Clarion Magazine

INVEST
in your own abilities
Clarion magazine

Welcome To The New ClarionMag Server

You have reached Clarion Magazine at its new server location! Along with the new location ClarionMag is sporting some refactored and upgraded code under the hood. Those changes have been tested and everything should be working normally, but if you notice anything that's broken please send an email to webmaster@clarionmag.com.
Dave Harms, Publisher
Posted Monday, July 28, 2003

Clarion Magazine Summer Schedule

The ClarionMag publication schedule is a bit irregular at the moment as I'm in the process of upgrading both the server and the server software. The next issue of Clarion Magazine will most likely be a double issue the first week of August. Following that I'm off for a bit of a holiday, but everything should be back on an even keel in September.
Dave Harms, Publisher
Posted Friday, July 25, 2003

Weekly PDF For July 13-19, 2003

All ClarionMag articles for July 13-19, 2003 in PDF format.
Posted Monday, July 21, 2003

A C6 Tagging Class Template Wrapper

Steve Parker recently published an article on an ABC compliant class for tagging on browses. When someone requested a template-enabled version, Jim Katz volunteered his services, and along the way discovered that creating a C6 ABC

Articles: XML

News: XML

Clarion Magazine **subscriptions** (online only) are $49 for six months, $95 for one year and $170 for two years.

Subscribe **Renew**

SoftVelocity

News

Icetips Web Site Rebuilt

xToolTip v1.3

UK Clarion Training Special Offer

New INN Bio, News Site

Insight Graphing 1.15 Beta

xAppWallpaper Manager v1.8

SURVEY

Which is your first choice for Clarion Internet application development?

Internet Connect	3.4%
Web Builder	2.3%
Clarion/ASP	9.2%
ClarioNET	11.5%
Fenix	20.7%
Other	11.5%
None	41.4%

87 responses

Previous Surveys

class template wrapper is easier than it used to be.
Posted Thursday, July 17, 2003

[Breaking Reports On Computed Fields]

Henry Plotkin is assigned a new report, "Open invoices, aged by date." Only the Invoice file contains fields for invoice number, date, amount, previous balance, payments and debits. No "open" flag. No "age." Henry ponders how to filter, order and group the report records without having those fields at hand.
Posted Thursday, July 17, 2003

[String Flinging Part 2]

What uses most of the memory in typical Clarion applications? Strings, most probably. And you may not realize that you're wasting a lot of memory by how you handle those strings. Jim Gambon digs into strings and comes up with some memory- and time-saving tips and techniques. Part 2 of 2.
Posted Tuesday, July 15, 2003

[String Flinging Part 1]

What uses most of the memory in typical Clarion applications? Strings, most probably. And you may not realize that you're wasting a lot of memory by how you handle those strings. Jim Gambon digs into strings and comes up with some memory- and time-saving tips and techniques. Part 1 of 2.
Posted Friday, July 11, 2003

[Clarion, COM, Soap, and HTTP]

Recently Shane Vincent wanted to add a no-cost solution to the credit card authorization process. His solution involved a couple of open source COM components, Jim Kane's COM interface generator, and a couple of Clarion wrapper classes.
Posted Thursday, July 10, 2003

[JaDu Technology Greenbar Templates Updated]

[SimTabTree Template 1.07 - Tooltips Added]

[xAppWallpaper Template 1.4 Clarion 6 Compatible]

[Fenix ASP.NET Generator Gold Release]

[PD Browse Button Lookup Version 60-04]

[xTipHotKey Class v2.3]

[UltraTree Summer Sale]

[PDF-XChange Scores Top Points In PCMag Review]

[xDataBackupManager Lite v1.5]

[Insight Graphing Beta 1.13]

[VirtualEIP Updated]

[CapeSoft Email]

[SealSoft xTipHotKey C6 Compatible]

[Backflash Half Price Sale]

[SimTabTree Template 1.06]

[Freeware xFunction Library C6 Compatible]

[Icetips Cowboy SQL 6.001]

One Year Ago In CM

[Another Look At Saving Printer Settings]

[Shrink-Wrapped Controls]

[The Program's Finished. Now What? (Part 3)]

Two Years Ago In CM

[Understanding Recursion - Part 1]

[Using The Web Browser OCX]

[Clarion News - August 2001]

Three Years Ago In CM

[Legacy to ABC: There is Another Way!]

[Clarion News - July 2000]

[Clarion 5.5 Gold Candidate: A First Look]

Four Years Ago In CM

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.

qbFUSE 1.3 Supports QuickBooks SDK 2.1

cwODBC 1.10

solid.software Holiday Schedule

EasyExcell 2.00 Bonus

Compad Holiday

New Icetips Bio Archive page

BoTpl Updated

xDataBackupManager Light v1.4

EasyResizeAndSplit 1.09

ClarionPost.com Links Exchange

Buggy 3.2.3 Available

Gitano Software Mail

xToolTip's Fixed Distribution Kits

New Icetips Xplore Build Available

VAT On Icetips Products

Icetips SQL Templates 6.0

RADFusion Site Updated

ImageEx 2.2 Released

Working With Control Files I

Clarion Challenge String Parser *Final* Results

Clarion News - July 1999

SQL Seminars Selling Out

SimTabTree Template 1.05

Contract Programming Site

CPCS C6 Beta Recompiled With EA5

Clarion6 EA5

Whitemarsh News

Clarion Book Released

Search the news archive

Clarion Magazine

Reborn Free CLARION online

Clarion Magazine's Publication Schedule

Published 2001-05-02

Clarion Magazine is a weekly magazine, published 48 times per year, with articles and news items published throughout the week. Because the immutable laws of mathematics say that 52-48=4, there are normally four weeks per year when Clarion Magazine doesn't publish. Occasionally, Clarion Magazine may skip publication on other weeks due to extenuating circumstances. In this event we may make up the missing issue in a double issue. If there is no subsequent double issue, don't worry - all subscriptions are automatically credited for missing issues (or to put it more accurately, subscription credits are only debited when an issue appears, but that's probably more information than you needed to know).

Weekly PDFs appear the Monday following, and the weekly summary notices are also emailed on Mondays (except when the Monday falls on a statutory holiday, or, to be honest, for any arbitrary reason deemed sufficient by the publisher).

Subscriptions begin on the first day of the month of your subscription purchase.

If you have any questions about your subscription, contact subscriptions@clarionmag.com.

Clarion Magazine

A C6 Tagging Class Template Wrapper

by Jim Katz

Published 2003-07-17

Steve Parker recently published an article on an ABC compliant class for tagging on browses. When someone requested a template-enabled version, I wrote Steve and volunteered my services in writing such a beast. Steve sent me his class files and a sample application.

Two things struck me when I looked things over. First, Steve was using two buttons to control the tag and untag operations on the browse, and second, the class was declared in the global data embedpoint. I like to use a single button for both tagging and untagging, and to set the button text according to whichever record is currently selected in the browse. Declaring the class globally is not always a bad thing, but it is sometimes necessary to declare it locally so that the tag queue does not interfere with other instances of the same browse procedure using the tagging operation.

Changing the tagging class

The solution to the first issue meant I needed to change the class files. In the class include file (shptag.inc) I added a new method to the `ShpTagClass`, which you may notice I capitalized: the ABC Viewer is case sensitive! The new method is called `SwapTag`, which does just what it says: it changes the record's state from either tagged to untagged or from untagged to tagged. The method code, which you can find in the source file (shptag.clw) and simply combines the `MakeTag` and `ClearTag` methods' code within an `IF-Then` structure.

The second issue, declaring the class locally, is the template's job.

Creating the template

To begin the new template I did the same thing I do whenever I start a new template from scratch: I open up an existing template that has most of the same elements as the new template I envision. Copy and paste is always the simplest way to start a new template.

Since this is a standalone template, not part of a chain, I started the first line with `#TEMPLATE`:

```
#TEMPLATE(JKSHPTaggingWrapper,'JK Template Wrapper ↵
    for SHP Tag Class'),FAMILY('ABC')
```

This project lends itself to a `CONTROL` template, i.e. the tagging button(s) are the Controls I'll place on the window. The tagging control is also dependent upon an ABC browse box, so it needs a `REQ` attribute:

```
#CONTROL(JKSHPTagButtons,'JKSHP Tag && Untag Buttons'), ↵
    DESCRIPTION('Tag/Untag buttons for: '&%JkShpListControl),↵
    REQ(BrowseBox(ABC))
```

Next I needed some basic symbols defined and set to handle this ABC compliant tagging class. The first one I generally call is `%OOPHiddenPrompts`, which is part of the ABC template chain. I call it like this:

```
#BOXED('Default Tag Class prompts'),AT(0,0),WHERE(%False),HIDE
   #INSERT(%OOPHiddenPrompts(ABC))
#ENDBOXED
```

The next one is the `%SetClassDefaults` along with the usual obligatory `%ReadABCFiles`:

```
#PREPARE
  #CALL(%ReadABCFiles(ABC))
  #CALL(%SetClassDefaults(ABC),'ShpTagClass','SHPTagObj:'& |
%ActiveTemplateInstance,%ShpTagClass)
#ENDPREPARE
```

These statements do a couple of things. First, they ensure that the ABC class can be found by the system, and second, they set the specified class or classes to be used by the template for generation of the virtual method embeds.

