Clarion Magazine

Clarion News

- » Charter Clarion.NET Subscription Extended
- » AppStats
- » FullRecord 1.85
- » Super Field-Filler 6.62
- » Icetips Magic Locks build 1.10.100
- » Icetips Utilities Beta 3.3
- » Save 25% On Automated Fax Engine
- » StrategyOnline Price Increases
- » J-Cal In Beta
- » Super Limiter 6.62
- » FullRecord 2.04
- » TagKeys 2.0
- » iQ-XML 1.27
- » RPMxt for C6.1 9026 thru C6.3 9050
- » Super Passcode 6.62
- » Icetips Previewer Build 2.60.100
- » Snazzy ListBox v1.34
- » Free Zip'N FTP Utility
- » Clarion Desktop 4.05
- » Free CHT Zip Utility
- » Icetips Utilities Beta 3.1
- » Data Management Center 1.0.0.9
- » CHT New Subscriptions Price Reduced 40 Percent
- » SpecialFolders Update
- » Whitemarsh November 2007 Announcement
- » C6 9057 Assertions On Code Generation
- » CHT 11D1.02
- » Clarion.NET Purchase Information
- » Clarion.NET Charter Subscription
- » C6.3 9057 Released
- » vuFileTools Newest Function
- » Data Manager Center 1.0.0.7
- » Smart-type Technical Support Unavailable Nov 6-13
- » SetupBuilder 6.6 Build 2040
- » SV Blog Post On Moving To .NET
- » Lodestar Products And C6.3 9057
- » Clarion Daily News Hits 10K
- » ClarionFAQ Temporarily Down
- » Super Security 6.62
- » PostgreSQL Multi-Master Replication
- » It's offically Clarion#

[More news]

[More Clarion 101]

Latest Subscriber Content

IMPORTANT: Change of Email Addresses

Clarion Magazine's email addresses have changed. Please visit our **contact page** and update your address book.

Posted Thursday, November 22, 2007

PostgreSQL Revisited:Porting From TPS

In this fourth installment in the PostgreSQL series Dave Harms shows how to create a database, set up users and roles, and convert the People application to PostgreSQL.

Posted Friday, November 30, 2007

Choosing Colors: Selected Tools

David Harms explains hex color notation (and why HTML colors are different from Clarion colors) and reviews several tools that make choosing a set of compatible colors less painful, if not easy.

Posted Thursday, November 29, 2007

Clarion# Language Comparison

Mike Hanson has prepared a cross-reference showing Clarion# equivalents to VB.NET and C# statements. Topics include program structure, comments, data types, constants, enumerations, operators, choices, loops, arrays, functions, strings, exceptions, namespaces, classes, interfaces, construtors, using object, structs, properties, delegates, events, and I/O. Based on a VB.NET/C# document by Frank McCown.

Posted Monday, November 26, 2007

Clarion.NET First Look

The first beta of Clarion.NET is out! No AppGen yet, but this release supports WinForms (desktop), WebForms (ASP.NET), and Compact Framework (mobile) development, as well as C# and VB.NET coding. Dave Harms reports.

Posted Tuesday, November 20, 2007

The Clarion.NET FAQ

A list of frequently-asked questions about Clarion.NET/Clarion#, and some hopefully informative answers.

Posted Saturday, November 17, 2007

BLOG: Clarion.NET Beta Released

The Clarion.NET beta has been released! As with C7 there's no AppGen yet but a C6 wizard is included which will generate a Clarion.NET application from your dictionary. You can explore and modify the source code using Clarion.NET. You can write console and WinForms apps from scratch, as well as ASP.NET and mobile apps. The IDE also accommodates C# and VB.NET. SV has opened a new newsgroup called softvelocity.public.clarionsharp. Details to follow.

Posted Saturday, November 17, 2007

A StringClass For Clarion

A common feature of object-oriented languages is a string class featuring common string-manipulation features. Rick Martin introduces a Clarion StringClass that supports creation and manipulation of strings of arbitrary length.

Posted Friday, November 16, 2007

PostgreSQL Revisited: Managing User Rights

In this third installment on the PostgreSQL database server David Harms covers basic access control issues and explores the concepts of users, roles, and rights.

Posted Wednesday, November 14, 2007

SoftVelocity Site Gets New Look

Latest Free Content

- » Clarion# Language Comparison
- » Clarion.NET First Look
- » The Clarion.NET FAQ
- » BLOG: Clarion.NET Beta Released
- » SoftVelocity Site Gets New Look
- » Source Code Library 2007.10.31 Available

[More free articles]

Clarion Sites

Clarion Blogs

Posted Tuesday, November 13, 2007

Source Code Library 2007.10.31 Available

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

Posted Wednesday, November 07, 2007

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

Source Code

The ClarionMag Source Code Library

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

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

More info • Subscribe now

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

PostgreSQL Revisited:Porting From TPS

by Dave Harms

Published 2007-11-30

This is the fourth article in my series on PostgreSQL, but it's only now that I'm ready to show how to actually use PostgreSQL with a Clarion application. If you haven't read those previous articles you may be wondering what's taking me so long. It's really a question of moving from dumb data to smart data, if you like. SQL databases typically add a great deal of processing power to your applications, at a cost of some additional complexity, and it's worth understanding what the database server requires of you and what it can do for you. If you haven't read the previous articles, I suggest you do so now. Part 1 covers installing PostgreSQL, Part 2 looks at the utilities and administration tools included with the install, and Part 3 examines the authentication/authorization system.

In this article I'll convert the People example application from TPS to PostgreSQL; along the way I'll show how to create a database, grant a user rights to use that database, and how to point the People application at a PostgreSQL data source.

Creating a database

What is a database? If you're accustomed to TopSpeed (TPS) files, you probably think of the collection of TPS files used by any one application as its database. Besides the raw data, each table (file) in that database probably has one or more keys, and those keys are part of the structure of the TPS file. But really this kind of a database is a logical construct - there's nothing forcing all of the TPS files into one cohesive structure. And nothing ties one table to another except the code in your application.

An SQL database, on the other hand, is a cohesive structure. In PostgreSQL, a database can contain numerous elements, including (but not limited to) tables, indexes, and constraints.

You have to explicitly create a database; you can do this with the CREATE DATABASE command, but it's easier to use one of the available GUI administration tools, such as pgAdmin III, which is included in the standard PostgreSQL installation.

Figure 1 shows pgAdmin III; you can see that it's connected to the server and is displaying some basic information about the server's existing databases.

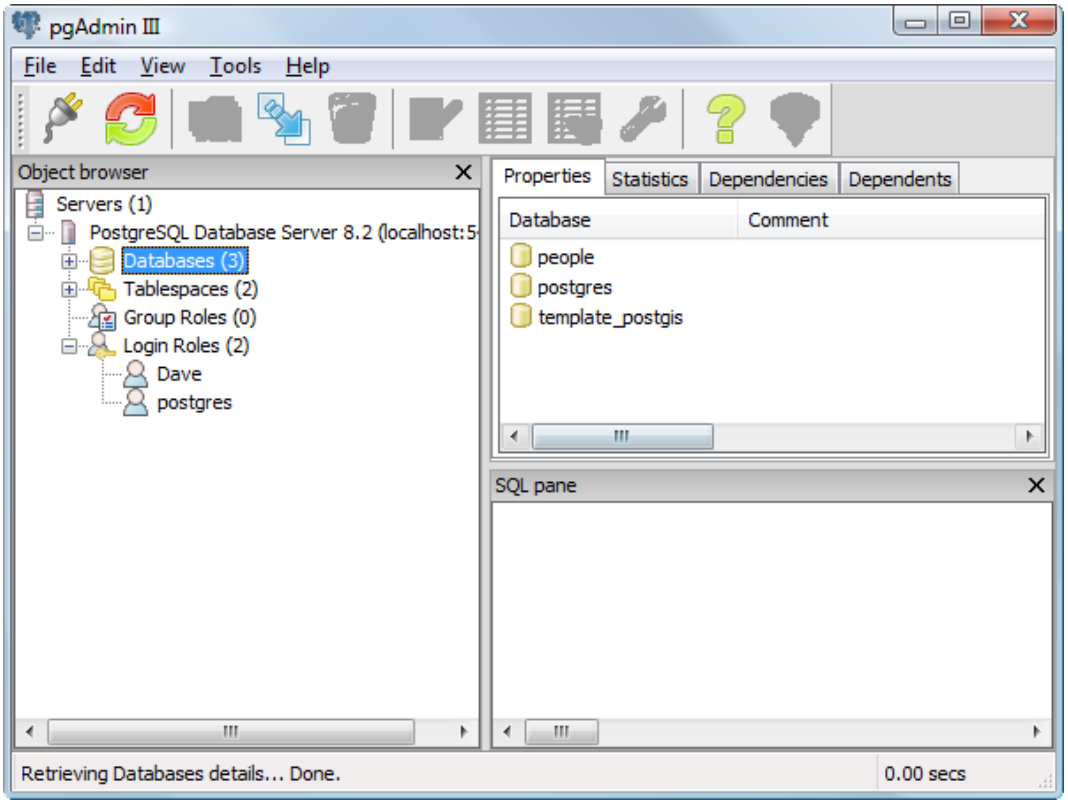


Figure 1. pgAdmin III

Right-click on the Databases node and select New Database.... You'll see the New Database form shown in Figure 2.

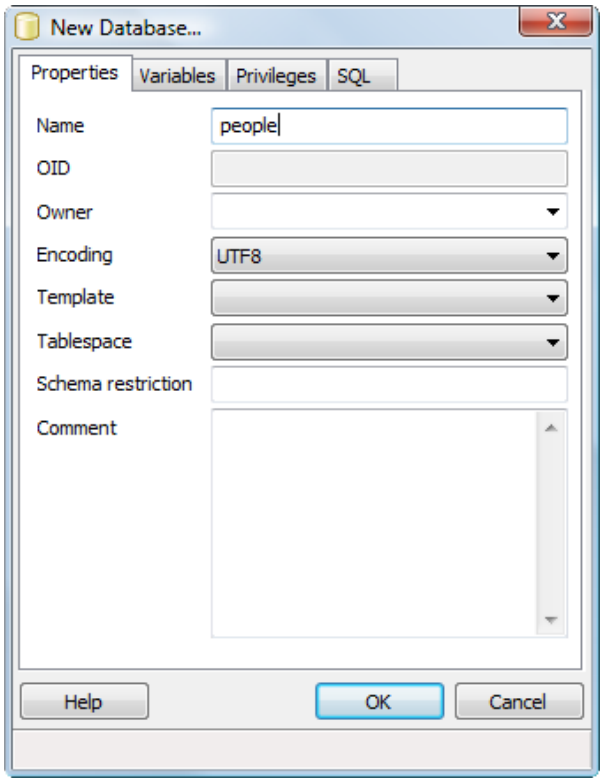


Figure 2. Adding a new database

There are only two required fields - you must enter a database name (I'm using people, which is the same name as the application and the one data file), and you must choose an encoding (which defaults to the selected or default tablespace encoding).

You'll probably want to enter the name of the database in lower case - I'll explain why in just a moment.

For encoding I suggest you give serious consideration to UTF8 encoding, which is an 8 bit Unicode format that is backward compatible with ASCII.

Encoding is the standard by which specific combinations of bits and bytes are resolved to specific characters. PostgreSQL supports a great many different encodings, including most single and double-byte systems. But the beauty of UTF-8, at least for North American users, is it only needs one byte for the 128 US-ASCII characters, while still supporting two, three, and even four byte characters if needed. Clarion 7 has Unicode support, so at least in theory (I haven't tested this) you could create a C7 app with a PostgreSQL UTF8 database and support all languages (although you would have to increase your field lengths as necessary - doubling the length would accommodate accented Latin, Greek, Cyrillic, Armenian, Hebrew, Arabic, Syriac, and Thaana - should you have customers in the Maldives - alphabets, and tripling would pretty much handle the rest as four byte characters are very rare).

If you absolutely want to stick with a single-byte non-US character set then you probably know your encoding and can choose it now.

If you click on the SQL tab you can see the actual SQL statement pgAdmin III will execute. Click OK and you should see your database listed under the Databases node. Note that PostgreSQL has supplied values for all the fields you did not enter (Figure 3).

Property	Value
Name	people
OID	16954
Owner	postgres
ACL	
Tablespace	pg_default
Encoding	UTF8
Default schema	public
Allow connections?	Yes
Connected?	Yes
System database?	No
Comment	

Figure 3. Database properties

Case sensitivity

You'll recall that I said you should use a lower case name for the database. In fact, you should use lower case names for all databases, tables, columns, and other entities in PostgreSQL. Here's why:

PostgreSQL automatically converts any labels in any SQL statements to lower case. This results in a kind of case insensitivity. If you have a field called lastname, you can write SQL that refers to it as lastname, Lastname, LastName, or lAStnaME for all I care, and PostgreSQL will convert your label to lastname, and all will be well. But if you use any upper case characters in your labels, you'll need to specify those labels in quotes or PostgreSQL will give you an error, even if what you type exactly matches the case of the label. And that can be a real pain, because you'll need NAME attributes on everything and you'll have to add quotes around labels whenever you write embedded SQL.

Happily, it's not that easy to create upper case labels by accident, at least not from SQL statements. For instance, the command CREATE DATABASE People... will yield a database named people, not People, because the label is folded to lower case. But CREATE DATABASE "People"... preserves the case. The place to watch out is when you're using a GUI administration tool such as pgAdmin III. If, when creating a database using the process just described, you specify something other than a lower case label, pgAdmin III will enclose that label in quotes, and you'll have to do likewise whenever you use the label in a query..

The rule of thumb, then, is to make sure that all your labels are lower case; then you can write the labels in your code using any case you like.

Creating a table

In SQL terminology, a *table* is generally equivalent to the data portion of what is in Clarion traditionally called a *file*. For instance, here's the file definition for People.TPS:

```
People          FILE,DRIVER('TOPSPEED'),CREATE,BINDABLE,THREAD
KeyId             KEY(PEO:Id),NOCASE,OPT,PRIMARY
KeyLastName         KEY(PEO:LastName),DUP,NOCASE
Record            RECORD,PRE()
Id                LONG
FirstName           STRING(30)
LastName            STRING(30)
Gender            STRING(1)
                END
              END
```

Note the CREATE attribute on that definition. If you don't have an existing People.TPS file, your application will create one.

Similarly you need to create a SQL table before you can use it, but advise you against letting the application do that job, primarily because the driver may not create the table the way you want it created (I'll describe one such situation a little later). You really want to see that SQL create script so you can review it and make any changes if needed.

Here's a PostgreSQL CREATE TABLE statement for people:

```
CREATE TABLE people (
  Id INT NOT NULL ,
  FirstName CHAR(30) NULL ,
  LastName CHAR(30) NOT NULL ,
  Gender CHAR(1) NULL
);
```

Do you see anything missing? Right, the TPS file has two KEY statements:

```
KeyId             KEY(Id),NOCASE,OPT,PRIMARY
KeyLastName         KEY(LastName),DUP,NOCASE
```

The first key is a primary key, which is used to store a unique identifier for each row. The second key provides a

convenient sort order, and I'll deal with that one first.

Here's the SQL statement to recreate the KeyLastName in PostgreSQL:

```
CREATE INDEX PEO_KeyLastName
  ON people (LastName)
;
```

There's an important difference, however, between KEY and INDEX. Just because you have a KEY defined in your Clarion data dictionary doesn't mean you'll always have a corresponding INDEX on the server (although you probably will); similarly you may have INDEXes on the server that are not represented in your data dictionary by a KEY.

The Clarion file access grammar lets you SET file processing order by KEYs, if you wish. But the SQL select statements generated by the file driver subsytem never references any keys declared in the dictionary; instead, the statement includes an appropriate ORDER BY clause, specifying the sort fields. Consequently, there can be a complete mismatch between KEYs defined in your dictionary and INDEXes on the database server, and your app will still run, though perhaps not optimally. The purpose of an SQL INDEX is simply to speed up access to data, not to enable access to data.

The important thing is that the fields in your FILE declaration correspond to fields in the SQL table (although the SQL table can also have as many additional fields as you like).

That takes care of the KeyLastName. Now, consider the SQL version of KeyID, which is a primary key. The equivalent here is not an INDEX but a CONSTRAINT:

```
ALTER TABLE people
  ADD CONSTRAINT PEO_KeyId PRIMARY KEY (Id)
;
```

A constraint is a limit on the kind of data that can be contained in a table or column. There are various kinds of constraints in PostgreSQL, and this one is of the type PRIMARY KEY. That means that each value of Id must be unique, and null values are not allowed. (You can create server-side autonumbered primary keys using *sequences*, but that subject is beyond the scope of this article.)

The DCT2SQL template

Probably the best way to create a PostgreSQL database is to create the tables in your data dictionary, and then use a template utility to generate the SQL scripts. Roberto Artigas' excellent DCT2SQL is one such utility; it includes templates for Firebird, Interbase, Mimer, MS_SQL, MySQL, Oracle, PSQL, and Sybase, as well as PostgreSQL (with contributions by Bo Schmitz, Eddie Sizemore, Vadim Berman, Lee White, Lew Strock, Matt Gorman, S. Jayashankar, Richard Bryce, and Russ Eggen). Some templates have several versions, and there are a number of additional utilities included. The utility is available from Steve Parker's Par2.com download page, but for PostgreSQL (at least at present) I suggest you use the version available at the end of this article, as I've made some changes to add the file prefix to index names to avoid name collisions. I've submitting my changes to Roberto for inclusion in his releases.

To use the template you must first unzip the files into your Clarion template directory. Then run Clarion and choose Setup|Template Registry. Click on Register, choose DCT2SQL.TPL and click Open. Close the registry and open your application - if you're following along with the example in this article, you'll want to make a copy of the People example directory and open that application.

From the main menu choose Application|Template Utility (or press Ctrl+U). Locate the Dct2PostgreSQL template and

double-click on it or press Select. You'll see the window in Figure 4.

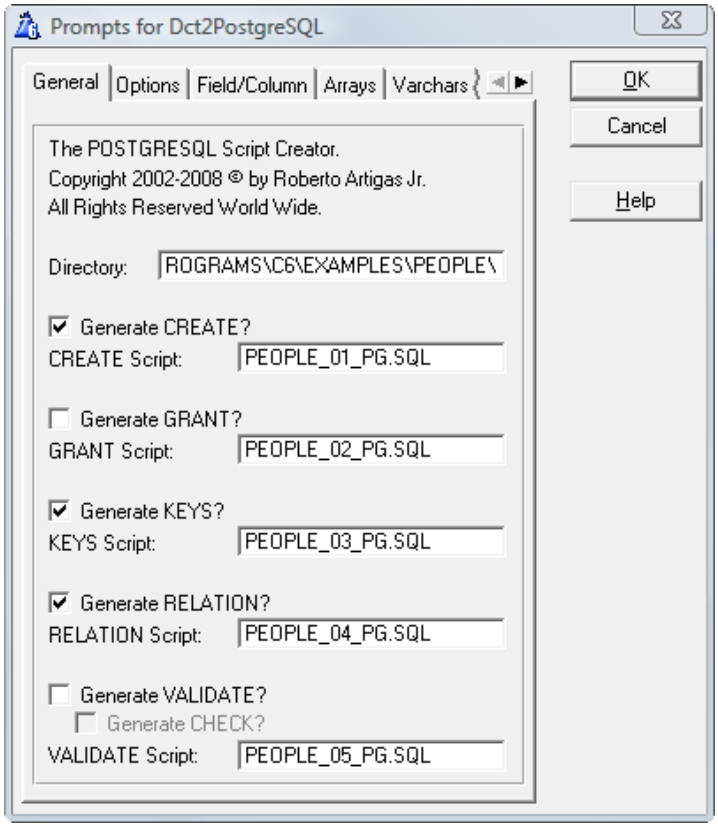


Figure 4. The utility template

The default settings should be fine - just click OK to generate the scripts. Three text files will be created, one to create the tables, one for indexes and constraints, and a third for relationships. If you're using the People app, only the first two will have SQL statements, since People only has one table.

Here again are the SQL statements generated for People.APP:

```
CREATE TABLE People (
  Id INT NOT NULL ,
  FirstName CHAR(30) NULL ,
  LastName CHAR(30) NOT NULL ,
  Gender CHAR(1) NULL
)
;
ALTER TABLE people
  ADD CONSTRAINT PEO_KeyId PRIMARY KEY (Id)
;

CREATE INDEX PEO_KeyLastName
  ON people (LastName)
;
```

I said earlier that it isn't usually a good idea to let the application create the tables automatically, and this is as good an example as any. If you let your app create the people table, it will be missing the primary key constraint - the driver instead generates code to create an INDEX on Id.

Now you're ready to run this SQL code on the server. In pgAdmin III select the people database node, then choose Tools|Query Tool, or click on the SQL toolbar button. Don't be misled by the term "Query Tool" - this is a utility that lets you execute SQL statements against the server. Those statements don't have to all be queries.

Paste the above script into the upper-left hand pane and press the green Execute button the toolbar (or press F5, or choose Query|Execute). Figure 5 shows the result.

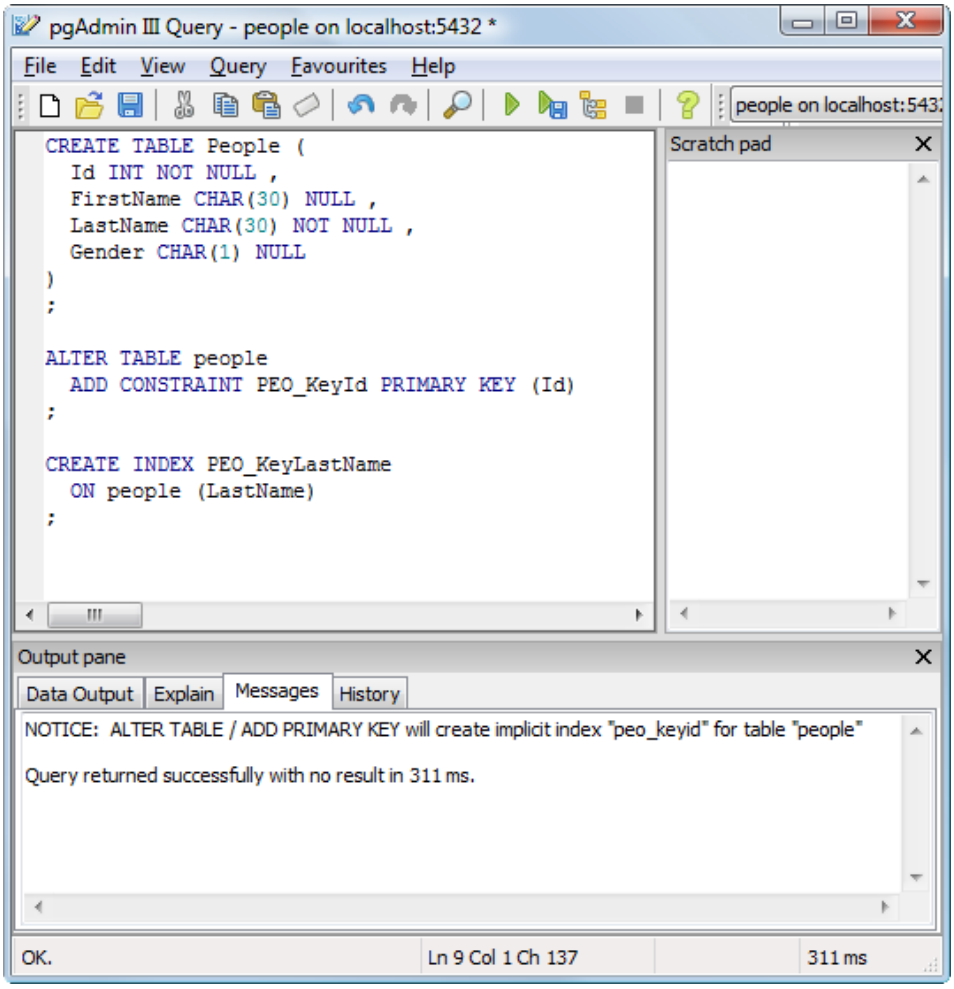


Figure 5. Creating a table

Go back to pgAdmin III and expand the following nodes: Databases|people|Schemas|public|Tables. Click on people (see Figure 6).

