

Clarion Magazine

Reborn Free

CLARION online

PDF For April, 2004

All Clarion Magazine articles for April, 2004 in PDF format.

Posted Tuesday, May 04, 2004

Two More Clarion Books

Coming!

We have two new softcover titles in production, and we anticipate taking pre-publication orders beginning Tuesday, May 17, for delivery in June. These books are: Clarion Databases & SQL, a collection of articles from Clarion Magazine on the subject of database design and deployment, and a new Clarion Magazine edition of Programming Objects in Clarion, Russ Eggen's highly-acclaimed book on understanding OOP and ABC.

Posted Tuesday, May 11, 2004

Understanding Clarion Templates Part 4: Writing Code Templates

David Harms continues his series on learning the template language with an introduction to writing code templates.

Posted Tuesday, May 11, 2004

Detecting Drive Types

Rumors to the contrary, SneakerNet is not dead! Sometimes you still need to copy data to physical, removable

Articles: 

News: 



News

[Clarion Third Party Profile Exchange Updated](#)

[etc2004 Third Party Handout Insert Deadline](#)

[SendTo - A New Capesoft Product](#)

[Hecticday Schedule For Lodestar & DeveloperPLUS](#)

[ADDA 1.0.3](#)

[BST Ver 3.5 Booking Template](#)

[Clarion 6.1 EA-2 Released](#)

[Free Clarion Application Icons](#)

[BST Booking Demo](#)

[Solid Software Back To Business](#)

[BST Marketing Survey](#)

[Crystal Clear Class 2.0](#)

[EasyExcel 3.05](#)

[PiFolio Word Reporter 4 For Clarion](#)

[Support For RADIntelliSense With Andy Ireland's COM Generator?](#)

[xWhatsNew Class 1.5](#)

SURVEY

How do you use XML with Clarion? (check all that apply)

- I don't do any XML  75.9%
- I read XML files  10.5%
- I write XML files  11.3%
- I use the SV XML templates  1.5%
- I use the Clarion XML classes directly  3.8%
- I use the CenterPoint wrapper classes  0.8%
- I use COM and MS Parsers  3.8%
- I use my own custom XML code  10.5%
- I use another solution not listed  6.0%

133 responses

[Previous Surveys](#)

media. As Andrew Guidroz shows, there are some API calls that can help.

Posted Friday, May 14, 2004

Cryptography and Clarion: Using the MSCrypto API

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Increasingly, cryptography is important to software developers, and with the US government having lifted restrictions on the export of strong encryption, a solution is now widely available to Windows developers. In this two-part article Ron Webb explains what cryptography is all about, and introduces a Clarion class wrapper for the MS Crypto API.

Posted Friday, May 21, 2004

Understanding Clarion Templates Part 5: Enhancing Code Templates

In this installment on code templates, David Harms shows how to properly display text in a template, and conditionally generate code.

Posted Friday, May 21, 2004

Clarion Magazine's Book/Subscription Sale Ends June 11

Clarion Magazine is having its biggest subscription sale ever! Not only have we dropped our prices for this **limited time offer**, but we're including *all* the

[etc2004, The Last etc and
The Biggest!](#)

[BRT Registration Tamer
Half Off Competitive
Upgrade](#)

[PlugIT 2.0 Documentation
Free Download](#)

[gReg Site Temporarily
Down](#)

[Hosting Service For
Clarion Developers](#)

[New Buttons in C6.1](#)

[Gitano Source Code Sale](#)

[Tracker Software Support
Email](#)

[BRT Registration Tamer
From Comsoft7](#)

[Free Font Change
Template](#)

[etc2004 Four Weeks Away](#)

[Jaguar: Common
Programming Tasks](#)

[Gitano Software Email
Outage](#)

[ABC Free Templates And
Tools Updated](#)

[Jaguar Java Source Code](#)

[Web Email Cloaking Utility
Update](#)

[ImageEx 3 Delayed](#)

[WEP Key Generator
Update](#)

[PlugIT Update](#)

[etc2004 Room Blocks
Disappearing Fast](#)

[etc2004 Third Party
Handout Inserts](#)

[1st Icon Design Special](#)

[Special Fenix Transition](#)

One Year Ago In CM

[Creating A
Designer
Interface In
Clarion \(Part 3\)](#)

[Weekly PDF For
June 1-7, 2003](#)

[The ClarionMag
Third Party
Product
Directory](#)

Two Years Ago In CM

[Positioning List
Box Totals](#)

[ETC III: SQL
Replication -
Shawn Mason](#)

[ETC III:
ClarioNET and
More - Bob
Foreman](#)

Three Years Ago In CM

[Using Dynamic
Indexes With
TPS Files](#)

[Understanding
Stack And Heap
Memory In 32
Bit Clarion
Applications](#)

back issues in every annual subscription/renewal. If you're a current subscriber, it's still a great rate. If you've never subscribed, this is by far the best deal we've ever offered! \$99.95 NEW SUB, \$69.95 RENEWAL. BACK ISSUES INCLUDED in all one year subscriptions/renewals! Sale ends June 11, 2004.

Posted Monday, May 24, 2004

Cryptography and Clarion: Using the MSCrypto API Part 2

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Increasingly, cryptography is important to software developers, and with the US government having lifted restrictions on the export of strong encryption, a solution is now widely available to Windows developers. In this two-part article Ron Webb explains what cryptography is all about, and introduces a Clarion class wrapper for the MS Crypto API. Part 2 of 2.

Posted Wednesday, May 26, 2004

Migrate Your Topspeed Application To Firebird, Part 1

Faced with a conversion from Topspeed to Firebird, Jimmy Rogers wrote an extensive template to automate the process as much as possible. Part 1 of 2.

Posted Thursday, May 27, 2004

[Offer](#)

[New Native OutlookBar](#)

[JAGUAR Samples](#)

[BoxSoft Super Security
6.05](#)

[Search the news
archive](#)

[Clarion News -
June 2001](#)

**Four Years
Ago In CM**

[ETC Wrapup:
Sights And
Sounds](#)

[ETC
Presentation
Summaries:
Bruce Johnson,
Andy Stapleton](#)

[ETC
Presentation
Summary:
James Fortune](#)

Looking for more? Check out the [site index](#), or [search the back issues](#).

This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

[Clarion Magazine](#)

Reborn Free

CLARION
online

Clarion Books

Clarion Magazine now has books in print! We'll be adding more titles in the near future, so to keep up to date be sure to get on our **mailing list!** If you **already have a free membership or a subscription**, put yourself on the list via the [member update form](#). If you're **not yet registered** with Clarion Magazine, join use the [new member form!](#) There's no charge. On either form, select the **Books & Specials** mailing list. This is a double opt-in list, and all emails that go out to you once you've signed up will have an unsubscribe link.

Coming soon!

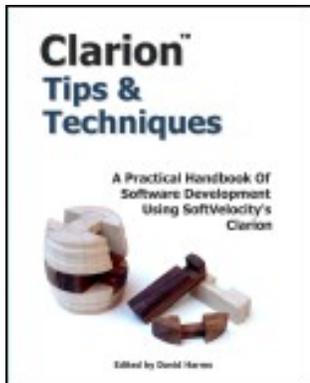
We have two new softcover titles in production, and both are expected to ship in June. You can pre-purchase these books now and save! This offer will end without notice.

- **[Clarion Databases & SQL](#) - Pre-purchase at [SAVE \\$20](#)**
Selected articles from Clarion Magazine on the subject of database design and deployment for both flat file (i.e. TPS) and SQL database systems. Approx 600 pages.
- **[Programming Objects in Clarion](#)**, by Russ Eggen - [SAVE \\$15](#)
Russ Eggen's highly-acclaimed book on understanding OOP and ABC will soon be available in a new Clarion Magazine edition, with index.

Tips Book On Sale!

Our very popular [Clarion Tips & Techniques](#) book is also on sale at **\$15 off** the regular price, and we also have a phenomenal subscription special happening. Subscriptions and renewals now include all the back issues, PLUS we're offering discounts off the usual rates! [Subscribe/renew now!](#)

Books in print



Clarion Tips and Techniques

Edited by David Harms

List price: US \$89.95

This 630 page softcover book contains selected articles from Clarion Magazine's first five years in publication, as well as a small number of articles originally published in Clarion Online. These articles were written for Clarion 5.0 and 5.5, but many are just as applicable to Clarion 6. CoveComm Inc. ISBN: 0-9689553-1-2

[See more book details](#)

[Buy the book now!](#)

Why is Clarion Magazine printing books?

Clarion Magazine, the online publication, is very popular with Clarion developers, and we're proud of the massive library of information we've published in HTML and PDF form. So why are we now printing books? Because as handy as HTML and PDF documents are, for many of us they still aren't as convenient and, well, enjoyable to read as a printed, bound book.

We are continuing to convert Clarion Magazine's HTML pages to book form. There is a stunning amount of material to choose from; in a 7.5" x 9.25" book format, the Clarion Magazine web site contains some 6500 pages of Clarion articles. That's about two feet of shelf space! If there is sufficient interest we will consider publishing all of this material, but it would be impossible to bring it all to press in a short period of time. As a result, we're focusing our efforts on areas likely to be of most immediate interest to Clarion developers. Stay tuned for more book news! And if you have any questions just send us an [email](#).

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

[Clarion Magazine](#)



Understanding Clarion Templates Part 4: Writing Code Templates

by **David Harms**

Published 2004-05-11

As you can see from the [previous article](#) in this series, the *concept* of code templates is quite straightforward. Fortunately, *writing* code templates can be equally straightforward.

One of the most common uses for code templates is to automate the writing of repetitious or potentially confusing code. In this article I'll show how you can use a code template to simplify the Clarion MESSAGE function.

The source code

Typically, your starting point for writing a code template will be some existing, and hopefully working, source code. In the case of the MESSAGE function, the large number of optional parameters mean that the function's usage can vary from the simple to the complex. Here's a very simple usage:

```
MESSAGE('This is a message!')
```

And here's something a little more complex:

```
IF (MESSAGE('Are you sure you want to order that take-out pizza?' |  
  , 'Food Delivery Decision Point', ICON:Question, |
```

```

    Button:Yes+Button:No,Button:Yes,1)) = BUTTON:Yes
    ! Place the order
ELSE
    ! Cancel the order
END

```

The Help for MESSAGE lists the prototype and parameters as follows:

```

MESSAGE( text [,caption] [,icon] [,buttons] [,default] [,style]
)

```

text A string constant or variable containing the text to display in the message box. A vertical bar (|) in the text indicates a line break for multi-line messages. Including '<9>' in the text inserts a tab for text alignment.

caption The dialog box title. If omitted, the dialog has no title.

icon A string constant or variable naming the .ICO file to display, or an EQUATE for one of Windows' standard icons (these EQUATES are listed in EQUATES.CLW). If omitted, no icon is displayed on the dialog box.

buttons Either an integer expression which indicates which Windows standard buttons (may indicate multiple buttons) to place on the dialog box, or a string expression containing a vertical bar (|) delimited list of the text for up to 8 buttons. If omitted, the dialog displays an Ok button.

default An integer constant, variable, EQUATE, or expression which indicates the default button on the dialog box. If omitted, the first button is the default.

style The style parameter is a bitmap integer constant, variable, EQUATE, or expression that specifies the type of modal behavior, and whether or not the text of the message can be copied to the Windows Clipboard.

As you can see, only the first parameter, the text to display, is a requirement for the MESSAGE function. And two of the parameters, icon and buttons, have standard equates available, as follows:

```
BUTTON:OK
BUTTON:YES
BUTTON:NO
BUTTON:ABORT
BUTTON:RETRY
BUTTON:IGNORE
BUTTON:CANCEL
BUTTON:HELP

ICON:None
ICON:Application
ICON:Hand
ICON:Question
ICON:Exclamation
ICON:Asterisk
ICON:Pick
ICON:Save
ICON:Print
ICON:Paste
ICON:Open
ICON:New
ICON:Help
ICON:Cut
ICON:Copy
ICON:Child
ICON:Frame
ICON:Clarion
ICON:NoPrint
ICON:Zoom
ICON:NextPage
ICON:PrevPage
ICON:JumpPage
ICON:Thumbnail
ICON:Tick
ICON:Cross
ICON:Connect
ICON:Print1
ICON:Ellipsis
```

Those are a lot of equates to remember, never mind the syntax of MESSAGE. If you're writing a lot of MESSAGE statements, you may eventually commit all of these various options to memory. Or you can create a CODE template that gives you a point and click interface to all of MESSAGE's intricacies.

Making sure the code works

Before writing any template, you will probably want to make sure that the code the template is replacing is working code, or at least that you know how to write the source code. If you write the template, and the generated source code doesn't compile, or worse, compiles but doesn't work, you want to be able to compare it to some working source code. Actually you want to compare it to working source code even if it does compile and appear to work, just to make sure that you are generating the source code correctly. Once you are satisfied that the template works properly, you can use it with confidence, and without having to repeatedly look at the source.

Designing the template

The first question to ask yourself when designing a template is what will the resulting source look like? The `MESSAGE` statement can, broadly, be used in two different ways. First, it can be strictly informational, showing a message to the user, but not requiring the user to make any choices. Second, it can present two or more buttons to the user, so that you can use the user's choice to proceed with various options.

If you're simply presenting the user with a message, then you will only need the template to build the `MESSAGE` function call based on options you choose in the template. If you also want to respond to the specific button the user presses, you have another choice to make: do you want the template to be able to respond to that choice, or do you just want to store the return value and then write some embedded source to handle the user's choice?

Although it may be tempting to put all the possible options into the template, you really have to think about how much of this code will be repeated throughout your application. If you typically take just one of several options after any message you present to the user, then yes, definitely consider how to implement this in a template. If you want to

make writing MESSAGE statements easier, then perhaps you would be better off storing the result of the MESSAGE call in a variable, and then writing embedded source code as needed based on the value stored in that value. This latter option is the one most likely to fit your needs, and that will be my working assumption for the rest of this article.

The EasyMessage template

Now it's time to create a template that will make MESSAGE a lot easier to handle. The first step is to create a new template chain, if you don't already have one handy. Why a new template chain?

As I mentioned in [Part 2](#), the only way to register a template with Clarion is if that template is part of a template chain. And all template chains start with a .TPL file, such as ABCHAIN.TPL or CW.TPL. These TPL files typically have #INCLUDE statements pointing to files with TPW extensions.

So why not just create your templates in a TPW file and add an #INCLUDE statement at the end of, for instance, ABCHAIN.TPL? You can do this, but you'll need to modify ABCHAIN.TPL again when you next upgrade your version of Clarion. It's generally considered bad practice to modify the standard templates in any way, even something as innocent as adding a new #INCLUDE.

Creating a new chain

If you don't already have a template chain of your own to use, create a new file called CMAG.TPL and save it in your Clarion template directory.

The first statement in any template chain is the #TEMPLATE statement. It takes two parameters: the template class name (which is a way of grouping templates by function or author) and the template description. A #TEMPLATE statement should also specify which template "family" it can be used with – in the shipping Clarion templates, this value can be ABC, CW20 (for all non-ABC templates), or both.

Enter the following line at the top of the file:

```
#TEMPLATE(ClarionMag, 'Clarion Magazine templates'),FAMILY('CW20','ABC')
```

`#TEMPLATE` statements can also have the `PRIVATE` attribute, which means that they cannot be regenerated from the template. The idea behind this is that you can distribute a registry with proprietary templates to other developers, and those developers would not be able to regenerate the templates from the registry. But really, template regeneration seems to be a complete non-issue with most, if not all, Clarion developers. So you probably won't want to add the `PRIVATE` attribute.

Now you have another choice to make – whether to put all of your templates into the TPL file, or to split them up into TPW files. You break templates up into modules simply to make the source code easier to find and maintain. Add the one more line to CMAG.TPL, so that the entire file looks like this:

```
#TEMPLATE(ClarionMag, 'Clarion Magazine templates'),FAMILY('CW20','ABC')
#include('CMAGCODE.TPW')
```

When you register CMAG.TPL, the registry will follow this `#INCLUDE` statement to the CMAGCODE.TPW file. So you had better create it! This file will contain the `EasyMessage` template. Writing templates is a lot like writing application source code – you often want to get the basics in place and then fine tune. So here's the template equivalent of a stub procedure:

```
#CODE(EasyMessage, 'Call the MESSAGE function')
```

Save the above text in CMAGCODE.TPW. At this point, the template doesn't do anything, but you should be able to register your new template chain. To do this, close any open application, and choose Setup|Template Registry, and click on the Register button. Select CMAG.TPL and click OK. You should be able to see the code template listed in the registry, as shown in Figure 1.

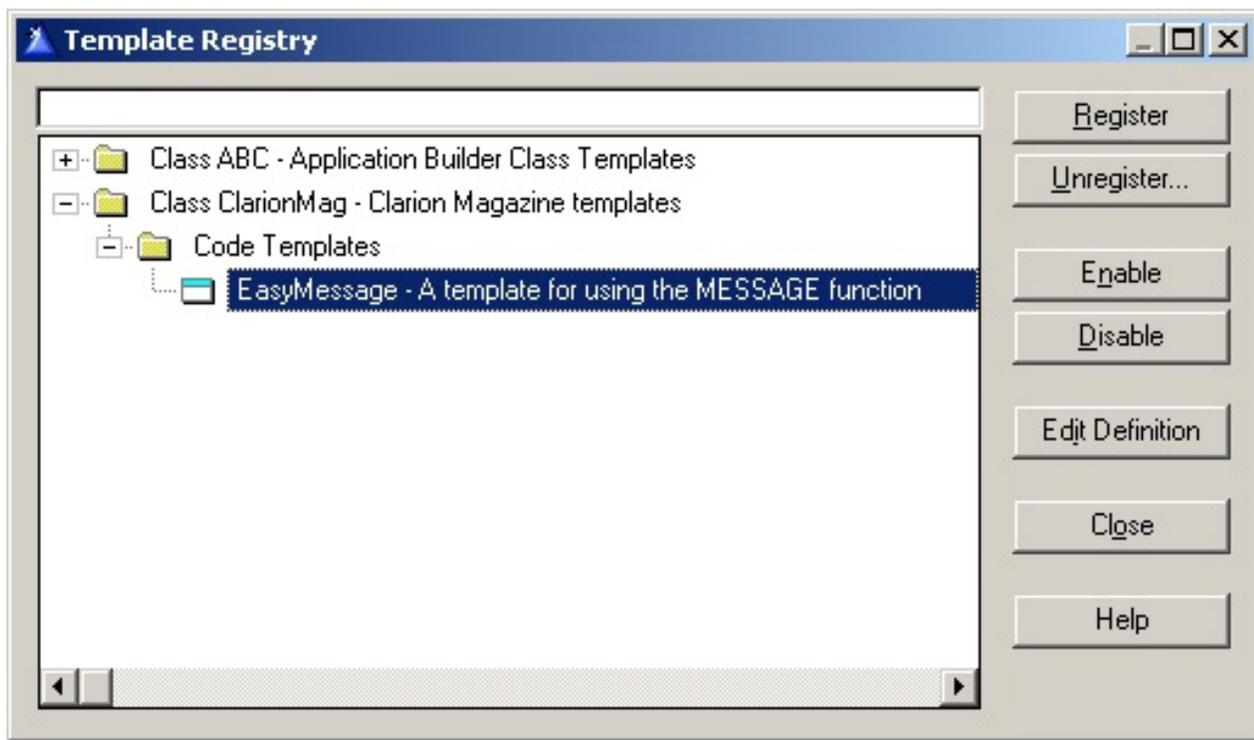


Figure 1. The registered code template.

