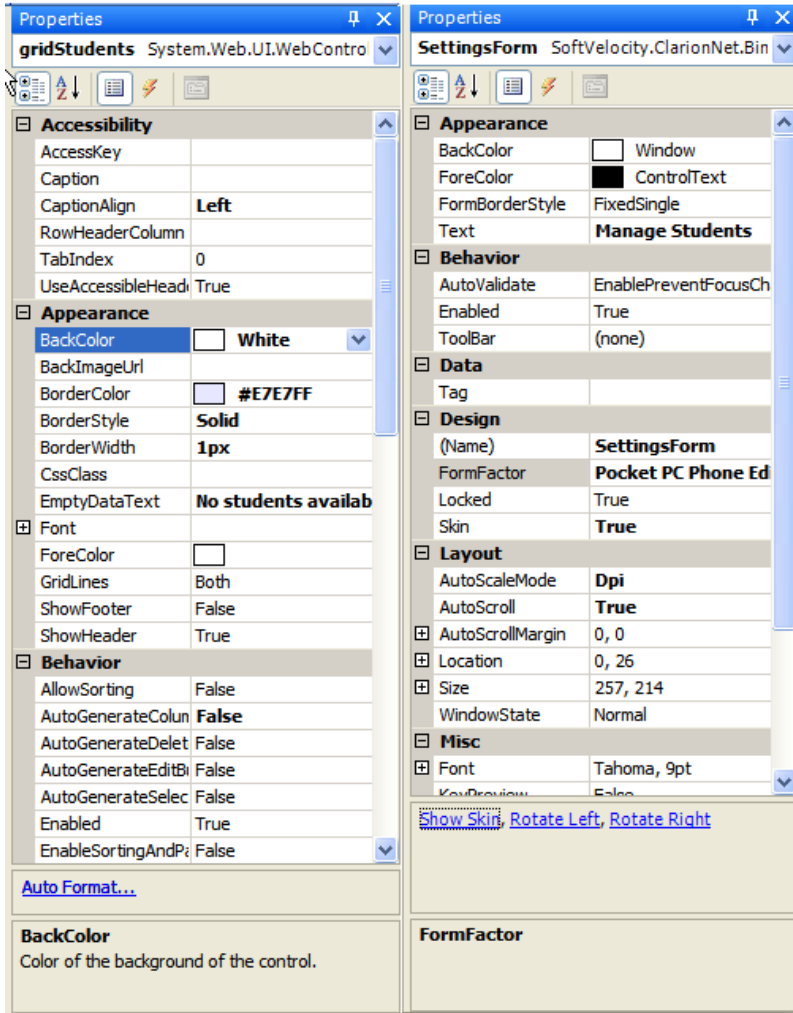
Property Grid as shown in the Window, Report and WinForm Designers

The image displays three side-by-side Property Grids from the Clarion IDE. Each grid shows the properties of a different control:

- Left Grid (STRING):** Properties include Color (BackGround, TextColor), Design (Locked, SuppressTranspare, TabIndex), Extra (Angle, DropID), General (Justification, Layout, StringValue, TextFont, Use), Help (Cursor), Mode (Disable, Hide, Scroll, Transparent), and Position (AT).
- Middle Grid (STRING):** Properties include Color (BackGround, TextColor), Design (Locked, SuppressTranspare), Extra (Angle, Extend), General (Justification, Layout, PictureToken, Text, TextFont, Use), Mode (Disable, Hide, Skip, Transparent), and Position (AT).
- Right Grid (DataGridView1):** Properties include Accessibility (AccessibleDescriptio, AccessibleName, AccessibleRole), Appearance (AlternatingRowsDe, BackgroundColor, BorderStyle, CellBorderStyle, ColumnHeadersBor, ColumnHeadersDef, ColumnHeadersHeig, ColumnHeadersVisit, Cursor, DefaultCellStyle, EnableHeadersVisu, GridColor, RightToLeft, RowHeadersBorder, RowHeadersDefaul, RowHeadersVisible, RowsDefaultCellSty, RowTemplate, ShowCellErrors, ShowCellToolTips, ShowEditingIcon, ShowRowErrors).

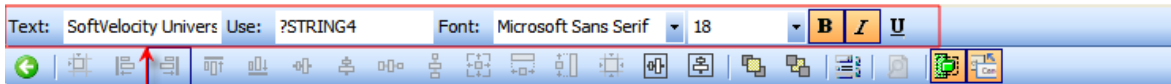
Each grid has an 'Edit Value' link at the bottom. The rightmost grid also includes a description for the 'AlternatingRowsDefaultCellStyle' property: 'The default cell style applied to odd-numbered rows.'

Property Grid in the ASP.Net and Compact Form Designers



The Designer Toolbars also share almost the exact same UI, the differences are that the Window and Report Designers take a trick from Clarion 6, and can optionally have the most common properties available for editing right on the Properties Toolbar, as shown here in the red boxes.

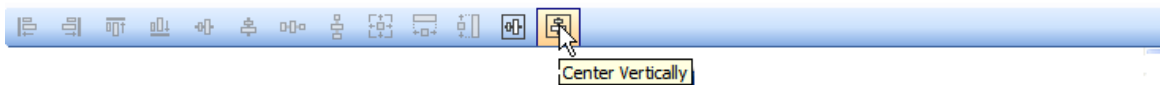
Window Designer with both Properties and Command Toolbars shown



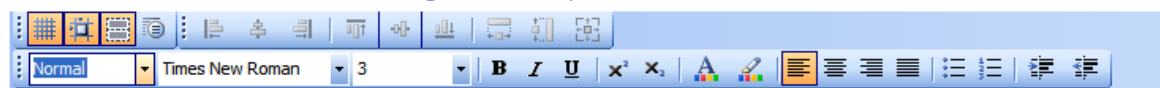
Report Designer with both Properties and Command Toolbars shown



WinForm Designer with Command Toolbar



ASP.Net Designer with Properties and Command Toolbar



Compact Form Designer with Command Toolbar



As you can see, learn them once and you're ready to use any of the Designers. There are a lot of cool features in the Designers, one of them that I'm sure you'll really like using is the "Snap Lines" Grid option. It's impossible to get a screen shot of that feature so here is a [very short video](#). I'll post a new article and a longer video tomorrow that goes a lot deeper into the new Designers.

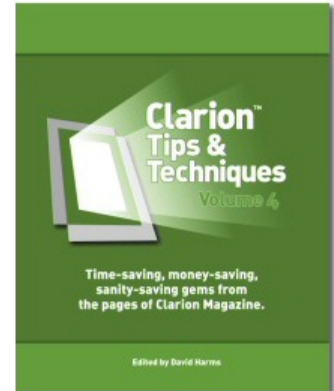
Reader Comments

[Add a comment](#)

Clarion Magazine

Clarion Tips & Techniques Volume 4

At over 650 pages, Clarion Tips & Techniques Volume 4 is another blockbuster brimming with terrific articles from Clarion Magazine.



Now in final editing - pre-purchase and save \$\$!

This book is now in final editing, and we expect it to ship in late June or early July. The list price is US\$79.95 - right now you can buy this book for just \$64.95, or \$59.95 if you have a current Clarion Magazine [subscription](#) (as a subscriber you also get big discounts on [e-books!](#)). When the book goes to the printer, the price goes up!

Pre-purchase now in the [Clarion Magazine store](#).

Contents

The draft table of contents (subject to minor changes) is [now available](#). Topics covered in this book include:

- Browsers & Forms
 - Internationalization standards
 - Dynamic listbox formatting
 - Edit-In-Place
 - Hot fields
 - Recursive inserts and updates
 - Beautifying Clarion apps
 - Automatically closing windows
 - Commonly used Clarion embeds
- Templates
 - Multiple lookups
 - Record status controls
 - A template debugger
 - Class wrappers
- Threading
 - Global variables
 - Critical sections
 - Background processes

- Controlling START
- Reports
 - Sending printer codes
 - Page loaded trees
 - Printing unknown queue fields
 - Printing directly to the printer
 - Printing directly to USB printers
 - Printing a "No records" report
- Databases
 - Mimer
 - The SQL Advanced settings
 - Creating SQL from XML with XSLT
 - External business rules
 - The in-memory driver
 - Using SQLIdentity
 - Multi-user primary keys
 - Embedded SQLite
 - PROP:SQL
- Windows Vista
 - Encryption and application signing
 - Manifests
 - Getting ahead with Vista and Office 2007
 - Vista-compliant INI files
 - Running C6 on Vista
- Version Control
 - CVS and WinCVS
 - Using CVS/WinCVS with Clarion 6.x
 - Understanding the C6 version control interface
 - Using MS Visual Source Safe with Clarion
- DLLs
 - Eliminating circular DLL calls
 - Smart DLL loading
 - DLLs and reusable code
 - Generic DLLs
 - Hand coding export files
- Tips & Techniques
 - Encrypting data
 - App shutdown options
 - Accessing private class data
 - Sorting queues
 - Displaying queues
 - Date calculations
 - Adding arrays to generic queues
 - Customizing deep assigns
 - Using finite state machines

- Using metadata
- Providing good customer service

Clarion Magazine

Rendering Text and Images on Vista Glass

by Randy Rogers

Published 2007-06-29

In the example from my previous article, [Adding Aero Glass Effects to Clarion Apps](#), I purposely avoided rendering text and images on the Vista Glass surface, primarily because such rendering requires the use of GDI+. In my search for information on GDI+ I came across an interesting article entitled [Aero Glass From Managed Code](#) on Tim Dawson's blog. The screen shot from his example program (Figure 1) caught my eye.

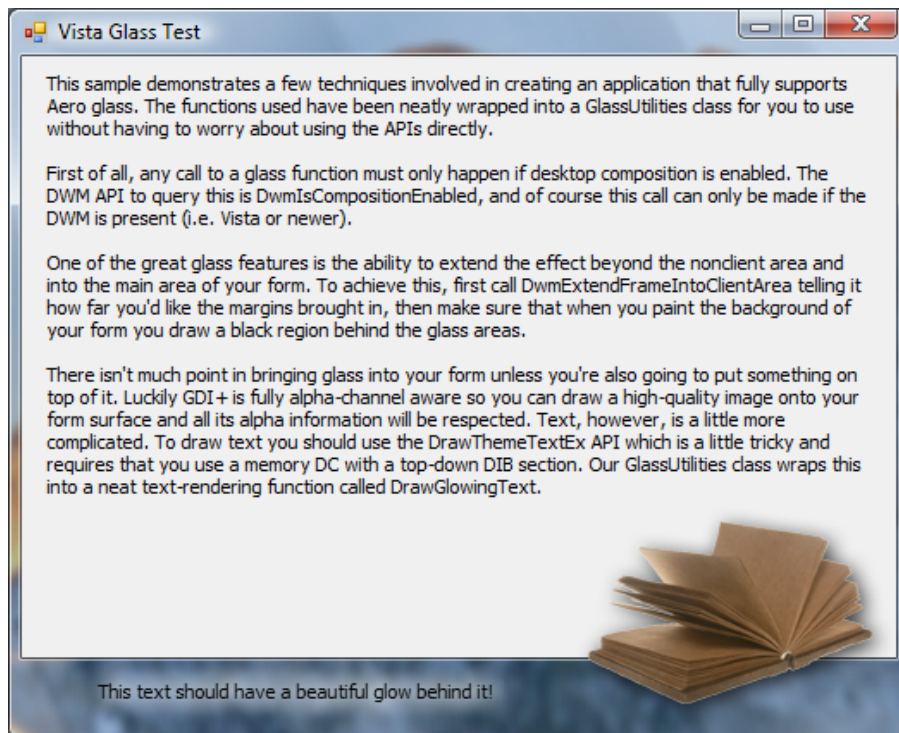


Figure 1. A C# example using GDI+ to render text and graphics

I wondered if I could recreate Tim's C# .NET example in Clarion.