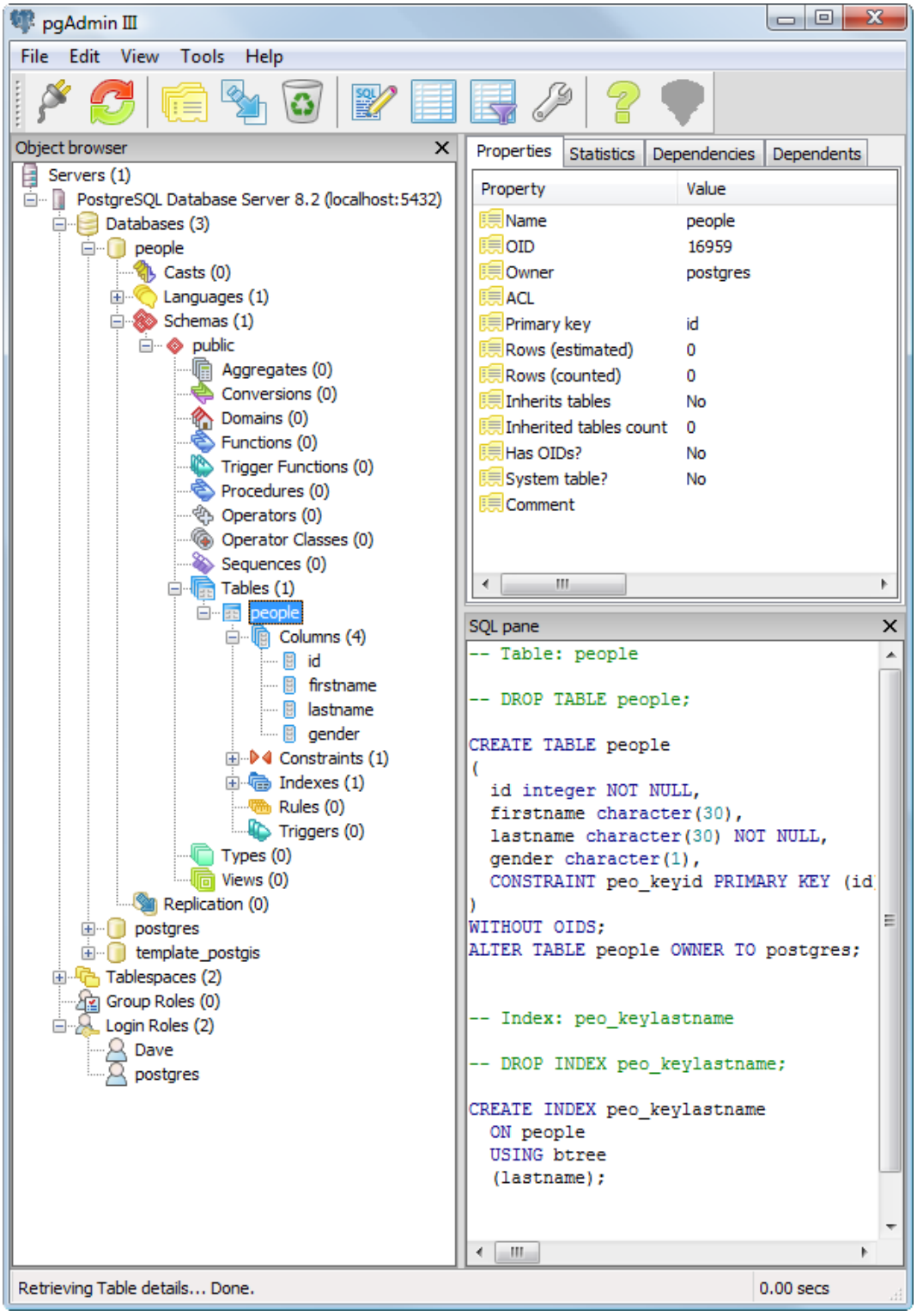


Figure 6. The people table's properties

There are a few things to note about Figure 6. Again, even though the CREATE statement used mixed case labels, all the actual labels are lower case. And in the SQL pane you can see the script pgAdmin III would use if it had to recreate your work. The main difference here is that the primary key constraint has been incorporated into the CREATE TABLE statement. Either way works.

There's something about Figure 6 that probably does look confusing. Databases and tables make sense, but what's all this business about schemas?

Schemas

The idea behind schemas is to accommodate users who want to perform queries across databases. In PostgreSQL you cannot have a single SQL statement that references tables from more than one database (although a Clarion VIEW can do that). Schemas are PostgreSQL's answer to the cross-database query problem.

The idea is that instead of creating multiple databases, you'll create one database with multiple schemas, which are like namespaces. The fully qualified name of any table is schemaname.tablename. If you have one schema called billing and another called shipping, you could create a JOIN using, say, two tables called billing.customers and shipping.shipment.

The public schema is the default schema; you don't have to prepend public. to its tables. That's to provide backward compatibility with applications that aren't aware of PostgreSQL's schema support.

Setting up a user

You're almost ready to use your new database, but first you may want to set up at least one role and one user. This isn't strictly necessary - as long as the login you use to access the database is the same login as the database's owner, you have full rights. But most SQL databases are used by a variety of users, so I'll walk though the process of setting up a user and a role. For a complete discussion of roles and users see Part 3 of this series.

First, set up a role. Go down to the bottom of the pgAdmin III tree and right-click on Group Roles. You'll see the window shown in Figure 7.

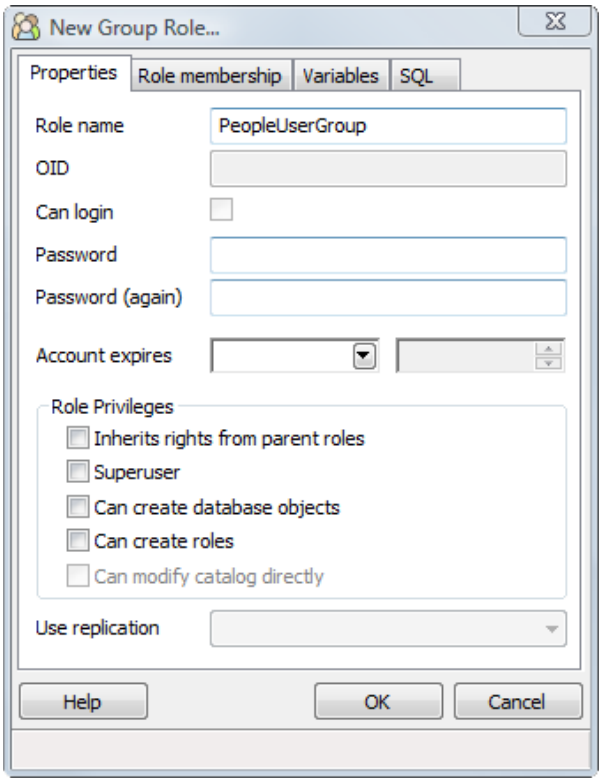


Figure 7. Adding a role

In my example I've called this role PeopleUserGroup. You'll set up the rights later - at the moment all you need to do is fill in the role name and click OK.

Next add a new user, by right-clicking on Login Roles and choosing a new login role (Figure 8).

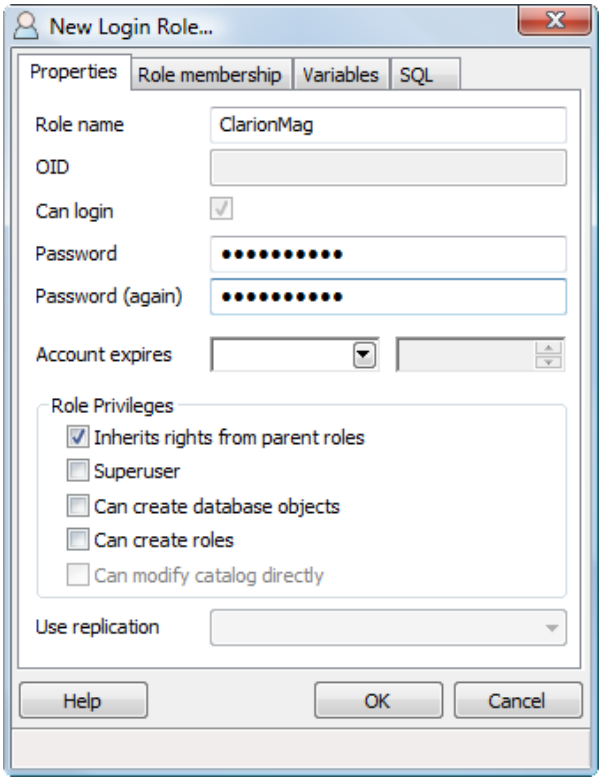


Figure 8. Creating a new login role.

Specify a name (ClarionMag in this example) and a password (also ClarionMag).

Make sure you check the Inherits rights from parent roles box. If this isn't checked the user won't receive any rights from its associated roles.

When you're finished with the Properties, click on the Role membership tab (Figure 9).

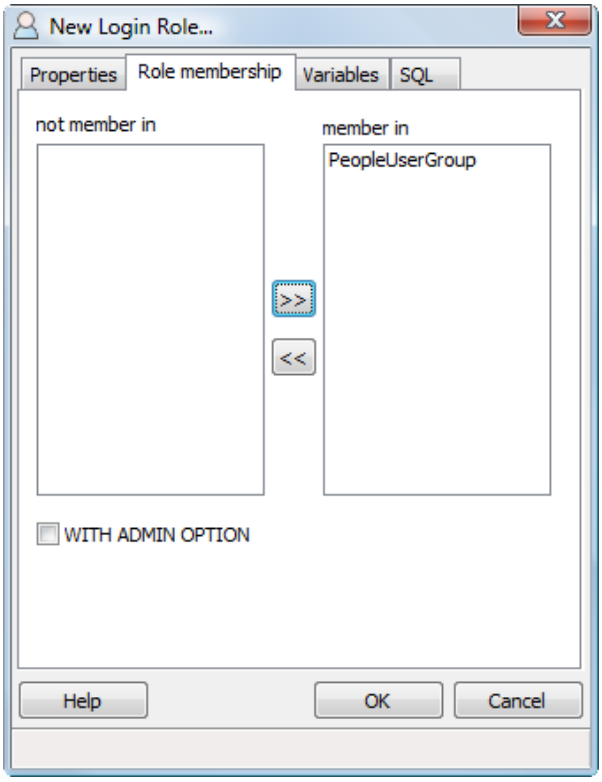


Figure 9. The Role membership tab

At first the PeopleUserGroup role will be in the left-hand list - select that role and click on the >> button, which adds that role to the ClarionMag user. Click OK to save. If you're curious you can see the generated SQL, either before or after creating the login:

```
CREATE ROLE "ClarionMag" LOGIN
  ENCRYPTED PASSWORD 'md557d6d6a23771b698d73cfaf5e063a3ad'
  NOSUPERUSER INHERIT NOCREATEDB NOCREATEROLE;
GRANT "PeopleUserGroup" TO "ClarionMag";
```

There's just one more step, and that's to give the PeopleUserGroup role access to the people database.

pgAdmin III has a tool for administering access rights. Right-click on the Schema or public node and choose Grant Wizard (Figure 10). Select the people table (it's the only one in this database) .

Figure 10. The Grant Wizard

Click on the Privileges tab (Figure 11), and select PeopleUserGroup from the drop list.

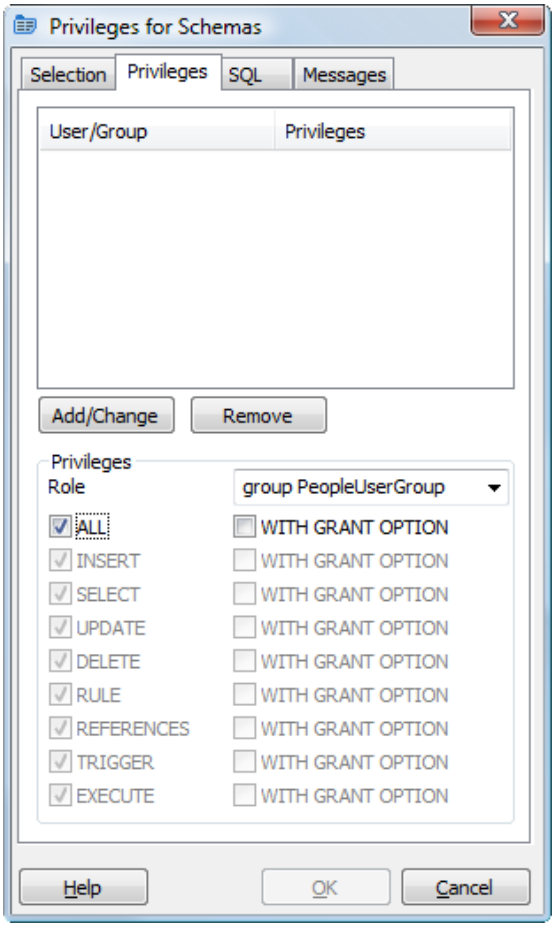


Figure 11. The Privileges tab

Now select All privileges, then click on the Add/Change button. The newly defined rights are then displayed (Figure 12).

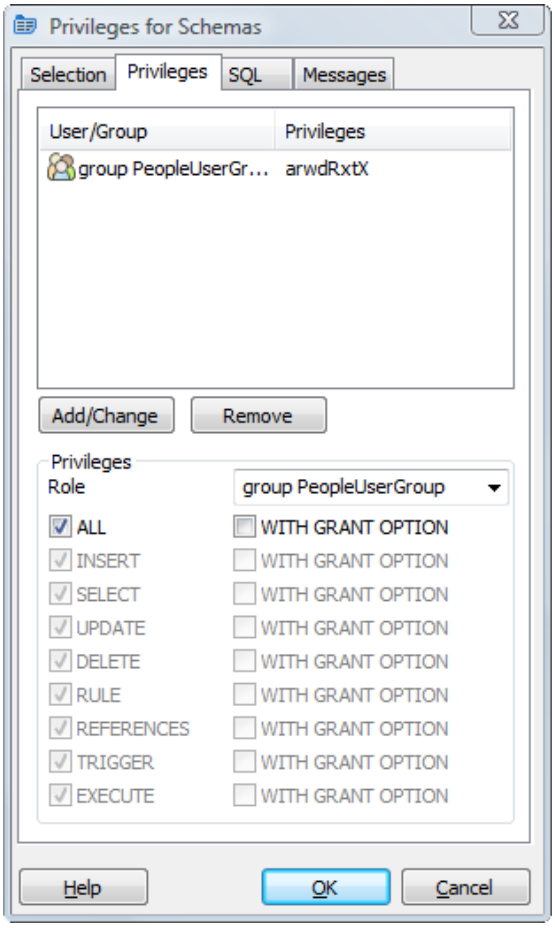


Figure 12. The Privileges tab showing rights

Click OK to apply the rights. You can preview the GRANT statement on the SQL tab; it looks like this:

```
GRANT ALL ON TABLE people TO GROUP "PeopleUserGroup";
```

You may only want to grant the usual CRUD functionality, in which case you deselect all except Insert, Select, Update and Delete. Click on Add/Change and the SQL statement becomes this:

```
GRANT SELECT, UPDATE, INSERT, DELETE
  ON TABLE people TO GROUP "PeopleUserGroup";
```

Click OK to assign the desired rights, then click Done.

Connecting to the database

Unless you're using Clarion.NET, you'll most likely connect to the database via ODBC. There are two ways to use ODBC. You can set up an ODBC data source, or you can use a "DSN-less" connection (which I'll explain later).

Creating an ODBC data source

To create a data source go to Control Panel|Administrative Tools|Data Sources (ODBC). Create a new data source and

choose either the PostgreSQL ANSI or the PostgreSQL Unicode driver. If you're using a non-UTF8 database then you'll want the ANSI driver; if you're on a UTF8 database then either seems to work well with Clarion 6. If you're building with Clarion 7 then I expect the preferred choice is UTF8 and the Unicode driver, although I haven't tested this. Set the values following the example in Figure 13.

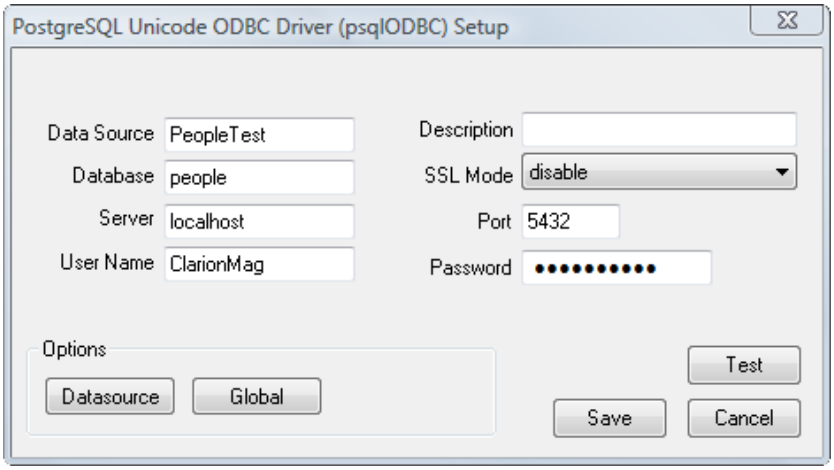


Figure 13. DSN settings

When you've filled in all the fields be sure to press the Test button to make sure you can connect to the database.

Updating the dictionary

Here come the application changes, finally! Open People.DCT and go to the People tables' properties. Set the driver to ODBC, the Owner to PeopleTest, and the pathname to public.people (see Figure 13). As noted earlier, I also advise you to turn off file creation.

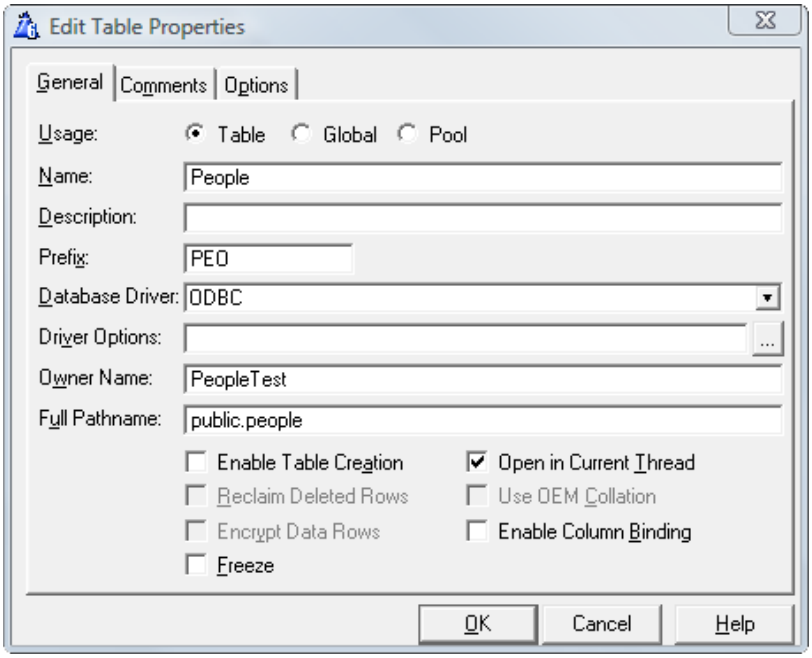


Figure 14. People table properties for DSN connection

You don't actually need to specify the pathname in this case as the public schema is the default, and in the absence of a pathname the driver will use the label. But be careful - there's still some DOS 8.3 compatibility code in the driver subsystem, and if your tablename is more than eight characters long and you don't have the table name specified in the Pathname field, the name will be truncated to eight characters and you won't be able to access the table.

Save your changes, load the People app, compile and run. You should now be connected to the PostgreSQL database and able to update records. If you get a rights violation while connecting make sure you've assigned the necessary rights to the role, and also verify that you've checked the Inherits rights from parent roles option.

DNS-less connection

You don't have to go to the trouble of setting up an ODBC data source. Here's the Owner Name setting for the example app, running locally (line break added):

```
Driver=PostgreSQL ANSI;Server=localhost;Port=5432;
 Database=people;Uid=ClarionMag;PWD=ClarionMag
```

If you're not comfortable with embedding a userid and password in your code you can leave them off, and you'll be prompted for them when the connection is made. Or you can use a variable for the Owner, store the connection string in encrypted form, and assign the Owner value at runtime.

Converting the data

To make it easier to play with the data I've added a conversion option to the sample application. This is a simple process procedure which loops through People.TPS.

There are only two embeds. In the Init embed, after the files are opened, you'll see this code:

```
People{prop:SQL} = 'delete from people'
```

This empties the people PostgreSQL table. In the TakeRecord embed (which gets executed once for each record in People.TPS, you'll see this code:

```
people.record = tpspeople.record
add(people)
```

That's pretty basic - no error checking, just a copy. Obviously if you really cared about the copy working perfectly you'd take a more sophisticated approach.

To load up the PostgreSQL data just select Convert! from the main menu.

Coming up...

No matter how complex your database, the basics of creating a database, creating tables, and giving users rights to access the data are all the same. But the People.APP I converted is a very simple application with a mere four fields of data, with data types that have direct equivalents in PostgreSQL.

In upcoming articles in this series I'll look at SQL data types vs TPS data types, and I'll demonstrate a more extensive, multi-table conversion.

UPDATE

Roberto has kindly folded my changes into the official DCT2SQL release; you can get the latest copy at Steve Parker's Par2 Download Center.

Download the source

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

Reader Comments

Posted on Thursday, December 06, 2007 by douglas johnson

It was surprising to me that a PostgreSQL index must be uniquely named in the database rather than be referenced in the same manner as a column (ie MyTable.ColumnName vs MyTable.IndexName).

Is this true of other SQL backends also?

.....

Posted on Thursday, December 06, 2007 by Dave Harms

I remember running into the same situation years ago with Oracle.

Add a comment

Clarion Magazine

Choosing Colors: Selected Tools

by Dave Harms

Published 2007-11-29

Last month I wrote about the importance of choosing appealing, or at least harmonious, colors for your applications and web sites. I touched on color theory and discussed some of the schemes designers use to choose compatible colors. But knowing a little about color theory isn't always helpful, especially if you're not artistically inclined in the first place. So what do you do?

You get yourself a utility that picks color sets, that's what you do.

There are a number of color picking tools out there; in this article I'll discuss two very different products, Color Cop and ColorImpact, and I'll list a bunch of additional resources. If you have a personal favorite I've missed, please post a reader comment below.