In the second `#CALL` statement, the second parameter is the class name exactly as defined in the include file (.INC). The last parameter representing the object type is purely arbitrary. Next, I added the actual controls to be populated:

```
CONTROLS
  BUTTON('&Tag'),AT(,,47,12),USE(?TagRecord)
  BUTTON('&UnTag'),AT(,,47,12),USE(?UnTagRecord)
END
```

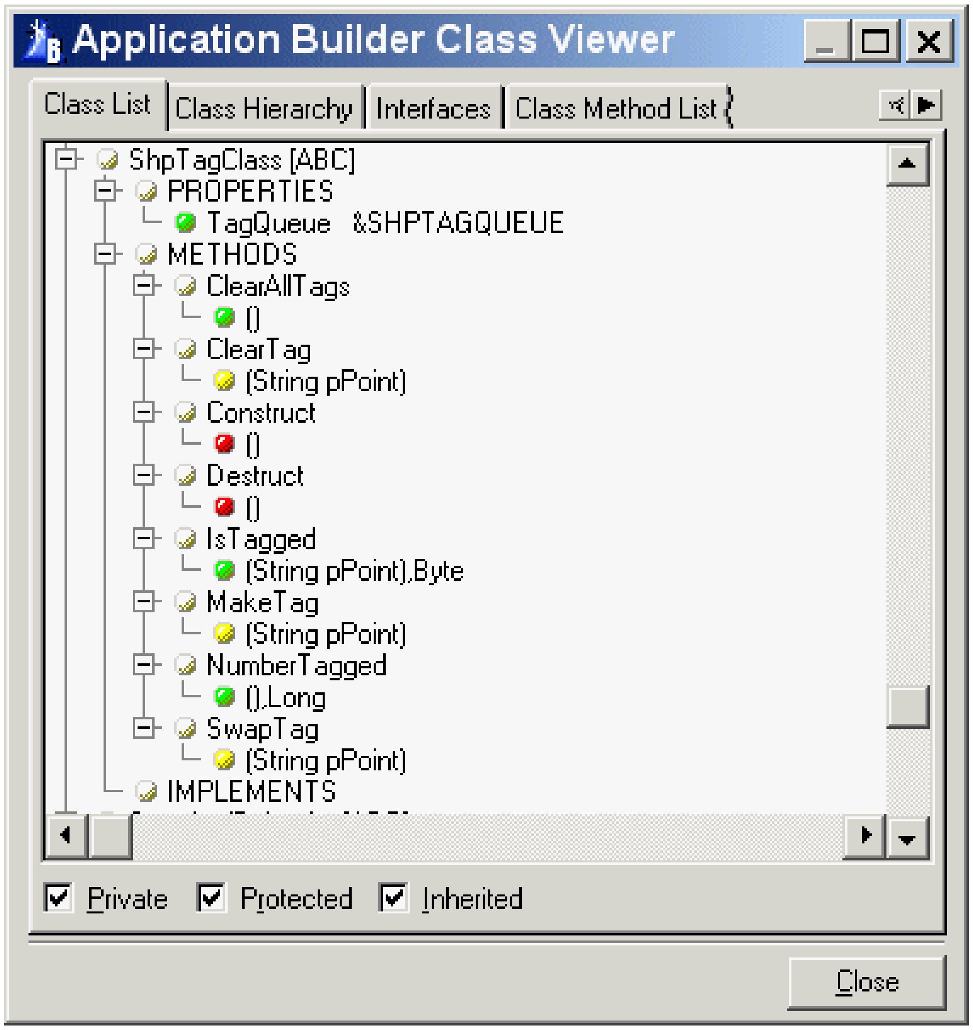
When I build the tab sheet for user input I like to start at the back, that is, with the class definitions, which as usual I mostly cut and paste from a convenient template: The `FROM` symbol, `%pClassName`, is a system symbol that is filled when the `ReadABCFiles` action occurs.

```
#TAB('Classes')
  #PROMPT('Default SHP Tag Class',  |  FROM(%pClassName)),↵
    %ShpTagClass,DEFAULT('ShpTagClass'),REQ
  #WITH(%ClassItem,'ShpTagClass')
    #INSERT(%ClassPrompts(ABC))
  #ENDWITH
#ENDTAB
```

As I said earlier, I want to extend the class by making it either Global or Local in scope, according to the developer's particular needs. So I add an option to the classes tab to allow for this scenario:

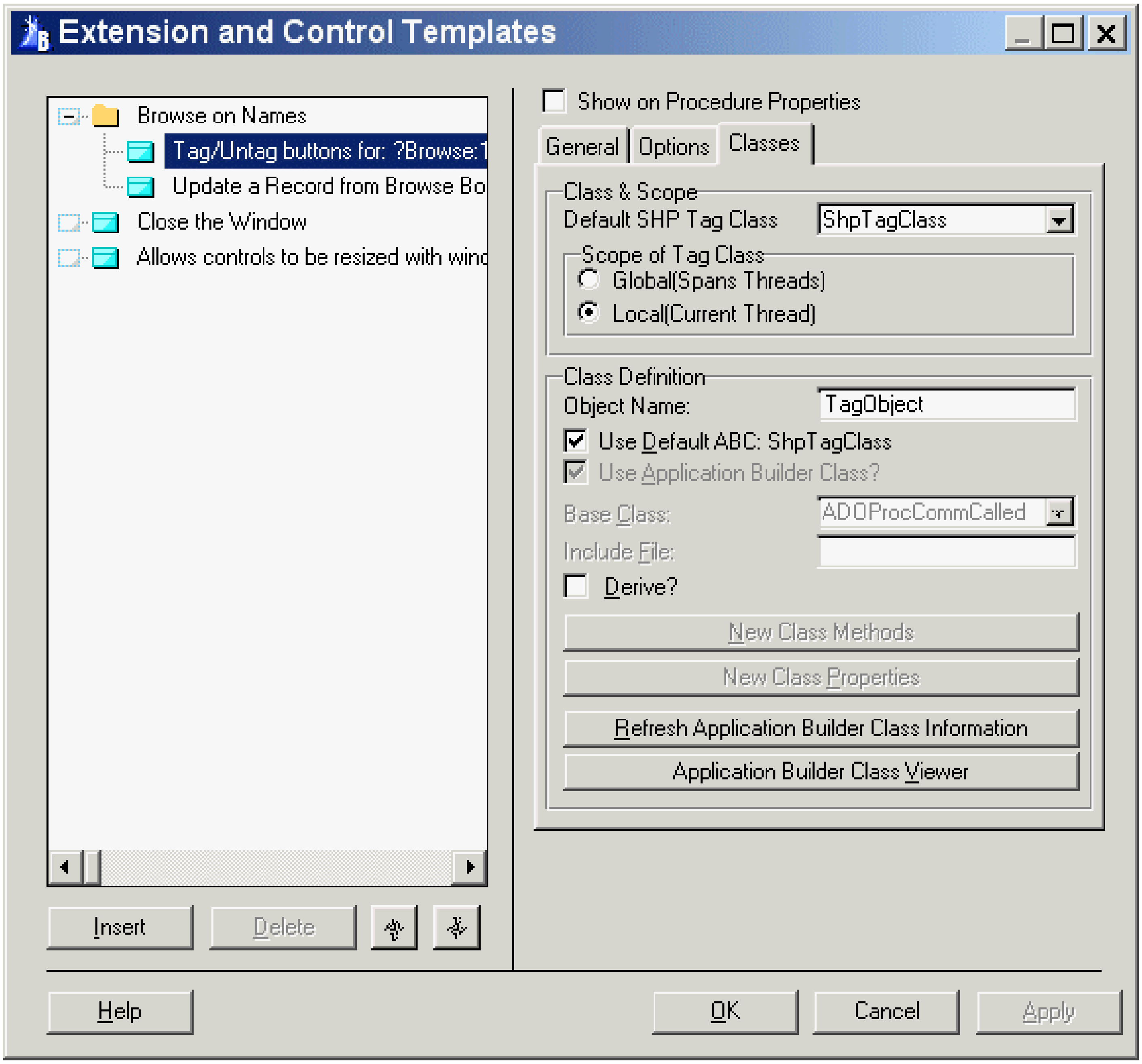
```
#PROMPT('Scope of Tag Class',OPTION),   |  ↵
    %JKShpTagClassScope,DEFAULT('Local')
  #PROMPT('Global(Spans Threads)',RADIO),VALUE('Global'),AT(15)
  #PROMPT('Local(Current Thread)',RADIO),VALUE('Local'),AT(15)
```

At this point I usually register the template and begin the continuous testing process. It's important that the ABC tagging class show up in the ABC Class viewer in the IDE (see Figure 1), and that it appears as expected when I place the control template on the appropriate ABC browse box (see Figure 2).



```
ShpTagClass [ABC]
  PROPERTIES
    TagQueue  &SHPTAGQUEUE
  METHODS
    ClearAllTags
      ()
    ClearTag
      (String pPoint)
    Construct
      ()
    Destruct
      ()
    IsTagged
      (String pPoint),Byte
    MakeTag
      (String pPoint)
    NumberTagged
      (),Long
    SwapTag
      (String pPoint)
  IMPLEMENTS
```

Figure 1. The tagging class in the ABC Class viewer


Extension and Control Templates

Browse on Names
Tag/Untag buttons for: ?Browse:1
Update a Record from Browse Bo
Close the Window
Allows controls to be resized with wind

[ ] Show on Procedure Properties

Class & Scope
Default SHP Tag Class: ShpTagClass
Scope of Tag Class
( ) Global(Spans Threads)
(●) Local(Current Thread)

Class Definition
Object Name: TagObject
[✓] Use Default ABC: ShpTagClass
[✓] Use Application Builder Class?
Base Class: ADOProcCommCalled
Include File:
[ ] Derive?