The wrapper classes

I decided to create several reusable classes to wrap the necessary API calls from the various Windows DLLs needed by the application. They are not complete API implementations; look at the number of functions in the [GDI+ library documentation](#) and you will understand why I only implemented what I needed. You can easily create your own derived classes to provide for extra functionality as required.

I'll describe the classes briefly here and then cover them in more detail later in the article.

ctSubclassedWindowManager

This class is derived from the ABC WindowClass. It subclasses the ClientWndProc and provides methods to handle the WM_PAINT and WM_ERASEBACKGROUND messages.

ctDesktopWindowManager

This class provides access to functions in the dwmapi.dll.

ctGdiPlusGraphics

This class provides access to functions in the gdiplus.dll.

ctUxTheme

This class provides access to functions in the uxtheme.dll.

ctFont

This class creates a font and provides a handle for use with the ctUxTheme class.

ctSystemEnvironment

This class provides a method that returns the OS Major Version number. It is used by the wrapper classes to determine the operating system version.

The source code

Using classes for all the API work made the project's main source code (GlassTextDemo.clw) relatively short.

First I include the standard Windows equates and group types (svapi.inc) and the declarations for the classes to be used. I also declare a constant equate for the size of the glass margin.

```
!Equates and Data Types
INCLUDE('svapi.inc'),ONCE
INCLUDE('ctSubclassedWindowManager.inc'),ONCE
INCLUDE('ctDesktopWindowManager.inc'),ONCE
INCLUDE('ctGdiPlusGraphics.inc'),ONCE
INCLUDE('ctUxTheme.inc'),ONCE
INCLUDE('ctFont.inc'),ONCE
MARGINSIZE EQUATE(40)
```

Next I include the standard Windows API prototypes:

```
!Procedure Map
MAP
INCLUDE('svapifnc.inc'),ONCE
END
```

The window manager is derived from the `ctSubclassedWindowManager`. It contains a couple of properties regarding the image being displayed and overrides four of the virtual methods of the base class.

```
!Global Data
thisWindow    CLASS(ctSubclassedWindowManager)
m_pImage      LONG
m_ImageSize   LIKE(SIZE)
Init          PROCEDURE(),BYTE,PROC,VIRTUAL
TakeEvent     PROCEDURE(),BYTE,PROC,VIRTUAL
OnPaint       PROCEDURE(HDC hDC),BYTE,PROC,VIRTUAL
OnPaintBackground PROCEDURE(HDC hDC),BYTE,PROC,VIRTUAL
END
```

I instantiate the helper classes globally. This will cause each class's Construct method to be called when the program initializes.

```
!construct the 'helper' class objects
dwm    ctDesktopWindowManager
gdip   ctGdiPlusGraphics
theme  ctUxTheme
textFont ctFont
```

This cstring contains the text that will be displayed on the window:

```
label1 CSTRING("This sample demonstrates a few techniques involved in creating an application that
fully supports Aero glass. The functions used have been neatly wrapped into classes for you to use
without having to worry about using the APIs directly.<13,10,13,10>First of all, any call to a glass
function must only happen if desktop composition is enabled. The DWM API to query this
is DwmIsCompositionEnabled, and of course this call can only be made if the DWM is present (i.e. Vista
or newer).<13,10,13,10>One of the great glass features is the ability to extend the effect beyond the
nonclient area and into the main area of your form. To achieve this, first call
DwmExtendFrameIntoClientArea telling it how far you'd like the margins brought in, then make sure
that when you paint the background of your form you draw a black region behind the glass
areas.<13,10,13,10>There isn't much point in bringing glass into your form unless you're also going to
put something on top of it. Luckily GDI+ is fully alpha-channel aware so you can draw a high-quality
image onto your form surface and all its alpha information will be respected. Text, however, is a little
more complicated. To draw text you should use the DrawThemeTextEx API which is a little tricky
and requires that you use a memory DC with a top-down DIB section. Our ctUxTheme class wraps this into
a neat text-rendering method called DrawTextOnGlass.")
```

Next is a simple resizable window:

```

Window WINDOW('Vista Glass Text Demo'),AT(,,351,249),|
    FONT('Tahoma',8,,FONT:regular),COLOR(0FEFCFCH),|
    ICON('windows.ico'),SYSTEM,GRAY,MAX,RESIZE,IMM
    TEXT,AT(6,4,339,162),USE(Label1),SKIP,TRN,READONLY,VSCROLL
END

```

This is the main program entry point:

```

CODE
thisWindow.run()
RETURN

```

The window manager's Init method first calls the parent Init method. If no errors occur, the ctGdiPlusGraphics class is used to load the book.png image file (which is the graphic to be drawn in the lower right corner), the window is opened, the ctFont class is initialized using the window font, and finally, if desktop composition is enabled, the glass is extended into the bottom of the client window area using the ctDesktopWindowManager class.

```

thisWindow.Init PROCEDURE() !,BYTE,PROC,VIRTUAL
returnValue BYTE(Level:Benign)
CODE
returnValue = PARENT.Init()
IF returnValue = Level:Benign
    !load the book image
    gdip.LoadImageFromFile(PATH() & '\book.png', |
    SELF.m_pImage, SELF.m_ImageSize)
    SELF.Open(Window)
    textFont.Init(Window)
    IF (dwm.IsCompositionEnabled() = TRUE)
        dwm.ExtendFrameIntoClientArea(Window, 0, 0, 0, MARGINSIZE)
    END
END
RETURN returnValue

```

The window manager TakeEvent method is used to position the text control when the window is resized after first calling the parent TakeEvent method.

```

ThisWindow.TakeEvent PROCEDURE() !,BYTE,PROC,VIRTUAL
returnValue BYTE(Level:Benign)
CODE
returnValue = PARENT.TakeEvent()

```



```

CASE EVENT()
OF EVENT:Sized
  SETPOSITION(?Label1,,Window{PROP:Width}-12,Window{PROP:Height}-87)
END
RETURN returnValue

```

The window manager OnPaint method is invoked, prior to processing by the original WndProc, when the client window receives the windows WM_PAINT message. This is where I use GDI+ to render the image and UxTheme to draw text on the window glass.

```

thisWindow.OnPaint PROCEDURE(HDC hDC)
bReturn    BYTE(FALSE)
pt         LIKE(POINT)
bounds     LIKE(_RECT_)

CODE
!Draw a book image in the lower-right corner
pt.x = SELF.m_clientRectangle.Right - |
  SELF.m_ImageSize.cx
pt.y = SELF.m_ClientRectangle.Bottom - |
  SELF.m_ImageSize.cy
gdip.DrawImage(hDC, SELF.m_pImage, pt)
IF (dwm.IsCompositionEnabled() = TRUE)
  !Draw some text
  bounds.left  = SELF.m_clientRectangle.left
  bounds.top   = SELF.m_clientRectangle.bottom - |
    MARGINSIZE
  bounds.right = SELF.m_clientRectangle.right - |
    SELF.m_ImageSize.cx
  bounds.bottom = SELF.m_clientRectangle.bottom
  theme.DrawTextOnGlass(SELF.MyWindow{PROP:ClientHandle},|
    'This text should have a beautiful glow behind it!',|
    textFont.GetHandle(), bounds, 10)
END
RETURN bReturn

```

The window manager OnPaintBackground method is invoked when the client window receives the windows WM_ERASEBACKGROUND message. It is called after processing of the message by the original WndProc. Here I use GDI+ to paint a black rectangle where I want the Vista Glass to appear.

```

thisWindow.OnPaintBackground PROCEDURE(HDC hDC)

```

```

bReturn    BYTE(FALSE)
rc         LIKE(_RECT_)
solidBlack  ARGB(OFF000000h)

CODE
IF (dwm.IsCompositionEnabled() = TRUE)
    !Fill the bottom with black, to indicate
    !glass should be present in that area
    rc = SELF.m_clientRectangle
    rc.top = rc.bottom - MARGINSIZE
    gdip.FillRectangle(hDC, rc, solidBlack)
    bReturn = TRUE
END
RETURN bReturn

```

The final result looks like Figure 2...

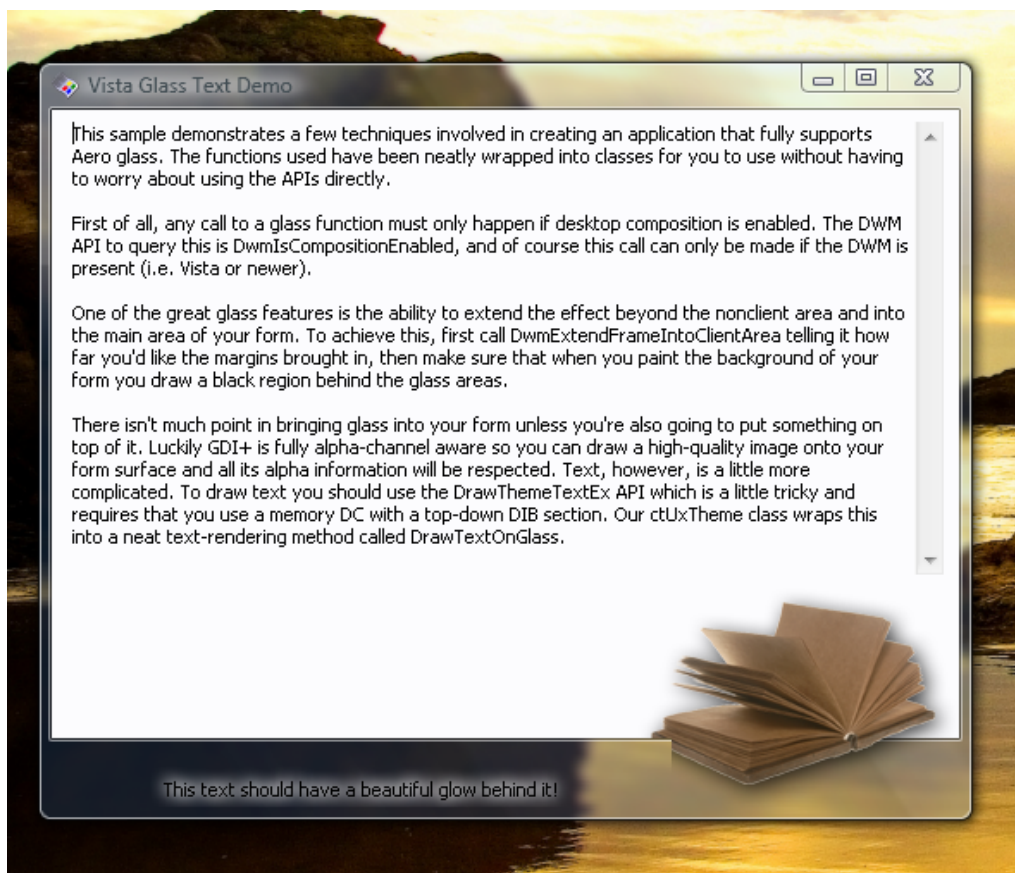


Figure 2. The Clarion version

...which is pretty much like the original.