About the only error you might make up to this point is to add an extra blank line between the `#TEMPLATE` and the `#INCLUDE` – if you do that, when you attempt to register the template you’ll get this error:

```
(cmag.tpl 2,0) Error: Expected a section header
```

This error message points to the beginning of the second line and is simply telling you that the registry doesn’t understand blank lines in this context. Although the template language has been around a long time, it does have its idiosyncrasies.

So now you have your first template! You can even use it in an application, although it won’t yet do a blessed thing. So your first order of business is to set the template up so that it will actually generate some code. Let’s take the simplest possible use of `MESSAGE` first.

The simplest EasyMessage template

In its simplest form, the `MESSAGE` function takes a single parameter – a string to display to the user. It’s true that such a simple use of `MESSAGE`

really doesn't warrant a template, but wait. This is only the beginning, and it will still serve to demonstrate the basics of code template use.

In order to generate a call to `MESSAGE`, you have to ask the developer what message to display. And in templates, you collect this kind of information with a `#PROMPT` statement. Open `CMAGCODE.TPW` and modify it so it looks like this:

```
#CODE(EasyMessage,'A template for using the MESSAGE function')
#PROMPT('Enter a message',@s100),%MessageText,REQ
MESSAGE('%MessageText')
```

The first parameter to `#PROMPT` is the text to display alongside the entry field, and the second parameter is actually a prompt type, which in this case is a standard Clarion entry picture for a one hundred character string. Prompt types can include not just strings and numbers, but template symbols as well, such as files and procedures. `%MessageText` is a template symbol, and also an implicit variable – you don't have to declare your own template symbols before their first use, although you can do so. The `REQ` attribute on the `#PROMPT` just means that this field is required.

Following the `#PROMPT` statement is the code that will be generated into the application. You can tell this isn't template-only code because the line begins with something other than `#`.

This code-to-be-generated line is a combination of a Clarion language statement, `MESSAGE`, and the `%MessageText` symbol declared in the `#PROMPT` line. When this code template is inserted in an embed point, at generation time the AppGen will take whatever value is in `%MessageText` and substitute it in the Clarion code.

Save `CMAGCODE.TPW`. Now load an application, such as the `AutoLog` app in the Clarion examples directory. For test purposes, you'll put the code in an startup embed point so the `MESSAGE` function is called when the app frame loads. The following instructions assume you're using C5.5 or later, with an ABC application. If you're using an earlier version or a Legacy app, simply choose any early embed such as `After Opening`

the Window.

Choose the application's Main procedure. From the Procedure Properties window, click on Embeds. Choose View|Contract All so you can find your navigation starting point. Choose Local Objects|ABC Objects|Window Manager|Init procedure, and then choose any embed under the CODE list. Press Insert. Assuming you remembered to register the template, you should see EasyMessage listed, as shown in Figure 2.

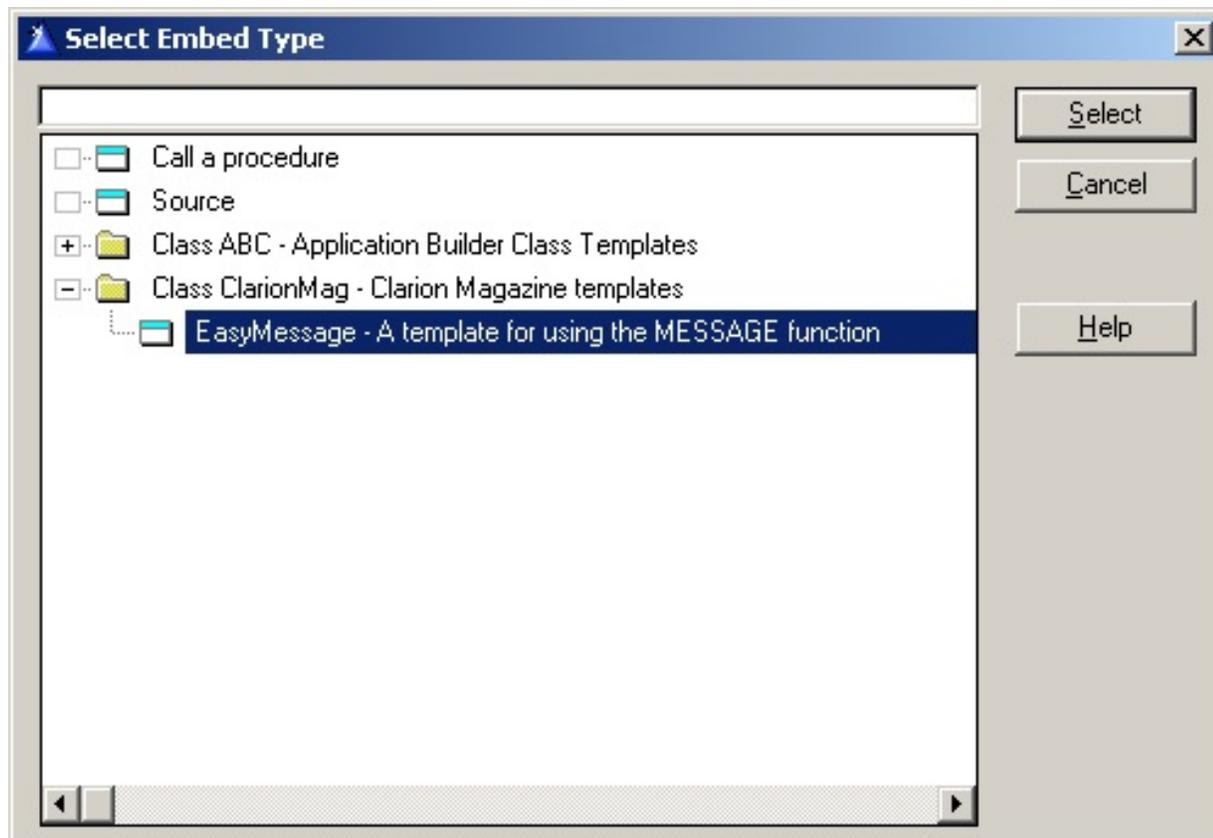


Figure 2. Picking the EasyMessage template

The template will now display a dialog, containing the #PROMPT statement, as in Figure 3.



Figure 3. Collecting input from the developer via #PROMPT

Click OK. The code template shows up in the embed list, as shown in Figure 4.

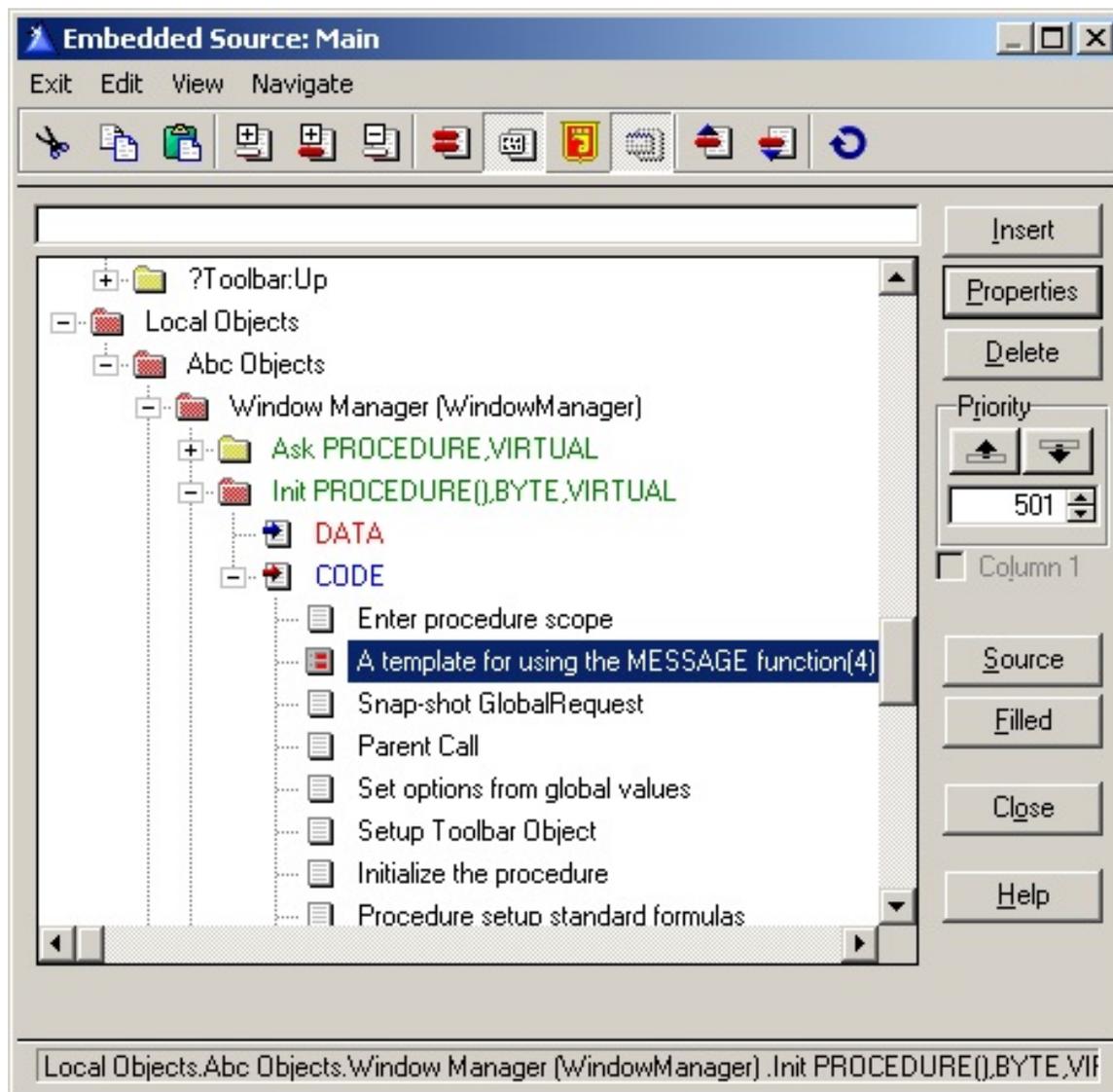


Figure 4. The code template in the embed list.

Save your changes, and compile and run the application. You'll see a message something like Figure 5 when your app starts up.



Figure 5. The template's resulting MESSAGE call.

Assuming you chose the same embed point as shown in Figure 4, the generated source code for `ThisWindow.Init` would begin as follows, with the code template-generated source shown in bold:

```
ThisWindow.Init PROCEDURE
ReturnValue          BYTE,AUTO

CODE
GlobalErrors.SetProcedureName('Main')
MESSAGE('This is my code template!')
SELF.Request = GlobalRequest
```

Note that the source code was generated at the correct indent level for the embed point, even though the `MESSAGE` statement in the template source is in column 1.

Summary

I did say that creating a template for such a simple use of `MESSAGE` is overkill. But `MESSAGE` is a powerful statement with a lot of options which can quickly become overwhelming. In the next article in this series I'll show how you can use a few template statements to create a point and click interface for `MESSAGE`, using techniques readily applicable to a variety of template programming tasks.

[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). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).

Reader Comments

[Add a comment](#)

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.



Detecting Drive Types

by Andrew Guidroz II

Published 2004-05-14

In the good old days, we had databases and we hard coded the paths and filenames. The key part of the word database was base. Everything the end user needed sat there in the server or PC. The machine hosting the database was a large stone monument to the importance of our data. And, just like anything that sits beneath a stone monument, our data was dead, dead, dead.

The true power of our data is in data exchange. We're able to send information to others within our company. We're able to send information to divisions outside our location. Now, with the internet, we're able to send information anywhere in the world to anyone at any time.

My projects lately have required me to FTP data, email data, and dice it up into XML and interface with weird stuff on the other side of the world. But every once in a while, I have to send data the way it was done before the Internet connected us all: via the sneaker-net.

Folks need data copied to floppies. They need it on CDs. They need it on all sorts of removable media like CompactFlash and Memory Sticks. Sometimes, they want to send data to a network drive from their local drive.

I needed to write some code that found out what drives were connected to a computer, as well as what types of media those drives handle. I also needed to know how much room was free on each of those drives. It was time to dig into the Windows API, with a little help from Google and MSDN.

The first function I found was `GetLogicalDriveStrings`. This API returns a nice long string with each drive letter and root path in it. Each is separated by nulls so you get a string like:

```
A:\<null>C:\<null><null>
```

Prototyping was easy:

```
GetLogicalDriveStrings(ULONG, *CSTRING), ULONG, PASCAL, RAW, |
    NAME('GetLogicalDriveStringsA')
```

The only tricky part is looping through the returned string pulling out the drive names. And even that isn't too tricky. Each drive ends with a null. The first drive you find that begins with a null is the end.

The second parameter is a `CSTRING` that will receive the list of logical drives on the PC. The first parameter is the maximum size that `CSTRING` can hold, less the terminating null character. In my case, I use the `SIZE` of my `CSTRING` minus 1.

The second thing I needed to figure out was what type of drive each drive is. Searching came up with `GetDriveType`. This API returns a numeric value for a given drive letter and root path, which in turn sounds exactly like what I'm getting from the `GetLogicalDriveStrings` API.

The following is a table of the values returned by `GetDriveType`, as taken from the Windows include files:

Value	Equate Name
0	Drive_Unknown
1	Drive_No_Root_Dir
2	Drive_Removable
3	Drive_Fixed
4	Drive_Remote
5	Drive_CDROM
6	Drive_RAMDisk

`Drive_Unknown` means Windows is thoroughly confused or you have given it an invalid string. `Drive_No_Root_Dir` means the root path is invalid in your string or, perhaps, there is no volume mounted. `Drive_Removable` applies to floppy disks, zip disks, and removable smart media-type devices. `Drive_Fixed` is your traditional hard

`drive.Drive_Remote` is any drive that you are connected to via a network. `Drive_CDRom` includes both CD ROM drives and DVD drives. And RAM disk applies to software devices that reside in memory.

The final part is getting the size of the drives. There are two functions that do this: `GetDiskFreeSpace`, for drives up to two gigabytes, and `GetDiskFreeSpaceEx`, for drives which may be over two gigabytes. `GetDiskFreeSpaceEx` appeared in Windows 95, SR2. I decided to use this newer API call, since today's drives can be massively large. `GetDiskFreeSpaceEx` uses data types that aren't standard in Clarion. The resulting size has to be defined as `DECIMAL(21)` in Clarion just to hold the maximum size that Windows can return.

But what happens if someone runs my application on a version of Windows older than Windows 95, SR2? To the rescue comes [Loading DLLs at Runtime](#) by Larry Sand. Larry's article shows how to code for functions that may or may not be available at runtime. I created a lib file from `Kernel32.DLL` and coded that API call. I borrowed Larry's prototypes and functions and I was off to the races. Here's the complete source for the example program:

```
PROGRAM

ULARGE_INTEGER GROUP,TYPE
LowDw          ULONG
HighDw         ULONG
                END

MAP
ULIntToDec     PROCEDURE(*DECIMAL, *ULARGE_INTEGER)
MODULE('WINAPI')
  GetLogicalDriveStrings(ULONG,*CSTRING),ULONG,|
    PASCAL,RAW,NAME('GetLogicalDriveStringsA')
  GetDriveType(*CSTRING),ULONG,PASCAL,RAW,NAME('GetDriveTypeA')
  GetDiskFreeSpaceEx(*CSTRING,*ULARGE_INTEGER,*ULARGE_INTEGER,|
    *ULARGE_INTEGER),BOOL,PASCAL,RAW,NAME('GetDiskFreeSpaceExA')
END
END

Loc:DriveQueue QUEUE
CurrentDrive   STRING(5)
DriveType     STRING(20)
TotalSize     DECIMAL(21)
TotalFree     DECIMAL(21)
                END

Window WINDOW('List of Drives'),AT(, ,316,197),FONT('Arial',8,|
  COLOR:Black,FONT:regular,CHARSET:ANSI),GRAY,DOUBLE
LIST,AT(5,3,303,166),USE(?List1),VSCROLL,FORMAT(' 24L(1)|M~Drive~@s5@83L(1)' &|
  'M~Drive Type~@s20@80R(1)|M~Total Size~@n21@20R(1)|M~By' & |
```

```

        'tes Free~@n21@'),FROM(Loc:DriveQueue)
    BUTTON('Close'),AT(114,176,45,14),USE(?Close)
END

```

```

Loc:APIResult                LONG
Loc:ListOfLogicalDrives     CSTRING(255) ! Looks like 'A:\<0>C:\<0><0>
Loc:StringStart              LONG
Loc:StringEnd                LONG
Loc:DriveCount              LONG
Loc:CurrentDriveLetter      CSTRING(5)
Loc:DisplayDriveType        STRING(20)
Loc:i64BytesAvailableThisUser LIKE(ULARGE_INTEGER)
Loc:i64TotalNumberOfBytes   LIKE(ULARGE_INTEGER)
Loc:i64TotalNumberOfFreeBytes LIKE(ULARGE_INTEGER)
Loc:BytesAvailableThisUser  DECIMAL(21)
Loc:TotalNumberOfBytes      DECIMAL(21)
Loc:TotalNumberOfFreeBytes  DECIMAL(21)