But before getting into the tool review, I need to explain a few things about color notation; if you don't have this bit right you may find that your Clarion programs display different colors than you expect.

Color notation

Last time I talked about color models, including CMYK (Cyan, Magenta, Yellow, and blacK), RGB (Red, Green, and Blue), and HSL (Hue, Saturation, and Lightness). The model typically used for web sites and application software is RGB. As you'll recall, RGB is an additive model - each pixel on the screen is made up of some combination of red, green, and blue light. If all are at full brightness you'll see white light; if red is full on and the others are off, you'll see pure red; if two or more colors are on you'll see some blended color.

The most common notation for RGB colors is hexadecimal, where each individual color has a value from 00 to FF. You'll recall that in hexadecimal (base 16) notation it takes four bits to represent values from 0 to F, and 8 bits (one byte) to represent values from 00 to FF. So in RGB notation you have 256 possible values each for red, blue and green. The total number of combinations is 256*256*256, or 16,777,216 colors for any one pixel.

Computer monitors may or may not be able to display all 16 million colors defined by hex RGB notation, depending on their color depth.

Color depth is the number of bits per color (that is, per the color displayed by a single pixel). While RGB hex notation assumes a maximum color depth of 24 bits (8 per color) not all displays support this depth.

In the old days EGA and early VGA displays had four bit color depth, for a maximum of 16 colors per pixel. Later, VGA and Super VGA upped the ante to 8 bit color depth, for 256 colors. These days most displays support at least HighColor, which uses either 15 or 16 bits per color for up to 65,536 colors per pixel.

Truecolor uses the same 8 bits per red, green, or blue color as described in RGB hex notation, for a total color depth of 24 bits or 16 million colors. You'll also see 32 bit color displays, but in this case usually only 24 bits actually describe the colors; the other bits are often used for non-color data.

Web-safe colors? Who cares?

But let me back up just a minute to the days of VGA and Super VGA. Although these displays could only generate 256 colors, they could achieve the effect of many more colors by dithering, a way of blending several colors together. But

while dithering achieve the effect of more colors, those weren't pure colors, and the effect could sometimes be unpleasant. You probably remember seeing background images or other solid colors that appeared to have a grainy pattern.

The web-safe color set (mostly) avoids this problem by using only hex RGB values that can be resolved to individual colors in the 256 color palette. This color set uses 6 bits per red, green and blue, for a total of 216 colors.

As you explore the tools I'll describe in this article you'll notice that many of them provide an option to display web-safe color sets. There's just one thing to keep in mind: *web-safe colors are almost completely obsolete*. It's pretty hard to find a display these days that won't handle at least 16 bit color - even handheld devices typically have this capability. So unless you have a bunch of clients stuck on VGA monitors you can safely forget about being "web-safe".

RGB or BGR?

Now, one more important item. For some reason I haven't yet fathomed, some Windows API functions require colors to be represented in BGR format rather than RGB. And presumably as a result of this, Clarion and some other Windows programming languages require you to present your colors in BGR format. This can be a real, um, hassle because many color pickers are designed for HTML, which uses RGB notation.

Programming language color notation can be different from HTML notation in other ways. HTML color codes begin with #; Clarion colors, like all hex numbers in Clarion, begin with 0 and end with H. Delphi expects a prefix of $00; in C++ the prefix is 0x00; and in Visual Basic it's &H. Depending on the tool you choose you may have to do a little translating when plugging values into Clarion.

The tools

There are a great many tools available to simplify choosing color schemes for web sites, applications, print material, and more. I'm going to discuss two of these, which I think give some idea of the range of product that's available.

Color Cop

Color Cop is a minimalist color-picking application notable for having the option to display color values in Clarion notation. This is thanks to Keith Neely, who requested the feature from Color Cop's author, Jay Prall. Figure 1 shows Color Cop's main window.

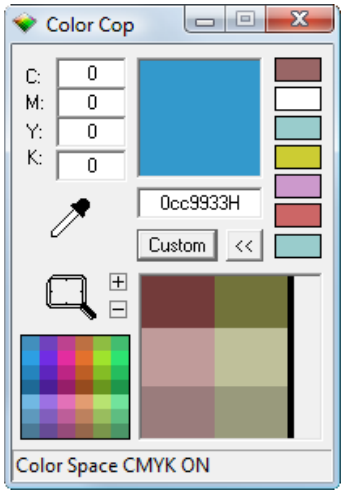


Figure 1. Color Cop

As with many other color tools, Color Cop has an eyedropper which you can use to sample a color from anywhere on your display. Drag the eyedropper and Color Cop will show you a magnified view of the selection area in its window, and at the same time update its display of 42 compatible colors. To retrieve a color from the 6x7 grid just click on it with

the eyedropper. Color Cop can also automatically copy the selected color to the clipboard.

Color Cop is free, but donations are requested.

ColorImpact

TigerColor's ColorImpact is the polar opposite to Color Cop's minimalist interface. Oriented around color wheels, ColorImpact is one of the heavyweights in the field, with numerous options for selecting and displaying matching colors. Figure 2 shows ColorImpact's main window, with the startup screen.

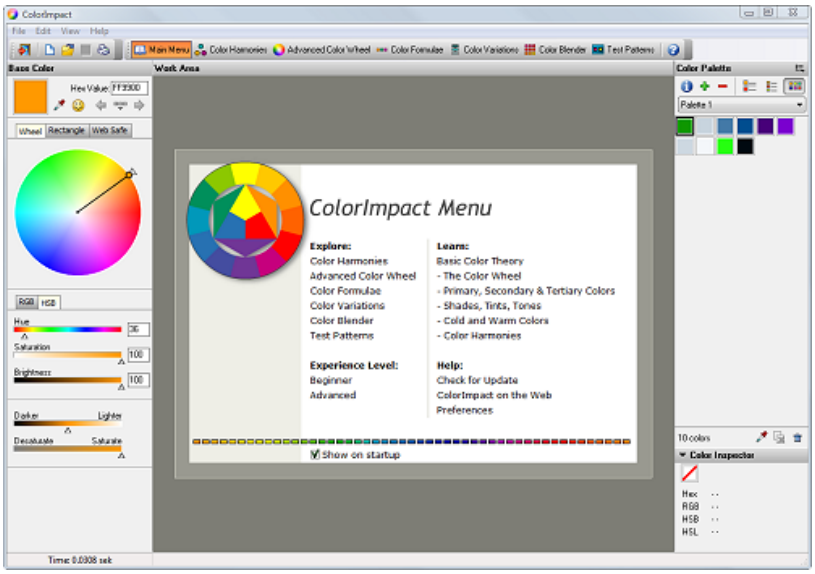


Figure 2. ColorImpact's main window (view full size image**)**

The startup screen (which you can disable) is basically a menu which either gives you some very basic information on color theory or takes you to one of ColorImpact's windows, which include:

- Color harmonies - a color wheel which you can customize to show various standard patterns of harmonious colors
- A highly customizable color wheel
- A page for displaying swatches based on standard color formulae
- A page for displaying varying versions of the same color (by lightness, saturation, etc)
- A page for displaying gradations between two colors
- A test page where you can see how your colors will look when used in a variety of standard designs

The basic workflow in ColorImpact is to choose one or more base colors, then use the available tools to expand the palette with additional shades of those colors.

Figure 3 shows the Color Harmonies page displaying an *analogous with complement* color harmony. There are eight other color harmonies to choose from, including complement, split-complement, triad, square, rectangle 1 & 2 (complementary pairs), analogous, and hexad (six equally spaced colors).

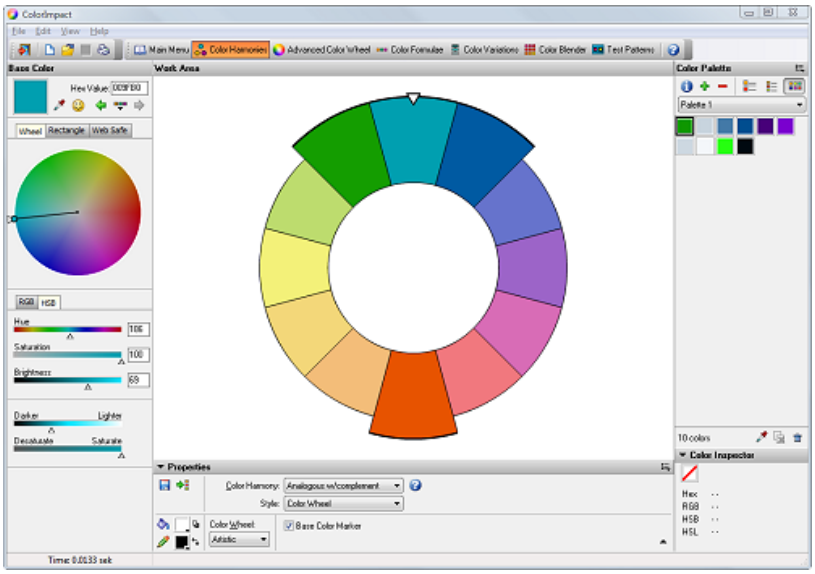


Figure 3. The Color Harmonies page. (view full size image**)**

Once you have some basic colors chosen you might want to look at the Color Variations page (Figure 4), where you can add different shades of your colors to your palette.

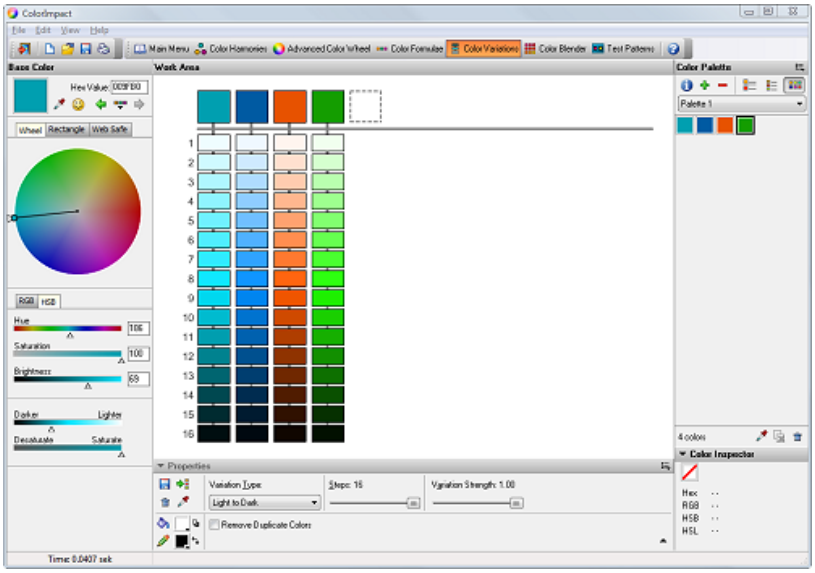


Figure 4. Color Variations (view full size image**)**

If you're not happy with the set of colors you've chosen you may find it easier to choose just one starting color, and then use the Color Formulae page. Figure 5 uses the same starting color as Figure 4, but the swatches are calculated using the *Monochrome w/contrasts & grays* formula.

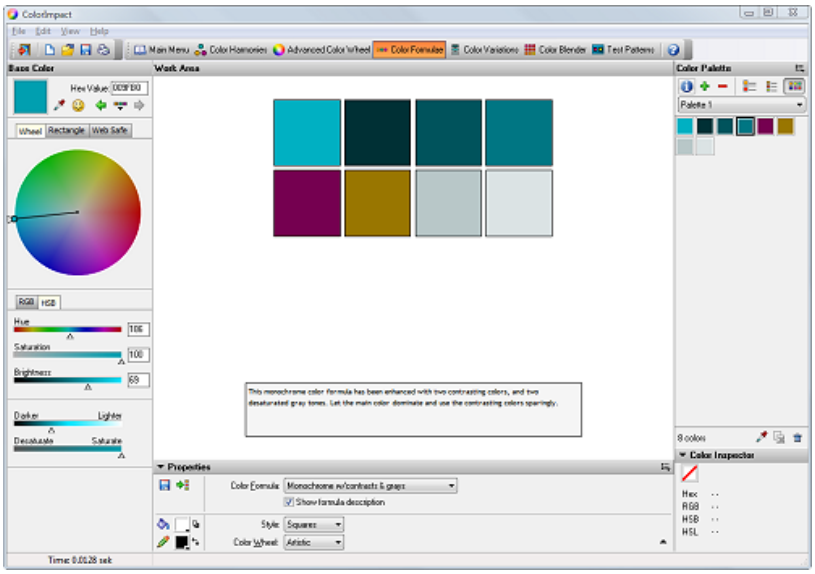


Figure 5. Color formula applied (view full size image**)**

The Color Formulae page lets you apply the schemes available on the Color Harmonies page, plus many more. You just don't see your results in the form of a color wheel.

Let's say you're happier with your latest set of colors, so set them as the palette. Want to see how they look? Click on Test Patterns and choose the Web Page 2 test pattern (Figure 6).

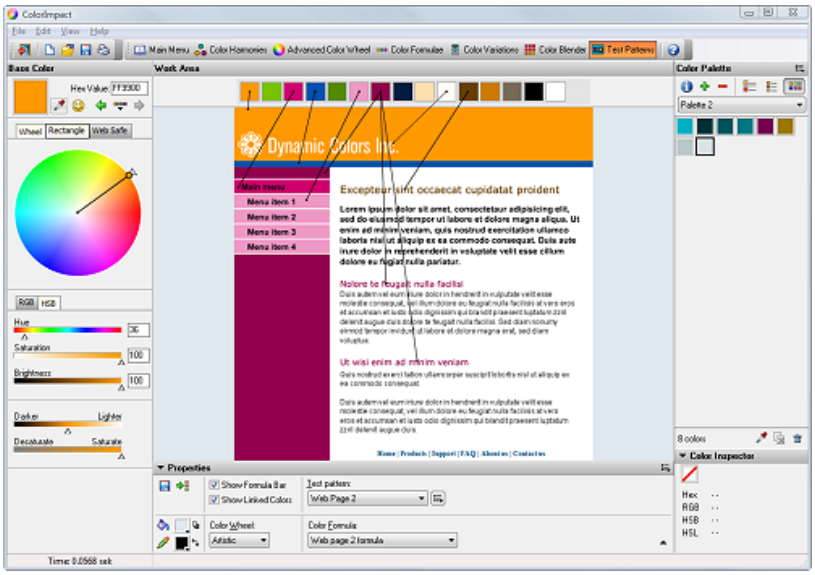


Figure 6. The Web Page 2 test pattern. (view full size image**)**

Now you can apply the colors from your palette to the test pattern (Figure 7).

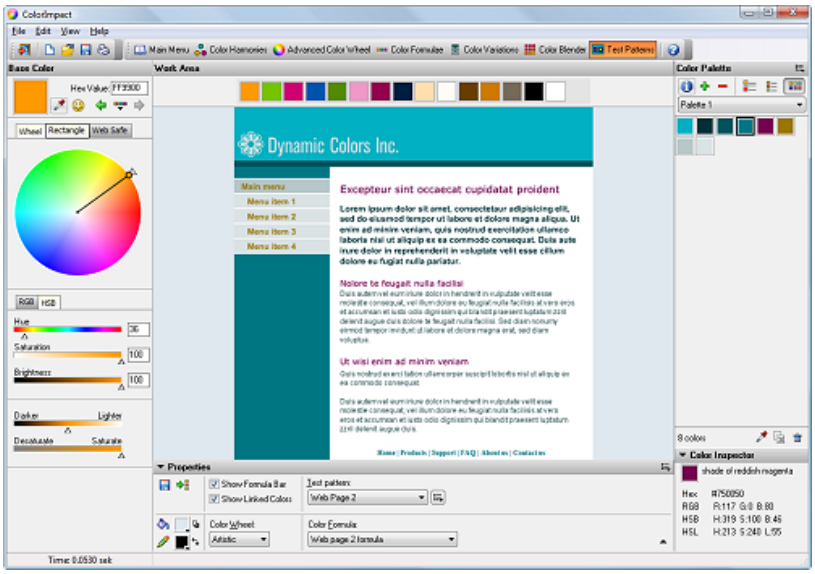


Figure 7. The test pattern (view full size image**)**

Keep in mind that with Color Impact, as with most other tools, you'll be presented with HTML values for colors, and you'll have to reverse the two outside hex pairs to get the Clarion value. Or you could leave your palette on screen and use Color Cop to sample the colors and create the correct Clarion notation.

Other tools

Color Cop and ColorImpact may represent the two poles of color-choosing software (at least on the desktop), but there's a whole lot of stuff in the middle. Here are just a few products you may want to look at.

- Color Wheel Pro - another heavyweight, this program doesn't have as much fine tuning capability as ColorImpact but it makes it easy to apply your colors to various preset schemes.
- Color Schemer - basic color selection tools, plus the ability to create a color palette based on a photograph
- GenoPal - An intuitive approach to choosing colors. Like Color Schemer it has the option of creating a palette from a picture.
- Interactive Color Wheel - online Java applet lets you experiment with colors
- Color Mixers - online color mixing app; just select RGB values and it does the rest.
- Color Schemes Generator 2 - A nifty online color scheme tool that also lets you see how your scheme appears to people with various kinds of color blindness.
- Colr.org - numerous online color schemes
- ColorCombos - a library of color combos, articles, and a tool for extracting a palette from a web page
- ColorBlender - simple online color blending
- ColorHunter - create palettes from images

Color theory, trends, and background

- Causes of Color
- Color Theory group on Yahoo
- Wikipedia entry
- Color Lovers

Color Blindness

Just because colors look good to you doesn't mean they look good to everyone else. Users with color blindness may find some elements of your design difficult to see. Here are some resources:

- Vischeck
- Colorblind Homepage
- Ishihara Test for Color Blindness
- Wikipedia entry

A special thanks to Leroy Schulz, Clarion Magazine's favorite design guru, who supplied many of the above links.

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

Reader Comments

Posted on Tuesday, December 04, 2007 by Carl Barnes

You don't say it specifically but by Clarion (and Windows) using BGR you mean to get a Clarion color:

Color = Blue*256^2 + Green*256 + Red

Seeing that Red is EQUATE(0000FFh)and Blue EQUATE(FF0000h) gives a big hint.

Add a comment

Comparison of C#, Clarion# and VB.NET

This is a quick reference guide to highlight some key syntactical differences between C#, Clarion# and VB.NET (version 2).

NOTE: Clarion.NET and the Clarion# language are currently in alpha test, and various areas of the documentation are incomplete and in a state of flux. Therefore, it's very likely that some of the entries will change as new information becomes available.

Program Structure

C#

```
using System;

namespace Hello {
   public class HelloWorld {
      public static void Main(string[] args) {
         string name = "C#";

         //See if an argument was passed from the command line
         if (args.Length == 1)
            name = args[0];

         Console.WriteLine("Hello, " + name + "!");
      }
   }
}
```

Clarion#

```
  PROGRAM
  NAMESPACE('Hello')
  USING('System')
  MAP
  END

Name &STRING

  CODE
  Name = 'Clarion#'
  !See if an argument was passed from the command line
  IF COMMAND('1') <> ''
    Name = COMMAND('1')
  END
  Console.WriteLine('Hello, '& Name & '!')
```

VB.NET

```
Imports System

Namespace Hello
   Class HelloWorld
      Overloads Shared Sub Main(ByVal args() As String)
         Dim name As String = "VB.NET"

         'See if an argument was passed from the command line
         If args.Length = 1 Then name = args(0)

         Console.WriteLine("Hello, " & name & "!")
      End Sub
   End Class
End Namespace
```

Comments

C#

```
//Single line


/* Multiple
   lines */

///<summary>XML comments on single line</summary>

/** <summary>
  XML comments on multiple lines
</summary> */
```

Clarion#

```
!Single line


!~ Multiple
   lines ~!

!!!<summary>XML comments on single line</summary>

!!!<summary>
!!!  XML comments on multiple lines
!!!</summary>
```

VB.NET

```
'Single line only
REM Single line only




'''<summary>XML comments</summary>
```

Data Types

C#

```
// Value Types
bool
byte, sbyte
char
short, ushort, int, uint, long, ulong
float, double
decimal
DateTime    //not a built-in C# type

// Reference Types
object
string

// Initializing
bool correct = true;
byte b = 0x2A;    //hex

object person = null;
string name = "Mike";
char grade = 'B';
DateTime today = DateTime.Parse("12/31/2007 12:15:00");
decimal amount = 35.99m;
float gpa = 2.9f;
double pi = 3.14159265;
long lTotal = 123456L;
short sTotal = 123;
ushort usTotal = 123;
uint uiTotal = 123;
ulong ulTotal = 123;

// Type Information
int x;
Console.WriteLine(x.GetType());         //Prints System.Int32
Console.WriteLine(typeof(int));         //Prints System.Int32
Console.WriteLine(x.GetType().Name);    //Prints Int32

// Type Conversion
float d = 3.5f;
int i = (int)d;    //set to 3  (truncates decimal)
```

Clarion#

```
! Value Types
BOOL
BYTE, SBYTE
CHAR, CSTRING, PSTRING, CLASTRING
SHORT, USHORT, SIGNED, UNSIGNED, LONG, ULONG, CLALONG
SREAL, REAL, BFLOAT4, BFLOAT8
DECIMAL, PDECIMAL, CLADECIMAL
DATE, TIME, CLADATE, CLATIME

! Reference Types
&OBJECT
&STRING

! Initializing
Correct  BOOL(True)
H        BYTE(02Ah)     !hex
O        BYTE(052o)     !octal
B        BYTE(01101b)   !binary
Person   &OBJECT
Name     &STRING
  Name = 'Mike'
Grade    CHAR('B')
Today    DATE
  Today = DATE(12,31,2007)
Amount   DECIMAL(35.99)
GPA      SREAL(2.9)
Pi       REAL(3.14159265)
lTotal   LONG(123456)
sTotal   SHORT(123)
usTotal  USHORT(123)
uiTotal  UNSIGNED(123)
ulTotal  ULONG(123)

! Type Information
X SIGNED
  Console.WriteLine(X.GetType())         !Prints System.Int32
  Console.WriteLine(TYPEOF(SIGNED))      !Prints System.Int32
  Console.WriteLine(X.GetType().Name)    !Prints Int32

! Type Conversion
D        SREAL(3.5)
I        SIGNED
  I = D            !Implicitly truncate to 3
  I = D AS SIGNED  !Explicitly truncate to 3
```

VB.NET

```
' Value Types
Boolean
Byte, SByte
Char
Short, UShort, Integer, UInteger, Long, ULong
Single, Double
Decimal
Date

' Reference Types
Object
String

' Initializing
Dim correct As Boolean = True
Dim b As Byte = &H2A    'hex
Dim o As Byte = &O52    'octal
Dim person As Object = Nothing
Dim name As String = "Mike"
Dim grade As Char = "B"c
Dim today As Date = #12/31/2007 12:15:00 PM#
Dim amount As Decimal = 35.99@
Dim gpa As Single = 2.9!
Dim pi As Double = 3.14159265
Dim lTotal As Long = 123456L
Dim sTotal As Short = 123S
Dim usTotal As UShort = 123US
Dim uiTotal As UInteger = 123UI
Dim ulTotal As ULong = 123UL

' Type Information
Dim x As Integer
Console.WriteLine(x.GetType())         'Prints System.Int32
Console.WriteLine(GetType(Integer))    'Prints System.Int32
Console.WriteLine(TypeName(x))         'Prints Integer

' Type Conversion
Dim d As Single = 3.5
Dim i As Integer = CType(d, Integer) 'set to 4 (Banker's rounding)
i = CInt(d)  'same result as CType
i = Int(d)   'set to 3 (Int function truncates the decimal)
```