Figure 2. Placing the control template on a browse

At this point I was ready to set up some basic options. I'd looked through the embedded code and I'd seen references to the browse list box as well as the browse queue. Instead of using the name of the queue as defined by the ABC browse template, I decided to simply use a reference to it. The reference is a property of the browse object, so I need to let the developer choose which ABC object:

```
#TAB('Options')
  #PROMPT('ABC List box to  | tag:',Control),↵
    %JKSHPListControl,HSCROLL,PROP(PROP:DropWidth,140)
  #PROMPT('&amp;ABC Browse Object Name: ',COMBO(%ObjectList)),↵
    %JKSHPBRWSObjectName,REQ,HSCROLL,PROP(PROP:DropWidth,140)
```

The `%ObjectList` symbol is a multi-symbol in the ABC chain; to make sure it gets filled correctly I added a `#CALL(%BuildObjectList(ABC))` statement to the `#PREPARE` section. The `PROP(PROP:DropWidth)` is a neat feature to make those long symbols or controls readable.

Next I needed a unique ID field from the primary table to store in the tag queue:

```
#PROMPT('Browse Field to ID tags:',FIELD(%PRIMARY)),↵
    %JKSHPUniqueField
```

Earlier I described adding a new method to the base class to handle single button tagging, but I also need to use some template options to control code generation, as well as fill the use variable symbols for the controls. I also like to have different options for the "Tag/Untag" text on the button.

```
#PROMPT('Use Two | Buttons?',CHECK),%JKSHPTwoButtonTagging,↵
    DEFAULT(%True) |
#PROMPT('Tag Button:',Control),%JKSHPTagButton,REQ,↵
    DEFAULT('?TagRecord') |
#ENABLE(%JKSHPTwoButtonTagging)
#PROMPT('Untag | Button:',CONTROL),%JKSHPUntagButton,↵
    REQ,DEFAULT('?UnTagRecord')
#ENDENABLE
#ENABLE(~%JKSHPTwoButtonTagging)
  #PROMPT('Tag Text:',@S32),%JKSHPTagText
  #PROMPT('Untag Text:',@S32),%JKSHPUntagText
#ENDENABLE
```

Finally, I like to access the tagging functions from the popup menu for the browse box, so I added that option to the mix:

```
 #PROMPT('Add to Browse | Popup?',CHECK),%JKSHPAddToPopup,↵
    DEFAULT(%True),AT(10)
```

Now I was ready to start writing the template code that generates the source. The usual way to do this is to run through the code to make sure that the AppGen can find the proper ABC class:

```
#ATSTART
  #CALL(%ReadABCFiles(ABC))
  #CALL(%SetClassDefaults(ABC), 'ShpTagClass',↵
    'SHPTagObj:'&%ActiveTemplateInstance,%ShpTagClass)
  #SET(%TagObject,%ThisObjectName)
#ENDAT
#AT(%GatherObjects)
  #CALL(%AddObjectList(ABC),'ShpTagClass')
#ENDAT
#AT(%AfterGlobalIncludes)
   INCLUDE('ShpTag.INC'),ONCE
#ENDAT
```

This code handles the optional code generation dependent upon `'Global'` or `'Local'` class scoping:

```
#AT(%GlobalData)
  #IF(%JKShpTagClassScope='Global')
%TagObject       ShpTagClass
  #ENDIF
#ENDAT
#AT(%DataSection)
  #IF(%JKShpTagClassScope='Local')
%TagObject          ShpTagClass
  #ENDIF
#ENDAT
```

Next I added the code to create tag/untag options in the browse’s popup menu, using the user-filled symbols to do the heavy lifting.

```
#AT(%WindowManagerMethodCodeSection, 'Init','(),↵
    byte'),PRIORITY(9550),WHERE(%JKSHPAddToPopup)
%JKSHPBRWSObjectName.Popup.AddItem('-')
  #IF(%JKSHPTwoButtonTagging)
%JKSHPBRWSObjectName.Popup.AddItemMimic('Tag',↵
    %JKSHPTagButton)
%JKSHPBRWSObjectName.Popup.AddItemMimic('UnTag',↵
    %JKSHPUnTagButton)
  #ELSE
%JKSHPBRWSObjectName.Popup.AddItemMimic('Tag',↵
    %JKSHPTagButton)
  #ENDIF
#ENDAT
```

Here’s the template code to handle the **Tag** button, with the conditional handling for the single button tagging options:

```
#AT(%ControlEventHandling,%JKSHPTagButton,'Accepted')
 Get(%JKSHPBRWSObjectName.Q,Choice(%JKSHPListControl))
  #IF(%JKSHPTwoButtonTagging)
 %TagObject.MakeTag(%JKSHPUniqueField)
  #ELSE
 %TagObject.SwapTag(%JKSHPUniqueField)
  #ENDIF
 ThisWindow.Reset(1)
 Select(%JKSHPListControl)
#ENDAT
```

If you’re following this through the example application, you will see that most of the code looks the same as Steve's original code, which I've commented out. The template simply uses the user-filled symbols instead of hard coded variable names, wherever practical. Most of us leave the Window Manager's name as `ThisWindow`, I hope.

The untag button handling code is fairly simple:

```
#AT(%ControlEventHandling,%JKSHPUntagButton,'Accepted')
 Get(%JKSHPBRWSObjectName.Q,Choice(%JKSHPListControl))
 %TagObject.ClearTag(%JKSHPUniqueField)
 ThisWindow.Reset(1)
 Select(%JKSHPListControl)
#ENDAT
```

Finally I added some code to handle the conditional button text for the single button tag/untag. To get this code to generate in the proper place I look to the `%ActiveTemplateParentInstance symbol,` which in this case will refer to the ABC browse box to which the tag template has been added. I am using the the browse's event handler to trigger the text change. I use `#SUSPEND` and `#RESUME` to avoid generating any code unless single button option is used.

```
#AT(%BrowserMethodCodeSection,↵
    %ActiveTemplateParentInstance,'TakeEvent',↵
    '()'),PRIORITY(100)
  #SUSPEND
  #IF(~%JKSHPTwoButtonTagging)
 %JKSHPBRWSObjectName.UpdateBuffer()
 IF %TagObject.IsTagged(%JKSHPUniqueField)
   %JKSHPTagButton{PROP:Text} = '%JKShpUnTagText'
 Else
   %JKSHPTagButton{PROP:Text} = '%JKShpTagText'
 End
  #ENDIF
 #RESUME
#ENDAT
```

The template wrapper is just about done. Next week I'll show how to set up embed points for the tagging class.

Download the source

Jim Katz has been working in Clarion since 1987, when he discovered version 1.1 for DOS. At that time he was developing internal applications for his family business, the BMW motorcycle franchise in Daytona Beach, FL. It was quite a thrill when CPD 2.0 came out with the Designer. Jim modified the table model to allow reverse sorting, and this modification was offered on one of the freeware tool diskettes at the time as REVRSTBL.MOD. In 1993 Jim left the motorcycle business to work full time as a programmer for a small computer shop in DeLand, Florida, using CPD 2.1 and LPM, and then in 1996 became an independent contractor. In 1998 he took over the maintenance of the DET/DEF *products originally developed by Tom Moseley. Jim's current project is an internet enabled multi-player board game with the client in Java and the game server engine in Clarion. He currently resides on ten secluded acres in rural Deleon Springs, FL with two cats and an impressive number of wild critters.*

Reader Comments

Add a comment

Clarion Magazine

online sales and delivery
for your applications & tools
DeveloperPLUS

Clarion News

Icetips Web Site Rebuilt

The Icetips Software has been completely rewritten, and now focuses entirely on Icetips Software products (the INN pages are now at www.icetips.net).
Posted Monday, July 28, 2003

xToolTip v1.3

xToolTip v1.3 is now available. New in this version: Some changes in xToolTip Class, including new and rewritten methods; For Init xToolTip you must use global variable (BYTE) with three values: 2 - Balloon ToolTip, 1 - Standard ToolTip, 0 - Disable All tips; New code template to change toolTtp type at runtime; Default icon balloon tip - Standard Info icon, Warning Icon, Error Icon and Balloon without icon at all; Tooltip support for region control. Updated demonstration program and install kits for Clarion 5, Clarion 5.5 and Clarion 6 are available.
Posted Monday, July 28, 2003