```

```
CODE
```

```
! Get the list of drives
```

```
Loc:APIResult = GetLogicalDriveStrings(SIZE(|
    Loc:ListofLogicalDrives) - 1, Loc:ListofLogicalDrives)
```

```
! Handle Each Drive
```

```
Loc:DriveCount = 0
```

```
Loc:StringStart = 1
```

```
Loop
```

```
    IF Loc:ListOfLogicalDrives [ Loc:StringStart : Loc:StringStart ] |
        = '<0>' ! At end of Drive List
        BREAK
    END
```

```
    END
```

```
    Loc:StringEnd = Loc:StringStart + 1
```

```
    Loop
```

```
        IF Loc:ListOfLogicalDrives [ Loc:StringEnd : Loc:StringEnd ] = |
            '<0>' ! At end of current drive
            BREAK
        END
```

```
        END
```

```
        Loc:StringEnd += 1
```

```
    END
```

```
! I now can grab the drive letter and path
```

```
Loc:CurrentDriveLetter = |
    Loc:ListOfLogicalDrives [ Loc:StringStart : Loc:StringEnd - 1 ]
```

```
Loc:DriveCount += 1
```

```
Loc:StringStart = Loc:StringEnd + 1
```

```
! Get the drive type
```

```
Loc:APIResult = GetDriveType(Loc:CurrentDriveLetter)
```

```
Case Loc:APIResult
```

```
    OF 0 ! DRIVE_UNKNOWN
```

```
        Loc:DisplayDriveType = 'Drive Unknown'
```

```
    OF 1 ! DRIVE_NO_ROOT_DIR
```

```
        Loc:DisplayDriveType = 'No root directory'
```

```
    OF 2 ! DRIVE_REMOVABLE
```

```
        Loc:DisplayDriveType = 'Removable'
```

```
    OF 3 ! DRIVE_FIXED
```

```
        Loc:DisplayDriveType = 'Fixed'
```

```

    OF 4 ! DRIVE_REMOTE
        Loc:DisplayDriveType = 'Remote'
    OF 5 ! DRIVE_CDROM
        Loc:DisplayDriveType = 'CD ROM'
    OF 6 ! DRIVE_RAMDISK
        Loc:DisplayDriveType = 'Ram Disk'
END

! Get the disk space information
Loc:APIResult = GetDiskFreeSpaceEx(Loc:CurrentDriveLetter, |
    Loc:i64BytesAvailableThisUser, Loc:i64TotalNumberOfBytes, |
    Loc:i64TotalNumberOfFreeBytes)
IF Loc:APIResult = 0 ! Failed
    CLEAR(Loc:BytesAvailableThisUser)
    CLEAR(Loc:TotalNumberOfBytes)
    CLEAR(Loc:TotalNumberOfFreeBytes)
ELSE
    ULIntToDec(Loc:BytesAvailableThisUser, Loc:i64BytesAvailableThisUser)
    ULIntToDec(Loc:TotalNumberOfBytes, Loc:i64TotalNumberOfBytes)
    ULIntToDec(Loc:TotalNumberOfFreeBytes, Loc:i64TotalNumberOfFreeBytes)
END

Loc:DriveQueue.CurrentDrive = Loc:CurrentDriveLetter
Loc:DriveQueue.DriveType = Loc:DisplayDriveType
Loc:DriveQueue.TotalSize = Loc:TotalNumberOfBytes
Loc:DriveQueue.TotalFree = Loc:BytesAvailableThisUser
ADD(Loc:DriveQueue)
END

OPEN(Window)

ACCEPT
    CASE Field()
    OF ?Close
        POST(EVENT:CloseWindow)
    END
END
END

```

```

ULIntToDec PROCEDURE(*DECIMAL dResult, *ULARGE_INTEGER I64)
Two_To_32 EQUATE(4294967296) ! 2 ^ 32
CODE
dResult = I64.HighDW * Two_To_32 + I64.LowDW
RETURN

```

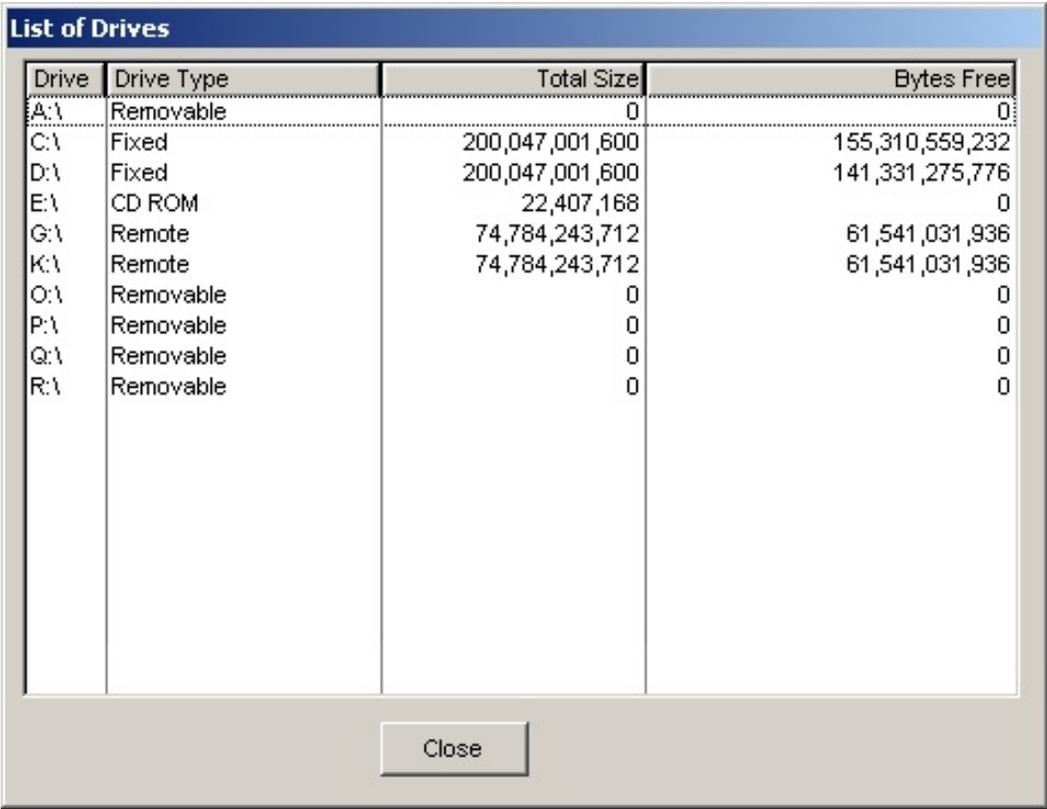
My app now makes a single call to the `GetLogicalDriveStrings`, passing it the length of my string and the string that will contain the list of drives, which I call `Loc:ListOfLogicalDrives`. I then loop through that string getting each drive name one at a time. Each is terminated by a `<0>`. If the drive name begins with a `<0>`, I know that I've processed them all.

Once I have a drive name, I call the `GetDriveType` API passing it the current drive name. The result is tested in a `CASE` statement to determine what type of drive I have as described in the earlier table in this article. Then I call the `GetDiskFreeSpaceEx` API function, passing the drive letter and variables that will get

return values from Windows. The first value is the bytes free on the drive that this user can use, which may not be the same as the physical space on the drive – Windows allows for profiles that limit the amount of free space on a drive that a user can use. The second value is the total number of bytes that the device contains. The third value is the total free bytes on the drive whether the user can use them or not. The `ULIntToDec` is a function documented in Larry Sand's article that converts the large numbers passed back by Windows into Decimal values.

Once I have all of the information about the current drive, I add all of the information to a queue that I will display in the window. This goes on for each drive.

In the end, I open a window and the list box with all of the data for each disk drive on my PC displays, as shown in Figure 1.



The screenshot shows a window titled "List of Drives" with a table containing the following data:

Drive	Drive Type	Total Size	Bytes Free
A:\	Removable	0	0
C:\	Fixed	200,047,001,600	155,310,559,232
D:\	Fixed	200,047,001,600	141,331,275,776
E:\	CD ROM	22,407,168	0
G:\	Remote	74,784,243,712	61,541,031,936
K:\	Remote	74,784,243,712	61,541,031,936
O:\	Removable	0	0
P:\	Removable	0	0
Q:\	Removable	0	0
R:\	Removable	0	0

At the bottom of the window is a "Close" button.

Figure 1. Displaying drive type information

This is fairly simple stuff. In my case, you can see, A:\ is a removable drive. That's a floppy with nothing in it. C:\ is a fixed hard drive with 200 billion bytes total (186 GB) and 155 billion bytes free (144 GB). D:\ is a fixed hard drive with 200 billion bytes total (186 GB and 141 billion bytes free (131 GB).

Drive E:\ is a CD Rom, in my case a DVD ROM, that shows 22 million bytes total size and 0 free. This isn't exactly accurate because it is a CD/RW that is mounted. The maximum capacity would be 650 MB and I have plenty free. But, in order to take

advantage of that space, I have to do things other than use the traditional copying of a disk file from one drive to another. I have to use something like the Windows IMAPI interface to write to it. So Windows, by telling me I have 0 bytes free, is really telling me I can't just copy to this drive.

Drive G:\ and Drive K:\ are big drives on a Windows server. The remaining four drives are all slots in a smart media reader device that are currently empty.

That's a lot of useful information from just a few API calls.

[Download the source](#)

Andrew Guidroz II, when he isn't traveling around the countryside watching his 2004 National Champion LSU Fighting Tigers, writes software for all facets of the insurance and finance industries. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the U.S. Andrew's Cajun website is www.coonass.com.

Reader Comments

[Add a comment](#)

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

[Clarion Magazine](#)

online sales and delivery
for your applications & tools

Developer **PLUS**

Clarion News

[Search the news archive](#)

[Clarion Third Party Profile Exchange Updated](#)

The Clarion Third Party Profile Exchange consists primarily of profiles of third party add-on products and vendors. This includes freeware templates and tools as well. Online and Downloadable Profiles available. Online product profiles include Product Internet URL, Order URL, Currency code, Dated Price Quote, Grouped by Category, Clarion 6 Compatible, Extended Description and Download Page Reference. Currently, there are 476 product profiles and 486 vendor profiles. You must have Product Scope 32 PRO Version 5.0 to view profiles with data files (downloadable profiles). 292 Clarion 6 compatible products as of this release.

Posted Monday, May 24, 2004

[etc2004 Third Party Handout Insert Deadline](#)

Wednesday is the deadline for third party inserts for the etc2004 conference materials.

Posted Monday, May 24, 2004

[SendTo - A New Capesoft Product](#)

SendTo will enable you to send your data exactly as it appears in the browse to Printer, or File (HTML, Excel, Word, CSV or PDF) or email it directly. Drop the SendTo button on any browse/list in your application, and you have all these options available without having to design extra reports or export procedures. No more designing reports to match the printout the user requires - simply display the data.

What you see is what you will get in the SendTo output. Requires NetTalk, Office Inside, WinEvent and Draw. Supports standard Clarion browse boxes and handcoded list boxes. Exports data to Excel, Word, HTML and CSV (PDF is in the pipeline). Price is \$79 while in beta.

Posted Monday, May 24, 2004

Hecticday Schedule For Lodestar & DeveloperPLUS

A hecticday schedule is sort of like a holiday schedule - but different! This Lee White's hectic day(s) schedule leading up to and through etc2004. The offices of Lodestar Software and DeveloperPLUS will be closed from 28-May-2004 through 13-June-2004. They'll be open again on 14-June assuming Lee has recuperated from the conference! The web sites will remain up and running during this period, but phone, fax and snail-mail will fall by the wayside. Lee will be handling email, as time permits.

Posted Monday, May 24, 2004

ADDA 1.0.3

ADDA (Advanced Data Dictionary Architect) version 1.0.3 is now available. Changes include: Bug fix - unique identifier type was not supported; Columns with null values are enabled; Default values can be now altered; Minor user interface improvements. ADDA is a multipurpose toolkit for designing, creating and maintaining the database layer throughout the entire application lifecycle.

Posted Monday, May 24, 2004

BST Ver 3.5 Booking Template

BST Big Scheduler Tamer Suite has a new booking template. BSTBooking features include: Mimics AddSoft's OCX; Daily by time duration; Weekly day duration booking; Monthly day duration booking; Yearly month duration booking; ODBC/SQL compatible - C55 ABC firebird example; Checks for overlaps in previous days, weeks, months; Defaults to yesterday for appointments, seven units(setable) for days and months. Many other features. Free upgrade for 3.0/3.1 users.

Posted Monday, May 24, 2004

Clarion 6.1 EA-2 Released

SoftVelocity has released the 6.1 EA-2 (patch). This is an Early Access build whose primary purpose is to allow 3rd party vendors to continue compatibility testing and their preparation of 6.1 versions of their tools. Make sure your rollback the EA1 patch if you have it installed. To do that, run any of the uninstx.exe versions in the Clarion root. It will prompt you for the log file. Select the Clarion6EE_61-EA1-Patch.log file. After rolling back the EA-1 patch, you should be able to run the EA2 patch and follow the prompts in the install.

Posted Monday, May 24, 2004

Free Clarion Application Icons

Free Clarion application icons for C5, C55, C6, and C61 are now available from Frost Bytes Development.

Posted Monday, May 24, 2004

BST Booking Demo

A basic booking demo of the Big Scheduler Tamer templates is now available. The TPS version is running well, work is continuing on the ODBC/SQL version. The demo is 644K.

Posted Monday, May 24, 2004

Solid Software Back To Business

Jens is back at work! Everything has been going well and he's already feeling much better now.

Posted Monday, May 24, 2004

BST Marketing Survey

Bo Schmitz is looking for input on future product directions for Big Scheduler Tamer.

Posted Monday, May 24, 2004

Crystal Clear Class 2.0

Crystal Clear Class 2.0 is now available. Since crwrap32.dll is not shipped with Crystal Reports 10 any more, the CCC wrapper has been reworked so as not to be dependent on this DLL. Existing features have been retained, new ones have been added. Demo available.

Update price for the existing CCC customers is \$49.95.

Posted Monday, May 24, 2004

EasyExcel 3.05

EasyExcel 3.05 is now available. New methods include: ; AddIns - returns the list of add-ins; AddInInstall - installs an add-in, available to Excel; SetFormatCondition - adds new or modifies an existing conditional format; SetFormatCondFont - sets a font for a formatted cells; SetFormatCondBorder - draws a frame border around the formatted cells; SetFormatCondColour - sets the background colour for the formatted cells; DeleteFormatCond - deletes a conditional format(s) for a range. New templates include: Set Conditional Format template - sets conditional format for a range of cells. This version is for Clarion 5.5 and 6.0.

Posted Wednesday, May 19, 2004

PiFolio Word Reporter 4 For Clarion

PiFolio Word Reporter (formerly known as ARCO Word Reporter) is now ready for use with Clarion introducing extended functionality and new options. PiFolio Word Reporter lets you generate full featured reports and complex documents out of your application. It also enables your customers to change the layout - or even create new ones - without having to learn a report generators interface because it can all be done directly in MS Word. The code has been completely rewritten in Delphi. Now PiFolio Word Reporter is more stable, faster and does not need VB runtimes any more. With the integration of Plugware's COM support the unsafe and slow Clarion OLE implementation has been eliminated, giving you all the benefits other development environments offer (clean and easy understandable code, early binding). Extended functionality and options include: Define your own bookmark delimiters; Set the filename for merge documents; Open the final document with write or edit protection; Give a password to restricted documents. PiFolio Word Reporter is available in three versions, Standard, Enterprise and Architect with different options included, and is available as an update from ARCO Word Reporter 3. Demo available. Requires MS Word 97 or above, Clarion 55g or above, Clarion 6. ABC and Legacy templates.

Posted Wednesday, May 19, 2004

Support For RADIntelliSense With Andy Ireland's COM Generator?

COM objects seem to be everywhere these days but they are not easy to use in Clarion unless you abandon the OLE controls syntax and opt for class wrappers around the native interfaces or late bound wrappers. Plugware has offered to write an add-on to RADIntelliSense, which will give you the option to generate the COM wrapper's with IntelliSense features. After the generation of the wrapper, RADIntelliSense will know all the methods and properties. Therefore, while typing your code, RADIntelliSense will offer you the available methods and make it a lot easier to integrate COM objects in Clarion so you can continue to concentrate on your business needs and not the technical requirements and intricacies of COM.

RADVenture would like to gauge the interest in such a solution before committing to the development of such a tool. This add-on would cost around \$199. Please email Peter Rakke if you are interested in this solution.

Posted Wednesday, May 19, 2004

xWhatsNew Class 1.5

Sealsoft has released xWhatsNew Class v1.5. Includes minor cosmetic changes.

Posted Wednesday, May 19, 2004

etc2004, The Last etc and The Biggest!

Lee White has reported that paid attendance for etc2004 has exceed the previous record, set in 2002. Registrations have now closed - the conference has reached capacity!

Posted Wednesday, May 19, 2004

BRT Registration Tamer Half Off Competitive Upgrade

Big Registration Tamer is on sale until May 31, 2004 at \$249 USD. The regular price is \$299, and you can save \$100 with a proof of purchase of a listed competing product.

Posted Saturday, May 15, 2004

[PlugIT 2.0 Documentation Free Download](#)

The User Guide for PlugIT 2.0 is available at the RadFusion site. In full color, complete with bookmarks.

Posted Saturday, May 15, 2004

[gReg Site Temporarily Down](#)

Gitano Software is moving gReg and the DNS will be updated in the next 24 hours. In the meantime use this temporary link.

Posted Saturday, May 15, 2004

[Hosting Service For Clarion Developers](#)

Robby Berthume's company, Epsilon Computers, has a hosting service for Clarion developers starting at \$7.77 a month. Epsilon Concepts has worked with many Clarion developers, and provides lots of space, bandwidth, and capabilities plus many free scripts and a control panel.

Posted Saturday, May 15, 2004

[New Buttons in C6.1](#)

As these screen shots by Lee White shows, Clarion 6.1 EA has professional looking buttons! These are good looking disabled buttons with icons, real XP buttons with icons, and real XP toolbar buttons.

Posted Saturday, May 15, 2004

[Gitano Source Code Sale](#)

Gitano Software is selling source code for 50% off for a limited time. Not valid with any other offer. Products include gCal, gCalc, gSec, gNotes, gFileFind and gQ. Price includes one year free upgrades.

Posted Monday, May 10, 2004

[Tracker Software Support Email](#)

Please note that Tracker Software has been forced to cease its previously published email support addresses due to a massive increase in spam. All users requiring contact should do so through Tracker Software's new web based ticketing system.

Posted Monday, May 10, 2004

[BRT Registration Tamer From Comsoft7](#)

Comsoft7 is releasing Big Registration Tamer (BRT) Ver 1.0. Special introductory Pricing until May 31, 2004 at \$249 USD. Features include: Easy of use; Mult-DLL support; Extensive documentation; All source included; Eval/demo/timeout mode; Network seats support; Feature registration levels; Subscription expiry; Web registration; All messages fully configurable; Branding support; and much more.

Posted Monday, May 10, 2004

[Free Font Change Template](#)

Simon Burrows has provided a new free template to change the font on all windows and controls. Add this template to your global extensions, and it changes all Windows and controls to whatever font you have asked for.

Posted Monday, May 10, 2004

[etc2004 Four Weeks Away](#)

If you're coming to East Tennessee for etc2004 don't wait too long to register. The block of rooms reserved for the conference is going fast.

Posted Monday, May 10, 2004

[Jaguar: Common Programming Tasks](#)

This page describes how Clarion developers can write Clarion-like embedded Java code for a Jaguar application. Tasks demonstrated include procedure calling, file access, queues access, messages displaying, etc. If you take a look at the differences between Clarion and JAGUAR code, you will see that, except for some punctuation issues (a ";" needed at the end of each statement, double-quotes instead of single-quotes, etc.) and the syntax of some control structures, the JAGUAR code is almost the same as the Clarion code you would write for the same purpose.

Posted Monday, May 10, 2004

[Gitano Software Email Outage](#)

Jesus Moreno at Gitano Software reports that he is currently have email connectivity problems. As soon as he is able to get corrected,

he will respond to waiting messages.