Constants

C#

```
const int MAX_STUDENTS = 25;

// Can set to a const or var; may be initialized in a constructor
readonly float MIN_DIAMETER = 4.93f;
```

Clarion#

```
! CONST and READONLY unsupported.
! Use EQUATEs or regular data types instead.
MAX_STUDENTS EQUATE(25)    !Instead of CONST; will auto-convert
MIN_DIAMETER SREAL(4.93)   !READONLY is unsupported
```

VB.NET

```
Const MAX_STUDENTS As Integer = 25

'Can set to a const or var; may be initialized in a constructor
ReadOnly MIN_DIAMETER As Single = 4.93
```

Enumerations

C#

```
enum Action {Start, Stop, Rewind, Forward};




enum Status {
   Flunk = 50,
   Pass  = 70,
   Excel = 90
};

Action a = Action.Stop;
if (a != Action.Start)
   Console.WriteLine(a + " is " + (int) a); //Prints "Stop is 1"

Console.WriteLine((int) Status.Pass);       //Prints 70
Console.WriteLine(Status.Pass);             //Prints Pass
```

Clarion#

```
Action  ENUM
Start     ITEM
Stop      ITEM
Rewind    ITEM
Forward   ITEM
        END
```

```
Action  ENUM  !Alternate syntax
          Start
          Stop
          Rewind
          Forward
        END
```

```
Status  ENUM
          Flunk(50)
          Pass (70)
          Excel(90)
        END

A       Action(Action.Stop)
  IF (A <> Action.Start)
    Console.WriteLine(A.ToString() &'is '& A)  !Prints "Stop is 1"
  END
  Console.WriteLine(Status.Pass + 0)           !Prints 70
  Console.WriteLine(Status.Pass)               !Prints Pass
```

VB.NET

```
Enum Action
  Start
  [Stop]    'Stop is a reserved word
  Rewind
  Forward
End Enum

Enum Status
  Flunk = 50
  Pass  = 70
  Excel = 90
End Enum

Dim a As Action = Action.Stop
If a <> Action.Start Then _
   Console.WriteLine(a.ToString & " is " & a)  'Prints "Stop is 1"

Console.WriteLine(Status.Pass)                 'Prints 70
Console.WriteLine(Status.Pass.ToString())      'Prints Pass
```

Operators

C#

```
// Comparison
==  <  >  <=  >=  !=

// Arithmetic
+  -  *  /
%                 //mod
/                 //integer division if both operands are ints
Math.Pow(x, y)    //raise to a power

// Assignment
=  +=  -=  *=  /=   %=  &=  |=  ^=  <<=  >>=  ++  --


// Bitwise
&   |   ^   ~   <<   >>
```

Clarion#

```
! Comparison
=  <  >  <=  >=  ~=  <> &=

! Arithmetic
+  -  *  /
%     !mod
/     !integer division
^     !raise to a power

! Assignment
=  +=  -=  *=  /=  %=  &=
:=    !"Smart" replacement for = and &=

! Bitwise
BAND(val,mask) BOR(val,mask) BXOR(val,mask) BSHIFT(val,count)
```

VB.NET

```
' Comparison
=  <  >  <=  >=  <>

' Arithmetic
+  -  *  /
Mod
\     'integer division
^     'raise to a power

' Assignment
=  +=  -=  *=  /=  \=  ^=  <<=  >>=  &=


' Bitwise
And   Or   Xor   Not   <<   >>
```

```
// Logical
&&   ||   &   |   ^   !

// Note:  && and || perform short-circuit logical evaluations

// String Concatenation
+
```

```
! Logical
AND OR XOR NOT

! Note:  AND and OR perform short-circuit logical evaluations

! String Concatenation
&
```

```
' Logical
AndAlso   OrElse   And   Or   Xor   Not

' Note:  AndAlso and OrElse perform short-circuit logical evaluations

' String Concatenation
&
```

Choices

C#

```
greeting = age < 20 ? "What's up?" : "Hello";

if (age < 20)
  greeting = "What's up?";
else
  greeting = "Hello";

// Semi-colon ";" is used to terminate each statement,
// so no line continuation character is necessary.
```

```
// Multiple statements must be enclosed in {}
if (x != 100 && y < 5) {
  x *= 5;
  y *= 2;
}
```

```
if (x > 5)
  x *= y;
else if (x == 5)
  x += y;
else if (x < 10)
  x -= y;
else
  x /= y;
```

Clarion#

```
Greeting = CHOOSE(Age < 20, 'What''s up?', 'Hello')

! One line requires THEN (or ;)
IF Age < 20 THEN Greeting = 'What''s up?' END
IF Age < 20; Greeting = 'What''s up?'ELSE Greeting = 'Hello' END

! Use semi-colon (;) to put two commands on same line.
! A period (.) may replace END in single line constructs,
! but it is discouraged for multi-line constructs.
IF X <> 100 AND Y < 5 THEN X *= 5; Y *= 2.   !Period is OK here
```

```
! Multi-line is more readable (THEN is optional on multi line)
IF X <> 100 AND Y < 5 THEN          | IF X <> 100 AND Y < 5
  X *= 5                            |   X *= 5
  Y *= 2                            |   Y *= 2
END                                 | .   !Period is hard to see here

! To break up any long single line use | (pipe)
IF WhenYouHaveAReally < LongLine |
AND ItNeedsToBeBrokenInto2 > Lines
  UseThePipe(CharToBreakItUp)
END
```

```
IF X > 5
  X *= Y
ELSIF X = 5
  X += Y
ELSIF X < 10
  X -= Y
ELSE
  X /= Y
END
```

VB.NET

```
greeting = IIf(age < 20, "What's up?", "Hello")

' One line doesn't require End If
If age < 20 Then greeting = "What's up?"
If age < 20 Then greeting = "What's up?" Else greeting = "Hello"

' Use colon (:) to put two commands on same line
If x <> 100 AndAlso y < 5 Then x *= 5 : y *= 2
```

```
' Multi-line is more readable
If x <> 100 AndAlso y < 5 Then
  x *= 5
  y *= 2
End If

' To break up any long single line use _
If whenYouHaveAReally < longLine AndAlso _
    itNeedsToBeBrokenInto2 > Lines Then _
    UseTheUnderscore(charToBreakItUp)
```

```
If x > 5 Then
  x *= y
ElseIf x = 5 Then
  x += y
ElseIf x < 10 Then
  x -= y
Else
  x /= y
End If
```

```
// Every case must end with break or goto case
switch (color) {                    //Must be integer or string
  case "pink" :
  case "red"  : r++;      break;
  case "blue" : b++;      break;
  case "green": g++;      break;
  default:      other++; break;   //break necessary on default
}
```

```
CASE Color              !Any data type or expression
OF 'pink' OROF 'red'
  R += 1
OF 'blue'
  B += 1
OF 'green'
  G += 1
ELSE
  Other += 1
END

CASE Value
  OF    0.00 TO   9.99;  RangeName = 'Ones'
  OF   10.00 TO  99.99;  RangeName = 'Tens'
  OF  100.00 TO 999.99;  RangeName = 'Hundreds'
  !etc.
  ELSE                ;  RangeName = 'Zillions'
END

EXECUTE Stage     !Integer value or expression
  Stage1          ! expression equals 1
  Stage2          ! expression equals 2
  Stage3          ! expression equals 3
ELSE
  StageOther      ! expression equals some other value
END
```

```
Select Case color      'Must be a primitive data type
  Case "pink", "red"
    r += 1
  Case "blue"
    b += 1
  Case "green"
    g += 1
  Case Else
    other += 1
End Select
```

Loops

C#

```
// Pre-test Loops
while (c < 10)
  c++;

// no "until" keyword

for (c = 2; c <= 10; c += 2)
  Console.WriteLine(c);

// Post-test Loop
do
  c++;
while (c < 10);

// Untested Loop
for (;;) {
  //break logic inside
}
```

Clarion#

```
! Pre-test Loops
LOOP WHILE C < 10
  C += 1
END
```

```
LOOP UNTIL C = 10
  C += 1
END

LOOP C = 2 TO 10 BY 2
  Console.WriteLine(C)
END
```

```
! Post-test Loops
LOOP
  C += 1
WHILE c < 10
```

```
LOOP
  C += 1
UNTIL C = 10
```

```
! Untested Loops
LOOP
  !Break logic inside
END
```

```
LOOP 3 TIMES
  Console.WriteLine
END
```

VB.NET

```
' Pre-test Loops
While c < 10
  c += 1
End While

Do While c < 10
  c += 1
Loop
```

```
Do Until c = 10
  c += 1
Loop

For c = 2 To 10 Step 2
  Console.WriteLine(c)
Next
```

```
' Post-test Loops
Do
  c += 1
Loop While c < 10
```

```
Do
  c += 1
Loop Until c = 10
```

```
' Untested Loop
Do
  //break logic inside
Loop
```

```
// Array or collection looping
string[] names = {"Fred", "Sue", "Barney"};
foreach (string s in names)
  Console.WriteLine(s);
```

```
! Array or collection looping
Names &STRING,DIM(3)
S     &STRING
      CODE
      Names[1] := 'Fred'; Names[2] := 'Sue'; Names[3] := 'Barney'
      FOREACH S IN Names
        Console.WriteLine(S)
      END
```

```
' Array or collection looping
Dim names As String() = {"Fred", "Sue", "Barney"}
For Each s As String In names
  Console.WriteLine(s)
Next
```

```
// Breaking out of loops
int i = 0;
while (true) {
  if (i == 5)
    break;
  i++;
}
```

```
! Breaking out of loops
I     SHORT(0)
      CODE
      LOOP
        IF I = 5 THEN BREAK.
        I += 1
      END
```

```
' Breaking out of loops
Dim i As Integer = 0
While (True)
  If (i = 5) Then Exit While
  i += 1
End While
```

```
// Continue to next iteration
for (i = 0; i < 5; i++) {
  if (i < 4)
    continue;
  Console.WriteLine(i);   //Only prints 4
}
```

```
! Continue to next iteration
      LOOP I = 0 TO 4
        IF I < 4 THEN CYCLE.
        Console.WriteLine(I)
      END
```

```
' Continue to next iteration
For i = 0 To 4
  If i < 4 Then Continue For
  Console.WriteLine(i)   'Only prints 4
Next
```

Arrays

C#

```
int[] nums = {1, 2, 3};
for (int i = 0; i < nums.Length; i++)
  Console.WriteLine(nums[i]);
```

```
// 5 is the size of the array
string[] names = new string[5];
names[0] = "David";
names[5] = "Bobby";    //Throws System.IndexOutOfRangeException
```

```
// C# can't dynamically resize arrays, so copy into new array
string[] names2 = new string[7];
Array.Copy(names, names2, names.Length);
// or
names.CopyTo(names2, 0);
```

```
float[,] twoD = new float[rows, cols];
twoD[2,0] = 4.5f;
```

Clarion#

```
Nums   SIGNED,DIM(3)
I      SIGNED
       CODE
       Nums[1] = 1; Nums[2] = 2; Nums[3] = 3
       LOOP I = 1 TO Nums.Length
         Console.WriteLine(Nums[I])
       END
```

```
! 5 is the size of the array
Names  &STRING,DIM(5)
       CODE
       Names[1] := 'David'
       Names[6] := 'Bobby' !Caught by compiler
       I = 6
       Names[I] := 'Bobby' !Throws System.IndexOutOfRangeException
```

```
! Clarion# can't dynamically resize arrays, so copy into new array
Names2 &STRING[]
       CODE
       Names2 := NEW STRING[7]
       Array.Copy(Names, Names2, Names.Length)  !or
       Names.CopyTo(Names2, 0)

TwoD   SREAL[,]
       CODE
       TwoD := NEW SREAL[Rows, Cols]
       TwoD[3,1] = 4.5
```

VB.NET

```
Dim nums() As Integer = {1, 2, 3}
For i As Integer = 0 To nums.Length - 1
  Console.WriteLine(nums(i))
Next
```

```
' 4 is the index of the last element, so it holds 5 elements
Dim names(4) As String
names(0) = "David"
names(5) = "Bobby"   'Throws System.IndexOutOfRangeException
```

```
' Resize the array, keeping existing values (Preserve is optional)
ReDim Preserve names(6)
```

```
Dim twoD(rows-1, cols-1) As Single
twoD(2, 0) = 4.5
```

C#

```
// Jagged arrays
int[][] jagged = new int[3][] {
  new int[5], new int[2], new int[3] };
jagged[0][4] = 5;
```

Clarion#

```
! Jagged arrays unsupported
```

VB.NET

```
' Jagged arrays
Dim jagged()() As Integer = { _
  New Integer(4) {}, New Integer(1) {}, New Integer(2) {} }
jagged(0)(4) = 5
```

Functions

C#

```
// Pass by value(in,default), reference(in/out), and reference(out)
void TestFunc(int x, ref int y, out int z) {
  x++;
  y++;
  z = 5;
}


int a = 1, b = 1, c;  //c doesn't need initializing
TestFunc(a, ref b, out c);
Console.WriteLine("{0} {1} {2}", a, b, c);  //1 2 5


// Accept variable number of arguments
int Sum(params int[] nums) {
  int sum = 0;
  foreach (int i in nums)
    sum += i;
  return sum;
}

int total = Sum(4, 3, 2, 1);    //returns 10


/* C# doesn't support optional arguments/parameters.
  Just create two different versions of the same function. */
void SayHello(string name, string prefix) {
  Console.WriteLine("Greetings, " + prefix + " " + name);
}
void SayHello(string name) {
  SayHello(name, "");
}
```

Clarion#

```
! Pass by value(in,default), reference(in/out), and reference(out)
TestFunc PROCEDURE(SIGNED X, *SIGNED Y, *SIGNED Z)
  CODE
  X += 1
  Y += 1
  Z = 5
  RETURN   !Optional, if not returning a value


A UNSIGNED(1)
B UNSIGNED(1)
C UNSIGNED      !C doesn't need initializing
  CODE
  TestFunc(A, B, C)
  Console.WriteLine('{{0} {{1} {{2}', A, B, C)  !1 2 5

! Accept variable number of arguments
Sum PROCEDURE(PARAMS UNSIGNED[] Nums),UNSIGNED
Result UNSIGNED(0)
I      UNSIGNED
  CODE
  FOREACH I IN Nums
    Result += I
  END
  RETURN Result

  Total# = Sum(4, 3, 2, 1)    !returns 10

! Optional parameters without default value
! (When omitted, value parameters default to 0 or an empty string)
! (Use OMITTED to detect the omission)
SayHello PROCEDURE(STRING Name, <STRING Prefix>)

! Optional parameters with default value
! (Valid only on simple numeric types)
! (OMITTED will not detect the omission — the default is passed)
SayHello PROCEDURE(STRING Name, BYTE Age = 20)
```

VB.NET

```
' Pass by value(in,default), reference(in/out), and reference(out)
Sub TestFunc(ByVal x As Integer, ByRef y As Integer, _
             ByRef z As Integer)
  x += 1
  y += 1
  z = 5
End Sub


Dim a = 1, b = 1, c As Integer    'c set to zero by default
TestFunc(a, b, c)
Console.WriteLine("{0} {1} {2}", a, b, c)    '1 2 5


' Accept variable number of arguments
Function Sum(ByVal ParamArray nums As Integer()) As Integer
  Sum = 0
  For Each i As Integer In nums
    Sum += i
  Next
End Function    'Or use Return statement like C#

Dim total As Integer = Sum(4, 3, 2, 1)    'returns 10


' Optional parameters must be listed last and have a default value
Sub SayHello(ByVal name As String, Optional ByVal prefix As String = "")
  Console.WriteLine("Greetings, " & prefix & " " & name)
End Sub

SayHello("Strangelove", "Dr.")
SayHello("Madonna")
```

Strings

C#

```
// Escape sequences
\r    //carriage-return
\n    //line-feed
\t    //tab
\\    //backslash
\"    //quote
```

```
// String concatenation
string school = "Harding\t";
school = school + "University";
// school is "Harding(tab)University"
```

```
// Chars
char letter = school[0];              //letter is H
letter = Convert.ToChar(65);          //letter is A
letter = (char)65;                    //same thing
char[] word = school.ToCharArray();   //word holds Harding
```

```
// String literal
string msg = @"File is c:\temp\x.dat";
// same as
string msg = "File is c:\\temp\\x.dat";
```

```
// String comparison
string mascot = "Bisons";
if (mascot == "Bisons")                 //true
if (mascot.Equals("Bisons"))            //true
if (mascot.ToUpper().Equals("BISONS"))  //true
if (mascot.CompareTo("Bisons") == 0)    //true
```

```
// Substring
Console.WriteLine(mascot.Substring(2, 3));     //Prints "son"
```

Clarion#

```
! Special Characters
<13>    !carriage-return
<10>    !line-feed
<9>     !tab
<n>     !character with the ASCII value=n (see above)
<<      !less-than
{{      !left-curly-brace
''      !single-quote
{n}     !Repeat previous character "n" times
```

```
! String concatenation
School &STRING
Univ   CLASTRING('University')   !Clarion string class
       CODE
       School = 'Harding<9>'
       School = School & Univ
       !School is "Harding(tab)University"
```

```
! Chars
Letter CHAR
Word   CHAR[]
       CODE
       Letter = School[1]            !Letter is H
       Letter = Convert.ToChar(65)   !Letter is A
       Letter = '<65>'               !Same thing
       Word   = School.ToCharArray   !Word holds Harding
```

```
! No string literal operator
Msg    &STRING
       CODE
       Msg = 'File is c:\temp\x.dat'
```

```
! String comparison
Mascot &STRING
       CODE
       Mascot = 'Bisons'
       IF Mascot = 'Bisons'                  !true
       IF Mascot.Equals('Bisons')            !true
       IF Mascot.ToUpper().Equals('BISONS')  !true
       IF UPPER(Mascot) = 'BISONS'           !true
       IF Mascot.CompareTo('Bisons') = 0     !true
```

```
! Substring
  Console.WriteLine(Mascot.Substring(2, 3))   !Prints "son"
  Console.WriteLine(SUB(Mascot, 3, 3))        !Prints "son"
  Console.WriteLine(Mascot[3 : 6])            !Prints "son"
```

VB.NET

```
' Special Character Constants
vbCrLf, vbCr, vbLf, vbNewLine
vbNullString
vbTab
vbBack
vbFormFeed
vbVerticalTab
""
```

```
' String concatenation (use & or +)
Dim school As String = "Harding" & vbTab
school = school & "University"
'school is "Harding(tab)University"
```

```
' Chars
Dim letter As Char = school.Chars(0)        'letter is H
letter = Convert.ToChar(65)                 'letter is A
letter = Chr(65)                            'same thing
Dim word() As Char = school.ToCharArray()   'word holds Harding
```

```
' No string literal operator
Dim msg As String = "File is c:\temp\x.dat"
```

```
' String comparison
Dim mascot As String = "Bisons"
If (mascot = "Bisons") Then                  'true
If (mascot.Equals("Bisons")) Then            'true
If (mascot.ToUpper().Equals("BISONS")) Then  'true
If (mascot.CompareTo("Bisons") = 0) Then     'true
```

```
' Substring
Console.WriteLine(mascot.Substring(2, 3)) 'Prints "son"
```

// String matching
// No exact equivalent to Like - use regular expressions

```
using System.Text.RegularExpressions;
Regex r = new Regex(@"Jo[hH]. \d:*");
if (r.Match("John 3:16").Success)   //true
```

// My birthday: Sep 3, 1964

```
DateTime dt = new DateTime(1964, 9, 3);
string s = "My birthday: " + dt.ToString("MMM dd, yyyy");
```

// Mutable string

```
System.Text.StringBuilder buffer =
    new System.Text.StringBuilder("two ");
buffer.Append("three ");
buffer.Insert(0, "one ");
buffer.Replace("two", "TWO");
Console.WriteLine(buffer);      //Prints "one TWO three"
```

! String matching
! No exact equivalent to Like - use regular expressions or MATCH

```
  USING('System.Text.RegularExpressions')
R &Regex
A &STRING
B &STRING
C &STRING
  CODE
  R := NEW Regex('Jo[hH]. \d:*')
  IF R.Match('John 3:16').Success          !true
  A = 'Richard'
  B = 'RICHARD'
  C = 'R*'
  IF MATCH(A,B,MATCH:Simple+Match:NoCase)  !true: case insensitive
  IF MATCH(A,B,MATCH:Soundex)              !true: soundex
  IF MATCH(A,C)                            !true: wildcard (default)
  IF MATCH('Fireworks on the fourth', '{{4|four}th', |
          MATCH:Regular+Match:NoCase)      !true: RegEx
  IF MATCH('July 4th fireworks', '{{4|four}th', |
          MATCH:Regular+Match:NoCase)      !true: RegEx
```

! My birthday: Sep 3, 1964

```
DT DateTime
S  &STRING
  CODE
  DT = NEW DateTime(1964, 9, 3)
  S  = 'My birthday: '& DT.ToString('MMM dd, yyyy')
```

! Mutable string

```
Buffer System.Text.StringBuilder('two ')
  CODE
  Buffer.Append('three ')
  Buffer.Insert(0, 'one ')
  Buffer.Replace('two', 'TWO')
  Console.WriteLine(Buffer)      !Prints "one TWO three"
```

' String matching

```
If ("John 3:16" Like "Jo[Hh]? #:*") Then    'true

Imports System.Text.RegularExpressions      'More powerful than Like
Dim r As New Regex("Jo[hH]. \d:*")
If (r.Match("John 3:16").Success) Then      'true
```

' My birthday: Sep 3, 1964

```
Dim dt As New DateTime(1964, 9, 3)
Dim s As String = "My birthday: " & dt.ToString("MMM dd, yyyy")
```

' Mutable string

```
Dim buffer As New System.Text.StringBuilder("two ")
buffer.Append("three ")
buffer.Insert(0, "one ")
buffer.Replace("two", "TWO")
Console.WriteLine(buffer)      'Prints "one TWO three"
```

Exception Handling

C#

// Throw an exception

```
Exception ex = new Exception("Something is really wrong.");
throw ex;
```

Clarion#

! Throw an exception

```
Ex &Exception('Something is really wrong.')
   CODE
   THROW Ex
```

VB.NET

' Throw an exception

```
Dim ex As New Exception("Something is really wrong.")
Throw  ex
```

```
// Catch an exception
try {
  y = 0;
  x = 10 / y;
}
catch (Exception ex) {    //Argument is optional, no "When" keyword
  Console.WriteLine(ex.Message);
}
finally {
  //Requires reference to the Microsoft.VisualBasic.dll
  //assembly (pre .NET Framework v2.0)
  Microsoft.VisualBasic.Interaction.Beep();
}
```

```
! Catch an exception
  TRY
    y = 0;
    x = 10 / y;
  CATCH(Exception Ex)    !Argument is optional, no "When" keyword
    Console.WriteLine(Ex.Message);
  FINALLY
    BEEP
  END
```

```
' Catch an exception
Try
  y = 0
  x = 10 / y
Catch ex As Exception When y = 0 'Argument and When is optional
  Console.WriteLine(ex.Message)
Finally
  Beep()
End Try

' Deprecated unstructured error handling
On Error GoTo MyErrorHandler
...
MyErrorHandler: Console.WriteLine(Err.Description)
```