Well, that was the easy part. Now it's time to roll up the sleeves and take a look at the classes.

For the sake of brevity I am not going to describe the class code in detail except where there is

something particularly interesting. I will try to provide enough information for you to be able to use the classes effectively. The reader is encouraged to study the downloadable source code for more details. All of the classes have Construct, Destruct, Init, and Kill methods. Typically the constructor will call the Init method and the destructor will call the Kill method. The Init and Kill methods are virtual and are to be used to provide application specific initialization and cleanup code.

The ctSubclassedWindowManager class

The first class in my example is the one that takes care of subclassing the window and provides two virtual methods, OnPaint and onerasebackground, to allow application specific processing of the Windows WM_ONPAINT and WM_ERASEBACKGROUND messages. The class contains a global threaded variable that is a reference to the class itself. The variable is initialized in the Init method and is used by the WndProc to update class properties.

```
this &ctSubclassedWindowManager,THREAD
```

Subclassing of the window is done in the open method, a virtual method inherited from the parent WindowManager class, after the call to the parent method. MyWindow is an &window property inherited from the parent WindowManager class.

```
ctSubclassedWindowManager.Open PROCEDURE(*WINDOW pWindow,|
    < *WINDOW pOwner >) !,VIRTUAL
```

CODE

```
PARENT.Open(pWindow,pOwner)
SELF.m_OrigWndProc = SELF.MyWindow{PROP:ClientWndProc}
SELF.MyWindow{PROP:ClientWndProc} = ADDRESS(WndProc)
SELF.m_hWnd = pWindow{PROP:Handle}
GetClientRect(SELF.m_hWnd, SELF.m_clientRectangle)
RETURN
```

The WndProc contains code to process the WM_PAINT, WM_SIZE, and WM_ERASEBACKGROUND messages.

```
WndProc PROCEDURE(HWND hWnd, UINT wParam, WPARAM |
    wParam, LPARAM lParam) !,LONG,PASCAL
```

ReturnValue LONG

hDC HDC

ps LIKE(PAINTSTRUCT)

CODE

```
CASE(wParam)
```

To support Vista Glass I need to do some special processing when the WM_PAINT message arrives. I call the BeginPaint API to start the painting process, call my virtual OnPaint method, and then complete the process by calling the EndPaint API.

```
OF WM_PAINT
    HDC = BeginPaint(hWnd, ps)
    this.OnPaint(HDC)
    EndPaint(hWnd, ps)
```

One of the features of this class is that it maintains a property that contains the size of the client area. To allow for a resizable frame, the class saves the client window size and forces a repaint of the window when a WM_SIZE message arrives.

```
OF WM_SIZE
    this.m_clientRectangle.right = |
        BAND(IParam,0FFFFh)
    this.m_clientRectangle.bottom = |
        BSHIFT(BAND(IParam,0FFFF0000h),-16)
    InvalidateRect(hWnd,this.m_clientRectangle,TRUE)
END
```

I now call the original WndProc to let the Clarion RTL do its work.

```
ReturnValue = CallWindowProc(this.m_origWndProc,|
    hWnd,wMsg,wParam,lParam)
```

In order to prepare my window for Vista Glass, I will need to do some special processing when the WM_ERASEBACKGROUND message arrives after the RTL has done its processing of the message. I call the virtual OnPaintBackground method to allow application specific processing.

```
CASE(wMsg)
OF WM_ERASEBKGND
    HDC = wParam
    this.OnPaintBackground(HDC)
END
RETURN ReturnValue
```

ctDesktopWindowManager

This class provides a wrapper for functions in the Windows desktop window manager API. The desktop window manager is new in Vista and provides the Aero Glass effect. In addition to the standard methods this class provides two virtual methods:

```
IsCompositionEnabled PROCEDURE(),BOOL,PROC,VIRTUAL
```

IsCompositionEnabled returns a true or false result indicating the state of Vista composition. It is necessary to check the composition state when using Vista Aero Glass effects.

```
ExtendFrameIntoClientArea PROCEDURE(WINDOW w, |
    LONG leftMargin, LONG topMargin,|
    LONG rightMargin, LONG bottomMargin),BOOL,PROC,VIRTUAL
```

ExtendFrameIntoClientArea is used to extend the window frame causing the Aero Glass effect to display inside the client area of the window.

The Init method deserves some explanation as its methodology is used in all of the API wrapper classes.

```
MAP
```

```
MODULE('dwmapi.dll')
```

```
    DwmExtendFrameIntoClientArea(HWND hWnd, *MARGINS MarInset)|
        ,HRESULT,PROC,PASCAL,RAW,DLL(__DwmExtendFrameIntoClientArea)
```

```
    DwmIsCompositionEnabled(*BOOL fEnabled),HRESULT,PASCAL,RAW,|
        PROC,DLL(__DwmIsCompositionEnabled)
```

```
END
```

```
END
```

```
__DwmExtendFrameIntoClientArea    LONG,AUTO,NAME('DwmExtendFrameIntoClientArea')
```

```
__DwmIsCompositionEnabled        LONG,AUTO,NAME('DwmIsCompositionEnabled')
```

Using the DLL attribute on the method prototypes with the long variables allows me to link the methods to the proper library code at runtime with the GerProcAddress API call.

```
ctDesktopWindowManager.Init PROCEDURE() !,BOOL,PROC,VIRTUAL
```

```
environment ctSystemEnvironment
```

```
szDwmApi    CSTRING('dwmapi.dll')
```

```
szProcName  CSTRING(65)
```

```
CODE
```

```
IF SELF.m_IsIntitalized = FALSE
```

Make sure the code is running on Vista or later OS, otherwise the library won't be there. I use the ctSystemEnvironment class to provide the OS Major Version number.

```
IF environment.GetOSVersionMajor() >= 6
```

```
    ! load DWMAPI.DLL
```

```
    SELF.m_hDwmApi = LoadLibrary(szDwmApi)
```

```
    IF (SELF.m_hDwmApi <> 0)
```

```
        ! locate functions in DWMAPI.DLL
```

```

szProcName = 'DwmExtendFrameIntoClientArea'
__DwmExtendFrameIntoClientArea = |
    GetProcAddress(SELF.m_hDwmApi, szProcName)
szProcName = 'DwmIsCompositionEnabled'
__DwmIsCompositionEnabled = |
    GetProcAddress(SELF.m_hDwmApi, szProcName)
END
!Check for Errors
IF __DwmExtendFrameIntoClientArea = 0 OR |
    __DwmIsCompositionEnabled = 0
ELSE
    SELF.m_IsIntitalized = TRUE
END
END
END
RETURN SELF.m_IsIntitalized

```

The ctGdiPlusGraphics class

The ctGdiPlusGraphics class wraps functions in the Windows GDI+ library. As mentioned earlier in this article the GDI+ API is quite extensive so only a few of the functions are wrapped in the class. I did, however, include and initialize several functions not used in this particular example since I already had the code written from some earlier attempts at using the gdiplus.dll.

There are four virtual methods of interest in this class:

```

DrawImage PROCEDURE(HDC hDC, LONG pImage, *POINT pt)|
    ,HRESULT,PROC,VIRTUAL

```

The DrawImage method is used to draw an image on the window, or more appropriately, a device context, at the specified location. In my example the pImage parameter is obtained by calling the loadimagfromfile method.

```

FillRectangle PROCEDURE(HDC hDC, *_RECT_ rc, ARGB argbColor)|
    ,HRESULT,PROC,VIRTUAL

```

FillRectangle is used to fill a rectangular area with an alpha channel color. In my example I use this method to create a solid black area where the Aero Glass effect is to appear. This method could also be use to paint a semi-transparent rectangle on a window.

```

LoadImageFromFile PROCEDURE(STRING sFilename, |
    *LONG pImage, *SIZE sz),HRESULT,|

```

PROC,VIRTUAL

LoadImageFromFile is used to load an image from a file. It returns the size of the image as well as a pointer to the image that can be used with the drawimage method.

The ctUxTheme class

ctUxTheme is the wrapper for the Windows User Experience Theme library. Of interest to the reader is the drawtextonglass method which uses the window caption theme to draw glowing text. The inner workings of the ctUxTheme DrawTextOnGlass method are too detailed for this article. However, it is worthy of closer study as it demonstrates creation of a device-independent bitmap (DIB) and use of DrawThemeTextEx to render text on the screen.

```
DrawTextOnGlass PROCEDURE(HWND hwnd, STRING sText, |
    HFONT hFont, _RECT_ ctrlct,
    LONG iglowSize)
```

The example program uses this method to draw text on the glass area.

```
DrawTextOnGlass(SELF.MyWindow{PROP:ClientHandle},|
    "This text should have a beautiful glow behind it!",|
    textFont.GetHandle(), bounds, 10)
```

The parameters are:

- hwnd – the handle of the window onto which the text will be drawn; use PROP:ClientHandle
- sText – the text to be drawn
- hFont - a handle to a font. I use the cFont class to create a font based on the window font and then call the cFont.GetHandle method to get the handle to the font for use with the DrawTextOnGlass method.
- ctrlct - the rectangle that positions and contain the text.
- iglowSize - the size in pixels of the 'glow' that will surround the text.

The ctFont class

ctFont is a simple class that uses the Windows API to create a font in its Init method. There are two flavours of Init, one that takes several parameters describing the font and one that takes a window as a parameter. This second version, used in the example program, uses property syntax to extract the windows font characteristics and then passes them as parameters to the first version to create the font. A GetHandle method is also available to get the font handle for use with other method calls.

The ctSystemEnvironment class

I got tired of hand coding the Windows GetVersionEx API call in my applications that require OS version checking so I created the ctSystemEnvironment class. It contains a single method, GetOSVersionMajor, to return the OS Major Version number.

Suggested Enhancements

In the ctDesktopWindowManager class all of the dwmapi.dll API functions have been prototyped but only the two used in the example are initialized. It is left as an exercise for the reader to complete the initialization code for the other functions.

The ctGdiPlusGraphics class could really use a method to return an image pointer to an image that has been linked into the application. As it stands now the the image file must be shipped with, and in the same folder as, the example application.

Conclusion

With a little bit of work it is possible to utilize some of the exciting new visual capabilities of Vista in your Clarion applications. By using classes, it is possible to encapsulate functionality in a way that is reusable, extendable, and easily maintained. I hope you have found this article interesting; if you are inspired to implement the suggested enhancements I also hope that you will share your code with the Clarion community.

[Download the source](#)

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

Reader Comments

[Add a comment](#)

Clarion Magazine

Finding Duplicate DLLs

by Maarten Veenstra

Published 2007-06-28

A question was raised in the newsgroups about the availability of a utility to find duplicate DLL's on a hard disk. This is simple to write in Clarion thanks to the DIRECTORY command, but the developer who asked the question also wanted to include the version info into this search, which made it a challenge.