Posted Wednesday, May 05, 2004

[ABC Free Templates And Tools Updated](#)

Changes to the ABC Free Templates and Tools as of 05/03/2004 include a beta release of a class wrapper template chain, VSABCCLS.TPL. This provides three extension templates per class, and one code template per method within each class.

Posted Tuesday, May 04, 2004

[Jaguar Java Source Code](#)

You can now view the generated source for the Jaguar Java demo applications. Also the examples page has been modified so you can download all the programs and run them on your own machine.

Posted Tuesday, May 04, 2004

[Web Email Cloaking Utility Update](#)

Ben Brady has updated the Web Email Cloaking Utility to correct an 'incompatibility' with The Bat! email client. The Bat! does not fully support the RFC 822 mailto: protocol so Ben has provided a workaround.

Posted Tuesday, May 04, 2004

[ImageEx 3 Delayed](#)

Jens Weiermann has had to delay the official release of ImageEx 3. The core library is done 99%, documentation 90%, example apps 60% and installer 100%. Jens will be on a medical leave for a couple of weeks, and should be back around the end of May. Now the good news: Anyone who bought (or will buy) ImageEx after March 1st will receive a free upgrade to version 3.x as soon as it's released! (You can still order at ClarionShop even while Jens is away.)

Posted Tuesday, May 04, 2004

[WEP Key Generator Update](#)

Ben Brady has updated the WEP Key Generator Utility to handle 152 and 256 bit WEP keys, as well as providing a method to communicate with the HTML configuration pages in most routers to make things

easier to copy and paste.

Posted Tuesday, May 04, 2004

PlugIT Update

For those people that have still not received a PlugIT 2.0 update, you need to email Andy Ireland with your licence and contact details in order to qualify. The vast majority of users have now done this and received an upgrade. An update including (on request) Process and Thread handle leak checking will go out this weekend as an update to pwutil.clw and the DLL. If anyone else has any recommendations for future releases, you can use our support email which is detailed on the contact page of the web site. All customers will also be receiving login details to an online bug tracking system so they can see the status of known bugs etc. and when they are updated.

Posted Tuesday, May 04, 2004

etc2004 Room Blocks Disappearing Fast

The special hotel room block Lee White has arranged for etc2004 will be dropping soon, so book now.

Posted Tuesday, May 04, 2004

etc2004 Third Party Handout Inserts

Interested third party vendors may supply handouts for etc2004 conference attendees. These must be received by Lee White no later than May 26.

Posted Tuesday, May 04, 2004

1st Icon Design Special

1st Icon Design is having an icon special, which includes the following: ; XP Collection; XP People Collection; Red Theme Collection; Yellow Theme Collection; Blue Theme Collection; Green Theme Collection; Silver Theme Collection. Bundle valued at \$800 not including several additional bonuses, price now \$299 including shipping.

Posted Tuesday, May 04, 2004

Special Fenix Transition Offer

Since RADventure is in a transition period between the Fenix ASP.NET Generator versions 1.5x and 2.0, the company is offering Fenix 1.5 for a special price of €350,- (Euro). This offer gives you the current version of Fenix 1.5x without including full support or future upgrades, which can be ordered separately. The upgrade to the Gold version 2.0 costs €199,- (Euro) and if you want to have e-mail support with that (besides to the free newsgroup support) you can order one year of Full Support (e-mail support and a minimum of four updates and one upgrade) for €250,- (Euro). As soon as Fenix version 2.0 goes Gold (estimated for the 4th quarter of this year) the prices will go back to normal.

Posted Tuesday, May 04, 2004

[New Native OutlookBar](#)

OutlookBar is a native Clarion component that enables you to write applications that has a navigation bar very similar to the navigation bars in Microsoft OutlookXP and Outlook 2003. Features include: Supports icons in headers and tasks; OutlookXP, Outlook2003 styles or define your own style.; "Easy-to-use" Control Template; Multiple controls per window; Mimic buttons; Dynamically create and delete headers and tasks; Hide/Unhide headers and tasks; Disable/Enable headers and tasks; Get and Set header and task info at runtime; Real, resizable gradients; Customizable colors; Multi-DLL support.; Clarion 5.5 and Clarion 6; ABC and Legacy; Full source code (no black-box DLL's); Windows 95/98/ME/NT/2000/XP support. Demo available.

Posted Tuesday, May 04, 2004

[JAGUAR Samples](#)

Demos of Jaguar apps, which are client-side Java apps generated with Clarion templates, are now available online.

Posted Tuesday, May 04, 2004

[BoxSoft Super Security 6.05](#)

Super Security 6.05 is now available. Changes include: A SSEC::Logon procedure, just like in ABC; An "Attempt auto-logon with network username" feature; An "Auto-Fill UserName from Previous Logon" feature; A Microsoft SQL example. For all purchase

and upgrade issues (including passwords), please contact Mitten Software. Their e-mail address is Mitten@MittenSoftware.com, and their phone numbers are (800) 825-5461 and (952) 745-4941.

Posted Tuesday, May 04, 2004

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.



Cryptography and Clarion: Using the MSCrypto API

by Ron Webb

Published 2004-05-21

My company produces electronic payment solutions that are used by dozens of banks and thousands of corporate users in Kenya and Zimbabwe, carrying nearly a million transactions a month. Data security is of critical concern, not only for the obvious need to protect financial transactions but, from a sales point of view, to overcome customers' objections and their not always rational fears!

Over the years I have had to reinvent the data protection wheel to suit multiple banks' security policies. Each re-write involved gut-wrenching changes of architecture to support different cryptography standards. The need for a flexible and abstracted cryptography layer was critical. I had been watching the Microsoft CryptoAPI for some time, and once the US Government lifted their embargo on the export of strong encryption technology, it was time to tackle the beast!

Before diving into the code, it is important to understand some of the principals and terminology used in cryptography.

What is Cryptography?

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography is not the only method of providing data security, but is one set of techniques. The goals are

to provide the following:

Privacy – keeping the data secret from anyone not authorised to view it.

Data integrity– being able to detect unauthorised changes to the data; insertion, deletion or substitution.

Authentication– being certain of the origin of the data and

Non-repudiation– preventing the data originator from denying having sent the data.

Cryptography is nothing new, and forms of its use have been found as far back as 4000 years ago in Egypt, where non-standard hieroglyphs were carved into monuments. These are not believed to be serious attempts at security, but rather to provide mystery, intrigue and amusement. The first substitution ciphers were in use by Hebrew Scholars around the period 550-500 BC. Cryptography has long been used by religious writers, where new ideas likely to offend existing thinking (and result in the early demise of the author) needed to be concealed, but still accessible to the initiated. Even ancient philanderers benefited from cryptography, as the Kama Sutra advises its use as a method for lovers to communicate without discovery!

In more modern times, cryptography was involved in the conviction, imprisonment and release of Alfred Dreyfus, a French captain accused of selling secrets to the Germans in the 1890s, and in the plotting which led to the execution of Mata Hari, the notorious first World War amateur spy. Conflict has always been fertile ground for cryptography, and crypto analysis is where some of the best mathematical brains are kept busy trying to crack the "other side's" information.

In the period between the world wars the first mechanical cipher machines appeared that made encryption easy and quick to use. These devices also brought about the first large scale organised attacks on ciphers. Marian Rejewski, probably the first crypto-analyst, cracked the early German Enigma system in 1932. This was followed by the establishment of the team of brilliant mathematicians at Bletchley Park in the UK that included Alan Turing. This is the team that was responsible for the sustained cracks of the later German

Enigma machines in use during World War II (sorry, Hollywood, it was the British who cracked Enigma *not* the Americans!). The Bletchley Park team evolved into the British Government Communications Headquarters (GCHQ), which is the equivalent to the US National Security Agency (NSA). Recently declassified documents show that it was the Bletchley Park team that developed public key cryptography, well before the publication of Diffie and Hellman's paper *New Directions in Cryptography*. The team also created schemes essentially identical to RSA encryption in 1973-4. (The British again!)

Encryption

Encryption protects clear, meaningful information by mathematically encrypting or "scrambling" the data using a key, so that the information can only be decrypted or "unscrambled" by using the correct key. It is also not possible to derive the key by examination of any forms of the data.

Data encryption relies on the assumption that there is no short-cut to solving the algorithms used to encrypt or scramble the data. It relies on complexity theory; one must try such an inordinately large number of possibilities to find the correct one that the search will not be worth it.

There are two types of encryption, private key and public key.

Symmetric or Private Key Encryption

Symmetric, or private key encryption, relies on the sender and the recipient each holding identical keys. Anyone holding the secret key can access the data, so key distribution is a critical issue. You cannot safely exchange a clear private key over an insecure medium.

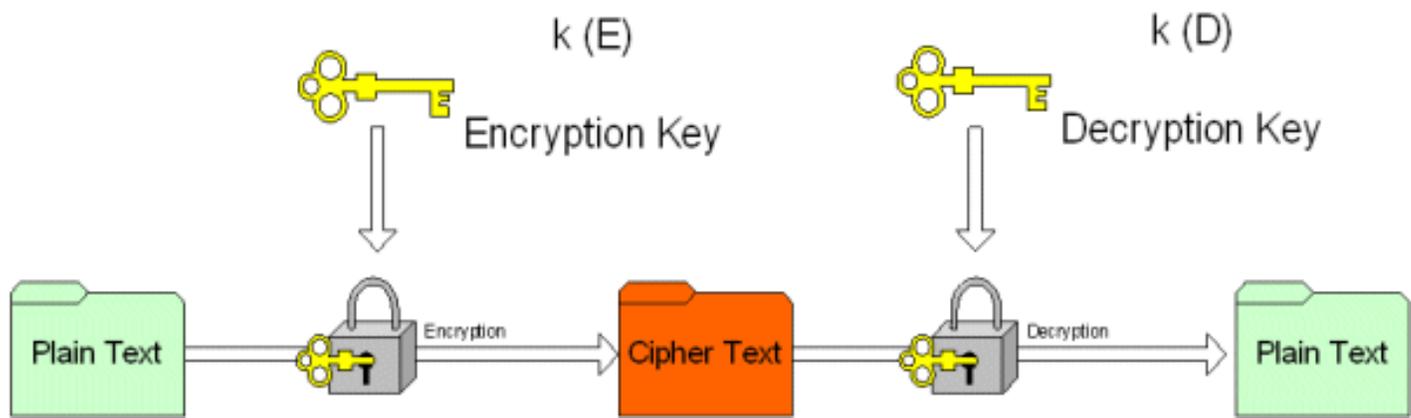


Figure 1. Private key encryption

Examples of symmetric encryption algorithms are DES, Triple-Des and AES.

Asymmetric or Public Key Encryption

Asymmetric, or public key encryption, was created to address the limitations of not being able to distribute keys over an insecure medium. This method uses key pairs - a public key and a private key. The public key does not need to be kept secret. A user could publish his public key in the newspaper and anyone could use it to encrypt and send data to him but only he could decrypt is using his private key.

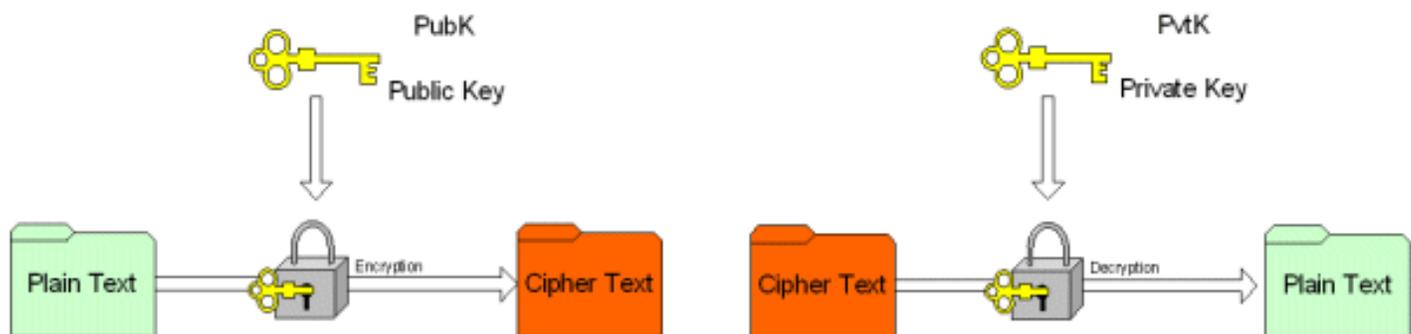


Figure 2. Public key encryption

If Alice and Bob want to secure a conversation using Asymmetric encryption, they would exchange public keys; Alice gives Bob her Public Key and Bob gives his Public Key to Alice. They could do this over the net, email telephone or even by using Personal adverts in the press. When Alice sends to Bob, she uses his Public Key to encrypt the plain text into cipher text. *Only* Bob can decrypt this back to readable plain text. Likewise, Bob uses Alice's Public key to send data to her that similarly *only* she can read.

Examples of asymmetric algorithms are RSA, RC2 and RC4.

Asymmetric Encryption Gotcha!

Asymmetric encryption seems to provide all the answers but for one big problem. By their nature, asymmetric algorithms are enormously mathematically complex. In fact they are so complex that to encrypt a plain text of one megabyte can take over an hour. The same time is needed to decrypt the cipher text back into plain text at the receivers end! The mathematics used in asymmetric encryption means that the time taken to encrypt grows exponentially as input data length (and key length) increases. It is simply too slow to practically use.

So, how on earth is asymmetric encryption of any use? The answer is simple! It is not used to encrypt plain text (input data), but it is used to encrypt symmetric keys. A crypto-random symmetric key is generated for the session (a session key) and this session key is used to encrypt the input plain text using a fast symmetric algorithm. The session key is then encrypted with the recipient's public key and sent along with the encrypted data. The recipient uses his private key to decrypt and extract the session key and then uses this session key to decrypt the actual data!

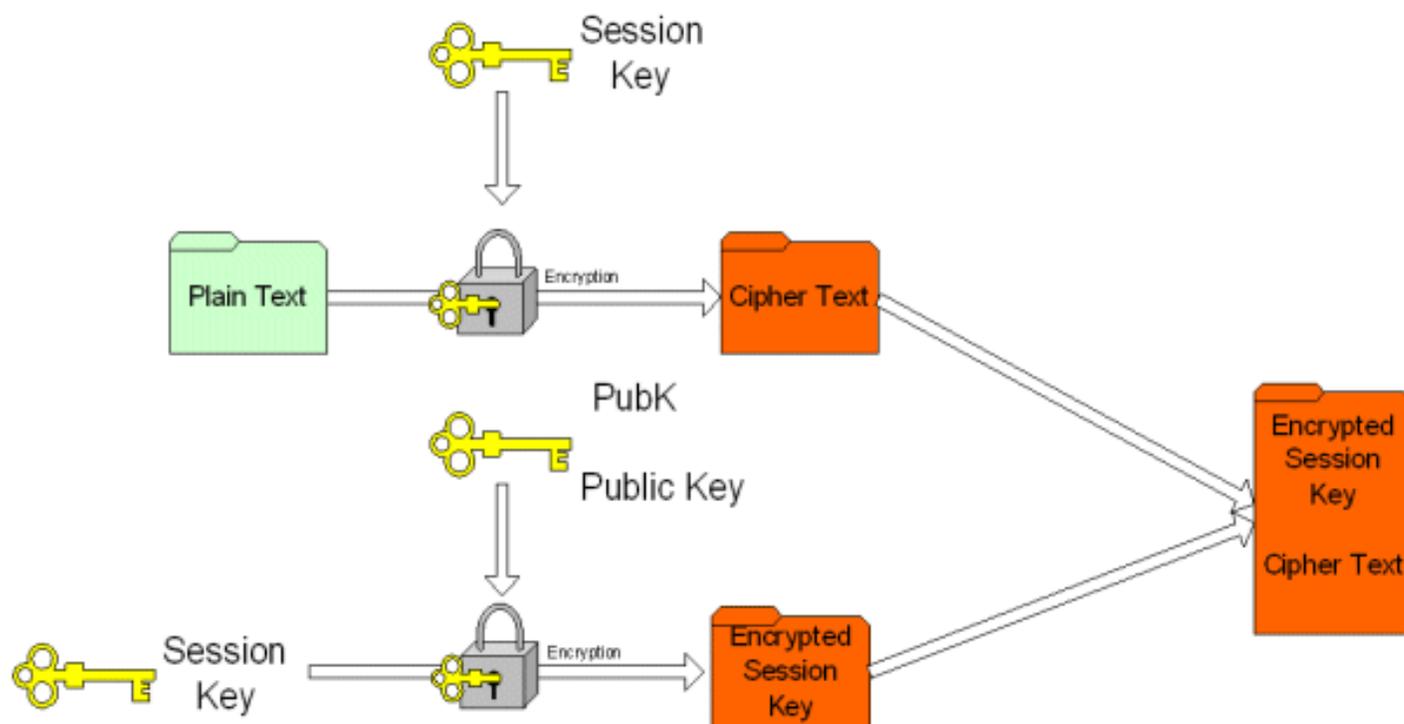


Figure 3. Asymmetric encryption using encrypted symmetric keys**Encryption Strength**

Encryption strength (resistance to crypto-analytic attack) is a product of the algorithm and key length used. So just what is "strong" encryption? Up to 2001 this was defined as anything using a key length over 40 bits. Straight DES encryption using a 56 bit key has been broken in 56 hours by the Electronic Frontier Foundation using a brute force attack. However, the attack took over eight months to set up and used 16,600 computer processors and 5000 MIP years to accomplish (a MIP year is defined as a one million instruction per second computer running for one year).

As key length increase, the probability of successful brute force attack decreases exponentially. I have read that it would take all of the processors that will ever be made and more time than the universe has been in existence to perform the same type of attack on 1024 bit keyed encryption using modern algorithms! Of course, new algorithms do come along.

For my financial transaction purposes, I use 128 bit symmetric encryption and 2048 bit asymmetric encryption. I have no doubt that in time I will need to increase this.

Block and Stream Encryption Algorithms

There are two type of symmetric encryption algorithms; block ciphers and stream ciphers. Block ciphers encrypt data a block at a time, rather than a sequential stream of bits, and are considered to be more secure than stream ciphers. A Block size of 64 bits is most common. Example block ciphers are DES, 3-DES and RC2. Stream ciphers operate faster and an example is RC4. Stream ciphers can be used to encrypt a data-communications channel in real time for example.

Next week

Now that you have the basics of cryptography in hand, it's time to look at how you can make use of this technology on the Windows platform. As I said

earlier, Microsoft's cryptography API is now widely available, and [next week](#) I'll introduce a class you can use to make use of this API.

[Ron Webb](#) was born in Zimbabwe and now lives in Kenya, East Africa. He has been using Clarion since CPD 2.1 and is the Technical Director and co-owner of [Paynet Limited](#) which provides financial transaction solutions to banks and corporate users in Africa. When not programming, he spends his time enjoying the great African outdoors and not enjoying golf! Ron is married to Karen and they have four children.