UK Clarion Training Special Offer

Posted Monday, July 28, 2003

New INN Bio, News Site

Icetips.net is the new domain for the Icetips News & Bios; from now on Icetips.com will be devoted to Icetips Software. Bios are also resuming, this week feature a person who has been a regular in the Clarion community for years. He has lived a few places, including Tennessee, Florida, Ohio and even Panama! (No, not Panama City, Florida - Panama, the country.;) On the work front, he talks about a project where he was programming blind, and how the abused code in Clarion templates helps him sleep at night. A first-time father after 21 years of marraige, he is also the first bio-ee to include an original poem. And don't miss his three pond photos at the end. The Icetips.net news page also includes a link to a website analyzer, and a little news from the Clarion third-party community.
Posted Monday, July 28, 2003

Insight Graphing 1.15 Beta

The Insight Graphing 1.15 beta is now available for download.
Posted Monday, July 28, 2003

xAppWallpaper Manager v1.8

SealSoft's xAppWallpaper Manager v1.8 is now available. New in this version: Now compatible with Clarion 6 (EA5); Small bugfix with randomly select pictures; New documentation. Password for installation kit is the same as for version 1.7.
Posted Monday, July 28, 2003

JaDu Technology Greenbar Templates Updated

The JaDu GreenBar individual ABC and Legacy TPL files have been updated to fix the bug with multiple copies of the extension in a single procedure.
Posted Monday, July 28, 2003

SimTabTree Template 1.07 - Tooltips Added

Version 1.07 of the SimTabTree template has been released. The help file and
Posted Monday, July 28, 2003

xAppWallpaper Template 1.4 Clarion 6 Compatible

SealSoft's xAppWallpaper Template 1.4 is compatible with Clarion 6 (EA5). Password for installation kit the same as for version 1.3
Posted Monday, July 28, 2003

Fenix ASP.NET Generator Gold Release

The Fenix ASP.NET Generator is now available as full product. With Fenix it is possible to create complete ASP.NET web applications in a fast way, based on the existing technology of Clarion. Fenix uses the Clarion Dictionary and the .NET Framework. VB.NET code is generated instead of Clarion Code (this means that existing Clarion applications cannot be converted to Fenix). Fenix not only supplies templates for the most common functions (Browse, Locators, Form, Lookup, Calendar, Save, Cancel, CrystalReports Previewer, etc.) but also generates a Business Object layer (BO). This supplies Clarion FileManager concepts and makes it easier for Clarion users to code in the .NET Framework. Requires Clarion 5 (or higher) Enterprise Edition; Windows OS server with IIS 5(or higher); .NET Framework (can be downloaded for free). Fenix can be ordered for the introduction price of EUR 660 until October 18 2003. After that date the price will be EUR 850.
Posted Monday, July 28, 2003

PD Browse Button Lookup Version 60-04

PD Browse Button Lookup Version 60-04 (Beta) source code class library is now available. This library, which supports both C55 and Clarion6 Legacy and ABC applications, provides type-ahead entry with a browse button. Among the options provided are: Optional matching record; Easy handling of related id and other fields; Disabling of lookups when changing a record; Direct updates with or without a browse or form; Options to reset a window on new selection or completion; Multiple lookups into the same file without aliasing; View, filter, and range limited lookups. The new release incorporates changes resulting from further testing with SQL and adds an option for entering both SQL Filter and Order properties plus a variety of other small enhancements. During the beta period, upgrades to the class library from the dll version are $79. New orders are $129.
Posted Monday, July 28, 2003

xTipHotKey Class v2.3

New in this version of xTipHotKey Class: Modifications in class and template for compatibility with xDataBackupManager, xAccessManager etc., where menu items, screen controls, have been created at runtime and hotkey assigned or changed at runtime. Updated Install Kit for Clarion 6(EA5), Clarion 5.5 and Clarion 5 available.
Posted Monday, July 21, 2003

UltraTree Summer Sale
Paragon Design & Development has a number of product specials available through July 31, 2003.
Posted Monday, July 21, 2003

PDF-XChange Scores Top Points In PCMag Review
Tracker Software Products' PDF-XChange, well-known to Clarion users, has received a four star rating from PC Magazine.
Posted Monday, July 21, 2003

xDataBackupManager Lite v1.5
xDataBackup Manager Lite v1.5 is now available. Fixes include: Program crash on backup, when xDBManager compliled in LIB mode; Wrong display of date, time and size of files in archives ("Restore Backup" option in xDataBackup Manager); Error backing up key files, index files and memo-fields (in some cases these files were not included in archive).
Posted Monday, July 21, 2003

Insight Graphing Beta 1.13
Insight Graphing version 1.13 beta is now available. This release fixes a couple of template bugs, and fixes all known font sizing, location, and quality bugs.
Posted Monday, July 21, 2003

VirtualEIP Updated
This release of VirtualEIP (ABC version) has a few changes to make it compatible with Clarion 6. Other changes include: Initial support for updates on a subset of columns in the listbox (instead of requiring all columns as at present); Allow VirtualEIP to work with a memory queue, rather than requiring an ABC browse. Existing users can download an updated version for the usual place. A demo program can be downloaded from the web site.
Posted Monday, July 21, 2003

CapeSoft Email
Due to a broken router at CapeSoft's ISP any emails sent in the last 24 hours will have bounced. Email is now working properly again.
Posted Tuesday, July 15, 2003

SealSoft xTipHotKey C6 Compatible
xTipHotKey v2.1 is now available. New in this version: Compatibility with Clarion 6 (EA5); Modifications in class for compatibility with xToolTip Class; Updated documentation.
Posted Tuesday, July 15, 2003

Backflash Half Price Sale

Sterling Software's Backflash, which provides a simple one-button backup facility, is on sale for 50% off until midnight, Friday, July 18 2003. BackFlash's features include: Compression using the royalty-free 32 bit ZLIB DLL; Calculation of required number of floppies; Unique serial numbers for each backup set; Any drive and path can be specified for backup, so large capacity drives such as ZIP drives or hard drives can be used; Unattended backup; User-specified backup locations; Logging; Security; Multi-language support - Croatian, Dutch, Danish, French, German, Norwegian, Portuguese & Spanish. Legacy/ABC compatible, includes all source code. Includes a free update to the C6 version.
Posted Tuesday, July 15, 2003

SimTabTree Template 1.06

Version 1.06 of the SimTabTree template has been released. This release handles the hiding/unhiding or disabling/enabling of tabs at runtime. The demo program has been updated to show this feature.
Posted Tuesday, July 15, 2003

Freeware xFunction Library C6 Compatible

SealSoft's freeware xFunction Library is now compatible with Clarion 6 (EA5)
Posted Tuesday, July 15, 2003

Icetips Cowboy SQL 6.001

Icetips Software has released build 6.001 of Icetips Cowboy SQL. This build fixes a problem with local variables and couple of problems with the Select button. If you are a registered owner of Cowboy CCS SQL templates C5 or C5.5 or Icetips Cowboy SQL C5 or C5.5 you can upgrade to the new version for US$99.00. The full purchase is still the same price, US$399.00.
Posted Tuesday, July 15, 2003

qbFUSE 1.3 Supports QuickBooks SDK 2.1

qbFUSE 1.3 is now available supporting QuickBooks SDK Version 2.1 in Clarion 5.5 and Clarion 6. Here are some of the new features exposed by SDK 2.1 for QuickBooks: Modifying Invoice and Purchase Order transactions; Support for QuickBooks Online Edition added to the QuickBooks Foundation Classes (QBFC); QuickBooks Remote Data Sharing allowing customers to run QuickBooks on one computer and your Clarion application on a different computer across a local area network; SalesOrder transaction supported; Expanded Support for Employee and Company Objects; Expanded Support for Reports and Queries. qbFUSE works with the following versions of QuickBooks: QuickBooks 2003 R7; QuickBooks 2003 R1 and R2; QuickBooks 2003, Canadian Editions; QuickBooks Online Edition v. 9; QuickBooks 2002 R2 and above; QuickBooks 2002 R1.
Posted Tuesday, July 15, 2003

cwODBC 1.10

cwODBC 1.10 is now available. This version corrects a problem caused by the ODBC hstmt being set incorrectly for the object after an SQL statment was sent to the server.
Posted Tuesday, July 15, 2003

solid.software Holiday Schedule

The solid.software office will be closed from now on until July 21. All support questions will be answered as soon as Jens is back. Purchases can still be done at ClarionShop.
Posted Tuesday, July 15, 2003

EasyExcell 2.00 Bonus

Twenty (20) customers who buy EasyExcel 2.00 (full version) will receive free a copy of Product Scope 32 PRO V4.5(a), Spreadsheet Version - A multi-purpose tool used to display, create, organize, and research profile exchanges. Some currently available profile exchanges are Clarion 3rd Party, Search Engine, Home Theater, Newsgroup and Email, and PRO Music USB. Profile Exchanges are collections of data with common themes, categories or interest; designed to help you find information on the Internet more quickly in an organized fashion, locating related data and products.
Posted Friday, July 11, 2003

Compad Holiday

Ronald van Raaphorst is on holiday for a week, so if you email him about his XPMenu Lookalike product he will answer next week.
Posted Friday, July 11, 2003

New Icetips Bio Archive page

Sue Pichotta is getting ready to fire up the Icetips news page. In preparation for that, she's made a web page that lists all the bios already done. While you're waiting for the next bio to be posted (which should be soon), you can browse the old bios, maybe read one you missed, or re-acquaint yourself with one you've forgotten.
Posted Friday, July 11, 2003

BoTpl Updated

BoTpl has been updated to be C6 compatible and has several bug fixes and new templates added. Bo_resc has been updated to now work correctly on W2K and XP Operating Systems. AppSpecsControl has been modified to get rid of errors when no dictionary is used. Dictionary appears to be a reserved word now in C6. BoAppInfo has been modified for more selective choices, and instring search of %ProcedureCategory and %ProcedureTemplate. New BarCodeLabelControl(Reports) dynamic resizing and fonts for inventory type application. LabelControl(Reports) dynamic resizing and fonts for standard mailing label type applications. ListHorzRuntime Runtime control over font and background color of listboxes. ListVertRuntime Runtime control over font and background color of listboxes.
Posted Friday, July 11, 2003

xDataBackupManager Light v1.4

xDataBackup Manager Lite v1.4 is now available. Compatible with Clarion 6 EA5, Clarion 5.5 and Clarion 5.
Posted Thursday, July 10, 2003

EasyResizeAndSplit 1.09

Changes in EasyResizeAndSplit ver 1.09 include: Updated demo; Global Default settings; GPF at init of RTF control fixed; Bug with tab order fixed; Syntax errors when used with TinTools calendar fixed.