I started looking for the APIs needed and found GetfileVersionInfoSizeA, GetFileVersionInfoSize and VerQueryValue which are located in the Version.DLL, supplied with Windows. I also found some examples (but not in Clarion) of the use of these APIs and realized that it would not be easy to translate the code into Clarion. Not wanting to reinvent the wheel, I searched the well known Clarion sources and, lo and behold, [Brian McGinnis wrote an article](#) about this in Clarion Magazine! He even supplied a class with his article. Thank you Brian.

Since I only wanted a simple utility, I decided to write it as a source procedure and not as an app (no overhead), but the code can easily be incorporated into an app.

The components

There are only a few pieces to this utility: a queue to hold the search-result; a listbox to display the result; a procedure which calls the DIRECTORY command; and Brian's class.

Brian's code is in bdVer.inc and bdVer.clw. I use the statement

```
INCLUDE('bdVer.inc')
```

so I can instantiate the VersionInfo class with this definition:

```
VerInfo CLASS(bdVersionInfo) !Instantiate Version class
END
```

My program's MAP structure has only one entry, ReadOneDir, which is the engine of this utility.

Here's the queue which holds the list of files:

```
FilQueue  QUEUE,PRE(FIL)
Name      STRING(FILE:MaxFileName)
Style     LONG
Path      STRING(FILE:MaxFilePath)
ShortName STRING(13)
Date      LONG
Time      LONG
Size      LONG
Attr      STRING(4)
FVersion  STRING(20)
PVersion  STRING(20)
```

END

Notice the Style field following Name. I use this field to set the color of the file name on display: normal for non-duplicated files, red for duplicated files.

Here's the window definition, containing the list box that displays the queue:

```
Window WINDOW('Directory Function Demo'),AT(,420,214),|
  FONT('MS Sans Serif',10,,FONT:regular,CHARSET:ANSI),|
  CENTER,CURSOR(CURSOR:Wait),SYSTEM,GRAY,DOUBLE,AUTO
STRING('Executing the DIRECTORY command'),AT(12,9),USE(?Mess1)
LIST,AT(10,24,400,178),USE(?List1),FLAT,VSCROLL,|
FORMAT('180L(2)|MYP~File~@s255@36R(2)|M~Date~C@d17@#5#' &|
'20L(2)M~Time~@t01@34R(2)|M~Size~@n10@18L(2)' &|
|M~Attr~@s4@50L(2)|M~FileVersion~@s16@50L(2)' &|
'~ProductVersion~@s16@'),FROM(FilQueue)
STRING(@s255),AT(10,204),USE(FIL:Path),|
  FONT(,COLOR:Navy,,CHARSET:ANSI)
END
```

The colors are specified in this setup code, after the window opens:

```
?List1{PROPSTYLE:TextColor, 1} = -1
?List1{PROPSTYLE:TextColor, 2} = Color:Red
?List1{PROPSTYLE:TextSelected, 2} = COLOR:Fuschia !COLOR:Red
```

The Y list box format string modifier tells the list box runtime code to apply these colors to the field. I can either insert this manually into the string or, more easily, check the Style option in the listbox formatter. Here's the portion of the format string that applies to the first displayed field:

```
180L(2)|MYP~File~@s255@
```

The next two fields in the queue, Path and Shortname, are not displayed. The Path data is displayed at the bottom of the listbox for the currently selected item, but if you enable the tooltips on this listbox (commented out in the source), the Path is also displayed as a tooltip when you hover the mouse over the filename.

By default, the list box formatter displays queue fields in the order they appear in the QUEUE structure. But I don't want to show Path and Shortname, so I have to tell the list box formatter to skip to the fifth field. Here's the relevant portion of the format string:

```
36R(2)|M~Date~C@d17@#5#
```

As with the Y modifier, I can add the field specifier manually or I can use the list box formatter and update the field number value.

ReadOneDir

The ReadOneDir function accepts two parameters: The first one is the starting path for the search and the second one is the extension (mask) to search for. You can easily extend this utility by asking the user for these values prior to calling the ReadOneDir procedure. They now default to the folder in which the utility is started and 'DLL'. Note: when you're asking for the extension, make sure it has no preceding dot and is in uppercase.

```

ReadOneDir  PROCEDURE(String pFolder, String pMask)

DocDirs    QUEUE(FILE:Queue),PRE(DFQ)
            END
SrcFolder   STRING(FILE:MaxFilePath)
Lc          LONG
Ec          LONG
L:DocExt    STRING(20)
lLen       LONG

CODE
?Mess1{PROP:Text} = 'Scanning: ' & pFolder ; display
SrcFolder = CLIP(pFolder) & '\*.*'
!
!   Queue , Mask , Attributes
DIRECTORY(DocDirs, SrcFolder, ff_:NORMAL+ff_:READONLY+
ff_:HIDDEN+ff_:SYSTEM+ff_:ARCHIVE+ff_:LFN+ff_:DIRECTORY)
!
TotalFiles += RECORDS(DocDirs)
LOOP Lc = 1 TO RECORDS(DocDirs)
  GET(DocDirs, Lc)
  IF BAND(DFQ:Attrib, ff_:DIRECTORY)
    IF DFQ:ShortName = '.' OR DFQ:ShortName = '..'
      TotalFiles -= 1
      CYCLE
    END
    !
    SrcFolder = CLIP(pFolder) & '\' & DFQ:Name
    ReadOneDir(SrcFolder, pMask)
    CYCLE
  END
  !
  L:DocExt = "
  lLen = LEN(CLIP(DFQ:Name))
  LOOP Ec = lLen TO 2 BY -1
    IF DFQ:Name[Ec] = '.'
      L:DocExt = UPPER(DFQ:Name[Ec+1 : lLen])
      BREAK
    END
  END
  IF pMask AND pMask <> L:DocExt THEN CYCLE .
  !
  FIL:Name = DFQ:Name
  FIL:Style = 1
  FIL:Path = CLIP(pFolder) & '\' & DFQ:Name
  FIL:ShortName = DFQ:ShortName

```

```

FIL:Date   = DFQ:Date
FIL:Time   = DFQ:Time
FIL:Size   = DFQ:Size
FIL:Attr   = '----'
IF BAND(DFQ:Attrib, ff_:READONLY) THEN FIL:Attr[1] = 'r' .
IF BAND(DFQ:Attrib, ff_:ARCHIVE) THEN FIL:Attr[2] = 'a' .
IF BAND(DFQ:Attrib, ff_:HIDDEN) THEN FIL:Attr[3] = 'h' .
IF BAND(DFQ:Attrib, ff_:SYSTEM) THEN FIL:Attr[4] = 's' .
FIL:FVersion = LOWER(L:DocExt)
FIL:PVersion = ""
IF INLIST(L:DocExt, 'DLL', 'EXE', 'COM')
  FIL:FVersion = '?'
  FIL:PVersion = '?'
  ! call the VersionInfo class methods
  IF VerInfo.Init(FIL:Path) = Level:Benign
    FIL:FVersion = VerInfo.GetFileVersion()
    FIL:PVersion = VerInfo.GetProductVersion()
  END
END
ADD(FilQueue, +FIL:Name, FIL:Size)
!
END
!
FREE(DocDirs)
RETUR

```

The ReadOneDir procedure calls the DIRECTORY command, starting in the specified folder. The returned queue is read and, when a sub-folder is found, this sub-folder is set as a new starting point and the ReadOneDir is called with this sub-folder. For every file found, the extension is checked against the (optional) supplied extension mask and stored in the result-queue which is used by the listbox.

Upon return of the (first) call to the ReadOneDir procedure the result-queue is scanned for duplicates by simply reading the next queue-record and comparing the name to the previous one. Here you can vary the results depending on your definition of a duplicate: is it Name + FileSize or Name + Date or Name + Version or ...? Alter the comparison to your liking, adding TMP: variables as needed.

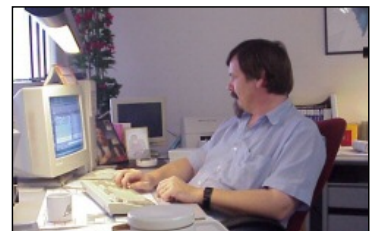
If you want to compile in C55 then change Level:Benign into False or 0 (zero).

This is just a working skeleton because I wanted to demonstrate the use of a recursive call to the DIRECTORY command, but you can turn it into a full-featured utility if you want to.

I could not have written this in less than 170 lines without Brian McGinnis' VersionInfo class, so thank you again Brian for your work.

[Download the source](#)

[Maarten Veenstra](#) began his computing career as a field service engineer on the DEC PDP11 family and a microprocessor-controlled punchcard machine. His first personal computer job was fixing CP/M computers and the early IBM clones, during which time he bought an MSX "computer" and taught himself BASIC and assembler. Maarten began using Clarion Professional Developer in 1988, wrote his first





Windows program with CW 2.0, and his first ABC program with C4. He is now the co-owner of [Nepucon](#), which provides service to public utilities. Maarten married in 1991, and he and his wife have adopted two beautiful Chinese girls.

Reader Comments

[Add a comment](#)

Clarion Magazine

The ToClipboard Class

by Tim Phillips

Published 2007-06-26

Users commonly interact with data via update forms and they tend to think of that data the way it is formatted and laid out on the form. And users often want to transfer this information to another person or program.

A print-screen utility lets the user capture an image of the form but this isn't always as useful as being able to capture and edit the form's text. In this article I'll show you how you can easily create a local class to copy prompts and field contents to the clipboard.

The supplied example program is based on the School example that ships with Clarion. The modifications are limited to a local class declared in the UpdateTeachers form window, a bit of setup code and a couple of To Clipboard buttons that call class methods. Figure 1 shows the UpdateTeachers form

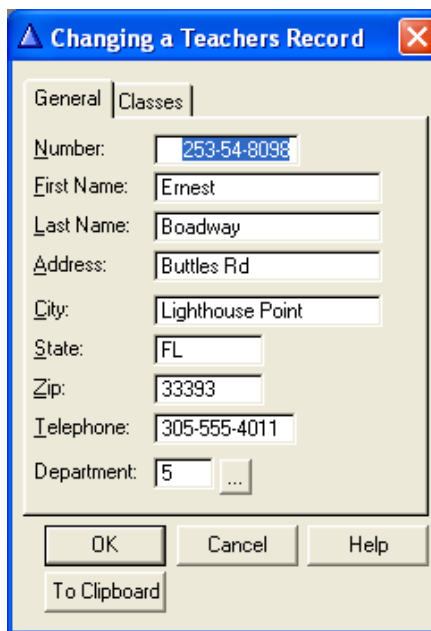


Figure 1. The UpdateTeachers form

And here's the text copied to the clipboard:

Number: 253-54-8098

First Name: Ernest

Last Name: Boadway
 Address: Buttles Rd
 City: Lighthouse Point
 State: FL
 ZIP: 33393
 Telephone: 305-555-4011f
 Department:

Local classes are a very handy way to organize code within a procedure. I prefer them over routines because of the ability to pass parameters in and receive return results to control logical behavior.