Reader Comments

[Add a comment](#)

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

Clarion Magazine



Understanding Clarion Templates Part 5: Enhancing Code Templates

by **David Harms**

Published 2004-05-21

In the previous installment in this series, I introduced a very simple code template that did nothing more than insert a MESSAGE statement into the selected embed point. And while that example served to demonstrate how a code template works, it didn't do much in the way of leveraging the template language's function of making the programmer's life easier. In this article I'll show you how to start adding some functionality to that template.

The template

As you might recall from the previous installment, the code template I wrote looks like this:

```
#CODE(EasyMessage,'A template for using the MESSAGE function')
#PROMPT('Enter a message',@s100),%MessageText,REQ
MESSAGE('%MessageText')
```

Recall also that this template resides in a file called cmagcode.tpw, which is referenced in a template chain with a TPL file called cmag.tpl:

```
#TEMPLATE(ClarionMag, 'Clarion Magazine templates'),FAMILY('CW20','ABC')
#INCLUDE('CMAGCODE.TPW')
```

Adding documentation

One of the first things you should consider adding to any code template is a little documentation, and you do this with `#DISPLAY` statements. Let's say you added some text like this:

```
#CODE(EasyMessage,'A template for using the MESSAGE function')
#DISPLAY('This template provides an easy-to-use interface
  for the MESSAGE() statement')
#PROMPT('Enter a message',@s100),%MessageText,REQ
MESSAGE('%MessageText')
```

Note that I've wrapped the `#DISPLAY` line only so it will fit on this page – in the template I have the entire `#DISPLAY` statement on one line. If I then attempt to use this code template, I'll see a template dialog like Figure 1:

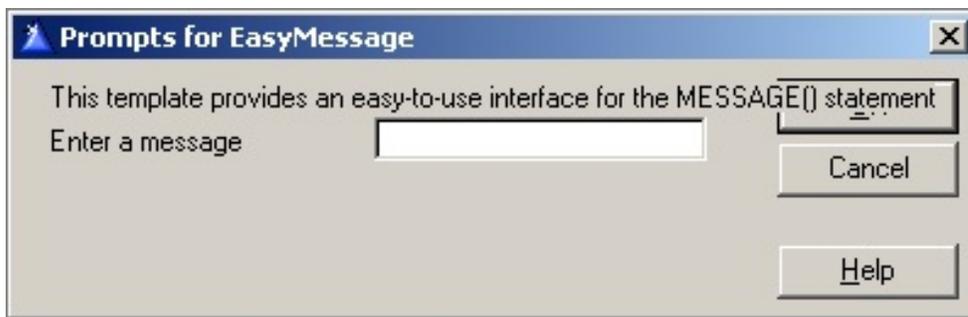


Figure 1. Houston, we have a problem

As Figure 1 clearly demonstrates, `#DISPLAY` statements in their barest form are remarkably unintelligent. They just write whatever text you tell them to write, just like the Clarion `SHOW` statement, and it's up to you to make sure that text doesn't obliterate something else on the screen or get clipped by the window border.

You can tidy up `#DISPLAY` statements with the `AT` attribute. Here's a version of the `#DISPLAY` statement that specifies a maximum width (again, wrapped here but not in the template):

```
#DISPLAY('This template provides an easy-to-use interface
  for the MESSAGE() statement'),AT(,,175,16)
```

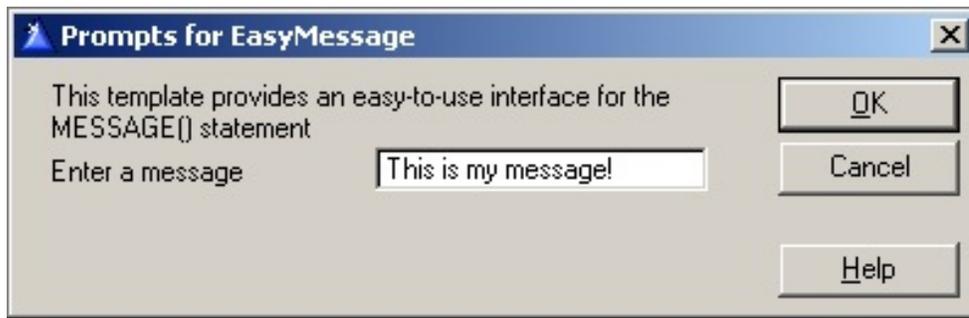


Figure 2. A cleaner display

Figure 2 definitely looks a little nicer. But what do those numbers in the AT parameter mean?

For starters, the first two parameters, the X and Y coordinates, are omitted, which means that the AppGen will place this text following any prior displayed text. This is the first instance, so the text begins at the upper left corner of the display area (the actual borders of this display area are determined internally by the AppGen, as is the appearance of the default OK, Cancel, and Help buttons).

The numbers passed to AT are dialog units, or DLUs. The Clarion Help has this to say (in part) about DLUs:

Dialog units are Windows' standard way of measuring distance on screen.

Dialog units are defined as one-quarter the average character width by one-eighth the average character height of the default font used on a window (creating a roughly square unit of measurement).

The actual size of a dialog unit can vary from one window to the next, depending upon your selection of the font to use for the window.

I'm using a width of 175, mainly because this is a very common value used in the templates that ship with Clarion, and I'm using a height of 8 * number of lines, or 16, since I expect two lines here.

You may see long template `#DISPLAY` statements written this way:

```
#DISPLAY('This template provides an easy-to-use interface ')
#DISPLAY(' for the MESSAGE() statement')
```

This approach has two disadvantages. First, if you decide later you want to add a bit more text in the middle, you have to handle all of the line breaks yourself. Secondly, the result doesn't look as good. The AppGen adds a little extra line spacing between `#DISPLAY` statements, and this extra spacing does not appear between lines inside a multi-line `#DISPLAY`. Your template dialogs will look neater if you specify multi-line `#DISPLAYs` using `AT`.

Adding optional prompts

As I discussed previously, the `MESSAGE` statement has a number of optional parameters:

```
MESSAGE( text [,caption] [,icon] [,buttons] [,default] [,style] )
```

The second parameter to the `MESSAGE` statement is the caption for the message window. To have the template generate the caption, first add a suitable `#PROMPT`:

```
#PROMPT('Enter a caption',@s100),%Caption
```

Like the `#PROMPT` statement for the message text, this statement declares a variable, called `%Caption`. But this time there is no `REQ` attribute, since the caption is an optional parameter to `MESSAGE`, and I want it to be optional in the template.

Sometimes you will want to vary how you generate code, based on whether or not a template prompt has any data. You could write the code to generate `MESSAGE` with an `#IF` statement, like this:

```
#IF(%Caption)
MESSAGE('%MessageText','%Caption')
#ELSE
MESSAGE('%MessageText')
#END
```

However, there is no functional difference between `MESSAGE('some text', '')` and `MESSAGE('some text')` so there's no need to go to this length. You can simply use this code:

```
MESSAGE('%MessageText', '%Caption')
```

Adding an icon pick list

The third parameter of the `MESSAGE` statement is an icon equate value, to indicate which standard icon Windows should display in that message box. These are the Clarion equates for the icons:

```
ICON:None  
ICON:Application  
ICON:Hand  
ICON:Question  
ICON:Exclamation  
ICON:Asterisk  
ICON:Pick  
ICON:Save  
ICON:Print  
ICON:Paste  
ICON:Open  
ICON:New  
ICON:Help  
ICON:Cut  
ICON:Copy  
ICON:Child  
ICON:Frame  
ICON:Clarion  
ICON:NoPrint  
ICON:Zoom  
ICON:NextPage  
ICON:PrevPage  
ICON:JumpPage  
ICON:Thumbnail  
ICON:Tick  
ICON:Cross  
ICON:Connect  
ICON:Print1  
ICON:Ellipsis
```

That's a lot of equates to remember! It would be much nicer to have those in a pick list:

```
#PROMPT('Choose an icon',DROP('ICON:None|ICON:Application|
ICON:Hand|ICON:Question|ICON:Exclamation|ICON:Asterisk|
ICON:Pick|ICON:Save|ICON:Print|ICON:Paste|ICON:Open|
ICON:New|ICON:Help|ICON:Cut|ICON:Copy|ICON:Child|
ICON:Frame|ICON:Clarion|ICON:NoPrint|ICON:Zoom|
ICON:NextPage|ICON:PrevPage|ICON:JumpPage|ICON:Thumbnail|
ICON:Tick|ICON:Cross|ICON:Connect|ICON:Print1
|ICON:Ellipsis')),%Icon,DEFAULT('ICON:None'))
```

I've added line breaks here for readability, but again, in the template that's all on one line. This #PROMPT has a DROP attribute after the prompt text, and the DROP contains a list of options, delimited by vertical bars.

At the end of the DROP attribute (and note the double closing parentheses, one for the DROP and one for the #PROMPT – this can be tricky to debug if you miss it) is the %Icon variable, which will hold the developer's choice, and a new attribute, DEFAULT. This attribute simply guarantees that ICON:None will be specified if the developer makes no other choice.

The code to generate the MESSAGE statement remains stunningly simple:

```
MESSAGE('%MessageText','%Caption',%Icon)
```

Figure 3 shows the template as it appears to the developer.

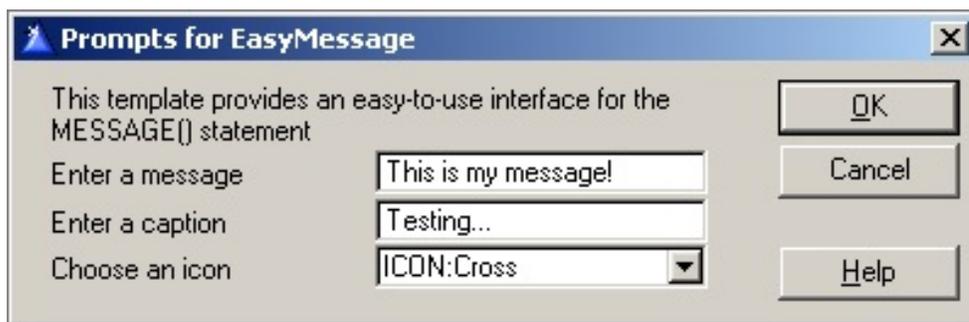


Figure 3. The template with an icon pick list.

That's three of the MESSAGE parameters taken care of, just three more to go! Note that I've been careful to ensure that no matter what choices the developer makes when applying the template, the code will generate without error, and will run as expected. That's been an easy

task so far, but it's about to get a bit trickier.

Adding button choices

The fourth parameter to the `MESSAGE` statement specifies which buttons should be displayed on the message. These equates are listed in `EQUATES.CLW` as follows:

```
BUTTON:OK           EQUATE (01H)
BUTTON:YES          EQUATE (02H)
BUTTON:NO           EQUATE (04H)
BUTTON:ABORT        EQUATE (08H)
BUTTON:RETRY        EQUATE (10H)
BUTTON:IGNORE       EQUATE (20H)
BUTTON:CANCEL       EQUATE (40H)
BUTTON:HELP         EQUATE (80H)
```

Note that these equates are not numbered sequentially – instead, they in the hex equivalent of the binary pattern 00000001, 00000010, 00000100 and so on. That way you can add these values together, pass them to `MESSAGE`, and `MESSAGE` can do a binary AND (`BAND`) on the patterns of each equate to determine which buttons you want to display. `MESSAGE` also returns the value of the selected button, so you can test for results this way:

```
IF MESSAGE('Do you want to continue?', |
  'Choices, choices',ICON:Question, |
  BUTTON:Yes+Button:No) = BUTTON:Yes
```

Clearly, a pick list will not work because you will often want to supply multiple buttons. A set of checkboxes, one for each button, makes more sense.

Adding the checkboxes is the easy part:

```
#BOXED('Buttons to display')
  #PROMPT('OK',CHECK),%ButtonOK
  #PROMPT('Yes',CHECK),%ButtonYes
  #PROMPT('No',CHECK),%ButtonNo
  #PROMPT('Abort',CHECK),%ButtonAbort
  #PROMPT('Retry',CHECK),%ButtonRetry
  #PROMPT('Ignore',CHECK),%ButtonIgnore
```

```
#PROMPT( 'Cancel' ,CHECK) ,%ButtonCancel
#PROMPT( 'Help' ,CHECK) ,%ButtonHelp
#ENDBOXED
```

Note the use of #BOXED/#ENDBOXED – this draws an enclosing box around the prompts, as shown in Figure 4.

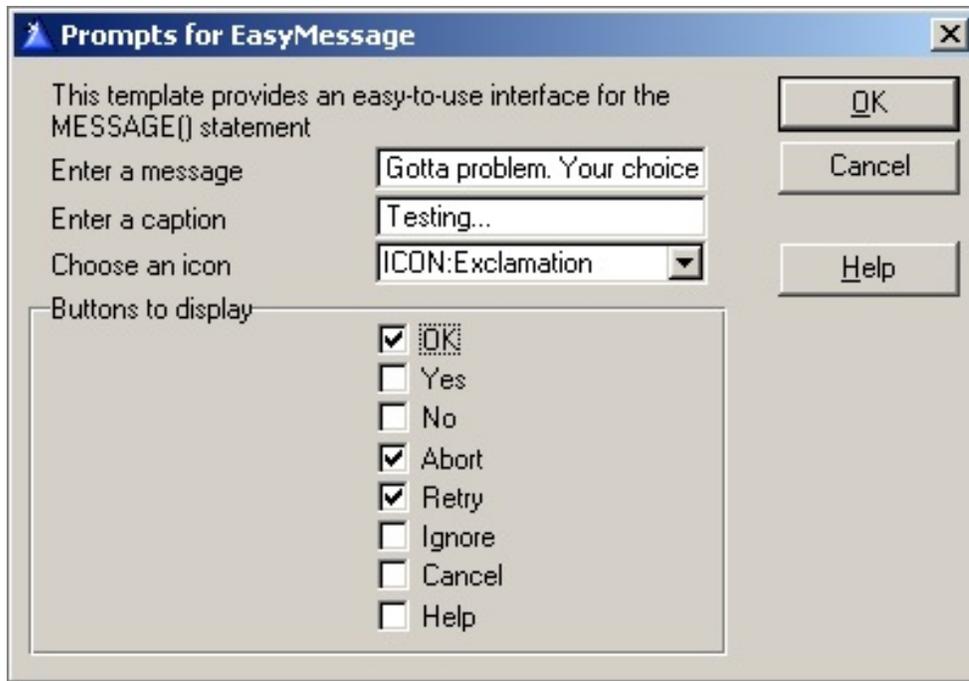


Figure 4. Adding the button checkboxes

Displaying the checkboxes is easy - the tricky part is generating the code. This is because it is no longer a simple question of whether or not a template variable has a value, but a situation where multiple variables may or may not have values, and if there is more than one value you have to add the values together.

The brute force method

As you'll see in a moment, the obvious solution isn't always the prettiest solution. There is an elegant way to solve this problem, and there is an ugly, brute force way. Since the elegant solution involves an aspect of the template language I haven't yet discussed, I'll consider the brute force method first.

The first thing I do is declare a variable to use as a true/false flag:

```
#DECLARE(%ButtonUsed)
#SET(%ButtonUsed,%FALSE)
```

I'll use this variable to determine if I've already found at least one checkbox, meaning that my parameter list already has one button equate, and I need to prepend the + character before I write the next button equate.

I start by writing out the first part of the MESSAGE call:

```
MESSAGE('%MessageText','%Caption',%Icon,|
```

Note the continuation character. The following code will generate each specified button equate on its own line:

```
#IF(%ButtonOK)
  BUTTON:OK |
  #SET(%ButtonUsed,%TRUE)
#ENDIF
#IF(%ButtonYes)
  #IF(%ButtonUsed)
  + BUTTON:Yes |
  #ELSE
  BUTTON:Yes |
  #SET(%ButtonUsed,%TRUE)
  #ENDIF
#ENDIF
#IF(%ButtonNo)
  #IF(%ButtonUsed)
  + BUTTON:No |
  #ELSE
  BUTTON:No |
  #SET(%ButtonUsed,%TRUE)
  #ENDIF
#ENDIF
#IF(%ButtonAbort)
  #IF(%ButtonUsed)
  + BUTTON:Abort |
  #ELSE
  BUTTON:Abort |
  #SET(%ButtonUsed,%TRUE)
  #ENDIF
#ENDIF
#IF(%ButtonRetry)
  #IF(%ButtonUsed)
  + BUTTON:Retry |
  #ELSE
```

```

    BUTTON:Retry |
        #SET(%ButtonUsed,%TRUE)
    #ENDIF
#ENDIF
#IF(%ButtonIgnore)
    #IF(%ButtonUsed)
    + BUTTON:Ignore |
    #ELSE
    BUTTON:Ignore |
        #SET(%ButtonUsed,%TRUE)
    #ENDIF
#ENDIF
#IF(%ButtonCancel)
    #IF(%ButtonUsed)
    + BUTTON:Cancel |
    #ELSE
    BUTTON:Cancel |
        #SET(%ButtonUsed,%TRUE)
    #ENDIF
#ENDIF
#IF(%ButtonHelp)
    #IF(%ButtonUsed)
    + BUTTON:Help |
    #ELSE
    BUTTON:Help |
        #SET(%ButtonUsed,%TRUE)
    #ENDIF
#ENDIF

```

And at the end, the code generates a closing parenthesis.

If, for example, you choose the OK, Abort, and Retry buttons, the template will generate code something like this:

```

MESSAGE('Gotta problem. Your choice!','Testing...',ICON:Exclamation,|
    BUTTON:OK |
    + BUTTON:Abort |
    + BUTTON:Retry |
)

```

And the message will display as shown in Figure 5.



Figure 5. A message box with multiple buttons.

Clearly the code works, but it's ugly. There's a lot of repetition there. And what do you do in your Clarion code when it gets repetitive? You generalize the repetitive code into a function (or a class).

Fortunately, the template language does provide for user-written functions, of a sort – they're called #GROUPS. Next time, I'll show you how you can use a #GROUP to clean up the MESSAGE generation code.

[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). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).*

Reader Comments

[Add a comment](#)

One minor enhancement would be to output text using the...

That's a good enhancement - I'll do some testing and...

Your explanation of #DISPLAY, the need to calculate line...

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

[Clarion Magazine](#)



Cryptography and Clarion: Using the MSCrypto API Part 2

by Ron Webb

Published 2004-05-26

[Last week](#) I introduced the subject of cryptography, and indicated that an API to perform cryptographic tasks is now widely available from Microsoft. In this article I'll provide some basics on calling that API, using a custom Clarion class which I've provided.

The Microsoft CryptoAPI

Microsoft's current approach to providing support for widely varying and changing cryptography needs is the CryptoAPI. The architecture of this API abstracts the raw cryptography functionality behind a common set of interfaces. The actual crypto engines, that is the bits that do the actual cryptographic work, are encapsulated as Cryptographic Service Providers (CSPs). Multiple CSPs can be installed on one system, providing support for everything from real-time data communications encryption through smartcard based authentication.