Posted Thursday, July 10, 2003

ClarionPost.com Links Exchange

Kelvin Chua is setting up a Links Exchange at ClarionPost. Third Party Developers can display their company or site logo on a running flash. These logos will link directly to the third party web site. This service will be on the basis that the third party developer displays the ClarionPost logo at their site, which will link visitors to www.clarionpost.com
Posted Tuesday, July 08, 2003

Buggy 3.2.3 Available

An update to the Buggy bug tracking tool is available to all registered users.
Posted Tuesday, July 08, 2003

Gitano Software Mail

If you have sent email recently to Gitano Software and it went unanswered, please send again.
Posted Tuesday, July 08, 2003

xToolTip's Fixed Distribution Kits

A problem with the xToolTip install kit updating Clarion redirection files has been fixed, and the update is available. Changes include: Multi-DLL program bug fix; Toolbar VCR Control bug fix; Tooltips for Buttons on merged toolbars support added; Enable/Disable Balloon ToolTips in runtime; Updated documentation; Updated screen shots. Demonstration program and Install Kits for Clarion 5, Clarion 5.5 and Clarion 6 are available.
Posted Tuesday, July 08, 2003

New Icetips Xplore Build Available

A new build of Icetips Xplore is now available. This build fixes problems with the filter that worked in some builds of Clarion but not others. This release is both Clarion 5.5 and Clarion 6 compatible.
Posted Tuesday, July 08, 2003

VAT On Icetips Products

Icetips' credit card company has implementing VAT on sales to EU customers. Please note that there may be minor differences (rounding differences) in the price with VAT as calculated on the Icetips site and on the online store site.
Posted Tuesday, July 08, 2003

Icetips SQL Templates 6.0

Icetips Software has released the Icetips SQL templates version 6, compatible with both Clarion 5.5 and Clarion 6. Version 6 contains a lot of new features, improvements and fixes. There is also a new 96 page PDF manual, with users guide and class reference. The upgrade price is US$99.00.
Posted Tuesday, July 08, 2003

RADFusion Site Updated

The RADFusion site is sporting a new look. Also you can now order Russ Eggen's "Programming Objects in Clarion" from the site.

Posted Tuesday, July 08, 2003

ImageEx 2.2 Released

ImageEx 2.2 is now available. Changes include: Drastically improved performance of the thumbnailer class when dealing with jpeg images; Added the Colorize method to the bitmap class for creating duo-tone images in any given color tone (hue).
Posted Tuesday, July 08, 2003

SQL Seminars Selling Out

Shawn Mason's Clarion/SQL Seminars Kings Langley, England and Boston, MA are both sold out. Las Vegas on Sept 10-12th has a few openings left. This will be the last of the Clarion/SQL Seminars for some time. The Las Vegas seminar is at the Circus-Circus hotel/casino which has tons of stuff for family if you want to bring yours.
Posted Tuesday, July 08, 2003

SimTabTree Template 1.05

A new version of SimTabTree Template (1.05) is now available. This release addresses three things: It handles the situation where a tab has been disabled; It ensures that the up and down arrow keys still operate properly on other list boxes; The help file has been updated and improved. A new demo is also available. The opening special is now over, and the price has reverted to $29 US. Registered users are entitled to the upgrade free.
Posted Tuesday, July 08, 2003

Contract Programming Site

Clarion developers looking for extra help or extra work may want to consider the RentACoder site. Robert Sorrells has reported successfully posting requirements, taking bids, and getting Clarion work done using this service. He recommends those looking for work to sign up as a programmer to this site, and bid on projects, and is confident that Clarion programmers will quickly rise over the VB programmers, especially for database projects.
Posted Tuesday, July 08, 2003

CPCS C6 Beta Recompiled With EA5

All CPCS C6 Beta products have been recompiled with C6 EA5 and new install files have been posted.
Posted Tuesday, July 08, 2003

Clarion6 EA5

Clarion 6 EA5 is now available to program participants. The major new feature in this release is the addition of the NOTIFY and NOTIFICATION functions, which provide a functional equivalent for SETTARGET(,thread) but can also be used for safe transfer of information between threads.
Posted Tuesday, July 08, 2003

Whitemarsh News

Since January, Whitemarsh has issued a number of interim releases to the metabase and one "dot" release. The ReadMe indicates all the changes. The official release is Version 5.08. The

SQL metabase release is under development, using the Mimer SQL Engine as the default DBMS back end via ODBC. It's very ANSI standard, quite good, and reasonable in price. The Whitemarsh recommended ER tool is DeZign for Databases from www.datanamic.com. Also available is a new paper, Comprehensive Metadata Management.
Posted Tuesday, July 08, 2003

Clarion Book Released

Russ Eggen's book, Programming Objects in Clarion, is now available for purchase from DeveloperPlus. Topics include: The theory of OOP; How to apply OOP in Clarion; The relationship of ABC and OOP; How to code objects; Writing template wrappers. 221 pages, available as a download ($60), CD-ROM ($70), or in print with the CD-ROM ($85).
Posted Tuesday, July 01, 2003

Clarion Magazine

Breaking Reports On Computed Fields

by Henry Plotkin

Published 2003-07-17

A new report, "Open invoices, aged by date," is required. It is assigned to me. The problem is that the `Invoice` file contains fields for invoice number, date, amount, previous balance, payments and debits. No "open" flag. No "age." I wonder how to filter, order and group the report records without having those fields at hand.

My supervisor constantly reminds me "It's just a loop." Because a report is "just" a `PRINT()` statement inside a loop, he feels that any data manipulation that can be tracked/monitored in a loop can be used to break and subtotal a report.

I would like to create the report using the templates, rather than controlling the report printing manually.

An "open" invoice has a balance. So, I can calculate whether an invoice is or is not open. Then I can filter the report on the calculation like this:

```
INV:Amount - INV:Credits + INV:Debits
```

Next, I need to group open invoices by age: under 30 days, 31-60 days, 61-90 days and over 90 days. I also need subtotals for each of these groups. Assuming data is sorted in date order, it is possible to compute an invoice's age grouping using a local variable:

```
If Today() - INV:Date > 90
  AgeStatus = 90
Elsif Today() - INV:Date > 60
  AgeStatus = 60
Elsif Today() - INV:Date > 30
  AgeStatus = 30
Else
  AgeStatus = 0
End
```

Thus, the absence of file variables does not impede getting the information I need. Unfortunately, while I can get the information, it does not seem to be in a useable form.

In my supervisor’s loop-centric view, I "just" need to compute which age group an invoice falls in, using code like that shown above. If it is the same as the previous record, increment subtotals and allow the line to print. If it is not in the same age group as the previous item, force the subtotal band to print, print the new group header band, reinitialize the subtotals and print the record.

It is actually a bit more complicated. Because I want to print a legend describing the age group at the top, I need to change the code to prime a string for the group header:

```
If Today() - INV:Date > 90
  AgeStatus = 90
  Legend = 'Over 90 Days'
Elsif Today() - INV:Date > 60
  AgeStatus = 60
  Legend = 'Over 60 Days'
Elsif Today() - INV:Date > 30
  AgeStatus = 30
  Legend = 'Over 30 Days'
Else
  AgeStatus = 0
  Legend = 'Under 30 Days'
End
```

I want to print the description for the first record read. I do not, however, want to print subtotals as there aren’t any when the first record is read. So, it looks like separate subtotal and group header bands. Only the header band is printed the first time. Thereafter, both the subtotal and group header are printed.