Namespaces

C#

```
namespace Harding.Compsci.Graphics {
  ...
}

// or

namespace Harding {
  namespace Compsci {
    namespace Graphics {
      ...
    }
  }
}

using Harding.Compsci.Graphics;
```

Clarion#

```
NAMESPACE('Harding.Compsci.Graphics')



!Progressive, nested namespaces unsupported



USING('Harding.Compsci.Graphics')
```

VB.NET

```
Namespace Harding.Compsci.Graphics
  ...
End Namespace

' or

Namespace Harding
  Namespace Compsci
    Namespace Graphics
      ...
    End Namespace
  End Namespace
End Namespace

Imports Harding.Compsci.Graphics
```

Classes / Interfaces

C#

```
// Accessibility keywords
public
private
internal
protected
protected internal
static

// Inheritance
class FootballGame : Competition {
  ...
}
```

Clarion#

```
! Accessibility keywords
PUBLIC
PRIVATE
INTERNAL
PROTECTED
PROTECTED INTERNAL
STATIC

! Inheritance
FootballGame CLASS(Competition)
  ...
            END
```

VB.NET

```
' Accessibility keywords
Public
Private
Friend
Protected
Protected Friend
Shared

' Inheritance
Class FootballGame
  Inherits Competition
  ...
End Class
```

```
// Interface definition
interface IAlarmClock {
  ...
}

// Extending an interface
interface IAlarmClock : IClock {
  ...
}


// Interface implementation
class WristWatch : IAlarmClock, ITimer {
   ...
}
```

```
! Interface definition
IAlarmClock   INTERFACE
  ...
              END

! Extending an interface
IAlarmClock   INTERFACE(IClock)
  ...
              END


! Interface implementation
WristWatch    CLASS,IMPLEMENTS(IAlarmClock),IMPLEMENTS(ITimer)
   ...
              END
```

```
' Interface definition
Interface IAlarmClock
  ...
End Interface

' Extending an interface
Interface IAlarmClock
  Inherits IClock
  ...
End Interface

' Interface implementation
Class WristWatch
  Implements IAlarmClock, ITimer
   ...
End Class
```

Constructors / Destructors

C#

```
class SuperHero {
  private int _powerLevel;

  public SuperHero() {
     _powerLevel = 0;
  }

  public SuperHero(int powerLevel) {
    this._powerLevel= powerLevel;
  }

  ~SuperHero() {
    //Destructor code to free unmanaged resources.
    //Implicitly creates a Finalize method
  }
}
```

Clarion#

```
SuperHero   CLASS
_PowerLevel   UNSIGNED,PRIVATE
Construct     PROCEDURE,PUBLIC
Construct     PROCEDURE(UNSIGNED PowerLevel),PUBLIC
Destruct      PROCEDURE
            END

SuperHero.Construct PROCEDURE
  CODE
  SELF._PowerLevel = 0

SuperHero.Construct PROCEDURE(UNSIGNED PowerLevel)
  CODE
  SELF._PowerLevel = PowerLevel

SuperHero.Destruct PROCEDURE
  CODE
  !Destructor code to free unmanaged resources.
```

VB.NET

```
Class SuperHero
  Private _powerLevel As Integer

  Public Sub New()
    _powerLevel = 0
  End Sub

  Public Sub New(ByVal powerLevel As Integer)
    Me._powerLevel = powerLevel
  End Sub

  Protected Overrides Sub Finalize()
    'Destructor code to free unmanaged resources
    MyBase.Finalize()
  End Sub
End Class
```

Using Objects

C#

```
SuperHero hero = new SuperHero();



// No "With" construct
hero.Name = "SpamMan";
hero.PowerLevel = 3;
```

Clarion#

```
Hero  &SuperHero
Hero2 &SuperHero
Obj   &Object
  CODE
  Hero &= NEW SuperHero

  ! No "With" construct
  hero.Name = 'SpamMan'
  hero.PowerLevel = 3
```

VB.NET

```
Dim hero As SuperHero = New SuperHero
'or
Dim hero As New SuperHero


With hero
  .Name = "SpamMan"
  .PowerLevel = 3
End With
```

```
hero.Defend("Laura Jones");
SuperHero.Rest();                //Calling static method


SuperHero hero2 = hero;          //Both reference the same object
hero2.Name = "WormWoman";
Console.WriteLine(hero.Name);    //Prints WormWoman

hero = null ;                    //Free the object

if (hero == null)
  hero = new SuperHero();


Object obj = new SuperHero();
if (obj is SuperHero)
  Console.WriteLine("Is a SuperHero object.");


// Mark object for quick disposal
using (StreamReader reader = File.OpenText("test.txt")) {
  string line;
  while ((line = reader.ReadLine()) != null)
    Console.WriteLine(line);
}
```

```
  Hero.Defend('Laura Jones')
  SuperHero.Rest()               !Calling static method


  Hero2 &= Hero                  !Both reference the same object
  Hero2.Name = 'WormWoman'
  Console.WriteLine(Hero.Name)   !Prints "WormWoman"

  Hero &= NULL                   !Free the object

  IF Hero &= NULL
    Hero &= NEW SuperHero
  END

  Obj &= NEW SuperHero();
  IF Obj IS SuperHero
    Console.WriteLine('Is a SuperHero object.')
  END

! No equivalent to USING(Resource) to mark for quick disposal
```

```
hero.Defend("Laura Jones")
hero.Rest()    'Calling Shared method
'or
SuperHero.Rest()

Dim hero2 As SuperHero = hero  'Both reference the same object
hero2.Name = "WormWoman"
Console.WriteLine(hero.Name)   'Prints WormWoman

hero = Nothing                 'Free the object

If hero Is Nothing Then _
  hero = New SuperHero


Dim obj As Object = New SuperHero
If TypeOf obj Is SuperHero Then _
  Console.WriteLine("Is a SuperHero object.")


' Mark object for quick disposal
Using reader As StreamReader = File.OpenText("test.txt")
  Dim line As String = reader.ReadLine()
  While Not line Is Nothing
    Console.WriteLine(line)
    line = reader.ReadLine()
  End While
End Using
```

Structs

C#

```
struct StudentRecord {
  public string name;
  public float gpa;

  public StudentRecord(string name, float gpa) {
    this.name = name;
    this.gpa = gpa;
  }
}


StudentRecord stu = new StudentRecord("Bob", 3.5f);
StudentRecord stu2 = stu;

stu2.name = "Sue";
Console.WriteLine(stu.name);     //Prints Bob
Console.WriteLine(stu2.name);    //Prints Sue
```

Clarion#

```
StudentRecord STRUCT
Name            &STRING,PUBLIC
GPA             SREAL,PUBLIC
Construct       PROCEDURE(STRING Name,SREAL GPA),PUBLIC
                  INLINE
                    CODE
                    SELF.Name = Name
                    SELF.GPA = GPA
                  END
              END

Stu  StudentRecord('Bob', 3.5)
Stu2 StudentRecord
     CODE
     Stu2.Name = 'Sue'
     Console.WriteLine(Stu.Name)     !Prints Bob
     Console.WriteLine(Stu2.Name)    !Prints Sue
```

VB.NET

```
Structure StudentRecord
  Public name As String
  Public gpa As Single

  Public Sub New(ByVal name As String, ByVal gpa As Single)
    Me.name = name
    Me.gpa = gpa
  End Sub
End Structure


Dim stu As StudentRecord = New StudentRecord("Bob", 3.5)
Dim stu2 As StudentRecord = stu

stu2.name = "Sue"
Console.WriteLine(stu.name)     'Prints Bob
Console.WriteLine(stu2.name)    'Prints Sue
```

Properties

C#

```
private int _size;

public int Size {
  get {
    return _size;
  }
  set {               //parameter is always "value"
    if (value < 0)
      _size = 0;
    else
      _size = value;
  }
}
```

```
// Usage
foo.Size++;
```

Clarion#

```
_Size UNSIGNED,PRIVATE
Size  PROPERTY,UNSIGNED,PUBLIC
        INLINE
        GETTER
          CODE
          RETURN SELF._Size
        SETTER    !parameter is always "Value"
          CODE
          IF Value < 0
            SELF._Size = 0
          ELSE
            SELF._Size = Value
          END
        END

! Rather than use INLINE, you may define Get_PropertyName
! and Set_PropertyName methods separately.
ClassName.Get_Size PROCEDURE
  CODE
  RETURN SELF._Size
ClassName.Set_Size PROCEDURE(UNSIGNED pValue)
  CODE
  SELF._Size = CHOOSE(pValue < 0, 0, pValue)
```

```
! Usage
  Foo.Size += 1
```

VB.NET

```
Private _size As Integer

Public Property Size() As Integer
  Get
    Return _size
  End Get
  Set (ByVal Value As Integer)
    If Value < 0 Then
      _size = 0
    Else
      _size = Value
    End If
  End Set
End Property
```

```
' Usage
foo.Size += 1
```

Delegates / Events

C#

```
delegate void MsgArrivedEventHandler(string message);

event MsgArrivedEventHandler MsgArrivedEvent;

// Delegates must be used with events in C#
```

```
MsgArrivedEvent += new
  MsgArrivedEventHandler(My_MsgArrivedEventCallback);
MsgArrivedEvent("Test message"); //Throws exception if obj is null
MsgArrivedEvent -= new
  MsgArrivedEventHandler(My_MsgArrivedEventCallback);
```

Clarion#

```
! Explicit DELEGATE keyword is unsupported, but PROCEDURE(...),TYPE
! declarations can be used to work with event handlers, etc.

  MsgArrivedEventHandler PROCEDURE(STRING Message),TYPE

! Explicit EVENT keyword is unsupported, although event properties
! associated with derived objects can be utilized.

! Consequently, this example assumes that MsgArrivedEvent has been declared
! as an "EVENT" property by a parent class.
```

```
  MsgArrivedEvent += My_MsgArrivedEventCallback

  MsgArrivedEvent('Test message')
  MsgArrivedEvent -= My_MsgArrivedEventCallback
```

VB.NET

```
Delegate Sub MsgArrivedEventHandler(ByVal message As String)

Event MsgArrivedEvent As MsgArrivedEventHandler

' or to define an event which declares a delegate implicitly
Event MsgArrivedEvent(ByVal message As String)
```

```
AddHandler MsgArrivedEvent, AddressOf My_MsgArrivedCallback
' Won't throw an exception if obj is Nothing
RaiseEvent MsgArrivedEvent("Test message")
RemoveHandler MsgArrivedEvent, AddressOf My_MsgArrivedCallback
```

```
using System;
using System.Windows.Forms;

public class MyClass {
  Button MyButton;



  public void Init() {
    MyButton = new Button();
    MyButton.Click += new EventHandler(MyButton_Click);
  }

  private void MyButton_Click(object sender, EventArgs e) {
    MessageBox.Show("Button was clicked", "Info",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
  }
}
```

```
  USING('System')
  USING('System.Windows.Forms')

MyForm         CLASS,TYPE,NETCLASS,PARTIAL
MyButton         &Button
Init             PROCEDURE
MyButton_Click   PROCEDURE(Object Sender, EventArgs E)
                 END

MyForm.Init PROCEDURE
  CODE
  MyButton &= NEW Button
  SELF.MyButton.Click += SELF.MyButton_Click

MyForm.MyButton_Click PROCEDURE(Object Sender, EventArgs E)
  CODE
  MessageBox.Show('Button was clicked', 'Info', |
      MessageBoxButtons.OK, MessageBoxIcon.Information)
```

```
Imports System
Imports System.Windows.Forms

Public Class MyForm
  Dim WithEvents MyButton As Button



  Public Sub Init
    MyButton = New Button
  End Sub


  Private Sub MyButton_Click(ByVal sender As Object, _
      ByVal e As EventArgs) Handles MyButton.Click
    MessageBox.Show(Me, "Button was clicked", "Info", _
        MessageBoxButtons.OK, MessageBoxIcon.Information)
  End Sub
End Class
```

Console I/O

C#

```
Console.Write("What's your name? ");
string name = Console.ReadLine();
Console.Write("How old are you? ");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("{0} is {1} years old.", name, age);
// or
Console.WriteLine(name + " is " + age + " years old.");


int c = Console.Read();   //Read single char
Console.WriteLine(c);     //Prints 65 if user enters "A"
```

Clarion#

```
Name &STRING
Age  UNSIGNED
C    UNSIGNED
  CODE
  Console.Write('What's your name? ')
  Name = Console.ReadLine()
  Console.Write('How old are you? ')
  Age = Convert.ToInt32(Console.ReadLine())
  Console.WriteLine("{{0} is {{1} years old.", Name, Age);
  ! or
  Console.WriteLine(Name + 'is '+ age + 'years old.');


  C = Console.Read()      !Read single char
  Console.WriteLine(C)    !Prints 65 if user enters "A"    0
```

VB.NET

```
Console.Write("What's your name? ")
Dim name As String = Console.ReadLine()
Console.Write("How old are you? ")
Dim age As Integer = Val(Console.ReadLine())
Console.WriteLine("{0} is {1} years old.", name, age)
' or
Console.WriteLine(name & " is " & age & " years old.")

Dim c As Integer
c = Console.Read()      'Read single char
Console.WriteLine(c)    'Prints 65 if user enters "A"
```

File I/O

C#

```
using System.IO;

// Write out to text file
StreamWriter writer = File.CreateText("c:\\myfile.txt");
writer.WriteLine("Out to file.");
writer.Close();
```

Clarion#

```
  USING('System.IO')

! Write out to text file
Writer &StreamWriter
  CODE
  Writer &= File.CreateText('c:\myfile.txt')
  Writer.WriteLine('Out to file.')
  Writer.Close()
```

VB.NET

```
Imports System.IO

' Write out to text file
Dim writer As StreamWriter = File.CreateText("c:\myfile.txt")
writer.WriteLine("Out to file.")
writer.Close()
```

```
// Read all lines from text file
StreamReader reader = File.OpenText("c:\\myfile.txt");
string line = reader.ReadLine();
while (line != null) {
  Console.WriteLine(line);
  line = reader.ReadLine();
}
reader.Close();
```

```
// Write out to binary file
string str = "Text data";
int num = 123;
BinaryWriter binWriter =
   new BinaryWriter(File.OpenWrite("c:\\myfile.dat"));
binWriter.Write(str);
binWriter.Write(num);
binWriter.Close();
```

```
// Read from binary file
BinaryReader binReader =
   new BinaryReader(File.OpenRead("c:\\myfile.dat"));
str = binReader.ReadString();
num = binReader.ReadInt32();
binReader.Close();
```

```
! Read all lines from text file
Reader &StreamReader
Line   &STRING
  CODE
  Reader &= File.OpenText('c:\myfile.txt')
  Line = Reader.ReadLine()
  LOOP WHILE NOT Line &= NULL
    Console.WriteLine(Line)
    Line = Reader.ReadLine()
  END
  Reader.Close()
```

```
! Write out to binary file
Str       &STRING
Num       SIGNED(123)
BinWriter &BinaryWriter
  CODE
  Str = 'Text data'
  BinWriter &= NEW BinaryWriter(File.OpenWrite('c:\myfile.dat'))
  BinWriter.Write(Str)
  BinWriter.Write(Num)
  BinWriter.Close()
```

```
! Read from binary file
BinReader &BinaryReader
  CODE
  BinReader &= NEW BinaryReader(File.OpenRead('c:\myfile.dat'))
  Str = BinReader.ReadString()
  Num = BinReader.ReadInt32()
  BinReader.Close()
```

```
' Read all lines from text file
Dim reader As StreamReader = File.OpenText("c:\myfile.txt")
Dim line As String = reader.ReadLine()
While Not line Is Nothing
  Console.WriteLine(line)
  line = reader.ReadLine()
End While
reader.Close()
```

```
' Write out to binary file
Dim str As String = "Text data"
Dim num As Integer = 123
Dim binWriter As New BinaryWriter(File.OpenWrite("c:\myfile.dat"))
binWriter.Write(str)
binWriter.Write(num)
binWriter.Close()
```

```
' Read from binary file
Dim binReader As New BinaryReader(File.OpenRead("c:\myfile.dat"))
str = binReader.ReadString()
num = binReader.ReadInt32()
binReader.Close()
```

Clarion Magazine

Clarion.NET First Look

by Dave Harms

Published 2007-11-20

The first Clarion.NET beta was released to subscription program participants on Saturday, Nov 17, 2007. Mark that on your calendar, because it's the beginning of a new era.

First, a few cautions about the beta. One, it's a beta, so you know some stuff won't work the way it should. Two, there's no AppGen yet, so you can't develop massive Clarion.NET applications (although there is a C6 wizard that will generate a basic app for you from your dictionary). More on the template later. And three, most importantly, is that these are early days; not only may there be unintentional errors in this document, but some things may change, and may even change wildly, between now and final release (in particular the wizard-generated source code).

Okay, with the disclaimers out of the way, let me just say that this is one rockin' product.

Like a lot of Clarion developers, I'd been waiting for this baby for a long, long time. And like many other folks I'd done some .NET programming in Visual Studio, and even a little in SharpDevelop. So I thought I had a pretty good idea of what to expect in Clarion.NET. And yet...

It's one thing to anticipate that Clarion will do .NET. It's another thing to open up the new IDE and see that you can use it to create:

- Windows Forms (WinForms) applications
- ASP.NET applications (Enterprise Edition only)
- Compact framework applications (Enterprise Edition only)
- C# applications
- VB.NET applications

I did mention that the AppGen isn't ready yet, right? And there may not even be C# or VB.NET templates any time soon, and okay, those languages are courtesy of the SharpDevelop IDE code base so let's not get all het up about it. But still.

Think about it.

We're talking Windows application development using an industry standard platform. ASP.NET web development with IIS. Mobile applications. Mixing and matching with other .NET languages. Full access to the many benefits of the .NET platform. A wild selection of add-on programming tools (profilers, code obfuscators, unit testers, etc.), not to mention instant compatibility with a massive number of controls and libraries (and for many existing Clarion third party vendors, a truly huge new market...).

I generally prefer my prose deathless rather than breathless, so before someone accuses me of channeling Russ Eggen, I'll try to curb my enthusiasm and get down to the nitty gritty.

The Clarion.NET IDE

First, as noted in the Clarion.NET FAQ, the name Clarion.NET refers to the IDE, and Clarion# is the .NET version of the Clarion language. If you've already seen the C7 IDE then you'll find Clarion.NET familiar (see Figure 1); both products use the same IDE, which is in part based on the SharpDevelop IDE (for which SV obtained a commercial code license).

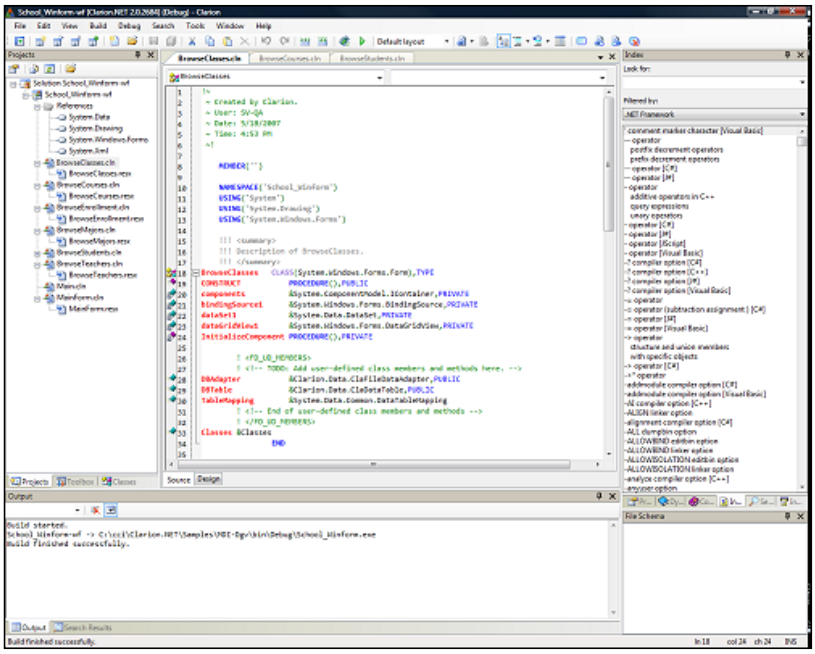


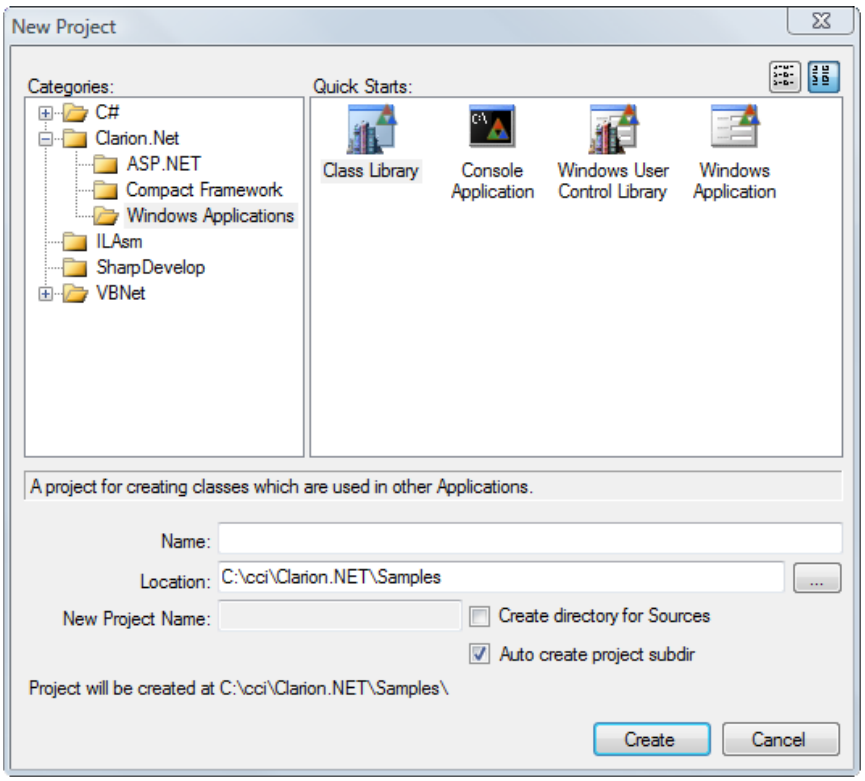
Figure 1. The Clarion.NET IDE, default layout (view full size image**)**

I won't go over the IDE in detail - if you want to see more screen shots I suggest you review the C7 alpha/beta articles. There are, however, a few significant additions in the .NET version.

.NET project types

As I indicated earlier, the beta release includes hand code support for all three Clarion.NET development targets: WinForms, WebForms (ASP.NET) and the Compact Framework (mobile devices).

Figure 2 shows the available project defaults for WinForms desktop apps.



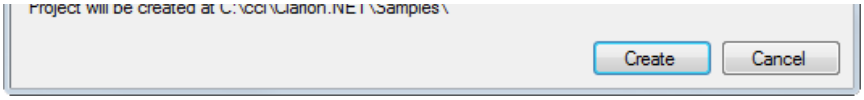


Figure 2. WinForms defaults

WinForms defaults include:

- Class Library - create just a class library with no EXE entry point. This is a bit like creating a WinAPI DLL.
- Console Application - create an application that writes to a command (i.e. DOS) window. You can still interact with a console application but only via the keyboard.
- Windows User Control Library - create your own custom window control derived from System.Windows.Forms.UserControl.
- Windows Application - the starting point for most application development; creates an executable .NET application.