Enumerating providers (using the CryptoAPI, of course) on my machine shows that the following CSPs are installed;

- Infineon SICRYPT Base Smart Card CSP
- Microsoft Base Cryptographic Provider v1.0
- Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
- Microsoft Base DSS Cryptographic Provider
- Microsoft DH SChannel Cryptographic Provider
- Microsoft Enhanced Cryptographic Provider v1.0
- Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
- Microsoft Enhanced RSA and AES Cryptographic Provider (Prototype)
- Microsoft Strong Cryptographic Provider
- Schlumberger Cryptographic Service Provider

These CSPs provide different cryptographic functionality in different ways or using different key lengths. The common CSPs I will be describing are the Microsoft Base Cryptographic Provider and the Microsoft Enhanced Cryptographic Provider. These are both general purpose providers that support symmetric and public key encryption, hashes and digital signatures. The Base provider was the first CSP released by Microsoft and limits encryption key length to 40 bits to comply with the then in force US embargo on the export of strong encryption.

In addition to providing the base cryptographic functions, each CSP also provides a persistent Key Database that securely stores key pairs and optionally session keys. This Key Database can contain multiple keys in Key Containers and can expose the containers on a machine-wide or per-user basis.

Here are the key lengths supported by the Microsoft CryptoAPI, up to and including Windows 2000:

Key Type and Provider	Minimum	Default	Maximum
RSA Base and Strong Providers - Signature Keys	384	512	16384
RSA Enhanced Provider - Signature Keys	383	1024	16384
RSA Base and Strong Providers - Exchange Keys	384	512	1024
RSA Enhanced Provider - Exchange Keys	384	1024	16348

Here are the key lengths supported by the Microsoft CryptoAPI starting with Windows XP:

Key Type and Provider	Minimum	Default	Maximum
RSA Base Providers - Signature and ExchangeKeys	384	512	16384
RSA Strong and Enhanced Provider - Signature and Exchange Keys	383	1024	16384

DSS Base Providers - Signature Keys	512	1024	1024
DSS Base Providers - Exchange Keys	N/A	N/A	N/A
DSS/DH Base Providers - Signature Keys	512	1024	1024
DSS/DH Base Providers - Exchange Keys	512	512	1024
DSS/DH Enhanced Providers - Signature Keys	512	1024	1024
DSS/DH Enhanced Providers - Exchange Keys	512	1024	4096

The following table lists the algorithms supported by the Microsoft Enhanced Cryptographic Provider.

Algorithm ID	Description	Comments
CALG_MD2	MD2 hashing algorithm	
CALG_MD5	MD5 hashing algorithm	
CALG_SHA	SHA hashing algorithm	
CALG_SHA1	Same as CALG_SHA	
CALG_MAC	MAC keyed hash algorithm	Block cipher MAC
CALG_HMAC	MAC keyed-hash algorithm	HMAC computation.
CALG_SSL3_SHAMD5	SLL3 client authentication algorithm	For more information, see SSL3 Client Authentication Algorithm

CALG_RSA_SIGN	RSA public-key signature algorithm	<p>Key length: can be set, 384 bits to 16,384 bits in 8 bit increments.</p> <p>Default key length: 1,024 bits.</p> <p>Signature conforms to PKCS #6.</p>
CALG_RSA_KEYX	RSA public-key exchange algorithm	<p>Key length: can be set, 384 bits to 16,384 bits in 8 bit increments.</p> <p>Default key length: 1,024 bits.</p>
CALG_RC2	RC2 block encryption algorithm	<p>Key length: 128 bits.</p> <p>Default mode: Cipher block chaining.</p> <p>Block size: 64 bits.</p> <p>Salt length: can be set.</p>
CALG_RC4	RC4 stream encryption algorithm	<p>Key length: 128 bits.</p> <p>Salt length: can be set.</p>
CALG_DES	DES encryption	<p>Key Length: 56 bits.</p> <p>Default mode: Cipher block chaining.</p> <p>Block size: 64 bits.</p> <p>No salt allowed.</p>
CALG_3DES_112	Two-key triple DES encryption	<p>Key Length: 112 bits.</p> <p>Default mode: Cipher block chaining.</p> <p>Block size: 64 bits.</p> <p>No salt allowed.</p>

CALG_3DES	Triple DES	Key Length: 168 bits. Default mode: Cipher block chaining. Block size: 64 bits. No salt allowed.
-----------	------------	---

The Clarion CryptoCL Class

My design criteria in developing the `CryptoCL` class was to hide as much of the complexity of the Microsoft CryptoAPI from the programmer as possible, while still providing a desired level of functionality. The `CryptoCL` class provides methods for the following functionality;

- Listing CSPs
- Checking for the existence of a CSP
- Managing Machine and User Key Containers
- Generating Key Pairs
- Public Key encryption and decryption for files
- Password encryption and decryption for files
- Hash generation and verification
- Support for different encryption and signature algorithms
- Support for different key lengths

All of the methods are "self contained" in that they do not persist any handles or context to the CryptoAPI. In other words once you call one method, you don't have to call another method to free up some memory allocated by the CryptoAPI. There is a slight performance penalty for this but I believe this is outweighed by the reduced complexity.

All of the file encryption/decryption functions use native Windows file functions. This is for speed as I found trying to use a Clarion DOS file driver prohibitively slow.

Properties

DebugMode Private property that turns on debug messages. Set in the init method.

hCryptProv	Private property. The main "handle" to the container and the CSP. Used by almost all CryptoAPI functions. Set in CryptAcquireContext called in most CryptoCL methods
Container	Holds the Key Container name.
Provider	Identifies the provider CSP by name that will be used for subsequent functions.
ProviderType	Identifies the Provider Type. You can request to acquire context to a provider without identifying which specific CSP to use. Types are; <ul style="list-style-type: none"> • PROV_RSA_FULL • PROV_RSA_AES • PROV_RSA_SIG • PROV_RSA_SCHANNEL • PROV_DSS • PROV_DSS_DH • PROV_DH_SCHANNEL • PROV_FORTEZZA • PROV_MS_EXCHANGE • PROV_SSL
ErrorDesc	Last Error description if any
ErrorCode	Last Error Code if any
ExchangeKeyLength	Sets the key length to use for key exchange key pairs
SignatureKeyLength	Sets the key length to use for signature exchange key pairs
SessionKeyLength	Sets the key length to use for session keys used in symmetric encryption.
AlgID	Sets the encryption or hash algorithm to use. CSP dependant.

Methods

Init	Simply sets the DebugMode
-------------	---------------------------

- CheckForProvider** Checks that a given CSP identified in `self.Provider` is installed. The CryptoAPI provides no simple way to do this so I loop through all the installed providers using `CryptEnumProviders` looking for the requested name.
- CheckForContainer** Checks that a given Key Container identified in `self.Container` exists in the CSP set in `self.Provider`. Currently checks only for User context containers.
- CreateContainer** Creates an empty Key Container identified in `self.Container` in the CSP set in `self.Provider`. The Container may not already exist.
- DeleteContainer** Deletes an empty Key Container identified in `self.Container` in the CSP set in `self.Provider`.
- GenKeyPairs** Creates Signature and Key Exchange Key Pairs in `self.Container`. Only do this once! If existing key pairs exist in the container, they will be overwritten and you would need to re-distribute your public keys.
- GetPublicKeyBlob** This method returns the public key from either the Exchange or Signature key pairs.

Parameters:

KeyType IN AT_KEYEXCHANGE

or

AT_SIGNATURE KeyBlob IN/OUT -
Address of a string to return the key blob. Allow for 2048 bytes

KeyBlobLen IN/OUT - Key Blob length

SessionEncryptFile Performs public key encryption on a file. The method generated a pseudo random Session Key and uses this to encrypt the file. The Session Key is then encrypted with the Recipient's Public Key Exchange Key and the encrypted Session Key is pre-pended to the file.

Parameters:

InFileName IN - Input file

OutFileName IN - Output file

RcpPubKeyBlob IN - Address of the recipients public key exchange key

RcpPubKeyLen IN - Length of the recipient's public key

AlgID IN - The Symmetric encryption algorithm to use. See the Microsoft CryptoAPI for a list of algorithms.

SessionDecryptFile Performs public key decryption on an encrypted file. The method extracts the encrypted Session Key from the front of the encrypted file and decrypts the key using the container's private key exchange key. The decrypted Session Key is then used to decrypt the data file.

Parameters

InFileName IN - Input file

OutFileName IN - Output file

PasswordEncryptFile Encrypts a file using a supplied password. The method generates an MD5 hash of the password and uses this hash to derive a Session Key. The file is encrypted using this Session Key.

Parameters:

InFileName IN - Input file

OutFileName IN - Output file

Password IN - Password used to encrypt the file

PasswordLen IN - Length of the password

AlgID IN - The Symmetric encryption algorithm to use.

PasswordDecryptFile Decrypts a file previously encrypted using the password.

Parameters:

InFileName IN - Input file

OutFileName IN - Output file

Password IN - Password used to encrypt the file

PasswordLen IN - Length of the password

AlgID IN - The Symmetric encryption algorithm to use.

HashPassword

Generates a hash of a supplied password.

Parameters:

Password IN - Password

PasswordLen IN - Length of the password

RetHash IN/OUT - Address of a string to hold the hash

RetHashLen IN/OUT - Pointer to a long to hold the hash length

CheckPasswordHash Verifies a hash against a supplied password.

Parameters:

Password IN - Password

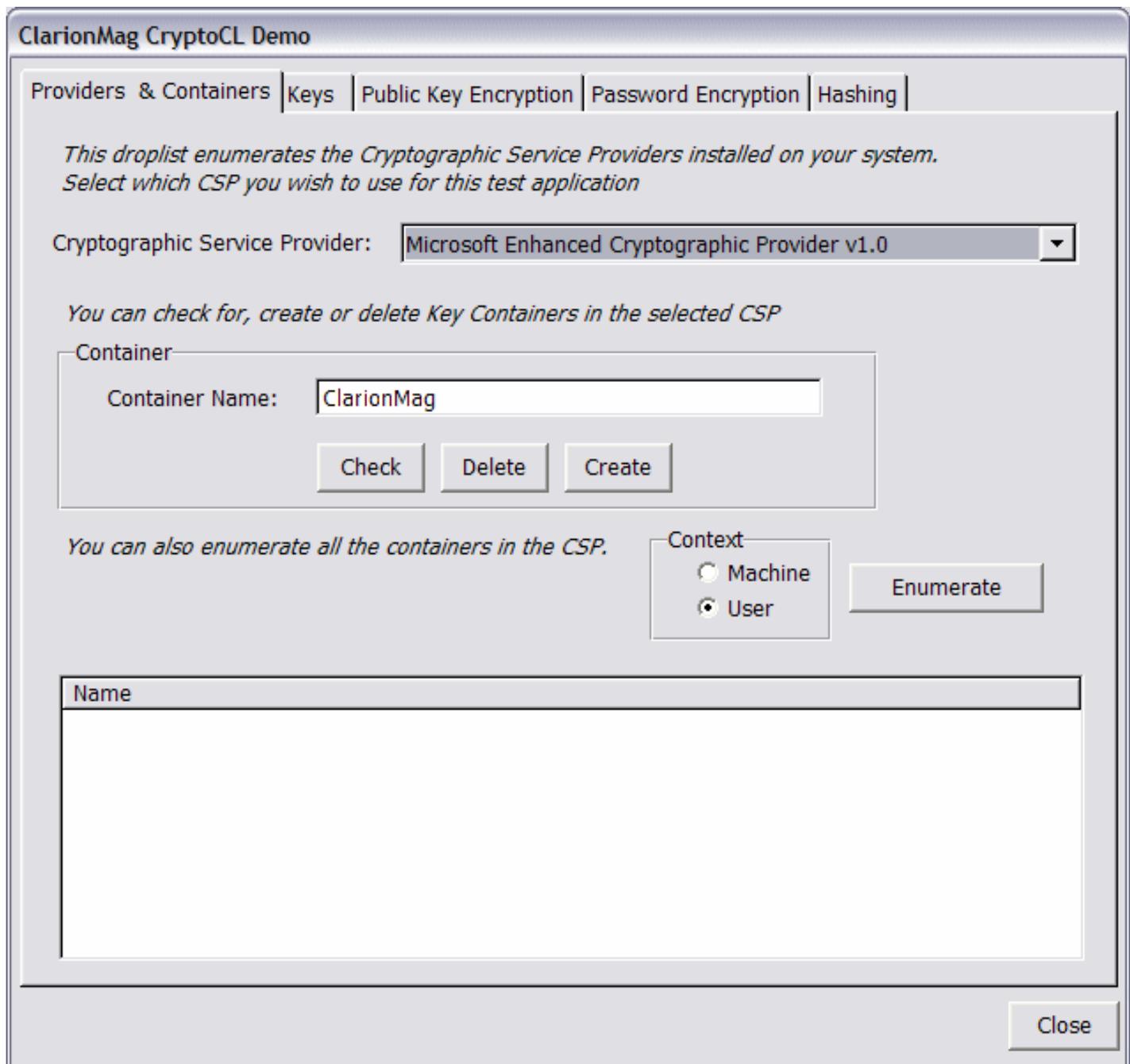
PasswordLen IN - Length of the password

InHash IN - Address of a string holding the hash

InHashLen IN - Hash length.

I have included an example application which demonstrates the functionality of the `CryptoCL` class. The app instantiates an object of `cryptocltype` called `cryptocl` which is used throughout the app. I have tested instantiating multiple `cryptocl` objects on multiple threads and this appears to work fine. For this example, a single instance is adequate.

Compile and run the application and you should see the following screen. The Microsoft CryptoAPI Base Provider was first distributed with Internet Explorer 3.0 so it should be pretty ubiquitous by now.



The provider drop list box is populated with the CSPs installed on your machine. The code calls `CryptoCL.EnumerateProviders(ProvQ)` in the `OpenWindow` embed. `ProvQ` is a queue created of type `ProvQType` which is declared in the include file. The method calls the CryptoAPI `CryptEnumProviders` function to populate the queue. You can now select any CSP to use for subsequent testing (note that not all CSPs support the types of encryption the code uses). The droplist sets the `CryptoCL.Provider` and `CryptoCL.ProviderType` properties.

Having selected a CSP you can now start working with key containers. These are stored per CSP (Microsoft seems to share containers between some of their CSPs). As I explained last week, containers also have a per user or machine-wide context. This version of the `CryptoCL` class only caters for user context although it will

enumerate containers for both user and machine contexts. If you need to support machine context (which you would need to do if you wanted multiple users to access common keys on a single machine) you change this by changing the `AcquireContext` calls in the methods from;

```
if ~CryptAcquireContext(SELF.hCryptProv, SELF.Container, |
    SELF.Provider, SELF.ProviderType, 0) THEN
```

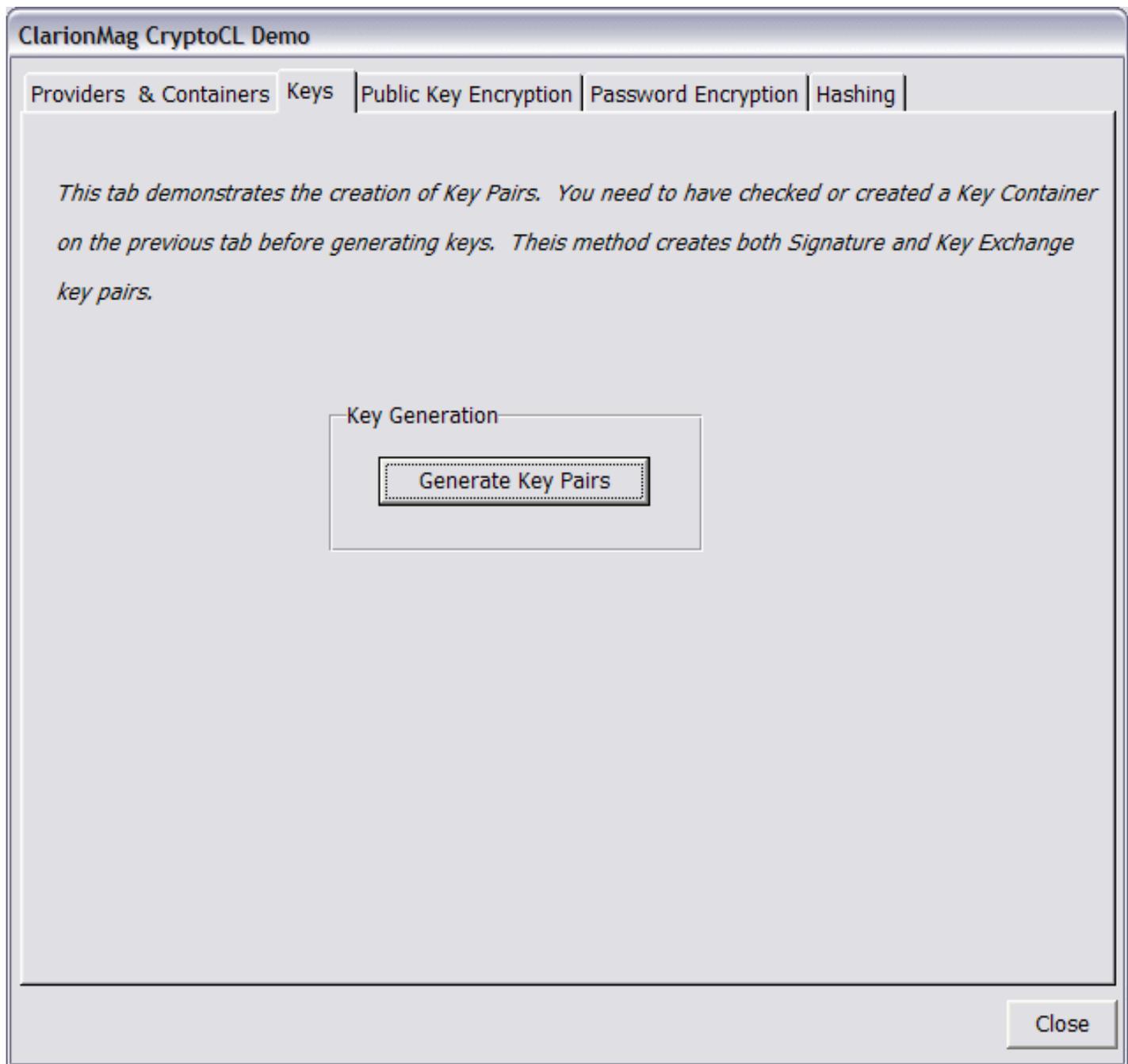
to

```
if ~CryptAcquireContext(SELF.hCryptProv, SELF.Container, |
    SELF.Provider, SELF.ProviderType, CRYPT_MACHINE_KEYSET) THEN
```

Three container management methods are provided; `CheckContainer`, `CreateContainer` and `DeleteContainer`. I have supplied a default container name of `ClarionMag` for test purposes which you need to create by clicking on the create button.