I always put the local class definition in Local Data | Other Declarations below the Field Validation, field coloring support variables embed.

The class for this usage has this definition:

```

LC          CLASS
ATC          Procedure(String DataEntryName,|
              <String DataItself>)
ClassesToClipboard Procedure()
ClearClipboard Procedure()
DoneMessage Procedure()
TeacherClassesToClipboard Procedure()
TeacherToClipboard Procedure()
          END
  
```

The class name is LC. It has six distinct methods, each of which has a brief descriptive comment at the declaration. Local class methods are entered at the Local Procedures embed. I always put each method in its own embed point so I can read the definition line of the method from the embed tree and easily find things.

Let's look at each method to understand its behavior, beginning with ClearClipboard.

```

LC.ClearClipboard Procedure()
Code
SetClipboard("") ! clear the windows clipboard
  
```

This method simply clears the Windows clipboard for a fresh start. Since this method doesn't add any extra code I could just type SetClipboard("") instead of using the class method, but I've had times in the past when I didn't encapsulate enough logic into the local class and had to rework things later. I'm typing about as many characters to use the method as using the direct command, but this way I can more easily accommodate a change in logic in the future.

Here's the AppendToClipboard procedure, abbreviated as ATC:

LC.ATC Procedure (String DataEntryName,)

Code

```

case CLIP(DataEntryName)
  of 'BlankLine'
    SetClipboard(CLIP(Clipboard()) & '<13><10>')
  of 'SeperatorLine'
    SetClipboard(CLIP(Clipboard()) |
      & '=====<13><10>')
  else
    if Omitted(DataItself)=True and CLIP(DataEntryName)>"
      SetClipboard(CLIP(Clipboard()) |
        & CLIP(DataEntryName) & '<13><10>')
    else
      SetClipboard(CLIP(Clipboard()) & CLIP(DataEntryName) |
        & ':' & CLIP(Left(DataItself)) & '<13><10>')
    end
  end
end

```

This method is the real workhorse of the LC class. It uses a combination of the Clipboard() and SetClipboard() commands to capture the current contents of the clipboard and then append the new data and put the new text in the clipboard.

ATC receives the DataEntryName string and the optional field of DataItself as parameters. If DataEntryName is equal to certain "commands" (BlankLine, SeparatorLine), then either a carriage return or a chunk of separating = marks are added to clipboard. Otherwise, if the value of DataItself has been omitted the value of DataEntryName and a carriage return are added to the clipboard. If the value of DataItself is present then the DataEntryName and DataItself are added to the clipboard with a separating colon (:) mark and a carriage return on the end.

Here's the code for the TeacherToClipboard procedure, which calls the LC.ATC method to copy the Teacher form data to the clipboard.

LC.TeacherToClipboard Procedure()

Code

```

LC.ATC('Number',format(TEA:Number,@P###-##-####P))
LC.ATC('First Name',TEA:FirstName)
LC.ATC('Last Name',TEA:LastName)
LC.ATC('Address',TEA:Address)
LC.ATC('City',TEA:City)

```



```

LC.ATC('State',TEA:State)
LC.ATC('ZIP',format(TEA:Zip,@n05))
LC.ATC('Telephone',TEA:Telephone)
MAJ:Number=TEA:Department
if Access:Majors.Fetch(MAJ:KeyNumber)=Level:Benign
  LC.ATC('Department',(format(TEA:Department,@n4) |
    & ' ' & CLIP(MAJ:Description)))
else
  LC.ATC('Department')
end

```

This method uses LC.ATC to add the fields (and their prompts, more or less) on the update form in the same order in which they appear on the window. A Fetch() is used to make sure that the description for a code number is available.

Notice that this method only adds the fields for the Teacher table and does nothing to clear the clipboard itself or inform the user that the action is complete. This is deliberate and allows reuse of this method in various ways - more on that shortly.

Next is the ClassesToClipboard method, which takes a similar approach to TeacherToClipboard, but with more lookups:

```

LC.ClassesToClipboard Procedure()
FoundChildRecord Byte
Code
LC.ATC('Classes') ! add a line that says Classes
! add a line that will serve as column headers
LC.ATC('Class Number Course Description')
! list the classes for the teacher
FoundChildRecord=False
CLA:TeacherNumber=TEA:Number
Set(CLA:KeyTeacherNumber,CLA:KeyTeacherNumber)
Loop
if Access:Classes.Next(<>Level:Benign
  break
end
if CLA:TeacherNumber<>TEA:Number
  break
end
COU:Number=CLA:CourseNumber

```

```

Access:Courses.Fetch(COU:KeyNumber)
LC.ATC((' ' & Format(CLA:ClassNumber,@P##-#####P) |
    & ' ' & COU:Description))
FoundChildRecord=True
end
if FoundChildRecord=False
    LC.ATC(' No Classes Entered For Teacher,')
end

```

This method uses the LC.ATC method to copy a select block of data from the Classes records related to the Teacher record to the clipboard. Only two columns of data are included (I've done this deliberately to match the child browse on the Teacher update form) and there is a provision to print out a message that no classes have been entered for the Teacher, if necessary.

Notice that the lines

```
LC.ATC('Class Number Course Description')
```

and

```
LC.ATC((' ' & Format(CLA:ClassNumber,@P##-#####P) |
    & ' ' & COU:Description))
```

have been "formatted" by using spaces. As long as the user pastes the clipboard contents into an editing window that is using a fixed font like Courier New, the columns will line up properly under the headers without issue. This technique will not work properly if the user insists upon only using a proportional font.

The DoneMessage method is, like ClearClipboard, a simple wrapper for a Clarion function call:

```

LC.DoneMessage Procedure()
Code
Message('Done!! Data copied to Clipboard|Ready ' |
    & 'to be pasted into Word, Outlook or wherever.')
```

Again, you could simply type the entire message manually instead of using a method, but if you need the code in more than one place you would definitely be doing a lot more typing. And editing the "done" message would be more involved if the code had been duplicated. This method means there is just one point of maintenance..

The TeacherClassesToClipboard method provides a way to combine a bunch of things into a single method call.

- a. Clear the clipboard
- b. Load all the Teacher data into the clipboard

- c. Add a blank line
- d. Load the classes data for the teacher to the clipboard
- e. Issue a done message so the user knows the data is now on the clipboard

Here's the code:

```
LC.TeacherClassesToClipboard Procedure()
code
LC.ClearClipboard()
LC.TeacherToClipboard()
LC.ATC('BlankLine') ! add blank link to clipboard
LC.ClassesToClipboard()
LC.DoneMessage()
```

When you actually add your To Clipboard buttons to the update form you only have to include a single method call, rather than remember these five lines of code.

On the UpdateTeachers form, I've opted to add two distinct To Clipboard buttons. The buttons are physically at the same X,Y coordinates on the window, but are on different tabs (the General and Classes tab). From the user's perspective, it will appear that there is a single To Clipboard button that has different behavior depending upon which tab is selected, but managing the code is less complex than it would be for one button responding to different tabs. Each button also has MouseRight set as an AlertKey so a right-mouse popup window can be used on the buttons to provide a greater level of user control.

The To Clipboard button on the General tab has code at two different embed points. First, at Control Events| ?ToClipboardTeacher | Accepted below the Generated Code:

```
LC.ClearClipboard()
LC.TeacherToClipboard()
LC.DoneMessage()
```

This code clears the clipboard, appends all the Teacher field data and issues a done message to the user whenever the To Clipboard button is pressed.

At Control Events | ?ToClipboardTeacher | AlertKey below the Generated Code:

```
case KeyCode()
of MouseRight
Execute Popup('Teacher+Classes To Clipboard')
Begin
LC.TeacherClassesToClipboard()
end
```

end

end

This code traps the condition of a user right-mousing on the To Clipboard button. When this happens, the user is shown a popup window with a single option: Teacher+Classes To Clipboard. If the user clicks on this, the `LC.TeacherClassesToClipboard()` method is called to load Teacher and Classes data onto the clipboard.

This architecture of the To Clipboard button allows a user to click on the button and simply get the data visible on the window, or right-mouse and select from the popup to get the visible information plus the information visible on the 2nd tab on the window also.

The code for the To Clipboard button on the Classes tab is basically the same, except it calls `LC.ClassesToClipboard()` instead of `LC.TeacherToClipboard()`. In fact, the code for the `AlertKey` embed is identical for both buttons. For the purposes of this article, I didn't fold the `AlertKey` embed code into yet another LC method, but in practice that is exactly what I would do to keep from having to maintain code in two places.

So, a few buttons and a local class and you have a setup where the user can easily copy select groups of data to the clipboard to be pasted into an email or document. By creating alternative methods and adding additional options to the `POPUP()` command, a user could be offered a varied selection of sets of data to match different needs and purposes.

I've found that users appreciate being able to easily convert what they can see on screen to something basic and easily editable. Wrapping the code up into a local class makes this functionality easy to use, maintain, and extend.

[Download the source](#)

[Tim Phillips](#) began programming Clarion with Version 3.0 for DOS. He currently works with Clarion 5.5/6 ABC. His preferred programming technique involves a lot of bass rock guitar on his cordless headset and vigorous application of the Keep It Simple Stupid principle. When not programming, he can be found trying to write fiction, "building Legos" with his two nieces, or being obsessive about television shows that have gone off the air.

Reader Comments

Posted on Wednesday, June 27, 2007 by Carl Barnes

A possible extension to the concept would be to pass the Prompt and Data FEQs. This would result in less code in the procedure since they are two longs, but more code in the class. Also saves formatting the data. Something like this:

```

LC      CLASS
FEQ      Procedure(<LONG PromptFEQ>, <LONG DataFEQ>)
Prmt STRING(256)
CODE
IF ~Omitted(2)
  Prmt=PromptFEQ{Prop:Text}
  IF PromptFEQ{prop:type}=create:prompt |
  OR PromptFEQ{prop:type}=create:button
    !Remove & from Prmt, handle && => &
    !this allows passing string
  END
END
IF ~Omitted(3)
  SELF.ATC(Prmt,DataFEQ{PROP:ScreenText})
ELSE
  SELF.ATC(Prmt)
END
RETURN

```

Note that PROP:ScreenText gives the formatted values (as seen on the screen), e.g. a date value of 76543 would have screentext=12/3/2007.

So the example would code as:

```

LC.FEQ(?TEA:Number:Prompt,?TEA:Number)
LC.FEQ(?TEA:FirstName:Prompt,?TEA:FirstName)
LC.FEQ(?TEA:LastName:Prompt,?TEA:LastName)
LC.FEQ(?TEA:Address:Prompt,?TEA:Address)
LC.FEQ(?TEA:City:Prompt,?TEA:City)
LC.FEQ(?TEA:State:Prompt,?TEA:State)
LC.FEQ(?TEA:ZIP:Prompt,?TEA:Zip)
LC.FEQ(?TEA:Telephone:Prompt,?TEA:Telephone)

```

I would also make a method that allowed passing a string for the DataEntryName and the Data FEQ since it seems like frequently the prompt text may be undesirable (e.g. 'ST' for State), but I like the simplicity of PROP:ScreenText.