Figure 3 shows the available project defaults for ASP.NET (WebForms) desktop apps.

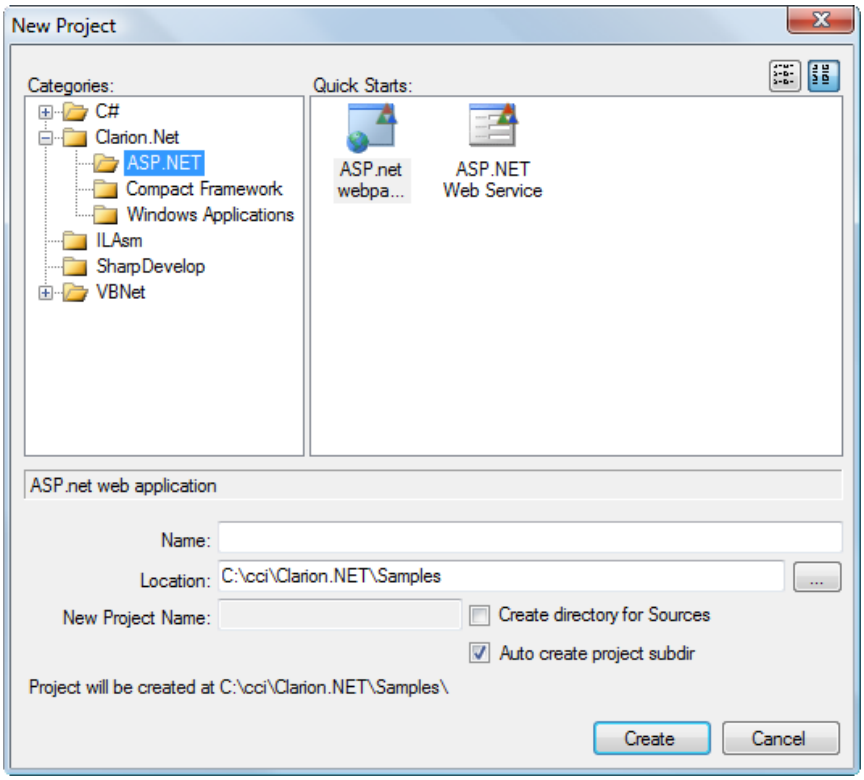


Figure 3. WebForms defaults

There are two ASP.NET options, a web application and a web service. There are already-created sample apps (see below) but you might as well create your own as this gives you the option of setting up the IIS virtual root. I couldn't get that feature to work on my machine but I suspect that's because I'd munged up my IIS configuration during testing.

The ASP.NET app provides a basic setup for an ASP.NET application, including a default.aspx file and a global.asax file, both with matching code-behind Clarion files. There is also a brief style sheet. The web app simply displays a "Hello from Clarion" page.

The services app provides a similarly basic framework for web services. The .NET library code supports both HTTP and SOAP protocols. The service generates a test web page which gives you helpful information on the XML file formats to use when calling using the SOAP 1.1 and 1.2 protocols. There are three simple functions included: add two numbers; say "Hello"; and say "Hello *name*" where you pass in the name. You can test these functions with the supplied page using the HTTP protocol, which returns the result in a simple XML page, like this:

```
<?xml version="1.0" encoding="utf-8"?>
```

<string xmlns="http://tempuri.org/">Hello Dave</string>

You'll need a SOAP client to test the SOAP 1.1 and 1.2 protocols. There's no such client included with Clarion.NET, but as you'd expect there's support for such a beast in the .NET framework and I found a C# example which you may want to try.

> **Update:** SoftVelocity's Deigo Borojovich showed me how to set up a Web Reference, which automatically creates the code to consume an HTTP web service. I'm have more on this in another article shortly.

Figure 4 shows the available project defaults for Compact Framework applications.

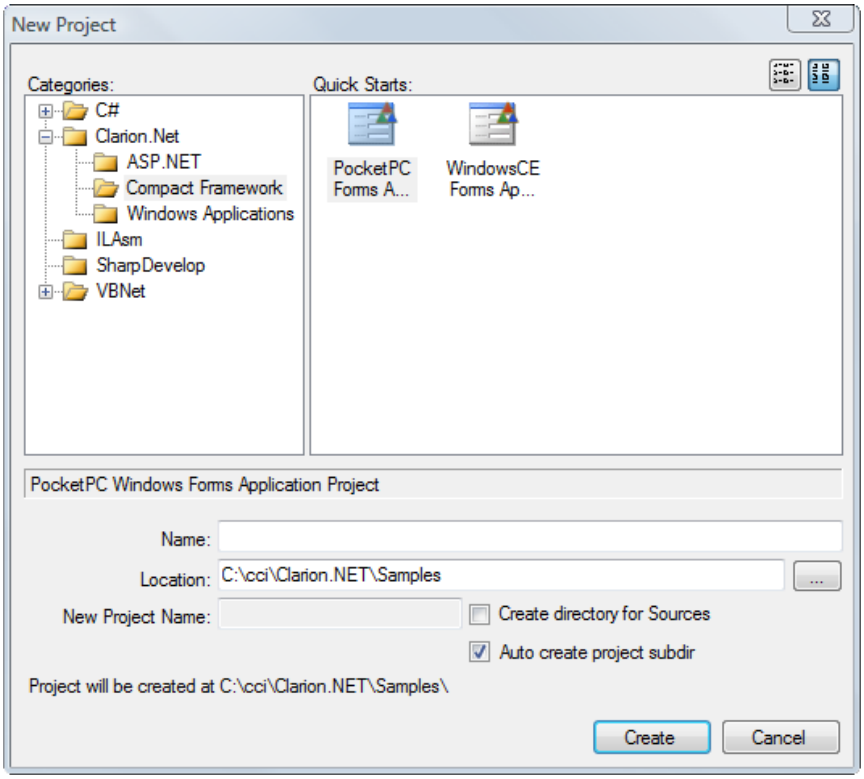


Figure 4. Compact Framework defaults

There are two CF defaults in the beta: Windows CE and PocketPC. At this point there doesn't seem to be any difference in the code created for these two defaults, but perhaps that will change in time.

Sample projects

Clarion.NET ships with a number of sample projects. These include:

- ADONetEx - console app that attaches to any ADO.NET data source.
- BlobBinaryToImage - read a TopSpeed BLOB image and display it in a .NET PictureBox control.
- CFDemo - a Compact Framework app using XML files for data storage.
- DataBind-Files-Records - databinding example using a TopSpeed file and the .NET 2.0 BindingSource and BindingNavigator controls. Also demonstrates binding a textbox control to a field in a record structure.
- DataBind-Queue - demonstrates binding a queue to a .NET datagrid control, among other things.
- DockingWindows - demonstrates docking the WeinfenLuo.WinFormsUI.Docking DLL. Also shows how to use the Clarion.NET IDE's docking features (movable MDI windows) in your own programs.
- DragandDropImage - drag and drop an image.
- DragandDropText - drag and drop text between list boxes.

- ForEachQueue - a console app showing how to use FOREACH with a queue.
- GlassButtons - demo of SV's glass button controls.
- ListControlTotallingTree - manipulate column headers and run dynamic totals on a list.
- MDI-Dgv - demo of TYPEd files, where each instance of the file has its own data buffer.
- MixedLanguages - calling C# from Clarion# and vice versa, using two projects in one solution.
- MyFirstWebService - launch a basic web service.
- MyFirstWebSite - a simple WebForms/ASP.NET application.
- NetRemoting - demo of .NET's inter-application communication capability.
- People - the People example for .NET.
- ScreenCapture - using the System.Drawing classes to capture the screen.

The sample projects alone provide many points for future study, and do a good job of illustrating the breadth of programming options available in Clarion.NET. I'll cover some of these samples in future articles.

WinForms

Drag yourself away from web apps and mobile apps for a moment. I mentioned earlier that Clarion.NET comes with a Clarion 6 wizard that lets you generate a WinForms app from a dictionary of your choosing. You won't get a complicated application, just a bunch of browse-form pairs, and you can't edit them in any AppGen, but these applications can be compiled in Clarion.NET and do work. This is an interim measure until AppGen arrives, and it mainly serves as a way to provide developers with Clarion# code they can study.

Clarion is originally a procedural language, with object-oriented extensions. .NET, however, is only an object-oriented platform. But Clarion# retains its hybrid nature, so you can still write procedural code if you want - your procedures are converted to class methods under the hood.

The generated code for the main module demonstrates this hybrid nature, and looks for the most part like the code at the start of any current Clarion program:

> **DISCLAIMER: The following wizard-generated code is purely for illustration purposes. It is not necessarily indicative of the code that will be generated by the Clarion# template chain. In particular I would *not* expect to see legacy-style utility procedures declared in the MAP.**

```
PROGRAM

 NAMESPACE('PEOPLEApp')
 USING('System')
 USING('System.Drawing')
 USING('System.Windows.Forms')

!!! <summary>
!!! Description of RequestType ENUM
!!! </summary>
RequestType ENUM
None         ITEM(0)
InsertRecord  ITEM(1)!  Add a record to table
ChangeRecord  ITEM(2)!  Change the current record
DeleteRecord  ITEM(3)!  Delete the current record
SelectRecord  ITEM(4)!  Select the current record
ProcessRecord ITEM(5)!  Process the current record
```

```
ViewRecord    ITEM(6)!  View the current record
SaveRecord    ITEM(7)!  Save the current record
          END

!!! <summary>
!!! Description of ResponseType ENUM
!!! </summary>
ResponseType ENUM
   None            ! 0
   RequestCompleted ! 1 Update Completed
   RequestCancelled ! 2 Update Aborted
          END

 MAP
  AutoIncColumn(System.Data.DataRow NewRow, |
    System.Data.DataTable AppDataTable, STRING FieldName)
  StopValidation(System.Windows.Forms.Control ctrlParent)
  CheckOpen(FILE pFile,bool OverrideCreate=true,|
    BYTE OverrideOpenMode=66),System.Int32
 END

GUIDStr                  CLASTRING(16),PUBLIC


!Region Files declaration
people       FILE,DRIVER('TOPSPEED'),CREATE,BINDABLE,TYPE
KeyId         KEY(Id),NOCASE,OPT,PRIMARY
KeyLastName    KEY(LastName),DUP,NOCASE
Record        RECORD
Id             System.Int32
FirstName       CLASTRING(30)
LastName        CLASTRING(30)
Gender          CLASTRING(1)
            END
          END
!endregion

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

There are a few statements right at the top that may look unfamiliar. The NAMESPACE directive indicates that anything newly declared in this file will automatically have its label prefixed with PEOPLEApp, followed by a period. That means, for instance, that RequestType is really PEOPLEApp.RequestType. Namespaces are how .NET avoids name collisions. You can have different classes called, say, String, provided they are declared in different namespaces.

Writing fully qualified class names, which means the namespace and the name, can get tiresome especially when you have nested namespaces such as System.Windows.Forms. The USING statements allow you to use a sort of shorthand - the compiler will automatically try the namespaces listed in USING statements when its attempting to resolve a label in your code.

You still have a PROGRAM statement, a MAP, some global equates and data, a file definition, and a CODE statement to kick off execution (and yes, to save space I've left off the definitions of the procs declared in the map). And then you have a couple of lines that again look a bit different.

Application is actually System.Windows.Forms.Application, a special class which you cannot derive, and which manages the WinForms application. The first method call enables visual styles, and the second call, the Run method, launches an instance of the MainForm object.

At this point things start to get just a little different. MainForm is actually a class, not a procedure, and it's derived from System.Windows.Forms.Form. You won't see an INCLUDE statement anywhere because it isn't needed; as long as MainForm is in the same project, that's enough.

MainForm, which contains the application frame and associated menu, is contained in two files. Figure 5 shows the project tree for PEOPLEApp.

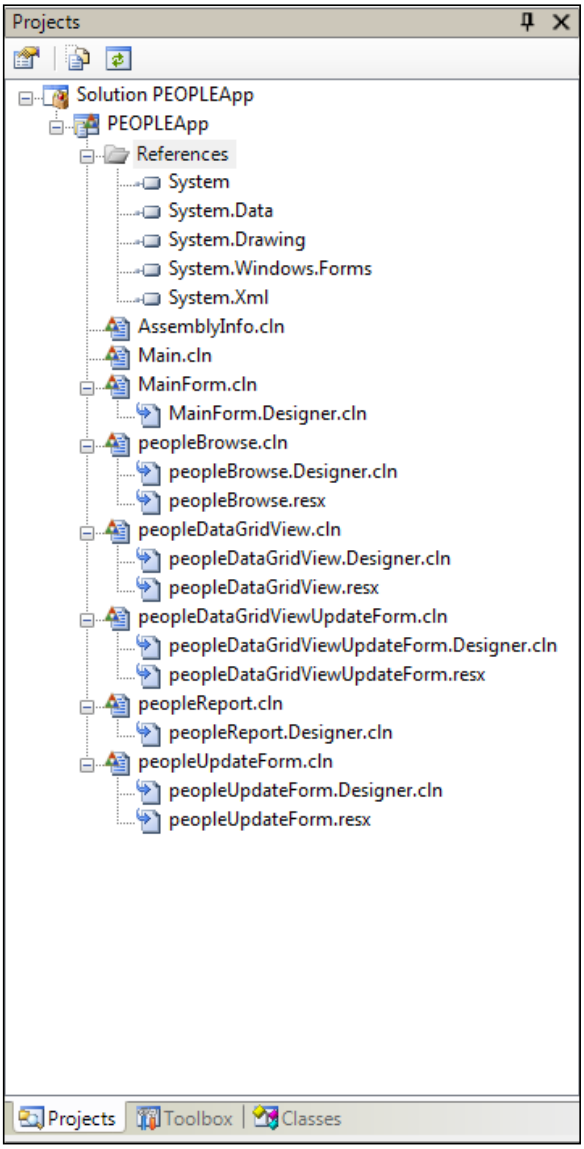


Figure 5. The PEOPLEApp project tree

As you can see there's a MainForm.cln file, and a MainForm.Designer.cln file. Together, these files make up the MainForm class code.

You're probably familiar with how Clarion separates classes into prototypes (INC files) and implementation (CLW files). This is something different, and it's called a partial class.

Partial classes

A partial class is a .NET feature whereby you can declare part of a class in one file, and another part in another file. One of the reasons for doing this is to make it easier to separate presentation (user interface) code from business logic. In Clarion.NET, as in C# and VB.NET, when it comes to window handling one part of the partial class is the bit where you add your code, and the other part is managed by the window designer.

Here are a couple of methods from the business logic half of the MainForm class (in MainForm.clw):

```
MainForm.peopleDataGridToolStripMenuItem_Click |
  PROCEDURE(System.Object sender, System.EventArgs e)
W &peopleDataGridView
    CODE
      W := NEW peopleDataGridView()
      W.MdiParent := SELF
      W.Show()


MainForm.peopleBrowseToolStripMenuItem_Click |
  PROCEDURE(System.Object sender, System.EventArgs e)
W &peopleBrowse
    CODE
      W := NEW peopleBrowse()
      W.MdiParent := SELF
      W.Show()
```

And here are a few methods from the window designer half of MainMenu (in MainForm.Designer.cln):

```
SELF.menuStrip := NEW System.Windows.Forms.MenuStrip()
SELF.fileToolStripMenuItem := |
  NEW System.Windows.Forms.ToolStripMenuItem()
SELF.printPreviewToolStripMenuItem := |
  NEW System.Windows.Forms.ToolStripMenuItem()
SELF.toolStripSeparator2 := |
  NEW System.Windows.Forms.ToolStripSeparator()
SELF.exitToolStripMenuItem := |
  NEW System.Windows.Forms.ToolStripMenuItem()
SELF.editToolStripMenuItem := |
  NEW System.Windows.Forms.ToolStripMenuItem()
! ...
SELF.menuStrip.Items.AddRange(NEW System.Windows.Forms.ToolStripItem[]|
  {SELF.fileToolStripMenuItem, SELF.editToolStripMenuItem, |
  SELF.dataGridToolStripMenuItem, SELF.browseToolStripMenuItem, |
  SELF.reportToolStripMenuItem, SELF.windowToolStripMenu, |
```

```
  SELF.helpToolStripMenuItem})
SELF.menuStrip.Location := NEW System.Drawing.Point(0, 0)
SELF.menuStrip.MdiWindowListItem := SELF.windowToolStripMenu
SELF.menuStrip.Name := 'menuStrip'
SELF.menuStrip.Size := NEW System.Drawing.Size(592, 24)
SELF.menuStrip.TabIndex := 0
SELF.menuStrip.Text := 'menuStrip'
```

As you can see, the business logic side determines what happens when the user clicks on a menu item, while the designer side creates and lays out the menu items.

What this means is that the window designing process, as far as the IDE and the runtime goes, is quite different from how it works in WinAPI Clarion. What we're used to is a WINDOW structure which is read by the designer, and which the compiler turns into a bunch of Windows API and runtime library calls to create and display the window. In Clarion# the window "structure" is actually stored using exactly the same code as will be needed at runtime to create and display the window.

There's a third file that can come into play in any window, and that's the resource file. This is an XML file specifying how images and other resources are to be handled, and it has the extension resx. Here's some code that loads an image from a resource file:

```
SELF.toolStripButtonClose.Image := |
 ((resources.GetObject('toolStripButtonClose.Image') |
 TRYAS System.Drawing.Image))
```

Reports

Unlike windows, Clarion.NET's report structures look pretty much unchanged, but with browses and forms, each report is a CLASS not a PROCEDURE. Basically a report opens a progress window, starts a timer, and on each timer tick prints a report detail:

```
peopleReport.Timer_Tick PROCEDURE(System.Object sender, |
                        System.EventArgs e)
 CODE
   IF NOT SELF.Print()
     SELF.Close()
   END

peopleReport.Print PROCEDURE()
 CODE
   SELF.RecordsThisCycle = 0
   LOOP SELF.RecordsPerCycle TIMES
      IF SELF.GetNextRecord()
        IF SELF.ProcessCancelled = false
          PRINT(SELF.detailBand)
        END
      ELSE
        SELF.timer.Stop()
        SELF.timer.Enabled := false
        RETURN False
```