At least I don’t have to create a variable and track reading of the first record. I can use the following code in the `ThisWindow.OpenReport` embed, after the parent call:

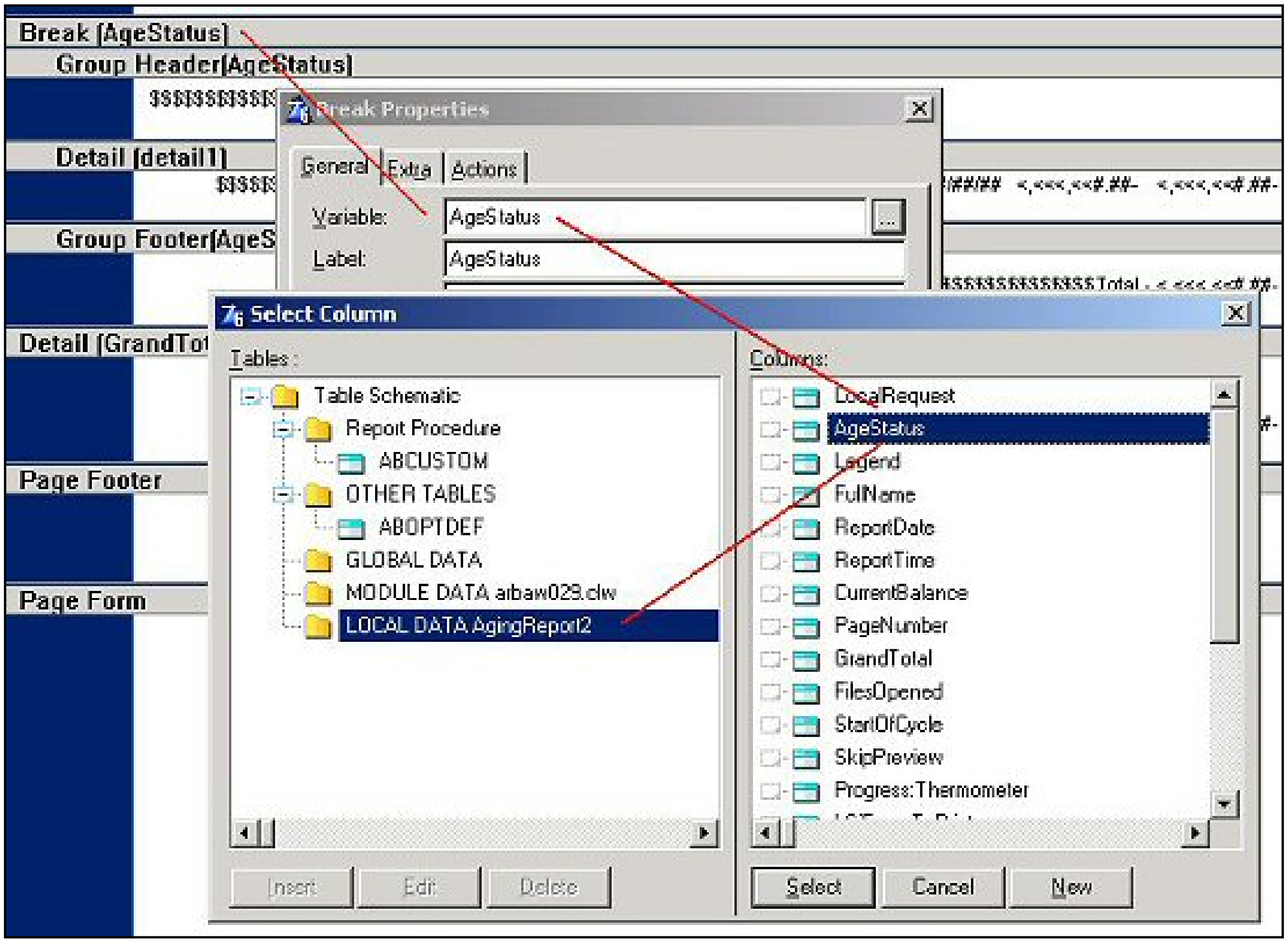
```
IF ~ReturnValue
  Legend = 'Under 30 Days'
  PRINT(RPT:LegendBand)
END
```

(If the report is sorted in descending date order, Legend = ‘Over 90 Days’.)

As my boss is also fond of saying, "This is beginning to sound like work." It needn’t be. There is, as I am about to demonstrate, an entirely template-based solution.

One need only read the article, "How the Print Engine Processes Report Sections at Runtime," in the help files (click the FAQs button on the help main menu). This article makes it quite clear that the printing of a Detail within a Break automatically checks for group Breaks. There is no restriction mentioned on the break variable, no reason to believe that it must be a file variable. This also implied that *I* don't have to manage headers and footers.

The only implied condition is that whatever variable is used, it must be set before the `PRINT()` statement is called (`TakeRecord`, 5001). If so, I can declare a break on the local variable, `AgeStatus`, entirely within the IDE:



Variable: AgeStatus
Label: AgeStatus

Figure 1. Local computed variable used a BREAK field

This done, I need not track first records or whether or not the age group has changed. The Report Engine will handle subtotaling for me. All I have to do is declare my two variables and compute a value for `AgeStatus` before `PRINT()`.

Summary

As the attached demo shows, you *can* use a local, computed variable to break and subtotal a report. You do *not* have to track changes in this variable manually or print header/footer bands by hand either (it's kind of cool when the boss turns out to be wrong and can't do anything about it).

The Report Engine does not offer embeds. But the Report Template allows tailoring of the behavior of the Report Engine, reducing the amount of code that must written. To paraphrase the

Sorting Hat, "It’s all there in the FAQs."

Download the source

"hp" in fact prefers Hewlett-Packard printers but will use whatever is available. Born in New York City, hp is a self-taught Clarion developer doing a sustantial amount of work for hospital gift shops.

Reader Comments

Add a comment

Clarion Magazine

String Flinging Part 2

by Jim Gambon

Published 2003-07-15

Last week I began discussing some of the intricacies of string handling in Clarion, and showed how to use reference variables to drastically reduce an application’s memory needs. This week I’ll wrap up this discussion with another example, and show you a few more interesting ways to work with `&STRING` and `&CSTRING` reference variables.

A larger example

To bring some of these ideas together I’ll show you how to create a simple program that works with the `GetEnvironmentStrings` Win32 API call. Unlike most string API calls (`GetTempPath` for example) where you pass the address of an existing string buffer to the function, `GetEnvironmentStrings` will allocate a buffer of sufficient size and return the address of that buffer. The environment strings are ASCII zero terminated strings within the buffer, with a zero length string to denote the end of the buffer. You could call this structure a double-null terminated CSTRING array. When you are done with the memory buffer you will have to call `FreeEnvironmentStrings` to release the memory.

In this experiment you will record each `CSTRING` in the memory buffer as an `&CSTRING` in a `QUEUE`. This will be the minimum amount of memory needed in a `QUEUE` to be able to easily retrieve each `CSTRING` later in the program. In order to show the problems you will have when you want to display that `QUEUE`, you will have a `QUEUE` of the `&CSTRING`s displayed and you will copy each `&CSTRING` in the `QUEUE` to a fixed-length `CSTRING` in another `QUEUE` for proper displaying. In a `WINDOW`, that you will open, you will show both the Storage Queue of `&CSTRING`s and the Display Queue of fixed-length `CSTRING`s.