Posted on Wednesday, June 27, 2007 by Carl Barnes

Forgot this code:

```

!Remove & from Prmt, handle && => &
LOOP x#=1 TO SIZE(Prmt)
  IF Prmt[x#]='&' THEN Prmt[x# : SIZE(Prmt)]=Prmt[x#+1 : SIZE(Prmt)].
END

```

[Add a comment](#)

Clarion Magazine

Using A Browse Popup Menu For Lookups

by Murray Gillespie

Published 2007-06-26

I recently had a client complain to me about the complexity of doing trivial updates (such as changing a staff member's department) using the browse/form paradigm. You know how it goes: select the record , click on Change, make the change, click on OK, select the next record, etc. I had to agree that it was a real time waster and, in agreeing to that, I inherently agreed to fix it.

The first obvious solution was to use Edit-in-Place droplists, but my opinion is that EIP can be a fiddly interface that confuses the user. EIP has its place but this was not it.

My client and I agreed that the best solution was to place the department lookups (as there were not many departments) onto a popup. The user would only need to right-click on the record and select a new department from the popup menu.

The first step was to add the lookup entries to the popup. The Clarion help lists two methods to add an action to a popup: AddItemMimic and AddItemEvent. AddItemMimic creates a popup entry that duplicates an existing button; as I didn't have a button for each lookup entry that approach was not of any use. The AddItemEvent method didn't look to be of much use either, as it just sends a specified event to a control (or thread). But then an event is just a number, just as a lookup ID is a number, so if I primed the AddItemEvent method with the lookup ID instead of an event equate, and sent that ID to a button control, I could then use the Event() function to retrieve the value (lookup ID) that I wanted to change.

Step 1 was to create a hidden button with the use variable of ?GChangeButton.

Step 2 was to embed the following code in ThisWindow.init (before prepare alert keys):

```
! Set the popups
set(GRO:NameKey)
! For each group (Department)
LOOP until access:groups.next()
! Add a popup entry with the group ID as its event
! and ?Changebutton as the target
BRW1.Popup.AddItemEvent(sub(GRO:Description,1,30),|
    GRO:GroupCode,?GChangeButton)
END
```

BRW1.Popup.AddItem('-', 'SeparatorPopup5') ! Add a separator

This code adds all the department entries and their codes to the popup.

Blake, Joe	jblake	Accounts
Bradby, Katrina	KBradby	Engineering
Brennan, Rosanne	Bren	Accounts
Close, Peter	pclor	Accounts
Cochrane, Amy	acoc	Engineering
Cole, Jack	jpcol	Maintenance
Cook, Pat	pacc	Sales
Cook, Betty	bcoc	Accounts
Cremin, Tim	crem	Accounts
Curnow, Murray	curn	Accounts
Davies, Penny	pmd	Engineering
Dawson, Patricia	daw	Maintenance
Elizabeth Sandiford	esar	Engineering
Fleming, Kate	kflen	Engineering
Garthwaite, Alan	agar	Maintenance
Gillespie, MurrayG	gilles	Sales

Figure 1. Popup with departments

Step 3 was to extract the event (that is, the lookup ID) in the ?GChangeButton accepted embed.

```
! Get the person record from the browse queue
currentsel = choice(?browse:1)
GET(Queue:Browse:1,currentsel)
COP:NameCode = Queue:Browse:1:COP:NameCode
! Get the person record from the person table
IF access:COPeople.fetch(COP:codekey) = level:Benign
! Set the new group (Passed as an event code from the popup)
COP:Group = event()
! Update and refresh the browse
access:COPeople.update()
Brw1.ResetFromFile()
ThisWindow.reset(1)
ELSE
MESSAGE('Cannot retrieve person record',|
'Browsepeople error 3',ICON:Exclamation,BUTTON:OK)
END
```

That's it! The popup interface is easy to use, cuts editing time drastically, and most importantly the customer is happy.

[Murray Gillespie](#) is a partner in [LyGil Software](#), specialising in bespoke business applications, and has been using Clarion on and off since 1989. He lives on the south coast of Western Australia. When not coding he spends his time raising Sussex cattle.

Reader Comments

Posted on Wednesday, June 27, 2007 by Djordje Radovanovic

Very nice!

Posted on Thursday, June 28, 2007 by Bjarne Havnen

Nifty use of Event().

To handle text based edition too, you can use AddItem's second parameter to create an entry with name . Then use AddItemEvent to force this selection to go to a button and use Popup.GetLastSelection() to get the label of the element. If you do not need submenu, you can bypass the AddItem method and only use AddItemEvent's first parameter as name.

I prefer to store the assigned name in a queue since AddItem may alter it to follow certain rules.

A snippet

```
Self.Popup.AddItem('Sett status','SETTSTATUS')
LastItem='SETTSTATUS'                Loop Until Access:Statuser.Next()
  Statusq.Status = Self.Popup.AddItem(Clip(Status:Desc),Status:Status,LastItem,2)
  Statusq.Original = Status:Status
  Add(Statusq)
  Self.Popup.AddItemEvent(Statusq.Status,Event:Accepted,?btnSettStatus)
  LastItem=Statusq.Status
End
```

Then on the accepted of ?btnSettStatus

```
Statusq.Status = BrwOrdrelinjer.Popup.GetLastSelection()
Get(Statusq.Statusq.Status)
!handle value
```

Posted on Thursday, June 28, 2007 by Murray Gillespie

Thanks Bjarne,

Your method looks good to, especially if you have a fixed list - maybe populated into a queue. Ypu can then just grab the 'name' and assign it to a field

Murray

[Add a comment](#)

Clarion Magazine

Converting A Process To A Stored Procedure/Trigger

by Bernard Groperrin

Published 2007-06-15

A Clarion programmer I know is porting his whole application from TPS files to [PostgreSQL](#). For now, this does not involve any redesign of the database, but some specific batch processes are begging to be ported to SQL. My job is to convert some of these lengthy processes into stored procedures and/or triggers for greater speed and reliability. This isn't as difficult as you might think, and in this article I'll show you how it's done.

In this particular application the processes are hand coded, but the conversion method is exactly the same for template-generated processes.

Here's the first block of Clarion code in a process that deletes, inserts, and updates records:

```

MessageString='Deleting Pending CG Activity Records'
!Logout(1,CasedGoodsActivity)
CLEAR(CGA:RECORD, -1)
Glo:SeqNo = POH:SeqNo
CGA:POHSeqNo = Glo:SeqNo
SET(CGA:POHSeqNoKey,CGA:POHSeqNoKey)
LOOP
  Access:CasedGoodsActivity.TryNext()
  IF ERRORCODE() OR CGA:POHSeqNo <> Glo:SeqNo
    BREAK
  END ! END IF
  ! prevent delete of production records
  If CGA:CasesIn + CGA:BtIsIn = 0
    ! prevent delete of adjustment records
    If ~CGA:Adjustment
      If CGA:Pending
        0{PROP:Text} = 'Deleting: ' & CGA:ProductID
        Access:CasedGoodsActivity.DeleteRecord(0)
        !if errorcode()
          ! Message('Error Deleting Records:' & errorcode())
          ! Break
        !End !end if errorcode
      End ! pending
    End ! pending
  End ! pending

```

```

    End ! if not adjustment
    End ! if not production
END !LOOP

```

This first block of code deletes records in the CasedGoodsActivity file according to some specific criteria. In SQL, this translates as follows (with the value of Glo:SeqNo determined at run time):

```

DELETE FROM cga WHERE pohseqno = Glo:SeqNo
    AND (casesin + btlsin) = 0 AND adjustment = 0 and pending = 1

```

That single SQL statement meets all the same conditions to delete as does the much lengthier block of Clarion code, and since the code executes on the server it also runs much faster. The Clarion code is retrieving each record across the network before it decides whether or not to do a delete.

If I were to run this statement with PROP:SQL the Clarion code would look like this:

```

CasedGoodsActivity{Prop:SQL} = 'DELETE FROM cga
    WHERE pohseqno = ' & Glo:SeqNo & ' AND (casesin + btlsin) = 0
    AND adjustment = 0 and pending = 1'

```

But as I'll explain shortly, I am not going to use Prop:SQL.

Here's the next block of Clarion code:

```

POH:WeightRemaining = 0  ! for recalc
CLEAR(POD:RECORD, -1)
POD:POHSeqNo = Glo:SeqNo
SET(POD:POHSeqNoKey,POD:POHSeqNoKey)
LOOP
    Access:PODetails.TryNext()
    IF ERRORCODE() OR POD:POHSeqNo <> Glo:SeqNo
        BREAK
    END ! END IF
    Clear(CGA:Record)
    0{PROP:Text} = 'Begin update of: ' & POD:ProductID
    CGA:CustomerID = clip(POD:CustomerID)
    CGA:CustPONumber = clip(POD:CustPONumber)
    CGA:ProductID = clip(POD:ProductID)
    CGA:CasesOut = POD:QtyOrdered - POD:ShippedToDate
    CGA:POHSeqNo = Glo:SeqNo
    CGA:BtlsOut = 0
    CGA:Date = POD:POReqDate
    CGA:Time = Clock()
    If CGA:Time = LOC:Time
        CGA:Time +=1

```

```

End
LOC:Time=CGA:Time
CGA:Pending    = True
If CGA:CasesOut+CGA:BtIsOut > 0
  Access:CasedGoodsActivity.Insert()
  !if errorcode()
  ! message('error adding:' & errorcode())
  !else
  !End !end if error on add
End ! CasesOut >0
!! Update the weight remaining in POD record
POD:WeightRemaining = POD:CaseWeight* |
  (POD:QtyOrdered-POD:ShippedToDate-POD:CasesLoaded)
Access:PODetails.Update()
0{PROP:Text} = 'Finish update of: ' & CGA:ProductID
POH:WeightRemaining += POD:WeightRemaining
END !LOOP

```

This code loops through a child file to create records in another file, and updates the child file records. Unlike the previous example, which could be resolved to a single SQL statement, this part of the process requires the ability to perform several actions for each record found, which would mean multiple SQL statements, any one of which could fail. A better solution is to create a server-side procedure.

Transactional SQL

All major RDBMS have a *transactional SQL*, that is, a procedural form of SQL which can't be used in embedded SQL from a client but is used on the server to write stored procedures and triggers (I use the term *stored procedure* here, but your database may call this a user function, function, procedure, etc).

I will translate all of the above Clarion code into PL/pgSQL, which is PostgreSQL's Procedural Language.

PL/pgSQL has a number of interesting features. An important one is that inside a function/procedure, COMMIT and ROLLBACK are implicit. But don't confuse transactions with BEGIN and END statements in PL/pgSQL, which only define blocks of code; the transaction always encompasses the entire function or trigger. From the [PostgreSQL documentation](#):

PL/pgSQL's BEGIN/END are only for grouping; they do not start or end a transaction. Functions and trigger procedures are always executed within a transaction established by an outer query --- they cannot start or commit transactions, since PostgreSQL does not have nested transactions.