```
    END
END
RETURN true
```

Keep in mind that this is a pretty simple report with just one detail.

Help! It's all OOP!

While the Clarion# code generated by the C6 wizard is salted with procedural code, the fact is that, yes, it is almost all object-oriented code. So what does that mean to you, if you don't know OOP at all, or aren't very comfortable with OOP?

It seems to me that OOP in Clarion# is a lot like OOP in ABC. If you can get by with ABC and AppGen, then you can probably get by in Clarion# as well. But you'll really be missing out on the power and flexibility of the .NET framework if you don't understand object-oriented programming. It's like travelling in a foreign country. You can get by with a phrase book, but you'll have a lot more fun and a lot less trouble if you can speak the language. You may want to get started by reading the three part series The ABCs of OOP; also keep your eyes open for Clarion# tutorials coming up in the magazine.

Data binding

In WinAPI applications you typically load data from a file into a queue, and then you link that queue to a list box with the FROM attribute. The queue is *bound* to the list control - change the data in the list box and the displayed data instantly changes accordingly. This is an example of data binding - the data store (in this case the queue) is tightly connected to the visual control (the list box). But how it all happens is hidden from the developer in a Clarion WinAPI application.

.NET takes the concept of data binding quite a lot further, exposing the plumbing and allowing a wide variety of classes to function as data sources for controls. Sample projects shipped with the beta demonstrate how to bind not just a queue to a control, but also a record and a file.

Visuals

Clarion.NET applications look marginally different from Clarion WinAPI applications. Figure 6 shows the Clarion.NET version of the people app, and Figure 7 shows the Clarion WinAPI version.

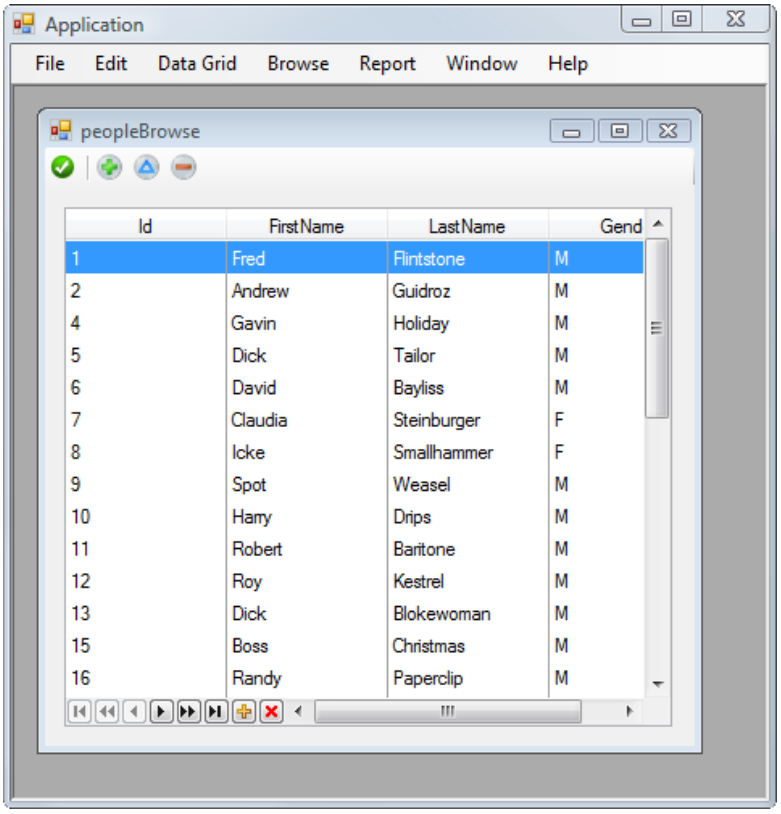


Figure 6. A Clarion.NET People application

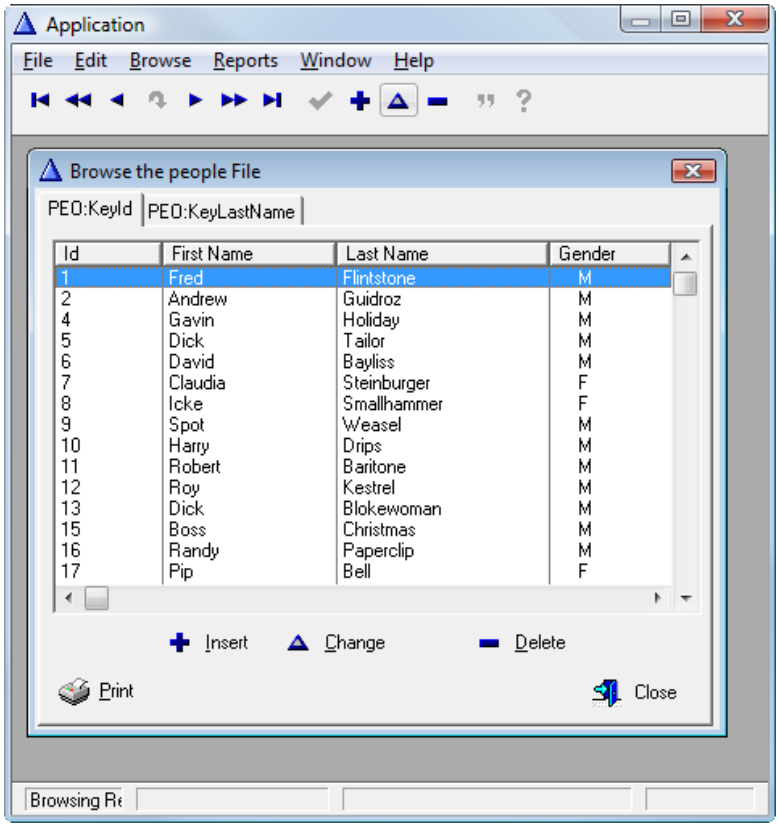


Figure 7. A Clarion WinAPI People application

There are some obvious differences not just due to the platform change. The older app has a navigation toolbar and large Insert/Change/Delete buttons, as well as a Print button, sorting tabs, and a status bar. The .NET app has a local toolbar and a VCR control. That doesn't mean the .NET app can't have tabs and buttons and all that, it just means the template wizard writer didn't put that stuff in.

There are some more subtle differences in the visuals as well. The list box headers are drawn differently, and the scroll bars, finally, are sized according to the portion of the data shown in the control (although I think you can expect that to work a little differently for page-loaded list boxes). There's also a gentle gradient in the background window color on the .NET browse (possibly because of the toolbar), while the older browse window is a solid color. If you mouse over menu items, buttons or column headers they will change color (at least on Vista - on my machine the highlight color is blue, presumably a feature of the color scheme I've chosen).

This is all pretty plain Jane Clarion stuff so far, but even Clarion is getting into the act. Figure 8 shows SoftVelocity's new glass buttons.

Figure 8. Glass buttons on image background

Hover your mouse over a glass button and you'll see a glowing highlight; hover over a flat button and the button outline appears. Click on the button and the background darkens. Figure 9 shows the buttons without the image background.

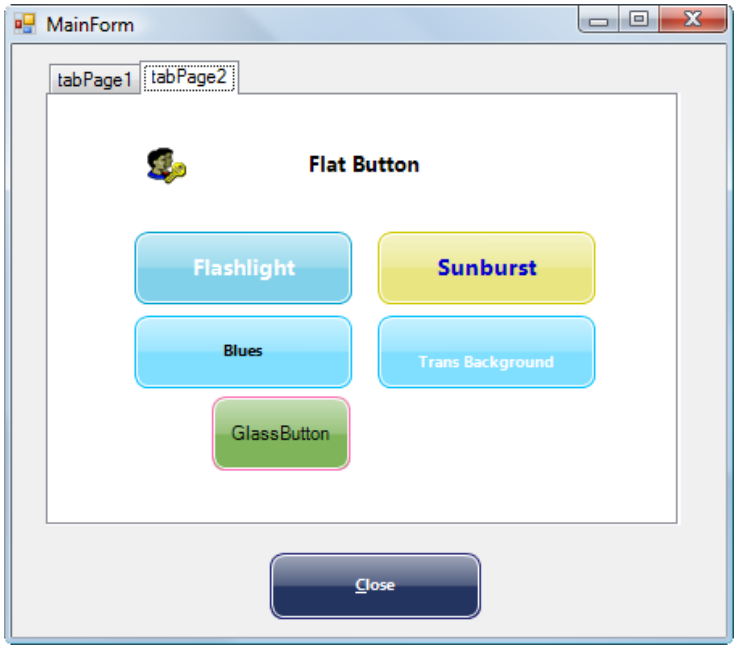


Figure 9. Glass buttons on white background

If the standard controls supplied by SoftVelocity and the .NET framework aren't good enough for you, there are a great many custom controls available as well (and of course you can create your own custom controls). Couple all that with .NET's data binding capabilities and you have a full toolbox for tackling the most unusual data management problems.

Summary

Clarion.NET is clearly a huge step forward for Clarion development. Yes, it's still in beta, and there will be bugs to squash, and we're still waiting on AppGen. But this is a real, working, .NET language. It's a triple crown of desktop, web, and mobile programming, all solidly in the mainstream of Windows technology. Clarion has never had that before.

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

Reader Comments

Posted on Tuesday, November 20, 2007 by Russell Eggen

Show them the debugger next! Suggest setting the breakpoint in the editor, THEN launching the debugger.

.....

Posted on Tuesday, November 20, 2007 by Bob Zaunere

Nice article Dave!

Add a comment

Clarion Magazine

The Clarion.NET FAQ

by Dave Harms

Published 2007-11-17

In this Frequently Asked Questions (FAQ) page I'll attempt to answer at least some of the questions that have been raised about Clarion.NET. If you log in you can post your own comments and questions below.

Terminology!

One of the problems in discussing Clarion.NET is finding a meaningful term for traditional (non-.NET) Clarion applications. Some developers use the term *Win32*, but that isn't always helpful since .NET applications can also be Win32 (or Win64).

As noted below, one of the key differences between the Clarion we now know and Clarion.NET is that the former is ultimately built on top of the Windows API. For that reason I will refer to "traditional" Clarion Windows applications as WinAPI apps, to differentiate them from .NET apps.

What did you just say?

I said that when I use the term *WinAPI app* I'm referring to a traditional Clarion (Windows) application.

But I don't use the Windows API in my applications.

You might not, but without the Windows API there would be no Clarion (for Windows) as we know it. All your apps use it all the time.

Is Clarion.NET available now?

Yes, the first beta was released to subscription program participants on Saturday, Nov 17, 2007.

What is Clarion.NET?

Clarion.NET is a version of the Clarion development environment specifically designed for Microsoft's .NET platform.

Okay, what is .NET?

From Wikipedia:

> The Microsoft .NET Framework is a software component that can be added to or included in the Microsoft Windows operating system. It provides a large body of pre-coded solutions to common program requirements, and manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering, and is intended to be used by most new applications created for the Windows platform.

I'll just add (for now) that .NET is an object-oriented framework which accommodates a great variety of programming languages.

Isn't .NET all about web development?

You might think so, given the name of the platform. While .NET contains many web/Internet-related classes, it also has extensive support for desktop development. As well, there's a version of the framework for mobile devices.

.NET is a comprehensive development and runtime environment suitable for most kinds of programming, but not usually for low level hardware-oriented stuff (although you *can* get an 80386 assembler that generates .NET code).

What's the difference between Clarion.NET applications and the Clarion WinAPI applications I write now?

There are a number of differences. One of the biggest is that the Clarion language (for Windows) is highly dependent on the Windows API. For instance, when you open a window, that window is created by the Clarion runtime library via calls to the Windows API. When you create a new variable or class instance, memory is allocated via the Windows API. This is procedure-oriented coding, and procedure-oriented applications (and operating systems) tend to be hardwired - they have only set ways of doing things. For instance, the Clarion window formatter only has a specific set of controls available, and you can't add your own custom widgets. You can use add-on components like ActiveX controls, but as anyone who's tried it in Clarion can tell you, it isn't all that easy either.

In a WinAPI application you often have to work hard to make different functional units of code work together because the Windows API isn't really geared to the concept of components.

Applications written for .NET, however, use the extensive .NET Framework class library rather than the Windows API. In .NET it's all about objects. You can assemble an application out of different classes and components much more easily because .NET provides a congenial framework for all these things to work together.

Is the .NET Framework that much better than the Windows API?

Yes, in almost every way. That's not toadying to Microsoft - it's just the evolution of programming. The Windows API is disorganized - you need to know what you're looking for or you'll get lost very quickly. The .NET Framework library is organized into namespaces. For instance, security classes are grouped together, as are diagnostic classes, data access classes, etc.

The .NET framework library is much more massive than the API and is growing larger. It provides a whole lot of code that you'd otherwise have to write yourself. There are many other benefits to the library but many of these are better described in the context of the Common Language Runtime (CLR).

Okay, I'll bite. What's the Common Language Runtime?

The CLR is an essential part of .NET. It's the layer that sits between you and the hardware, and which provides important services to your applications including, but not limited to: memory management; thread management; exception handling; garbage collection; security; introspection and reflection. The CLR means many tasks are easier in .NET. For instance, if you NEW something you don't (usually) have to DISPOSE it - the CLR's garbage collector will detect when the object is no longer in use and will clean it up automatically.

Are there any other weird acronyms I should know?

Well, there's CIL, the Common Intermediate Language (often just called IL). The CLR doesn't actually run the code you write; instead, all .NET language compilers translate their code into this CIL (IL) bytecode, and that's what the CLR executes.

So .NET is a giant interpreter? This seems like a step back from true compiled languages - didn't CPD generate pseudocode **which had to be interpreted?**

You're right - CPD did in fact generate pseudocode, and it's a fair analogy. The advantage of IL is that since all .NET languages compile to an intermediate language, they all share a common platform. You can take some classes compiled in, say, Fortran.NET (no, I'm not kidding) and easily use them in C# or Boo or Clarion.NET.

What is Clarion#?

Clarion# is the .NET version of the Clarion programming *language*. The Clarion IDE for .NET is, at least at present, called Clarion.NET. But a lot of developers already use the terms Clarion# and Clarion.NET interchangeably.

Clarion is both a procedural and an object-oriented language, but .NET is an OO framework - can I still write procedure code in Clarion.NET?

Yes, you can still write procedural code in Clarion.NET. Your procedures are converted to object-oriented code when they're compiled to IL code, but you don't need to know or care about that if procedural code is what makes you happy.

Are Clarion 7 and Clarion.NET the same product?

No, they are separate products, but they share the same integrated development environment (IDE).

Does that mean if I buy Clarion 7 and Clarion.NET I'll end up with just one installed IDE able to work with both platforms?

Most likely, although there may be some versioning issues that make that more difficult, at least during the beta process. So for a time you may have two IDEs installed. I really don't know.

What can I do with Clarion.NET/Clarion# beta in the first release?

The beta includes a template wizard you can use to generate a .NET application, but since those templates run in C6, not the new IDE (AppGen isn't ready yet), you can't yet create and maintain a .NET APP file in the same way you do in C6. No word yet on when the AppGen will be ready, but meanwhile you can hand code not just desktop applications, but also ASP.NET and mobile applications.

There are a number of example solutions shipped with the beta. These include demonstrations of connecting to a database with ADO.NET, displaying a BLOB image onscreen, a mobile app, data binding, drag and drop, a FOREACH with QUEUE example, glass buttons, new listbox features, the SCHOOL application, mixing Clarion# and C#, a simple web service, a "Hello world" ASP.NET web application, a .NET remoting example, the PEOPLE app with data grid and browse procedures, and a screen capture utility.

Since Clarion# is a full .NET producer/consumer language, you will have full access to all of the .NET framework library as well as the rich supply of third party .NET tools.

What will I be able to do with Clarion.NET/Clarion# in the long run?

When complete Clarion.NET will include templates to generate desktop (WinForms) applications, web (ASP.NET) applications, and mobile (Compact Framework) applications.

What is the Compact Framework?

The Compact Framework is .NET for mobile devices, and includes about 30% of the full framework plus classes specific to mobile devices, and takes up about 1/10 of the space, mostly due to file compression.

What is ASP.NET?

ASP.NET is Microsoft's Active Server Pages web application framework for .NET. You create ASP.NET pages using a development approach similar to desktop development: you place controls on a page, and write code for those controls; ASP.NET then renders the pages accordingly and executes the code on the server side when data comes back from the browser.

How hard is it to write .NET applications?

Writing .NET applications can be both easier and harder than writing WinAPI apps, and the comparison between the two brings to mind the differences between DOS and Windows API development. You could argue that Windows development was a lot harder because you had to do so much more to create even a simple Hello World application (at least in C, if not in Clarion). On the other hand, Windows provided standard capabilities like a windowing library; in DOS you had to write or otherwise obtain a windowing library to achieve a consistent, accessible user interface. In DOS you had to know how to talk to every printer you wanted to use; in Windows you talked to the printer driver, and the printer driver sorted out the back end. Similarly .NET does a lot of the heavy lifting you now have to code in WinAPI apps, leaving newer and more complex tasks to your wily programming brain.

One big difference between WinAPI and .NET is that the latter is exclusively object-oriented. To get the most out of Clarion.NET apps you will definitely want some basic OO programming knowledge. With that knowledge in hand, I think you'll find .NET easier and safer to use than the Windows API. If you've had to deal with ActiveX or (heaven help you) directly call COM functions you'll truly find .NET an easier place to code.

.NET introduces some new language concepts that may take some getting used to, such as delegates, which are a sort of type-safe object-oriented callback mechanism.

Is .NET open source?

.NET is not open source, but Microsoft is in the process of releasing source code for some parts of the framework.

Will my third party tools work in .NET?

The majority of third party tools will need to be ported to Clarion.NET. Templates that don't generate any source code and do not depend on a particular template chain are likely to work without modification, but there aren't many that fall into that category. You're probably best off assuming you'll need new versions until you hear otherwise from the vendor.

I've often said that .NET is a two-edged sword for third party vendors. If what you provide is readily available as part of the .NET framework, then zing!! off goes your head as you step into .NET land. On the other hand, if you have a great product and you can port it to .NET, you're ready to carve a swath through a programming market that numbers in the millions of coders.

Will my customers want .NET and if so why?

Some customers will want .NET just because it's a buzzword. That's an easy sale.

Other customers may or may not know whether they want .NET. Clearly what they want is software that does what they need it to. If their needs run to eye candy or very unique user interfaces, then .NET presents some distinct advantages. There are a bazillion custom controls out there for .NET, all of which you can use with Clarion.NET. And there are many class libraries that handle important behind-the-scenes tasks as well.

I find it difficult to overstate the importance of ready access to all of that existing code. With Clarion WinAPI apps you have to be concerned about prototyping functions, register passing conventions, the arcana of COM, etc. etc. With .NET you just drop in the library and start to use it, no matter what language it's written in.

.NET offers programming benefits as well. For instance, your code compiles to IL code which is run under the watchful eye of the CLR. That means the CLR can detect problems with your code and present far more detailed information to you than you get from, say, a GPF in a WinAPI application. This makes debugging easier and faster.

Can I run .NET apps on Linux or the Mac? What is Mono?

Mono is a Novell-sponsored project to port the .NET platform's functionality to multiple platforms, including Linux, Mac OS X, Solaris, BSD, and Windows. Mono necessarily lags behind Microsoft's efforts, and currently has completed support for .NET 1.1 and mostly-complete support for .NET 2.0.

Versions? What are all these .NET versions?

Microsoft released .NET 1.0 in 2002, and 1.1 in 2003. You may see some computers with 1.1 as the latest version, (and some computers without .NET installed at all) but the standard at present is .NET 2.0, released in 2005. Clarion.NET targets 2.0 apps - there was talk in the early days of 1.1 being supported as well but the only reason I can see for supporting 1.1 is for Mono compatibility, given that Windows Forms 2.0 support is now scheduled for Mono 2.5, which does not have a release target.

For most of us, .NET 2.0 will be just where we want to be.

What about .NET 3.0? Or 3.5?

The first thing to keep in mind about .NET 3.0 is that it is not a replacement for .NET 2.0. It's a bunch of new stuff added to 2.0, including Windows Presentation Foundation, Windows Workflow Foundation, Windows Communication Foundation, and Windows Card Space. The base class library is unchanged from 2.0.

.NET 3.5 uses the same CLR as 2.0 but it adds some new stuff to the base class library, in particular support for the LINQ query language. 2.0 apps will still run fine on 3.5.

Will I need to learn C# or VB.NET?

You will not need to learn C# or VB.NET or any other .NET language, but you may want to. In particular there's a lot of C# source code out there, and you may want to adapt some of it to your own uses. If you can read object-oriented Clarion code you won't have much trouble reading C#.

What is ADO.NET?

ADO.NET is Microsoft's data access layer for .NET, and consists of data providers (i.e. drivers) and DataSets. A DataSet is a set of objects that model the database elements (tables, views, columns, rows, relations, etc.).

Will my Clarion (.clw) programs compile in Clarion#?

While much of the Clarion language is unchanged in Clarion#, it's unlikely that any single WinAPI application will compile as Clarion# code without modification.

What will I have to do to port my apps to .NET?

Porting applications to .NET is a bit of a gray area at the moment. Theoretically it can be done; the question is, is it worth the work? Clarion WinAPI apps are built on a traditional, client-server model. Is this a good approach to take into the .NET world? Do we really want ABC.NET? Perhaps a multi-tier design would be more appropriate, particularly one where you could easily reuse your business logic in desktop, web, and mobile versions of your application. SV has alluded to this kind of design but it isn't clear yet what kind of desktop application templates will be included with Clarion.NET.

Can we mix and match .NET objects with Win32 API objects easily?

You can include WinAPI code in a .NET app and vice versa. Easy is a relative term. See Wade Hatler's series of articles.

Can I get my Clarion 7 and earlier programs to use .Net components I create using Clarion.Net or other .Net languages?

Most likely you could (see the article series above) but I think this would be a stopgap measure at best.

How secure is .NET code? Can it be easily decompiled?

Code security is a legitimate concern in .NET and yes, IL code can be decompiled much more easily than native Windows executable code. .NET obfuscators alter label names and use various code scrambling approaches to make it very difficult for anyone to make sense of the decompiled result. Or you could just hire someone who's a natural at writing unreadable code.

I've heard .NET programs run slower than native code. Will I notice the difference?

.NET uses just-in-time (JIT) compiler technology, meaning that IL code is compiled to native code the first time that IL code is needed by the application. That means there is a small startup penalty but once the code is compiled it runs just like any other native code. Theoretically code produced by a JIT compiler can outperform code issued by a standard compiler because the JIT compiler can tailor the code to the hardware.

Can I include a .NET runtime with my apps so I don't have to require my customers to install .NET?

There is a tool called the Salamander .NET Linker, Native Compiler and Mini-Deployment Tool that will do just this. It's a bit expensive, and I don't know how well it works. Apparently Thinstall will also create self-contained .NET installs.

If both new Clarions deliver desktop applications - why should I buy both? Would Clarion# not be enough?

I'll assume here you mean *after* AppGen is released for C7 and Clarion.NET. While you can create real desktop apps already with Clarion.NET, most developers will want to use AppGen for larger apps.

So yes, you can create desktop apps with both. Why would you still want C7? Here are a few reasons:

- Better productivity with the new IDE
- Ability to work with different versions of Clarion within the same IDE
- C7's runtime improvements including eye candy and Unicode/ClearType support.
- Stability - templates are well established and the runtime is solid
- Potentially smaller installs - no need for the .NET runtime

Can a Clarion# class inherit from a C# class?

Definitely. And vice versa. The same goes for all .NET languages.

What are .NET's minimum requirements?

According to Microsoft, the minimum requirements for the .NET 2.0 redistributable are:

- 400 Mhz processor (800 Mhz recommended)
- 96-128 Mb memory (256 or better recommended)
- 280 Mb hard disk space (610 for 64 bit), 1 gig recommended
- 800x600 256 color, 1024x768 high color recommended

As with most Microsoft platform requirements, you're probably not going to be very happy at the low end of the spectrum. I suggest you take the "recommended" values as the minimum values.

Supported x86-based operating systems:

- Microsoft Windows 98
- Microsoft Windows 98 Second Edition
- Microsoft Windows 2000 Professional with SP4
- Microsoft Windows 2000 Server with SP4
- Microsoft Windows 2000 Advanced Server with SP4
- Microsoft Windows 2000 Datacenter Server with SP4
- Microsoft Windows XP Professional with SP2
- Microsoft Windows XP Home Edition with SP2
- Microsoft Windows XP Media Center Edition 2002 with SP2
- Microsoft Windows XP Media Center Edition 2004 with SP2
- Microsoft Windows XP Media Center Edition 2005
- Microsoft Windows XP Tablet PC Edition with SP2
- Microsoft Windows XP Starter Edition
- Microsoft Windows Millennium Edition
- Microsoft Windows Server 2003 Standard Edition
- Microsoft Windows Server 2003 Enterprise Edition
- Microsoft Windows Server 2003 Datacenter Edition Microsoft Windows Server 2003 Web Edition

x64-bit based systems

- Microsoft Windows XP Professional x64 Edition
- Microsoft Windows Server 2003, Standard x64 Edition
- Microsoft Windows Server 2003, Enterprise x64 Edition
- Microsoft Windows Server 2003, Datacenter x64 Edition

Itanium-based systems

- Microsoft Windows Server 2003 with SP1, Enterprise Edition for Itanium-based Systems
- Microsoft Windows Server 2003 with SP1, Datacenter Edition for Itanium-based Systems

Should I be learning C# or VB.NET, or some other .NET language?

Although Clarion is a full-fledged .NET language, it probably will be to your advantage to learn at least one other .NET language. There's a wealth of .NET programming information out there, and the vast majority of books and articles deal with either C# or VB.NET. So which language should you learn?

VB.NET in general has more Clarion-like syntax; there are certainly differences, but VB.NET is more of a "plain English" programming language. C# on the other hand has C-like syntax which isn't to everyone's liking. It's also easier to make non-obvious mistakes with C#. On the other hand, C# is the .NET reference language, so you can expect it to support all the latest .NET features, and it's generally a better source of programming examples.

If you have C, C++, or Java experience, choose C#. If you've never worked in a language with C-like syntax then you'll probably be better off with VB.NET. Carl Barnes recommends Programming VB .NET: A Guide For Experienced Programmers, which is also available as a free download - look for the Free eBook Download link on that page.

Additional reading

- Clarion Magazine articles on Clarion.NET
- All Clarion Magazine articles related to .NET
- Clarion Magazine articles on mixing Clarion 6 and .NET
- How to subscribe to Clarion Magazine
- CapeSoft's Clarion.NET FAQ

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

Reader Comments

Posted on Saturday, November 17, 2007 by Wolfgang Orth

If both new Clarions deliver desktop applications - why should I buy both? Would Clarion# not be enough? It seems to be the most evolved as it covers Desktop, somehow Web-applictions and mobile devices. So the traditional Clarion 7 with its WinAPI-programming is oldfashioned history from now on, even before it got gold? Or am I wrong on this?

.....

Posted on Sunday, November 18, 2007 by Dave Harms

Thanks Wolfgang - see answer above.

Dave

.....

Posted on Wednesday, November 21, 2007 by Carl Barnes

A concern I don't think I see addressed in the FAQ are the system requirements for the End User computer for a Clarion# application.

A "classic" Clarion Win32 app will typically run on Windows 98 or newer. Probably even Windows 95. No patches are needed. Nothing to download. 3rd party tools can change this.

A Clarion# .Net app requires the 2.0 .Net Framework. That is more picky about systems. Here's a list I found:
Windows 2000 SP3; Windows 98; Windows 98 SE; Windows ME; Windows Server 2003; Windows Vista; Windows XP SP2

Note that XP users must have SP2. I don't see NT 4 or 95.

Some (or many) users will have to download and install 2.0. That's a 23MB file that takes a 280MB of diskspace. I wonder if the 23MB is just the installer and it downloads more.

In summary if you are trying to reach the maximum users, that may be running older hardware, the .Net way might cost you a few.

Posted on Friday, November 23, 2007 by Dave Harms

Thanks Carl - I've updated the FAQ.

Add a comment

Clarion Magazine

A StringClass For Clarion

by Rick Martin

Published 2007-11-16

In the last year or so I have done a quite a bit of development work in Python and C#. One of the nice features of these languages is a built-in string class that supports creation and manipulation of strings of arbitrary length. While Clarion obviously supports references to dynamically allocated strings, it clutters your code to have memory allocation and de-allocation in with your business logic; besides, it can be tedious to have to write similar code over and over.

Well, we all know (or should know) the answer to that problem: write a class.

I found myself dealing with a number of situations where I was calling procedures, many of which were out of my control, which returned a string where I didn't know the length of the return value in advance. This led me to create StringClass. I don't know about you, but I hate to declare a string larger than I think will ever be needed to handle an unknown value. First, I generally allocate way more memory than I normally need. Second, if the return value ever exceeds the length I guessed then I'm in trouble.

So I had a few goals in mind when I created StringClass.

1. Avoid declaring fixed length variables for unknown length strings.
2. Eliminate repetitive code to allocate and de-allocate strings that clutters source code logic.
3. Eliminate repetitive code for handling certain string operations.

An example

Let's look at an example. Say I want to loop through the records of a table and create a comma delimited string of all the company names for customers in Florida. I have no idea how many records are in the customer table, nor how many of their addresses are in Florida. If I don't want to take a wild guess and declare a big string to hold the customer names, I need to dynamically allocate the return string as I find each customer.