Here is the program, with included comments describing what is happening:

```
! Example 6
PROGRAM

    ! Declare the API calls you will make.
    MAP
      MODULE('Win32API')
```

```
          GetEnvironmentStrings          |
               PROCEDURE(),LONG,PASCAL,  |
               NAME('GetEnvironmentStringsA')
          FreeEnvironmentStrings         |
               PROCEDURE(*STRING TheBuffer),SIGNED, |
               RAW,PASCAL,PROC,NAME('FreeEnvironmentStringsA')
       END !MODULE
     END !MAP
     ! Declare some local variables

 MyBuffer    &STRING    ! The memory buffer
 BufferSize  LONG        ! It's size
 StringCount LONG        ! The count of CSTRINGs in MyBuffer
 TempCString &CSTRING    ! A CSTRING ref. Var.
 TempAddress LONG,AUTO    ! A memory address
 StorageQ    QUEUE        ! The &CSTRING Storage Queue
 TheCString    &CSTRING
             END !QUEUE
 DisplayQ    QUEUE        ! The CSTRING Display Queue
 TheCString    CSTRING(255)
             END !QUEUE
 Counter     LONG        ! A LOOP counter
     ! Define the Display Window.
 Window WINDOW('GetEnvironmentStrings Example'),     |
               AT(,,316,122),IMM,SYSTEM,GRAY
        STRING('StorageQ of &CSTRINGs'),AT(6,4),USE(?String1)
        STRING('DisplayQ of Fixed Length CSTRINGs'),  |
               AT(163,3),USE(?String2)
        LIST,AT(3,16,152,82),USE(?List1),VSCROLL,    |
               FORMAT('255L(2)|MS(255)'),FROM(StorageQ)
        LIST,AT(160,16,152,82),USE(?List2),VSCROLL,  |
               FORMAT('1024L(2)|MS(1024)'),FROM(DisplayQ)
        PROMPT('Buffer Size:'),AT(4,107),USE(?Prompt1)
        STRING(@N10),AT(46,107),USE(BufferSize)
        STRING('String Count:'),AT(118,107),USE(?String4)
        STRING(@N_10),AT(168,107),USE(StringCount)
      END
   CODE
   ! Get the Memory Address of the Environment String Buffer
   ! and cast that address as a &STRING. This will be used
   ! in the FreeEnvironmentStrings function later.
   MyBuffer &= ( GetEnvironmentStrings() )

   ! Initialize some LOOP variables.
   !    1) Get the Address of the STRING Buffer.
   TempAddress = ADDRESS(MyBuffer)

   !    2) Reference assign (by address) the first CSTRING.
```

```
    !  At this point TempCString refers to the first
    !  CSTRING in the MyBuffer, the LEN() will be the
    !  length of that CSTRING due to the ASCII zero
    !  terminating character. Note: SIZE(TempCString)
    !  would be zero if you were testing on it.
    TempCString &= ( TempAddress )

    ! Since &CSTRINGs have a Length irrespective of Size
    ! You can use that to determine when you have found the final
    ! NULL CSTRING at the end of the buffer.
    LOOP WHILE LEN(TempCString) <> 0
    ! Add the LEN() to the BufferSize, taking into account the
    ! terminating ASCII zero.
      BufferSize = BufferSize + (LEN(TempCString) + 1)
    ! Add to the StringCount (duh)
      StringCount += 1
    ! Reference Assign the TempCString to the Storage
    ! Queue's &CSTRING and add it to the QUEUE.
      StorageQ.TheCString &= TempCString
      ADD(StorageQ)
    ! Move the TempAddress Pointer to the end of
    ! the current CSTRING and reassign the TempCString,
    ! once again, by using assign-by-address "casting".
      TempAddress = TempAddress + LEN(TempCString) + 1
      TempCString &= ( TempAddress )
    END !LOOP

    ! Add the last NULL byte to the BufferSize
    BufferSize += 1

    ! Now that you have a QUEUE of &CSTRINGs you can do
    ! anything you need to do. Like create a Display Queue
    ! to show in a ListBox.
    LOOP Counter = 1 TO StringCount
      GET(StorageQ,Counter)
      DisplayQ.TheCString = StorageQ.TheCString
      ADD(DisplayQ)
    END !LOOP

    ! Open the Window and Start an ACCEPT loop.
    ! This is a simple display window, so the ACCEPT
    ! loop will automatically BREAK upon an EVENT:CloseWindow
    OPEN(Window)
    ACCEPT
    END !ACCEPT

    ! Free the Storage. Notice that, since you did not allocate
    ! any memory for the StorageQ &CSTRINGs you do not have to
```

```
  ! do anything special in FREEing the Queue. If you had used
  ! allocated memory or ANY's you would have to LOOP through
  ! the StorageQ to DISPOSE or NULL the variables.
  FREE(DisplayQ)
  FREE(StorageQ)
  ! Free the Environment String buffer now that you are done.
  FreeEnvironmentStrings(MyBuffer)
  RETURN
```

If you wanted to halve the storage space, and substantially decrease your processing time, you could store the `&CSTRING` addresses in an array of `LONG`s instead of the Queue of `&CSTRING`s. By using `&LONG` reference variables assigned by address to store the `CSTRING` addresses in a dynamically created `STRING` buffer you would also be able to emulate a re-dimensionable array of `LONG`s. A rough example is included in the downloadable source code (Example 7). By "rough" I mean that this situation really demands the creation of a `CLASS` for a dynamic `LONG` array. Aside from the ease of use and re-use of a `CLASS`, the best reason to create a `CLASS` in this case would be to make sure you deallocate the dynamic array buffer in the `CLASS`'s `Destruct` method, thus avoiding potential memory leaks. However, I wanted to stay within the framework defined by the above example, so the included example does not show a `CLASS` implementation.

The salient points in Example 7 are 1) loop through the environment buffer first to count the `CSTRING`s to store, 2) allocate enough storage space for the array of `LONG`s that will hold the memory addresses (note: Clarion does not have an actual "re-dimension" command so I use a simple `NEW` string to hold the "array" of `LONG`s), and 3) loop through the environment buffer a second time using `&LONG` references to store the address of each `CSTRING` in the environment buffer into the memory buffer just allocated. A portion of that code (edited for clarity here) looks like this:

```
    ! DATA
AddressArray        &STRING
AddressArrayValue   LONG
AddressArrayRef     &LONG

  ! CODE
  ! … at the end of the first loop, we have a cstring
  ! count, so create a STRING large enough to hold
  ! all the addresses.
  ! Note SIZE of a LONG is 4.
  AddressArray &= NEW(STRING( StringCount * SIZE(AddressArrayValue)))

  ! Now, inside the second loop, fill the
  ! new "array"
  TempAddress = ADDRESS(MyBuffer)
  TempCString &= ( TempAddress )
  Counter = 1  ! Reset the position counter.

  LOOP WHILE LEN(TempCString) <> 0
    ! Position the &LONG, using address casting,
    ! to the LONG (i.e. the four bytes) desired…
    AddressArrayRef &= ( ADDRESS(AddressArray) + |
           (SIZE(AddressArrayValue) * (Counter - 1)) )
    ! …and change it's value to the CSTRINGs address.
    AddressArrayRef = TempAddress
    Counter += 1  ! Move to the next "array" position.
    TempAddress = TempAddress + LEN(TempCString) + 1
    TempCString &= ( TempAddress )
  END !LOOP
```

At the end of this you will have an `&STRING` of `LONG` memory addresses that you can even return to a calling program as a `*STRING`. Please remember to `DISPOSE` the `NEW`'ed memory at some time before you lose the reference. This technique of creating a pseudo-array of `LONG`s will run substantially faster than the use of a `QUEUE` of `&CSTRING`s. Since you cannot use the `&CSTRING QUEUE` for immediate display anyway, having the data in an "array" string may be just as useful.

Tricky tricks with string reference variables

Recall that by using address casting to assign just an address to an `&STRING`, or to an `&CSTRING` reference variable, you only get an address and not a size. The size stays zero. You would not be able to perform any string slicing on the `&STRING` without turning off the "array bounds" checking in the debug project settings. Jim Kane, in a newsgroup message a long while ago, pointed out a trick you can use if you really have to be able to set the size component of a `&STRING` or `&CSTRING`. Consider the following structures.

```
RefGroup         GROUP
MyBuffer            &STRING
                  END !RefGroup
OverRefGroup      GROUP,OVER(RefGroup)
TheAddress          LONG
TheSize             SIGNED
                  END !GROUP
```

What this does is gives you access to the individual components of the `&STRING` by placing a `GROUP` of values over the same memory space being used to store the `&STRING`. Therefore, instead of doing a reference assignment by address (i.e. "casting"), and not having a size, you can just set the components like this:

```
OverRefGroup.TheAddress = WhateverAddressYouGot
OverRefGroup.TheSize = WhateverSize
```

After that, the `RefGroup.MyBuffer` is available for any action including string slicing. Interestingly, although `SIZE(RefGroup.MyBuffer)` will give you the full size of the referred string, `SIZE(RefGroup)` will always be 8 bytes (two `LONG`s in 32-bits) because that is the size of the allocated reference variable.

The downloadable examples show the above Example 6 written to include a small example of this technique (Example 8). Please be aware that this technique is using the undocumented structure of a string reference variable. This structure may change in future versions of Clarion (although that is unlikely). Also be aware that `&STRING`s are available inside a `GROUP`, but `&GROUP` types cannot be. Therefore you cannot use this same technique to set the size component of `&GROUP` references to work around the problem I mentioned earlier of `CLEAR()` on a group reference causing a GPF.

Although not addressed in this article, in the interest of speed you should note that Clarion has an "atomic" string type called `ASTRING`. This data type can be handy if you have situations in your program where the same string values are being used in numerous places, or have numerous incidents (the tags in an XML document come to mind). An `ASTRING`, once created, is not disposed until your application terminates, and any `ASTRING` of the same value will always point to the one `ASTRING` of that value in memory. This makes string comparisons very fast. This is very similar to the Win32 concept of an ATOM. An `ASTRING` may actually be an ATOM behind the scenes, but it is not explicitly defined as such.

A preface in conclusion…

Several years ago I was involved in a large data conversion project. One of the preliminary steps in the conversion was to move all of the data from a very old RDBMS, which did not have an ODBC driver, into a more modern RDBMS where we could then use Clarion and other tools to massage the data. The only way we could get all the data (including some very inconvenient array fields) was to use the old RDBMS's raw dump format. This was a space delimited format with header and footer records. Since we were moving this data into an SQL engine with table structures that exactly matched the old structure, my choice was to take each dump file, parse it, and create a text file of SQL `INSERT` statements. Doing this allowed me to crunch the data into a usable format without having to build Clarion dictionary entries for all of the tables.

My first attempt, using a parsing class I wrote for the occasion, moved parts of the "dump" file around in memory using `STRING` parameters. It worked OK, but to process the one hundred megabyte of data took over six hours! That was going to be unacceptable.

I rewrote the parsing class using some of the techniques I show in this article. When I ran the new version I thought I had obviously neglected something. The processing time for the data was less than thirty minutes! Yet the resulting `INSERT` statements were exactly the same as the six-hour version. I was so excited about the speed I ran it several times just to watch it fly.

Since that time I have looked more closely at my usage of strings, and I use reference variables, and string-slicing, more often. Not every situation calls for the use of a reference variable, but, as you can see, their proper use will reduce the memory footprint of your programs, and you will gain potentially dramatic speed increases. And, if your program is running inside a terminal server, you will be able to run more instances on the same hardware.