By default PostgreSQL is in auto-commit mode: if there is no error while executing a function or trigger the changes are COMMITed, otherwise a ROLLBACK is performed. There is no reason to use explicit COMMIT and ROLLBACK statements in a stored procedure. (It's different with embedded SQL, where either COMMIT or ROLLBACK is needed to terminate a BEGIN block. Remember that PL/pgSQL and SQL are two different languages, each with its own rules.)

Another nice feature of PL/pgSQL is the ability to LOOP through query results without needing to explicitly declare, and then open and close, a cursor. This makes PL/pgSQL LOOPS look much like Clarion LOOPS.

Here's a translation of the second block of code into a stored procedure, minus the insert code:

```

DECLARE
  arecord RECORD;
BEGIN
  FOR arecord IN SELECT * FROM pod
    WHERE pohseqno = someparameter ORDER BY pohseqno
  LOOP
    -- at this point, arecord contain one record from pod.
    -- To get a value from it, I can write x=pod.column
  END LOOP;

```

To create records in the other table I need an INSERT statement. If I supply values for only a few columns, all other columns will get their default values. Something like this will do the job:

```

INSERT INTO cga (customerid, custponumber, productid,
  casesout, pohseqno, btlout, date, time, pending)
VALUES (arecord.customerid, arecord.custponumber,
  arecord.productid, (arecord.qtyordered - arecord.shippedtodate),
  arecord.pohseqno, 0, arecord.poreqdate, now())

```

However, in the Clarion process the insert is conditional. The server-side code also needs to test for the condition:

```

IF (arecord.qtyordered - arecord.shippedtodate) > 0 THEN
  INSERT INTO cga (customerid, custponumber, productid, casesout,
    pohseqno, btlout, date, time, pending)
  VALUES (arecord.customerid, arecord.custponumber, arecord.productid,
    (arecord.qtyordered - arecord.shippedtodate),
    arecord.pohseqno, 0, arecord.poreqdate, now())
END IF;

```

Here's the code so far:

```

DECLARE
  arecord RECORD;
BEGIN
  --First block of code in Clarion process
  DELETE FROM cga WHERE pohseqno = someparameter
  AND (casesin + btlsin) = 0 AND adjustment = 0 AND pending = 1

```

```
-- Second block of code in Clarion process
FOR arecord IN SELECT * FROM pod
WHERE pohseqno = someparameter ORDER BY | pohseqno
LOOP
  IF (arecord.qtyordered - arecord.shippedtodate) > 0 THEN
    INSERT INTO cga (customerid, custponumber,
      productid, casesout, pohseqno, btlsout, date,
      time, pending)
    VALUES (arecord.customerid, arecord.custponumber,
      arecord.productid, (arecord.qtyordered - arecord.shippedtodate),
      arecord.pohseqno, 0, arecord.poreqdate, now())
  END IF;
END LOOP;
```

Looking at the original Clarion code, I see that the PoDetails record is updated at the end of the second loop. This translates to:

```
UPDATE pod SET weightremaining = arecord.caseweight *
(arecord.qtyoredered - arecord.shippedtodate - arecord.casesloaded)
WHERE podetailsid = arecord.podetailsid;
```

And the last line inside the loop sums the value just calculated. So it might be better to declare two local variables to make this easier to read and to store the temporary value:

```
DECLARE
  arecord      RECORD;
  loc_weightremaining  numeric(6,0);
  loc_totalweight    numeric(6,0);
```

With these additions the code is changed as follows:

```
loc_weightremaining = arecord.caseweight *
(arecord.qtyoredered - arecord.shippedtodate - arecord.casesloaded);
UPDATE pod SET weightremaining = loc_weightremaining
WHERE podetailsid = arecord.podetailsid;
loc_totalweight = loc_totalweight + loc_weightremaining;
```

Here's the whole loop so far:

```
FOR arecord IN SELECT * FROM pod
WHERE pohseqno = someparameter ORDER BY pohseqno
LOOP
  IF (arecord.qtyordered - arecord.shippedtodate) > 0 THEN
    INSERT INTO cga (customerid, custponumber, productid,
```

```

        casesout, pohseqno, btlsout, date, time, pending)
        VALUES (arecord.customerid, arecord.custponumber,
arecord.productid, (arecord.qtyordered -
arecord.shippedtodate), arecord.pohseqno,0,
arecord.poreqdate, now());
END IF;
loc_weightremaining = arecord.caseweight *
    (arecord.qtyoredered -| arecord.shippedtodate -
arecord.casesloaded);
UPDATE pod SET weightremaining = loc_weightremaining
    WHERE podetailsid=| arecord.podetailsid;
loc_totalweight = loc_totalweight + loc_weightremaining;
END LOOP;

```

Outside of that loop, in my Clarion process, there is one more block, a single line of code, but still a very important one:

```

!! Update the total remaining in POH record
0{PROP:Text} = 'Begin update of PO Header '
Access:POHeaders.Update()

```

Hmm, does that mean that at this point the PoHeaders buffer is available to me, and I can simply update it? Looking at my Clarion application, I realize that this process is called from a Clarion form, inside the ThisWindow.Kill embed.

Because this process is called when an action is performed on a record, it is a good candidate for a trigger, not a stored procedure! Triggers can happen before or after a particular action is taken on a record. If I make this a BEFORE event trigger, I can modify the values of the parent record before they are actually written to the table, and any error in the process will cause a ROLLBACK. Because the code is deployed as a trigger, both the PoHeaders record change and the various actions that are part of the trigger are contained in a single transaction. If there is any error anywhere in the process, all those changes will be rolled back. Sounds good!

For PostgreSQL, a trigger simply calls a specific function, so there are not many differences between writing a trigger and a function/stored procedure. But from inside a trigger, all the columns from the table to which he trigger is linked to are available. The prefix NEW identifies access to the values just changed or inserted by he user, while the prefix OLD give access to the previously existing values. Obviously OLD is not available during an insert, and NEW is not available during a delete.

Once I move my code into a trigger, the last line of code becomes:

```
NEW.weightremaining = loc_totalweight
```

And the "someparameter" I was using before can now be NEW.seqno.

The whole function is now:

```
CREATE OR REPLACE FUNCTION
```



```

"public"."poh_insdel_function" () RETURNS trigger AS
$body$
DECLARE
    arecord          RECORD;
    loc_weightremaining  numeric(6,0) = 0;
    loc_totalweight    numeric(6,0) = 0;

BEGIN

    /* First Block of Clarion Code */
    DELETE FROM cga
    WHERE pohseqno = NEW.seqno
    AND (casesin + btlsin) = 0
    AND adjustment = 0
    AND pending = 1;
    /* Second Block of Clarion Code */

    FOR arecord IN SELECT * FROM pod WHERE pohseqno = NEW.seqno
    LOOP
        IF (arecord.qtyordered - arecord.shippedtodate) > 0 THEN
            INSERT INTO cga (casedgoodsactivityid,
                customerid,
                custponumber,
                productid,
                casesout,
                pohseqno,
                btlsout,
                date,
                time001,
                pending,
                seqno)
            VALUES ((SELECT max(casedgoodsactivityid) FROM cga) + 1,
                arecord.customerid,
                arecord.custponumber,
                arecord.productid,
                (arecord.qtyordered - arecord.shippedtodate),
                arecord.pohseqno,
                0,
                arecord.poreqdate,
                1,
                0,

```

```

        (SELECT max(seqno) FROM cga) + 1);
    END IF;
    loc_weightremaining = arecord.caseweight *
        (arecord.qtyordered - arecord.shippedtodate -
        arecord.casesloaded);
    UPDATE pod
    SET weightremaining = loc_weightremaining
        WHERE podetailsid= arecord.podetailsid;
    loc_totalweight = loc_totalweight + loc_weightremaining;
    END LOOP;
    /* Notice that, being inside a trigger, there is no need to UPDATE
    the parent record... */
    NEW.weightremaining = loc_totalweight;
    /*For PostgreSQL, there are only functions, no procedures,
    so you always have to return "something", even if the \
    something is undefined....*/
    RETURN NULL;
END;
$body$
LANGUAGE 'plpgsql' VOLATILE CALLED ON NULL INPUT SECURITY INVOKER;
/* Notice that, being inside a trigger, there is no need to UPDATE
the parent record... */
NEW.weightremaining = loc_totalweight;
/*In PostgreSQL there are only functions, no procedures,
so you always have to return "something", even if the
something is undefined....*/
RETURN NULL;
END;

```

There is one last thing that I want to add to this code, and it is to detect if I have any child records for my parent. If there aren't any child records, I don't want to set NEW.weightremaining = 0.

PostgreSQL has a few built-in variables; I will use FOUND here and change the beginning of my second block of code to be:

```

SELECT * FROM pod WHERE pohseqno = NEW.seqno;
IF FOUND THEN

```

I will also add an END IF just before the RETURN NULL statement

If the SELECT statement finds at least one row FOUND will be true, and the loop will execute. If there are no child records the trigger won't do anything.

As I said earlier, the trigger itself is a transaction which has to execute entirely without error to be committed. I don't need additional error checking, but if I wished I could use RAISE ERROR to artificially create errors.

As you can see, writing a trigger is no more difficult than writing embed code in Clarion, once you have some familiarity with SQL. And triggers and stored procedures generally involve fewer lines of code than the Clarion equivalent.

The code for the trigger which calls the function is quite short:

```
CREATE TRIGGER "poh_insdel_trigger" BEFORE UPDATE
  ON "public"."poh" FOR EACH ROW
  EXECUTE PROCEDURE "public"."poh_insdel_function";
```

Pros and cons

There are a few downsides to writing stored procedures and triggers. You'll have to learn a different language, and there is some inconvenience in splitting code between your application and the back end.

On the other hand, stored procedures and triggers offer some important advantages over client side code:

- Speed is a very big factor in large database processing. As the trigger is compiled in the database engine itself, and as it process all records locally rather than across the network, you get all the speed your server can give you. Typically, a trigger like this, with about 40 to 100 child records, will take about 30 to 60 milliseconds to execute.
- Because all processing is on the server, network conditions are not a factor, and the transaction is more robust.
- Server-side code increases data integrity. If any other application updates or inserts a record in that particular table, your "rule" will be applied, always. When you are working with any SQL backend, you should keep in mind that anyone with minimal MS Office training (and suitable permissions) can easily access a database via ODBC and play with the data, being totally unaware of the consequences for relational integrity.

I hope this simple, real-world example will help you get started writing your own triggers and stored procedures, in whatever SQL database you use.

[Bernard Groperrin](#) is a native of France, and has been a big fan of everything American since his teens. He began visiting the US in 1996, and moved to San Antonio in 1998, where he discovered his love of Mexican food. He and his lovely wife Gloria, who he met in Tulsa, now live in California. Bernard has been programming and designing software and databases for 14 years, primarily with Clarion. His hobbies include flying radio-controlled airplanes and riding his motorcycle. He loves aircraft of all kinds, and can't miss an opportunity to fly, whether it's in a glider, a World War II trainer, or a general aviation Piper or Cessna.

Reader Comments

Posted on Friday, June 15, 2007 by S Jayashankar

Hi BG,

Nice Article. I do agree that using stored procedures is faster and more efficient than processes but the biggest stumbling block is if the processing time is slightly long, how does one show the progress/status of the execution to the user? Also, if we use PROP:SQL there is no feedback from the backend database on completion. The only solution that I can think of is using another query (a COUNT() query) on a timer to check the progress and completion. Any others that you can think of.