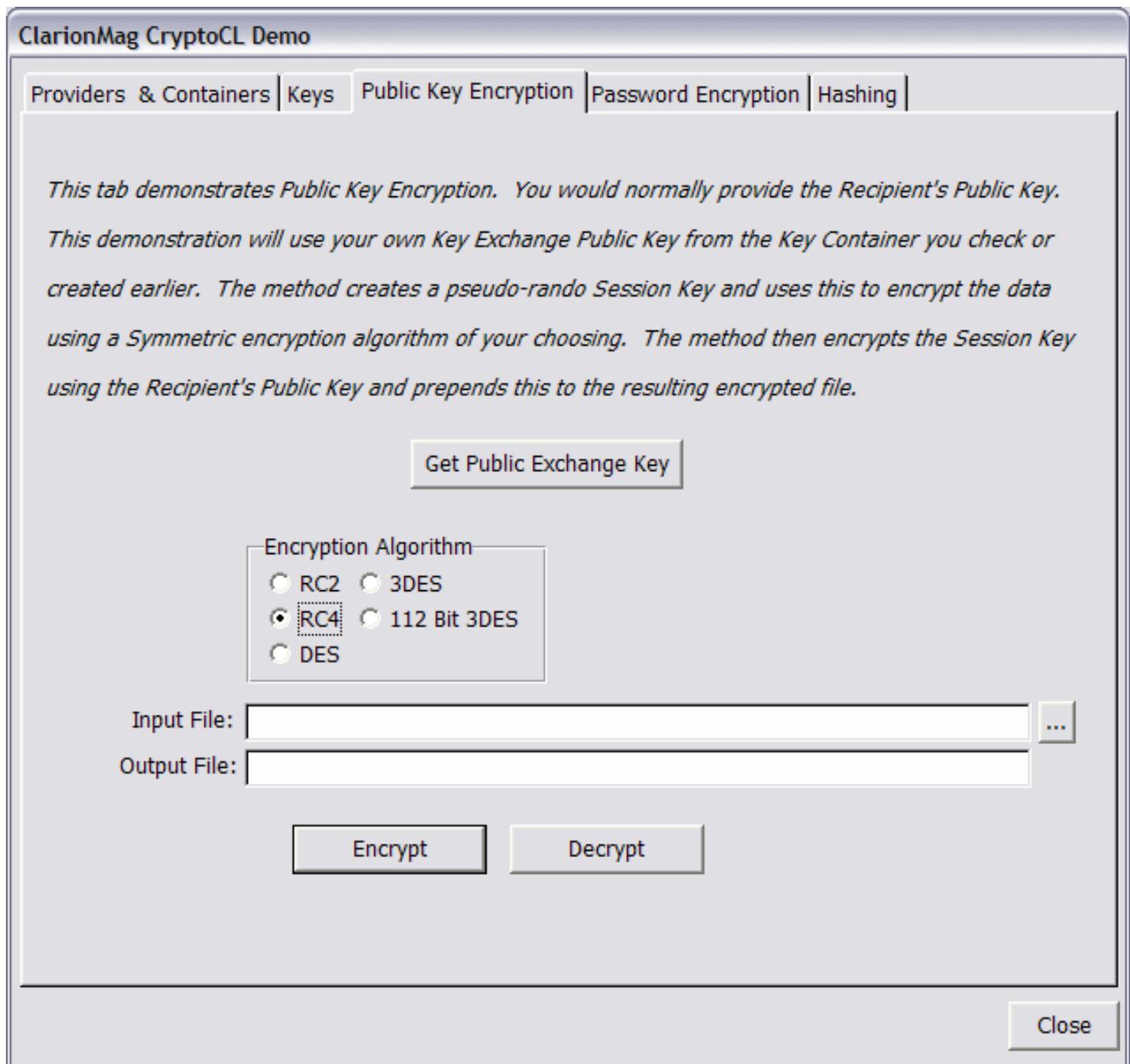
The enumerate button lists container for the selected provider and calls `CryptoCL.EnumerateContainers` which populates `ContQ` declared from `ContQType`. Enumerating machine context containers is much slower than user context containers.

The **Keys** tab allows you to create key pairs in the selected container. When you create a new container, it is empty so needs to be populated with keys before you can use it.



The `CryptoCL.GenKeyPairs` method generates both Key Exchange and Signature Key pairs for use with Public Key Cryptography and stores them in the container. You can also store persistent session keys or even public keys in a container although the `CryptoCL` does not currently support this.

Now on to some actual encryption!



To perform public key encryption you require the recipient's Public Key Exchange Key. The code retrieves your own Public Key Exchange Key from the container so that we can actually decrypt the file as well! This calls

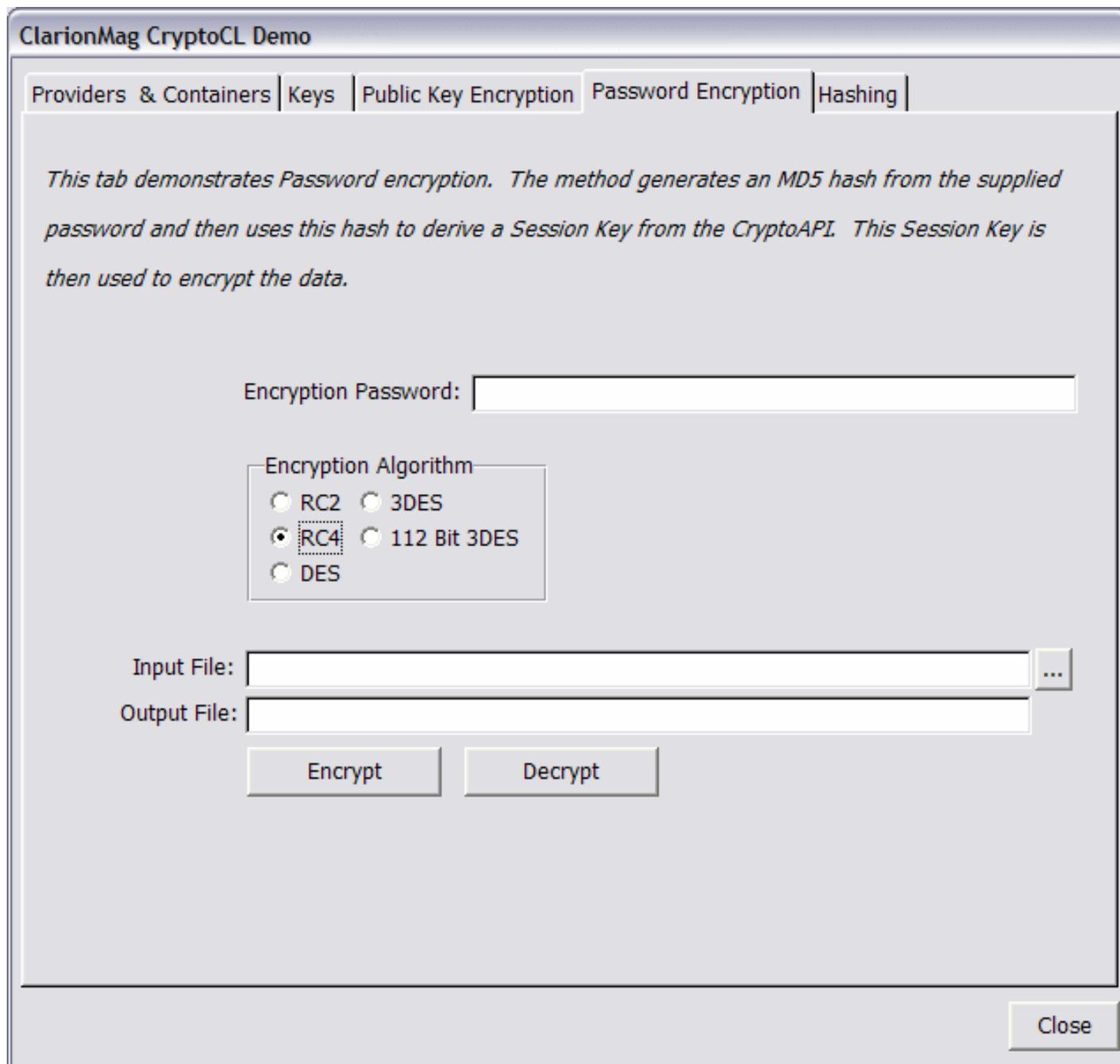
`CryptoCL.GetPublicKeyBlob` and asks for the Key Exchange public key which it exports to a string.

You can now select an encryption algorithm which sets `CryptoCL.AlgID`, provide input and output filenames before encrypting which calls

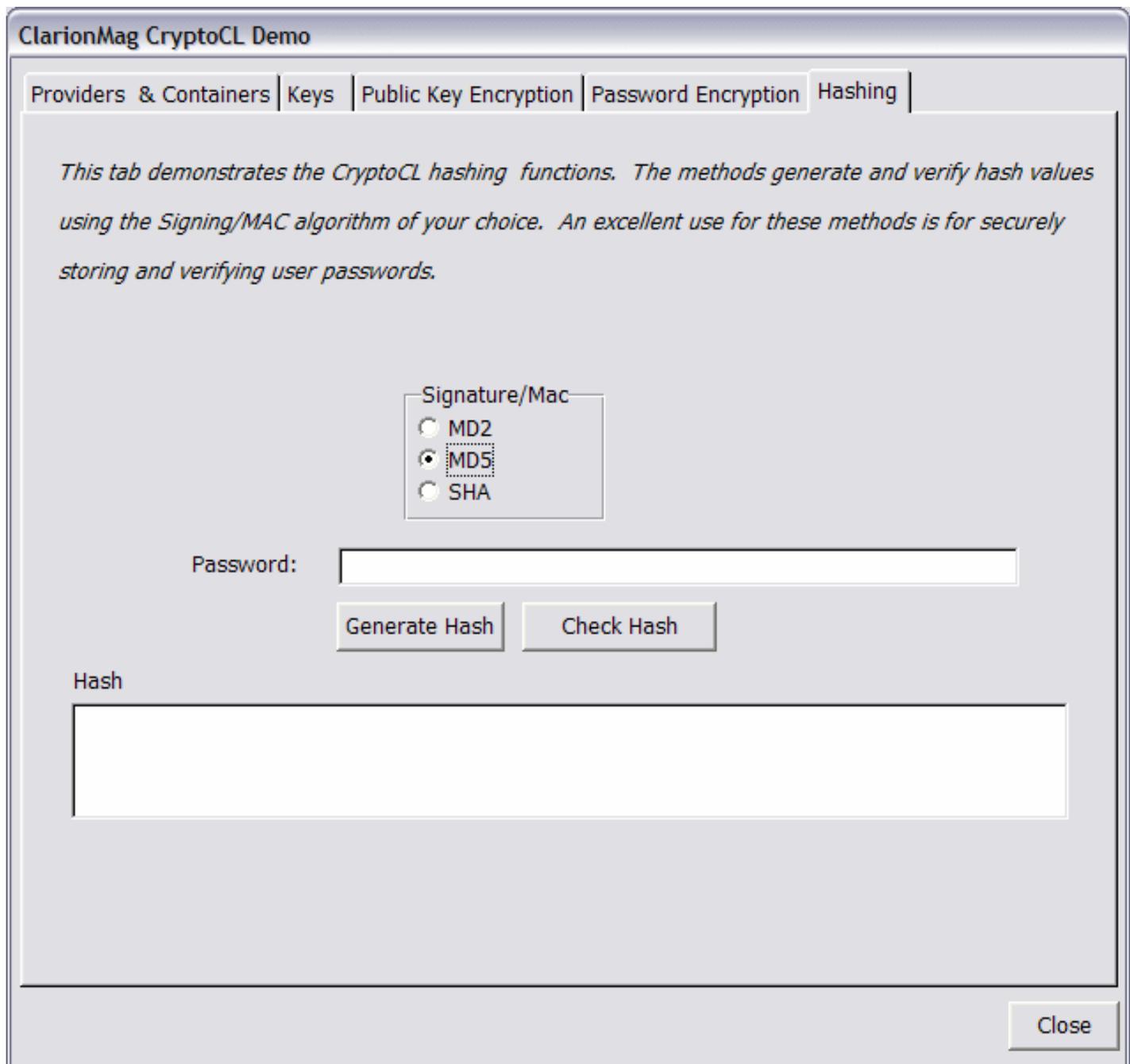
`CryptoCL.SessionEncryptFile` or decrypting with `CryptoCL.SessionDecryptFile`.

The Password Encryption tab demonstrates the `CryptoCL.PasswordEncryptFile`

and `CryptoCL.PasswordDecryptFile` methods. You do not need any key pairs to use password encryption.



Finally, hashing is demonstrated where a hash is generated from a password and can then be validated calling `CryptoCL.HashPassword` and `CryptoCL.CheckPasswordHash`.



These functions are ideal for securely storing passwords. Create a hash and store the hash rather than the password in the user table or similar.

I feel that I have barely scratched the surface of cryptography, but hope that the bit I have scratched is of use. If there is sufficient interest I will expand on the `CryptoCL` class to include support for string/blob encryption, certificates, certificate stores and PKCS#7 messages.

[Download the source](#)

[Ron Webb](#) was born in Zimbabwe and now lives in Kenya, East Africa. He has been using Clarion since CPD 2.1 and is the Technical Director and co-owner of [Paynet Limited](#) which provides financial transaction solutions to banks and corporate users in Africa. When not programming, he spends his time enjoying the great African outdoors and not enjoying golf! Ron is married to Karen and they have four children.

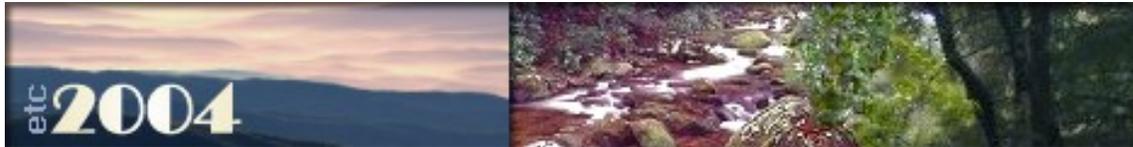
Reader Comments

[Add a comment](#)

Great article about encryption. Very useful to get...
Much needed Article!!! With the demands put on IT staff...
The label warnings should not affect functionality, but I...

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

[Clarion Magazine](#)



Migrate Your Topspeed Application To Firebird, Part 1

by Jimmy Rogers

Published 2004-05-27

About two years ago I worked on a template to migrate my Topspeed applications to Interbase 5.0. The price of the ODBC driver was what eventually made me stop work on this project. Now that an open-source ODBC driver is available and Interbase has migrated to a much richer Firebird 1.5 I decided it was time to start testing this SQL code again.

My template worked but needed to allow more options. After making my changes it appears the template now does what is needed to help create the tables for the Firebird database, change the application dictionary and write conversion files to move the Topspeed data to the Firebird tables.

The supplied template can do almost everything you need to at least get your existing application up and running for testing in Firebird. You will need to follow these steps:

1. You will need to [download](#) and install Firebird.
2. Load the [open source ODBC driver](#).
3. You will need to create the database yourself and run the CREATE TABLE scripts generated by the template. You can read more information on the scripts below. I used the [IBExpert](#) personal free version for this.
4. You will need to run the utility to change an exported TXD (more

later) from an existing dictionary and import this into a new dictionary, which you will use for your existing application. More information on this below.

5. Some third party templates may need to be removed. I had to disable Capesoft FM3 and Capesoft SecWin just for testing. SecWin worked two years ago while I was testing Interbase, but I have not tested with Firebird.
6. Local and global data may need to be changed (e.g. use Date and Time instead of Long).
7. Run the utility to create the data conversion files. Make and run the one conversion file named CompileAndRunAllConverts.Exe. More information on this below.
8. Recompile your existing application with the new dictionary. Set your new ODBC driver connect string and test your application.

For testing purposes I wanted to use the standard ABC browse. Some of my browses have a slight delay. This is with no indexes created from the scripts in Firebird. Kelvin Chua tells me that his [Icetips CCS SQL Template](#) browses are fast. The point is your browses that use the standard templates are not going to be at their best performance. Be aware of this in your testing.

Some people import their script-created database into a Clarion dictionary. That will break an existing application if the database column names that were created from this template use the external names in the old Clarion dictionary. This fix may be just some minor change to the application or the problem may be more serious, since the id numbers in the dictionary link to the application. The only way I know of to get around this is to export the application as a TXA and then import into a new application.

The SoftVelocity [news groups](#) are the best place to find Clarion-related information on Firebird or SQL.

Thanks to Roberto Artigas Jr. for his Reserved Words template work, which I added to this template.

Thanks to Terry Ho for testing out the template, and thanks to Kelvin Chua for his time spent helping with Firebird.

Some Firebird Information/Links

First of all, here are some links to general information about Firebird.

- The Firebird [home page](#)
- Firebird 1.5 [downloads](#), including release notes. This is a must read to see the latest changes and Microsoft Windows specific communication-related parameters.
- Firebird 1.5 [What's New](#)
- [Article](#) comparing Interbase/Firebird To MSSQL article. Please note that Firebird is *free*.

Check out the following newsgroups on [news.atkin.com](#):

- egroups.firebird-tools
- egroups.ib-support

There is also a support group on Yahoo:

- <http://groups.yahoo.com/group/firebird-support/>

There are several [mailing lists](#) at SourceForge, including sourceforge.firebird-dev and sourceforge.firebird-odbc-devel.

You can find additional Firebird links at [Fingerbird](#).

[Case Studio 2](#) is a popular CASE tool which supports Firebird.

Connection Information

Here's how to create a DSN-Less connection using the open source ODBC driver. The following connect string means that a data source is not required:

```
GlobalConnectVariable = 'DRIVER=Firebird/InterBase(r) driver;' |  
& 'UID=SYSDBA;PWD=masterkey;DBNAME=myServer:/path/myDatabase.fdb'
```

Here's an example of my connect string

```
Glo:dbOwner = 'DRIVER=Firebird/InterBase(r) driver;' |  
  & 'UID=SYSDBA;PWD=masterkey;      '|  
  & 'DBNAME=Bigdog:D:\C55\Apps\Fxits\Firebird\fxits.fdb'
```

If you install the open source ODBC driver and have a problem with it follow these steps:

1. Turn on the windows ODBC trace not the Clarion.
2. Zip the trace with detail information about what the problem is
3. Post on the news group news.atkin.com at sourceforge.firebird-odbc-devel. For instance, I have communicated with [Vladimir Tsvigun](#) about a date time problem that he then fixed.

Step 1: Generate Topspeed Table Structures

If you are going to convert your Topspeed data to Firebird then you will have to run the Step 1 utility template, shown in .Figure 1.