```
ManualProcessData Procedure(STRING pState)
CustomerString &String
TempString  &String
  CODE
    SHARE(Customer)
    SET(Customer)
    LOOP
      NEXT(Customer)
      IF ERRORCODE()
        BREAK
      END
```

```
    IF CUS:State = pState
      IF CustomerString &= NULL   !This is the first one
        CustomerString &= NEW(STRING(LEN(CLIP(CUS:Company))))
        CustomerString = CLIP(CUS:Company)
      ELSE
        !Some customers for the state have already been found
        TempString &= NEW(STRING(LEN(CustomerString)))
        TempString = CustomerString
        DISPOSE(CustomerString)
        CustomerString &= |
          NEW(STRING(LEN(TempString & ',' & CLIP(CUS:Company))))
        CustomerString = TempString & ',' & CLIP(CUS:Company)
        DISPOSE(TempString)
      END
    END
  END
  CLOSE(Customer)
  MESSAGE(CustomerString)
  DISPOSE(CustomerString)
```

This example has another problem. The string is allocated memory in the procedure, and every NEW must have a DISPOSE or you have a memory leak. But that would mean disposing of CustomerString before the procedure ends, which would prevent you from returning the string! One way around this problem is to pass a reference to a string to the procedure and use the reference in place of CustomerString. I am not fond of this solution because the memory is allocated in one procedure and must be de-allocated in another. This is just begging for a memory leak.

Now this example is a little extreme. The code required to handle the memory is far larger than the logic of concatenating the company names, but it shows that it is a lot of work to handle the string length dynamically vs. just allocating a string of 2000 characters (or whatever) and hoping the result string is never longer than 2000.

Now let's look at the same procedure using StringClass.

```
TestClass.ProcessData   Procedure(STRING pState)
CustInState StringClass
  Code
  Share(Customer)
  SET(CUS:KeyCustNumber)
  LOOP
    NEXT(Customer)
    IF ErrorCode()
      Break
    END
    IF CUS:State = pState
      CustInState.Append(Choose(CustInState.Length()=0,'','')|
        &CLIP(CUS:Company))
    END
```

```
END
Close(Customer)
MESSAGE('Inside ProcessData|'& CustInState.Get())
Return CustInState.Get()
```

There are a lot fewer lines of code needed to build the string using the StringClass. The code is also much easier to read and you don't have to recreate the memory logic the next time you need to do something similar. Another bonus is you can return the string (more accurately a copy of the string, assuming the return data is a value type, not a reference type) from the procedure; the class destructor contains code to de-allocate the string memory when the class is terminated, which happens *after* the procedure in which it is declared ends.

Now that the ball is starting to roll let's add other features to make processing strings easier. Often I need to take a string that has some kind of delimiter, like comma or carriage return/line feed pairs, and process each entry. The StringClass. Split method lets you do this.

```
ResultsString.Assign('This is a comma,delimited,String of undetermined length')
ResultsString.Split(',')
LOOP I# = 1 TO ResultsString.Records()
  MESSAGE(ResultsString.GetLine(I#))
END
```

Internally, StringClass breaks up the passed string using the delimiter characters and builds a queue of sub-strings. The memory for this queue is automatically de-allocated when the class destructs or when Split is called again.

StringClass is rounded out with a number of methods to handle many string functions:

- Replace - Replace the first sub-string with the second sub-string. An optional parameter allows you to replace all occurrences or to specify how many occurrences to replace.
- Contains - Return the offset or zero of the sub-string in the current StringClass value.
- Count - Returns how many times a sub-string occurs in the current StringClass value.
- SubString - Return the sub-string specified by the start and stop positions passed.
- Length - Return the length of the current string value.

While you can write straight Clarion code for all of these operations, it is better to use class methods rather than re-write the code over and over.

ANY and BSTRING

There was a thread on the Clarion newsgroups recently on dynamic length strings. It was suggested that you can use either an ANY variable or the undocumented BSTRING data type to work with strings of dynamic length. Both of these data types allow assign and reassigning string values so you can write code like this:

```
ANYString   ANY
   ANYString = 'This is the first value added to the string'
   ANYString = ANYString & ' and now we are adding addititional text'
   ANYString = '1234'
Bstr    BSTRING
```

```
Bstr = 'This is the first value added to the string'
Bstr = Bstr & ' and now we are adding addititional text'
Bstr = '1234'
```

After reading this I started to think that in many situations I could replace the StringClass with ANY or BSTRING. You can argue whether or not it is a good idea to use an undocumented feature of the language. The BSTRING datatype is used in a number of the ABC classes and has been around for years, so I don't think it will be removed anytime soon; it should be pretty safe to use. There is another advantage to using these datatypes vs. string references variables; you can return them as a STRING from a procedure or method call. This eliminates the issue of returning the dynamically allocated string in the example above.

Is my StringClass in danger of elimination? Well, maybe not quite.

Performance tests

I decided to run some performance tests. There are a couple methods in the included source that I used for performance testing. They take a filename as a parameter and read the file using the DOS driver in 50,000 byte chunks and build a string with the contents of the file. I ran the test using BSTRING, ANY, and the StringClass as the destination variable. Here's what I found (all times are in seconds):

File size (bytes)	StringClass	BString	ANY
3,742,102	1.67	1.38	2.63
79,409,790	845.51	695.07	1398.16

Clearly you do not want to use an ANY variable for this operation. The overhead involved with the ANY datatype made it significantly slower that either of the other two. However, the BSTRING solution was actually faster than using StringClass by about 21%. That's probably not too significant unless you are dealing with vary large strings or thousands of assignments. For most situations StringClass is close enough to BSTRING's performance.

My final argument for continuing to use the StringClass is encapsulation. StringClass hides the details of the actual data declaration for the string. In fact, converting the class to use BSTRING or ANY would be easy and none of the existing code using StringClass would be affected. Also, if I dropped StringClass and converted my code to use BSTRING I would once again be duplicating the string manipulation logic everywhere it was used. The class lets me hide that complexity behind method calls.

I use StringClass all the time, even when I'm not necessarily dealing with unknown string lengths. It's nice to use a consistent interface when dealing with strings, and I'm continuing to add methods as I come across new problems to solve.

Examples

I have included both a Clarion 6 project file and a Clarion 7 solution so you can run the example in either environment. I have also included a template to add support for the StringClass to ABC or Clarion template chain APP files. If you are going to use the class and template in an ABC APP project then you need to put the class INC and CLW files in the LibSrc folder and add the template to any APP that will use StringClass, as well as to the data DLL for an ABC-based multi-DLL project.

The Build Results from Data test which creates a comma delimited string of all customers for a particular state requires the CUSTOMER.TPS file from the DLLTutor example.

Download the source

Reader Comments

Posted on Saturday, November 24, 2007 by Paul Howard

Thanks for the excellent and useful class.

I often get field delimiters confused with field separators.

I remind myself that the comma "separates" two string values rather than "delimiting" either. That is why there is no comma at the end of the string before the end of line CRLF. If there were, then a null value should be returned for the last field and the field count would include that null field.

A text file containing comma separated (optionally quote delimited) string values is AFAIK called Comma Separated Value {CSV} or Standard Data Format (SDF) or BASIC Data Format rather than Comma Delimited Value.

When dealing with strings containing separators (,), the separator within a delimited string value ("string,string") needs to be handled.

"john,mary","smith","New York",12345,"5124,458"

and the field delimiters (") could be optionally removed before returning the value.

These are the fun parts in parsing strings, and would be a welcome addition to this class. <g>

Hope this helps,
Paul

Add a comment

Clarion Magazine

PostgreSQL Revisited: Managing User Rights

by Dave Harms

Published 2007-11-14

In the first article in this series on PostgreSQL I showed how to install the server on Windows (Vista, in my case), and in the second I covered some of the tools and utilities included with PostgreSQL. There's just one more topic to cover before getting into creating and using databases, and that's security.

If you're accustomed to TPS or other flat files, you may be wondering at this point if SQL is really worth the trouble. First you have to install the server, then you have to figure out how to administer the server, and now you have to deal with security? Why does everything have to be such a big deal in SQL?

The short answer is that SQL databases are more complex to administer because they're more than just simple flat file data repositories because SQL databases contain data *and* code. You can update, delete and even insert massive amounts of data with a single command, or you alter the structure of the database itself. That level of power introduces some significant risk that things will go wrong. And to raise the stakes higher, applications communicate with the SQL servers using English-like syntax. That allows users the benefit of free-form queries, but if you let users (or application developers) execute their own SQL statements against the database you make it easier for an incompetent or malicious person to wreak havoc.

Authentication and authorization

In PostgreSQL's security, as with almost any security system, there are two important issues: authentication and authorization. Authentication is identifying the user; authorization is determining what the user is permitted to do. Authentication typically involves the user entering an id and password, although it's also possible to authenticate solely by IP address or dispense with authentication entirely.

Before you can be either authenticated or authorized, however, you have to get connected to the database server.

The pg_hba.conf file

PostgreSQL's pg_hba.conf configuration file has several important security-related functions. It controls the nature of the client's connection to the database server (encrypted vs unencrypted), the source of the connection (by IP address or range of addresses), the destination database(s), and the allowed user names. It also determines the authentication method that will be used. (It's important to note that pg_hba.conf's job is done once you're connected to the database; it doesn't care what you do once you're there. For control over specific database actions see the discussion of users, roles and privileges, below.)

Typically you'll find pg_hba.conf in the data directory. On my machine that directory is C:\Program Files\PostgreSQL\8.2\data.

After a bit of introductory commentary, the file ends with the following information:

```
# Put your actual configuration here
# ----------------------------------
#
# If you want to allow non-local connections, you need to add more
```

```
# "host" records. In that case you will also need to make PostgreSQL listen
# on a non-local interface via the listen_addresses configuration parameter,
# or via the -i or -h command line switches.
#



# TYPE  DATABASE    USER       CIDR-ADDRESS         METHOD

# IPv4 local connections:
host    all         all        127.0.0.1/32         md5
# IPv6 local connections:
host    all         all        ::1/128              md5
```

As you can see, there are two lines of text (called access records) in the example which specify allowable connections, one for Internet Protocol version 4 (the one most of us use), and the other for IPv6, the new standard which is needed because we're running out of IPv4 addresses. You will need at least one access record, and the format of each record can be any of the following:

```
# local      DATABASE  USER  METHOD  [OPTION]
# host       DATABASE  USER  CIDR-ADDRESS  METHOD  [OPTION]
# hostssl    DATABASE  USER  CIDR-ADDRESS  METHOD  [OPTION]
# hostnossl  DATABASE  USER  CIDR-ADDRESS  METHOD  [OPTION]
```

The first field is the connection type, and can be one of the following:

- local - unix socket
- host - any TCP/IP socket, plain or encrypted
- hostssl - encrypted TCP/IP socket)
- hostnossl - unencrypted TCP/IP socket

On a Windows machine you'll use one of the host settings; local is only for *nix systems.

The second field is the database, which can be any of the following:

- all - all databases
- sameuser - allowed access to a database with the same name as the user's login name
- samerole - allowed database with the same name as the user's role (more about roles later)
- a specific database, or a comma-delimited list if more than one
- if prefixed by @, the name of a file containing a list of databases

The third field is the user name, which can be any of the following:

- all - any user
- a specific user name
- if prefixed by @, the name of a file containing a list of user names
- if prefixed by +, a user group name

The fourth field is the IPv4 or IPv6 address or addresses from which connections will be accepted, in the form of the

address and a netmask. For a single IP address use a netmask of /32, or /128 if you're on IPv6. This field is omitted if the connection type is local.

The fifth field is the authentication method. Options include:

- trust - no authentication is needed from the address or IP range specified, and the user can log in as any user without using a password
- reject - never authenticate from the address or IP range specified
- md5 - require an md5 encrypted password
- crypt - require a crypt()-encrypted password. This is for compatibility with older clients; md5 is now the preferred method.
- password - send an unencrypted password (not recommended)
- krb5 - use Kerberos V5 (only available on TCP/IP connections)
- ident - use the client's ident server
- ldap - use LDAP authentication
- pam - use Pluggable Authentication Modules (on Linux/Solaris)

When you install PostgreSQL, local access to all databases is allowed by default. If you've installed on a separate server you'll need to create an appropriate access record so you can connect to the database.

Users, roles and privileges

The first time you connect to PostgreSQL you do so using the superuser name (default is postgres) and password you specified during the installation process. Keep in mind that this is not the same login required by the PostgreSQL server process, which on Windows runs as a service. It's easy to get them confused as both default to postgres. One is a Windows system login, the other is a PostgreSQL server login, and they should have different passwords.

If you're running a test server you might be tempted to stick with just the postgres user, and particularly if it's a local server on your development machine that isn't a big risk. After all, if you lose the password you can always reconfigure pg_hba.conf to allow trusted connections from localhost. Log in, change the password, restore pg_hba.conf, and you're off to the races.

On the other hand, if your server is a bit more exposed you probably don't want to user the superuser password any more than necessary. Set up another user with the necessary rights and use that login for your development work. You may also want to create other logins for testing, mimicking the rights you'll be assigning your users.

Creating a user or role

There are several ways to create a PostgreSQL user. One is to issue a CREATE USER statement from your SQL client (such as psql); you can also use the CreateUser command line utility (which is a wrapper for CREATE USER) or a GUI like PgAdmin III.

Actually, CREATE USER is just an alias for CREATE ROLE. And that's a source of potential confusion, because users and roles in PostgreSQL are pretty much identical.

The concept of users and roles (sometimes called groups) is common to security systems. The idea is that while you may have any number of users, you will probably have fewer unique combinations of rights. In a SQL database, you might assign one kind of user select-only permission, another select and insert for only some tables, and a third kind may get full access. If you have dozens or hundreds of users it's a lot of trouble to have to grant the same rights to each individual; it makes more sense to package up each unique set of rights and then link each user to the appropriate set (or sets).

In PostgreSQL you create a set of rights by creating a role and giving the role the necessary rights. Then you assign users to roles (and a user may belong to more than one role).

What's confusing is that in PostgreSQL, roles and users are pretty much identical; you can assign rights to a user directly, just as you assign rights to a role. And CREATE USER and CREATE ROLE are actually the same command. The

only difference is a user has the ability to log in; a role cannot be used to log in.

Here's the syntax for CREATE ROLE, which is exactly the same as CREATE USER:

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]
where option can be:

    SUPERUSER | NOSUPERUSER
  | CREATEDB | NOCREATEDB
  | CREATEROLE | NOCREATEROLE
  | CREATEUSER | NOCREATEUSER
  | INHERIT | NOINHERIT
  | LOGIN | NOLOGIN
  | CONNECTION LIMIT connlimit
  | [ ENCRYPTED | UNENCRYPTED ] PASSWORD 'password'
  | VALID UNTIL 'timestamp'
  | IN ROLE rolename [, ...]
  | IN GROUP rolename [, ...]
  | ROLE rolename [, ...]
  | ADMIN rolename [, ...]
  | USER rolename [, ...]
  | SYSID uid
```

So if CREATE ROLE and CREATE USER are the same command, why have both? It all comes down to the LOGIN/NOLOGIN option. The default value for CREATE ROLE is NOLOGIN, and the default for CREATE USER is LOGIN. You can create a user with CREATE ROLE by specifying LOGIN, and you can create a role with CREATE USER by specifying NOLOGIN.

Assigning permissions

You assign permissions to users/roles with the GRANT statement, which can be used a number of ways. Here's the syntax for granting privileges on a database:

```
GRANT { { CREATE | CONNECT | TEMPORARY | TEMP } [,...] | ALL [ PRIVILEGES ] }
   ON DATABASE dbname [, ...]
   TO { username | GROUP groupname | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

And here's the syntax for granting privileges on a table:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER }
   [,...] | ALL [ PRIVILEGES ] }
   ON [ TABLE ] tablename [, ...]
   TO { username | GROUP groupname | PUBLIC } [, ...] [ WITH GRANT OPTION ]
```

You can also grant privileges on sequences (which are a little bit like autonumber keys), functions, languages, schemas, tablespaces and triggers. And you have the option of passing along the ability to grant the same permissions you've just granted, which is something you need to think about carefully. For instance, just because you grant a user the right to create a specific table, that doesn't always mean they should be able to pass along that same right to anyone else.

The word PUBLIC has special meaning - granting a right to PUBLIC is the same as granting that right to each and every user.

One thing missing here is column level permissions. In some SQL databases, including MySQL and MS SQL, you can control a user's access to specific columns as well as to specific tables. In PostgreSQL you can create views if you want to hide certain columns, but these views are read-only. There is a workaround, as noted in the help for CREATE VIEW:

> You can get the effect of an updatable view by creating rules that rewrite inserts, etc. on the view into appropriate actions on other tables. For more information see CREATE RULE.

PostgreSQL rules go far beyond the capabilities of triggers and stored procedures; they basically let you rewrite queries before they are executed. They're one of those dang cool things I'll probably never need to use. PostgreSQL is littered with them.

Granting existing roles to existing users

You can also associate an existing user with an existing role using the GRANT ROLE statement:

```
GRANT role [, ...]
  TO username [, ...]
  [ WITH ADMIN OPTION ]
```

Keep in mind that users and roles are synonymous, so you can use this syntax to granting one person's rights to another, as well as granting role rights. But you'd probably be best off sticking to the concept of using roles for groups of rights, and assigning users to one or more roles.

Revoking rights

Just as you can grant rights, so can you revoke rights. The syntax is almost identical to GRANT. Here's the syntax for revoking table rights:

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | REFERENCES | TRIGGER }
    [,...] | ALL [ PRIVILEGES ] }
    ON [ TABLE ] tablename [, ...]
    FROM { username | GROUP groupname | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Note that you can cascade a grant revocation. If you issue GRANT with the option to let the user grant those rights to other users, a REVOKE...CASCADE will revoke all of the specified rights granted down the chain. However, if a user has rights also granted by another user those rights will not be revoked. The default value is RESTRICT - in other words, not to cascade grant revocation.

Reviewing rights

You can list the current rights (at least those visible to you) in the psql command line client by issuing a \dp or a \z command. Consider an example from the PostgreSQL documentation. Let's say you have a user named miriam who creates a table called myTable, and then issues the following GRANT statements:

```
GRANT SELECT ON mytable TO PUBLIC;
GRANT SELECT, UPDATE, INSERT ON mytable TO GROUP todos;
```

The rights output looks like this:

```
lusitania=> \dp mytable
        Access privileges for database "lusitania"
 Schema |  Table  |           Access privileges
--------+---------+---------------------------------------
 public | mytable | {=r,miriam=arwdRxt,"group todos=arw"}
(1 row)
```

There are three sets of access modifiers. The first one, =r, indicates the public has read access (i.e. can issue SELECT statements). The user miriam, as the creator of the table, has full access, and the group todos has add, read, and write access. Here's the complete list of rights identifiers:

```
          =xxxx -- privileges granted to PUBLIC
       uname=xxxx -- privileges granted to a user
  group gname=xxxx -- privileges granted to a group

          r -- SELECT ("read")
          w -- UPDATE ("write")
          a -- INSERT ("append")
          d -- DELETE
          R -- RULE
          x -- REFERENCES
          t -- TRIGGER
          X -- EXECUTE
          U -- USAGE
          C -- CREATE
          T -- TEMPORARY
     arwdRxt -- ALL PRIVILEGES (for tables)
```

Rights and GUIs

There are a few administrative tools out there that will let you manage rights without having to resort to the command line. PGAdmin III, which is included with PostgreSQL, has a usable grant wizard, although it takes some getting used to. I'll have more to say about the grant wizard in the next installment.

Summary

SQL databases can seem overwhelming at times. So far I've covered installation, administrative tools, user authentication and authorization. That's a lot of detail to absorb just to get ready to create a database.

Fortunately, the fun stuff is about to begin. Next time I'll show how to convert a TPS application to PostgreSQL.

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

Reader Comments

Add a comment

Clarion Magazine

The ClarionMag Blog

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

XML All blog entries
XML All new items, including blogs

Blog Categories

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

Clarion.NET beta impresses

Direct link

Posted Saturday, November 17, 2007 by Dave Harms

SoftVelocity released the long-awaited beta of Clarion.NET today. It's been a looooooong wait for this baby, and I'm very happy to have it installed on my computer at last. As expected, this release does not include the AppGen (that will come later) but it does include a C6 template utility that lets you generate a functioning browse/form application from the data dictionary of your choosing. You load up an app in C6, run the utility, and then look for the Clarion.NET solution in a newly created subdirectory under your app directory.

The Clarion# code that utility templates generates is instructive on a few levels, and I'll get back to that in a minute.

This beta isn't just about desktop (that is, WinForms) applications. You can also use Clarion.NET to create compact framework (mobile) apps, as well as ASP.NET web applications. Basic examples of both are provided, along with a bunch of other stuff. See the just-updated Clarion.NET FAQ for more. And Clarion.NET is a multi-language IDE, which makes sense since it's a marriage of SV's IDE code and the SharpDevelop IDE. So you can write apps in C# and VB.NET as well, and you can mix languages within a solution.

While the lack of templates/AppGen means large scale development is a ways off for most of us, there is a whole lot you can do right now with Clarion.NET.

Now, about that source code. It isn't ABC, that's for certain. There are a lot of classes in the namespace Clarion.Windows.Forms, but there are also some familiar constructs like views, and source code that manipulates views for display in list

controls, and stuff that otherwise looks like Clarion code. And there are event handlers that functionally are pretty similar to virtual methods that contain ABC embed points. In other words, while the code isn't ABC, it's not entirely unfamiliar either, and it's not that hard to see how the concepts behind the ABC template set can be adapted to this new Clarion#/WinForms code.

I hope some of that makes sense - I'll have more to say in upcoming mag articles when I've had a chance to absorb all this a little better.

Clarion.NET Runtime 100% Verifiable

Direct link

Posted Tuesday, November 13, 2007 by Dave Harms

You've probably noticed the new SV web site, complete with information on Clarion.NET. One item in particular jumped out at me:

> The Clarion Runtime Library has been ported to 100% verifiable .Net code

That may not matter to most Clarion.NET developers. After all, there's a good chance there's still some Win32 code in the file driver system (although I could be wrong about that). But a 100% .NET runtime is great news for anyone considering writing Clarion.NET applications for Mono, since any dependence on Win32 code would be a huge problem on a non-Windows platform.