Regards

.....

Posted on Friday, June 15, 2007 by Wayne Freeman

Hi BG,

You said early on in the article that you weren't going to use Prop:SQL, but then, as far as I could tell, you didn't give an example of how you were going to call the stored procedure without using Prop:SQL.

Good article, by the way.

Wayne

[Add a comment](#)

Clarion Magazine

CapeSoft World Tour Las Vegas

by John Rae and Brian Reid

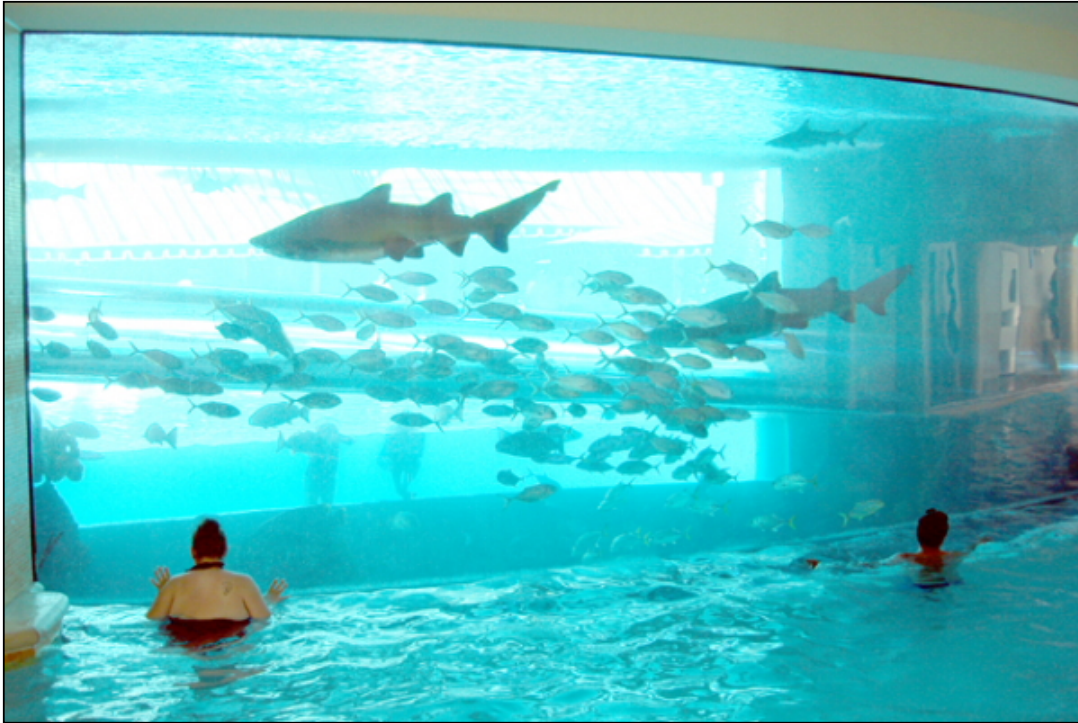
Published 2007-06-13

At the end of May over fifty Clarion developers met in Las Vegas for the only North American stop on the [CapeSoft World Tour](#).

Anyone that thought this was a sales tour missed an excellent opportunity to discuss and develop all things Clarion.

The venue was the [Golden Nugget](#) hotel, which provided nice and relatively inexpensive accommodation, although the downtown Las Vegas location provided plenty of distractions to consume our spare change after classes finished each day.





The program ran for five days beginning early on a Tuesday and finishing up around dinner time on Saturday. Training ran from 7:00 a.m. to 7:00 p.m. and included theory sessions as well as dedicated two hour slots for question and answer on any Clarion-related topic, not just CapeSoft products. This was an excellent format as it allowed ample time for discussion and development of the concepts covered in the theory sessions.



I saw members of the CapeSoft team rolling up their sleeves* and literally coding solutions for folks who were having trouble with various Clarion projects. If attendees were stuck on problems with other third party tools, Bruce or members of his team took the time to contact those vendors to resolve issues.

All CapeSoft staff were knowledgeable and keen to assist with development issues and ideas.



Several sessions were repeated throughout the five days to allow everyone to get to as many of the concurrently run workshops as possible. To take full advantage of the sessions you really needed to come prepared and organize your time effectively.



The style and presentation generated a peer-to-peer type of atmosphere in the classes that made the whole experience very positive and productive. CapeSoft came well prepared with a course CD and two books made specifically for the World Tour event. As John Rae remarked, "It was really good to be able to chat with the actual authors of the CapeSoft tools that you were using and as they related to the sample projects that you were working on. There were nuances of their usage that would be difficult to discover otherwise."

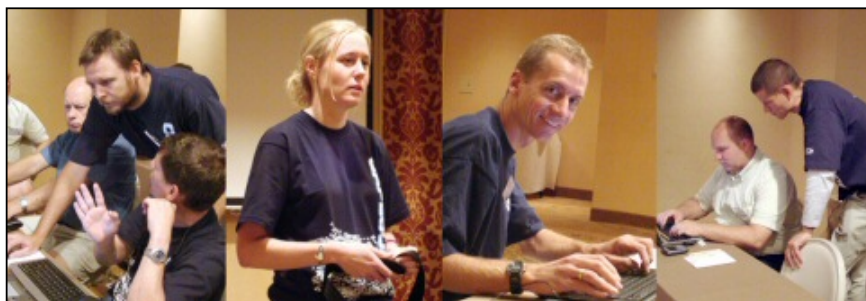
For John, myself, and many others, one of the highlights of the tour was the detailed information on CapeSoft's Web Server product. What an amazing technology. The sessions also included a great educational opportunity for anyone with an interest in networking fundamentals, FTP, mail, HTTP, TCP/IP protocols, SSL, etc. The information presented here wasn't something you could glean from a text or even Google in five days' time.

Another interesting product that CapeSoft has been working on is Right Reports, a report writer for your applications. It's early days yet, but this looks promising.

The week wrapped up with a session on VISTA from a developer's perspective. The challenges and accommodations necessary to bring products from XP to VISTA were covered in some detail. And finally there was a brief discussion of C7 and the impact of the new IDE on Clarion developers. For the most part there were no alarm bells going off during the dialogue, just a lot of folks anxious to see how it will improve their Clarion work environment.

It's hard to capture in a few words the volume of ideas, development strategies, and the ever present comradery of a room full of Clarion developers. Take a look at the faces in the photos. These were serious days of development for many of the attendees. CapeSoft did a superb job in supporting all the attendees in whatever aspect of Clarion that they came to spend time on. Hats off to Bruce and his team for a superb five days of intensely focused Clarion education and development.

*The presenters all wore CapeSoft Tee shirts, as did many of the attendees by the end of the week)



More photos

- [Leroy Schulz's gallery](#)
- [Jono Woodhouse's gallery](#)

Reader Comments

[Add a comment](#)

Clarion Magazine

Kalashnikov Programming

by Tim Phillips

Published 2007-06-11

Inspiration for a programming style can come from the oddest of places. I call my general philosophy *Kalashnikov Programming*. It is named after [Mikhail Kalashnikov](#) and what I see as the extraordinary attributes of the series of assault rifles (AK-47, AKM, AK-74, AK-103) that he has designed.

Let's run down through the design goals that I see as striking in the Avtomat Kalashnikova line and see how it is an excellent model for good Clarion programming.

Build rugged

The AK is renowned for being "tough". There is a reason they are so common in third world countries where repair facilities are non-existent: these rifles simply keep working day-in/day-out in dust, rain and gunk. The Kalashnikov is a weapon you can depend upon to keep working when you need it the most.

In Clarion terms, this means keeping as much as you can within the well-debugged world of templates and classes. It means making changes with care and anticipating what could go wrong and providing a graceful way to handle the error.

Build simple

AKs are mostly stamped steel and wood. Expensive custom machining is kept to a minimum, as is the total number of parts. This keeps the cost to build a rifle down and means repairs/maintenance are cheaper and easier.

In Clarion terms, this means writing as little hand code as you humanly can. Templates and classes are your friends. I am generally willing to trade in some performance if it leads to a simpler design that I can build or maintain more easily.

Chrome-plate where it counts

Despite inexpensive stamped sheet metal parts intended for mass production during wartime shortages, the AK line has always benefited from chrome-lining of the barrel and receiver. The hard chrome protects the machined barrel from rust caused by inclement climate and corrosive pitting caused by poor

grade ammunition. It is a relatively expensive step that enormously benefits a weapon that must function for potentially decades on end.

In Clarion terms, this means paying careful attention to where something extra is needed. You do not blindly make every little piece perfect, but you do commit time and energy when it is clear the long-term effect will be large, or where the issue is key to the use of the program.

Anticipate unskilled operation

The AK was built for service in the mass-conscription armies of the old USSR. Training was often minimal and hindered by the dozens of languages spoken by the masses.

In Clarion terms, this means recognizing that many of your users won't read the printed manual, will ignore the Help, will not pay for the training, and will still expect the software to do what they want. You need to work on being able to "mentally erase" everything you know about your software, much of what you know about computers and even much about what your software actually does. You need to become a new user with only a) some vague notation of what you may want to do and b) a need for results *now!* The less users have to know to successfully use your software, the better it will be.

Reuse where you can

The original AK-47 has led to the AKM, the AK-74, the RPK, the AK-101 and Saiga-12 (to name just a handful of "children"). There are submachine guns, assault rifles, light machine guns, hunting rifles and automatic shotguns based upon the AK design. In each case the original design was tweaked only as needed to match the new use.

In Clarion terms, this means being diligent about the traditional "code once, use everywhere" concept.

Do whatever works for you to keep track of the myriad of templates, classes and blocks of procedural code within the scope of your programming environment (I personally have a code snippets program where I keep code samples and step-by-step how-tos for adding specific features). Invest time into figuring out how to build at least basic templates and classes of your own. The time you invest will be repaid handsomely every time you use your templates or classes.

"Borrow" good ideas

The concept for the AK is reputed to have come from the German intermediate cartridges and various "storm guns" used by the Germans on the Russian Front during WWII. If Kalashnikov "stole" the idea of the AK, he at least had the design sense to recognize something good when he saw it. The standard personal weapon for soldiers world-wide now is some form of assault rifle; Kalashnikov's design was not only one of the very first, but has been successful for some sixty years now.

In Clarion terms, this means keeping an open mind to solutions and not getting caught up in the "not invented here" syndrome. If you can buy a template that fills your need, don't reinvent the wheel, just

"borrow" the good idea by buying the template.

Six simple concepts.

If the six simple concepts I've outlined here lead to even a fraction of the success that Mikhail Kalashnikov's designs have had down through the years, you should be pleased.

[Tim Phillips](#) began programming Clarion with Version 3.0 for DOS. He currently works with Clarion 5.5/6 ABC. His preferred programming technique involves a lot of bass rock guitar on his cordless headset and vigorous application of the Keep It Simple Stupid principle. When not programming, he can be found trying to write fiction, "building Legos" with his two nieces, or being obsessive about television shows that have gone off the air.

Reader Comments

[Add a comment](#)