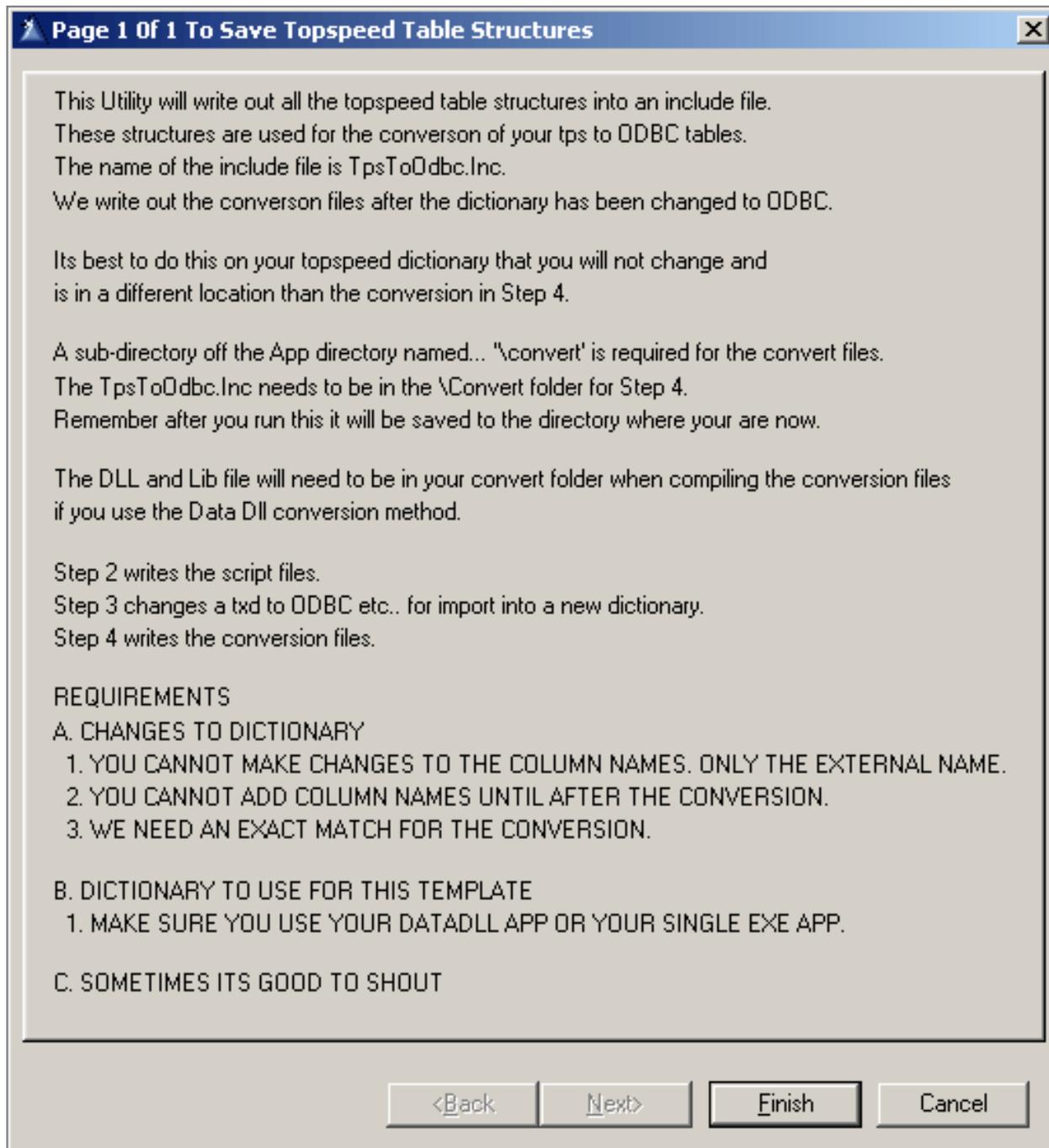


Figure 1. Page 1 of Step 1 – saving your Topspeed table structures.

This utility needs to be run from an Application that uses your dictionary with your Topspeed tables. It can be an EXE or a DLL application. The template will save the table structures in an include file named TpsToOdbc.Inc. This file is used when creating the convert files for each table.

It would be best to copy the dictionary with your Topspeed tables to the \Convert folder. Create this folder if it does not exist. Copy or

create an application that uses this dictionary so you can create the TpsToOdbc.Inc file in the proper location.

If you are going to use a data DLL for your conversion then you can copy an existing application to the same folder as the dictionary or create a new application. By compiling the DLL in the convert folder you have the LIB and DLL files in their proper location.

Step 2: Creating The Firebird Scripts

The utility will create scripts for ODBC or Topspeed table types only. If you create a UserOption of LOCALFILE for a table, no script is created for this table. The USEROPTIONS are checked for LOCALFILE and this table is skipped if found.

There are conditions for creating the Firebird scripts. If you click the check box for **Check for conditions below and if found cancel writing script files?** then all errors are written out to a file named SqlErrors.Txt: No script files are created.

Always use the check box that prints a report of any found problems. Keep using this until you have no more problems in the dictionary. You will have no more errors because you manually change the dictionary, or you let Step 3 change a TXD of the existing dictionary and import the new TXD into a new dictionary. Run the Step 2 Utility again to see if any errors have been removed from the import of the new TXD.

The most important thing to remember is if you are going to be using an existing application's dictionary then you do not want the script file column names to be different than the dictionary column names or the external column names.

These scripts use the external name of the column if one exists. If you import a table from your database into your dictionary and the table had external column names in the dictionary before, you could have just broke your existing application. You for sure broke your future

conversion files. This is easy to modify, so just know what happened and why.

The following conditions are checked

1. Reserved words are not allowed - use the dictionary utility to correct.
2. EXTERNAL NAMES are checked for UPPERCASE - use the dictionary utility to correct
3. Field Names > 27 characters. Maximum in Firebird for object names is 31. You will need to change this manually in the dictionary by using the Name attribute to shorten the name. If you end up with less than 31 characters you can still use the name. The Firebird docs say that some system object names that are created add a prefix to the name so be warned.
4. No primary key in a table in the dictionary is not allowed. It will cause errors at runtime. You will need to add a Primary key to this table manually. Remember if you add one that the Topspeed conversion structure does not have you will probably need modify this conversion procedure. You will get an error when it compiles.

Dictionary Settings

There are specific settings that can be set in the dictionary that will determine how the scripts will be written. You may need to look at some of your scripts and see how they were created. For instance:

Initial values

The template takes a special approach to initial values in the dictionary columns. Initial values become a Firebird default for the column in the script.

1. You can check to not allow them to be created in the Firebird script. This is the default.
2. Set a filter on your existing values and convert them to something Firebird understands, or make it not create a default for just this condition. Any not filtered pass thru as is. For example:

```
Today( ) = 'NOW'
```

will make all initial values that have Today() convert to default 'NOW'.
Or consider a year conversion:

```
YEAR( Today( ) ) =
```

All initial values that have YEAR(Today()) will, in this case, be converted with no default value for the column.

Use the report for all initial values in the dictionary to determine what to set the scripts for. If there is an initial value then a default is written into the script. This will only happen if you check this option on.

Boolean values

You can turn off script writing of validity checks, except Cannot Be Zero Or Blank. The template writes out your validity checks into the scripts. Bad dictionary entries could cause a problem. The template assumes a string Boolean is not surrounded by single quotes, and so adds the quotes.

In the dictionary, 'Y' is *not* allowed, but Y is.

If there is an initial value, then a default is written into the script if you uncheck this check box.

Must be in Numeric Range

The template writes numeric range checks as a Firebird check on the column. If there is an initial value then a default is written into the script.

Must be in List

The template writes "must be in list" checks as a Firebird check on the column. If there is an initial value then a default is written into the script.

Utility Options And Prompts

The following figures show the template prompts for Step 2.

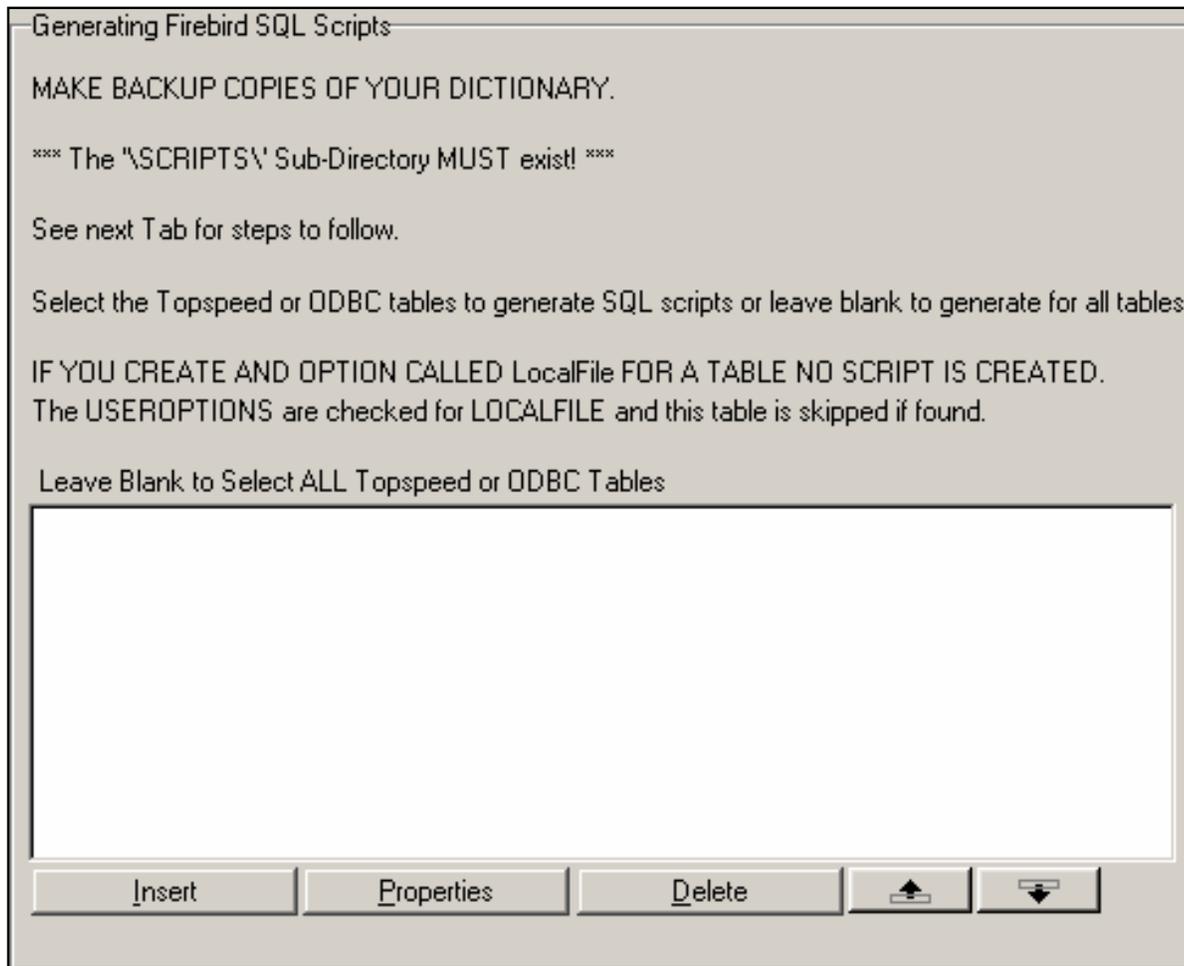


Figure 2. Page 1 of Step 2 – creating the scripts.

Conditions when creating Firebird scripts

Name of the SQL file that is created for errors - SqlErrors.Txt:

Check for conditions below and if found cancel writing script files?

Conditions Checked

1. Reserved words are not allowed - use EXTERNAL NAME to correct.
2. EXTERNAL NAMES are checked for UPPERCASE.
3. Field Names > 27 Characters. Maximum in Firebird for object names is 31.
4. No primary key in a table in the dictionary is not allowed. It will cause errors at runtime.

** Template ASSUMES a string Boolean value has no ' on say the Y N or a T F and adds them.
This is NOT allowed 'Y' but this is Y. The Initial Value entered value must have ' around it.

The dictionary changes are described in Step 3.
Step 3 makes changes in a txd file for import into a new dictionary.
Always use the check box that prints a report of any found problems.
Keep using this until you have no more problems in the dictionary.

Remember that if you want your scripts to allow an existing application to work for testing then you want the column and file names to match with the dictionary.
Use the external name attribute on your columns or let this template do it for you.

Figure 3. Page 2 of Step 2 – creating the scripts.

Conversion Settings

Be consistant with the next 2 column Date and Time types.
If you do not change them to a Date and Time type in the Firebird Script then do not change them in Step 3 that changes the TXD.

Change All LONGs With @D To DATE field type
 Change All LONGs With @T To TIME field type

Be consistant with the next column type change.
If you do not change them to a BLOB type in the Firebird Script then do not change them in Step 3 that changes the TXD.

Make STRING CSTRING PSTRING above this size a BLOB for text
Example: 2000 makes any of these strings that are 2000 or more in size into BLOBS
Enter a number or uncheck the box

Figure 4. Page 3 of Step 2 – creating the scripts.

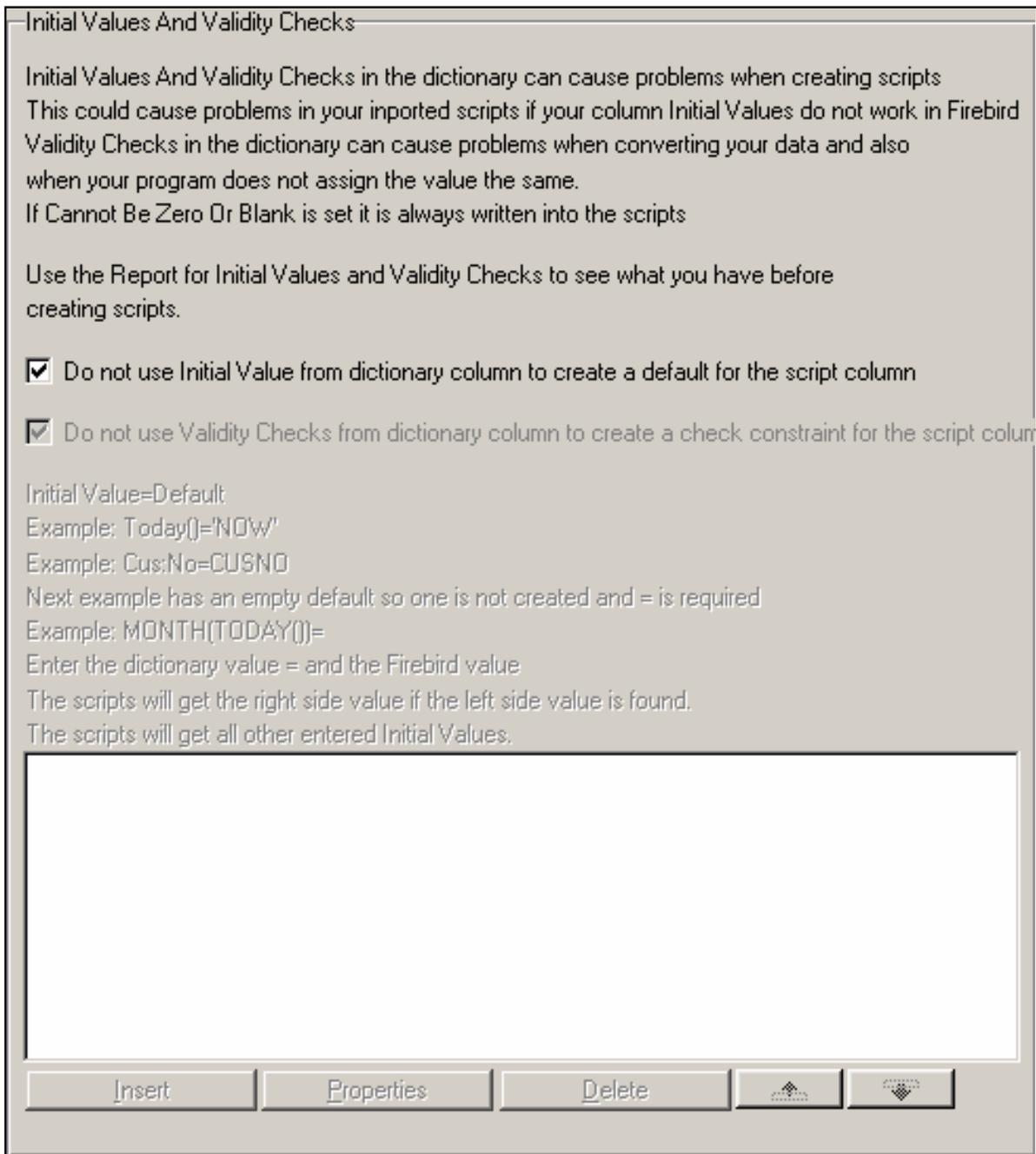


Figure 5. Page 4 of Step 2 – creating the scripts.

Listed below are the options you have for creating the Firebird scripts. Step 3 allows you to change a TXD of your dictionary. When you import the new TXD into a dictionary some of the options here will already be taken care of. You may want to create some scripts before Step 3 so you can see what errors you may have in your dictionary. You still should look at Step 3 to see what changes can be made to your dictionary.

1. Select Topspeed or ODBC tables

2. Check for conditions below and if found cancel writing script files
3. Change All LONGS With @D to DATE field type
4. Change All LONGS With @T to TIME field type
5. Make STRING CSTRING PSTRING above this size a BLOB for text
6. Enter the size
7. Do not use Initial Value from dictionary column to create a default in the script
8. Do not use Validity Checks from dictionary column to create a check constraint in the script

Date and Time

Column types that are a long and have a @D or @T picture type can be made to change to a Firebird Date and Time type.

Be consistent with this. If you do not change them to a Date and Time type in the Firebird Script, then do not change them in Step 3 that changes the TXD.

Memo and Blob

You have the option to change all memos to a blob in Firebird. They cannot stay a memo or blob in the dictionary. So you must manually change them or let Step 3 change the memo to a string in the txd which is the correct data type for Firebird.

Blobs of binary in the dictionary are changed to a not text blob in Firebird.

Strings, Cstrings and Pstrings

You have the option to change all of these strings types above a specific size to a Firebird blob.

Example: Using a value of 2000 makes any of these strings that are 2000 or more bytes in size into a text BLOB

If you do not want them to change to a Firebird BLOB type then do not change them in Step 3 that changes the TXD.

Auto numbering

Triggers are created for auto numbering. There are two types of triggers created. One is set to increment the primary numeric index using a generator before insert. The other one is the same except it adds a condition to the trigger that checks if the primary key column is Null. If it is Null then the generator gets the next primary key column number. If it is not null then it does nothing. This should work together with another Stored procedure script that is created by the template to return a parent generator primary key value so you can assign to the children and parent. When the parent trigger gets called for the insert, the backend trigger will not assign a new value but use the already assigned value. This is one way recommended in Firebird if you use a trigger with a generator.

Next week I'll look at the scripts the template creates.

[Download the source](#)

[Jimmy Rogers](#) has been using Clarion since the 2.1 days. His projects include manufacturing and sales software for hot sauce and a software product for dispatch and tracking of fire extinguisher equipment. Jimmy is presently involved in the never ending sales and improvement to his consignment retail software. He is always waiting to win the lotto.

Reader Comments

[Add a comment](#)

Copyright © 1999-2004 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.