Faster with less memory? It seems like old times.

Download the source

After receiving a Mechanical Engineering degree from the Georgia Institute of Technology, Jim Gambon decided that he enjoyed working with computers too much to actually get a job in the engineering field. He began programming banking and small business applications in Clarion 2.1 and has kept at it ever since. Jim and his wife Nicole own JNData, LLC in Dawsonville, Georgia.

Reader Comments

Add a comment

I saw a posting from Alexey recently and he shows a way to...

Clarion Magazine

String Flinging Part 1

by Jim Gambon

Published 2003-07-11

With the use of Web Servers, Terminal Servers, and Clarionet it seems that, more, and more, our programs do not get the luxury of being one of the few "productivity" applications running on the end-user's machine. The server may be hosting a hundred copies of not only our application, but word processors, spreadsheets, and e-mail clients. Just throwing hardware at a slow, or memory hungry program may not be as viable of a solution as it was just a few years ago. In fundamental ways the mainframe is back, and our programs are going to have to play nice.

What uses most of the memory in typical Clarion applications? Strings, most probably. Business applications are filled with names, addresses, and descriptions of all kinds. Data comes in and out of Clarion programs in various text formats like XML and Comma Separated Variable (CSV). And Clarion `STRING`s can be used as memory buffers for Windows API functions like `GetTempPath` and `QueryDOSDevices`. If your app has a lot of strings, moving and copying those strings around probably takes the most processing time. One of the best ways to increase speed and reduce memory usage is to reduce the movement of these strings.

In this article I will concentrate on the use of the old standbys `STRING` and `CSTRING`, and their related reference types `&STRING` and `&CSTRING`. I cannot promise you will learn anything new, but I hope to encourage you, with some examples, to take another look at how you are using strings in your programs.

The usual method

What follows is a simple program that illustrates some of the points I want to discuss. This program includes a string parsing function (`GetFirstName`) that extract someone's First Name from their Full Name; code like this might be used thousands of times in your program:

```
! Example 1


    PROGRAM
    MAP
      GetFirstName    PROCEDURE(STRING FullName),STRING
    END !MAP


TheName      EQUATE('Philbert MoleWhacker')
AFirstName   STRING(40)
AFullName    STRING(60)


    CODE
    AFullName = TheName
    AFirstName = GetFirstName(AFullName)
    IF LEN(CLIP(AFirstName)) <> 0
      MESSAGE('The First Name of ' & CLIP(AFullName) & ' is ' |
              & CLIP(AFirstName) & '.')
    ELSE
      MESSAGE('Cannot find First Name of ' & CLIP(AFullName) & '.')
    END !IF
    RETURN


GetFirstName   PROCEDURE(STRING FullName)   ! ,STRING
ReturnString   STRING(40)
InStringPos    LONG

    CODE
    InStringPos = INSTRING('<32>',FullName,1,1)
    IF InStringPos > 0
      ReturnString = SUB(FullName,1,InStringPos)
    ELSE
      ReturnString = ''
    END !IF
    RETURN ReturnString
```

I know for a fact that I have written functions just like this one, and I was darn proud of them at the time. But look at what the code is doing. The call to `GetFullName` specifies that the string to be parsed is sent as a copy, in memory, to the function. Then, inside the function, another forty bytes of memory is allocated for the return value. When the function completes, that return string is placed into another string variable in the calling code. In total, one hundred and forty bytes of memory have been allocated, and more than that has had to be moved around in the variable assignments and in the procedure call and return.

(Very) small improvements

If you did nothing else to make this function faster, you could put the `AUTO` qualifier on each of the string variables. Clarion, by default, will clear all variables to default values. Strings will get cleared to spaces (ASCII 32), but by using `AUTO` you are telling the compiler to not waste time clearing the memory. Still, leaving `AUTO` out of the declaration is great if you are expecting the memory to be nice and empty when you start using it.

Another way to reduce some processing is to use `CSTRING`s instead of `STRING`s. A `CSTRING(60)` will use the same amount of memory as a `STRING(60)`, but you do not have to `CLIP()` when you concatenate. That might save some time when you are trying to display the data, but it is not going to reduce the memory usage, nor the processing time in the call to `GetFirstName`.

No, the big savings will always be if you can keep from copying and moving the strings around in memory. Once a string has been created, you should try to find ways of just leaving the string where it is. The way to do this is through the use of *reference variables*.

Introducing string reference variables

Many programming languages have a data type called a *pointer*. A pointer is a variable that holds (or "points to") a memory address, and is of a certain data type. For example, in the C programming language, a pointer to the beginning of a string is declared as `*char`. The asterisk defines that the data type is a pointer, but the char defines what is being "pointed at." This way the compiler can do some data type checking, because, in reality, a pointer is just a value (often a 32-bit integer). If you change the value you can point to, and modify, anything in memory within your process's memory space. Of course, therein lies the danger of a pointer: you accidentally point to an address location that you did not allocate. This is the source of problems like the infamous "buffer over-run" that virus writers exploit.

Clarion has something similar to, but not exactly like, a pointer: it has *reference variables*. A reference variable is declared with an ampersand in front of the data type. A declaration of `&STRING`, which is a reference to a string, is a data type that does indeed have a "pointing" aspect to it. As well, the `&STRING` and `&CSTRING` reference variables also hold the size (i.e. allocated memory) of the referred `STRING` or `CSTRING`. But while a pointer can be used to point at any kind of data, a reference can only point at its own kind of data. (Actually there *are* ways around this enforced typing, but they are beyond the scope of this article.)

A `&STRING` or `&CSTRING` reference variable uses only eight bytes of memory no matter the size of the referred string, and can be used in many, but not all, of the places in your program that calls for a `STRING` or `CSTRING`. As I will show, these reference variables are perfect for referring to an arbitrarily large section of an existing string buffer.

Declaring and using references

To use a reference string you first must declare the type, like this:

```
AstringRef      &STRING
```

To use the reference as a string you must perform some assignment of the reference to existing string memory, such as:

```
AstringRef &= AFixedLengthString
```

Now you’re ready to use the reference just as you would a "normal" string. But note carefully the use of &= rather than just =. Without the & you’re not assigning a reference, you’re just making the contents of the reference equal to the contents of the string.

Referencing string slices

One of the nice aspects of Clarion’s `STRING` handling is its ability to use `STRING`s both as a singular data type and as an array of characters. Because of this you can use "string slicing" (i.e. array referencing) to have reference variables refer to parts of an existing `STRING`. For example:

```
! Example 2
    PROGRAM
    MAP
    END !MAP


MyString    STRING('This is a Fixed Length String')
MyStringRef    &STRING

    CODE
    MyStringRef &= MyString[11 : 22]
    MESSAGE('Characters 11 through 22 are "' & MyStringRef & '."')
```

The message displayed by this program is ‘Characters 11 through 22 are "Fixed Length."’ Note that the `MyStringRef` variable not only points to the desired twelve characters in `MyString`, but it also knows the desired size (i.e. `SIZE()` and `LEN()` return what you would expect.) Of course, for this simple example, you could concatenate the string slice right into the `MESSAGE` display string and get rid of the `MyStringRef` altogether. However, the points this example illustrates are:

1. `MyStringRef` uses only the memory needed for an `&STRING` variable (8 bytes in 32-bit) no matter how big the slice is,
2. You did not have to declare another `STRING(12)` to hold the section that you are interested in (namely "Fixed Length", and
3. The concatenation operator (&) uses the variable `MyStringRef` as just another string.

Both `&STRING` and `&CSTRING` have their limitations, however. In particular, you cannot use reference string variables as "use" variables for data entry, nor as strings in a `QUEUE` for a listbox. These require declared fixed length strings. Trying to use reference strings for data entry will get probably get you a GPF when the window opens, and using them to display data in a listbox will show garbage characters. If you really want to use a reference variable for an entry control you will have to assign the reference before calling your procedure, and you need to pass the reference into the form as a `*STRING` parameter. That way the string data already exists when the window is being created in memory.

References and * return types

Here's a rewrite of the program above that concentrates on using string references where appropriate.

```
! Example 3
    PROGRAM
    MAP
      GetFirstName   PROCEDURE(*STRING FullName),*STRING
    END !MAP


TheName      EQUATE('Philbert MoleWhacker')
AFirstName   &STRING,AUTO
AFullName    STRING(60),AUTO

    CODE
    AFullName = TheName
    AFirstName &= GetFirstName(AFullName)
    IF NOT AFirstName &= NULL
      MESSAGE('The First Name of ' & CLIP(AFullName) & ' is ' |
               & AFirstName & '.')
    ELSE
      MESSAGE('Cannot find First Name of ' & CLIP(AFullName) & '.')
    END !IF
    RETURN


GetFirstName   PROCEDURE(*STRING FullName)

ReturnString   &STRING,AUTO
InStringPos    LONG,AUTO

    CODE
    InStringPos = INSTRING('<32>',FullName,1,1)
    IF InStringPos > 0
      ReturnString &= FullName[1 : InStringPos - 1]
    ELSE
      ReturnString &= NULL
    END !IF
    RETURN ReturnString
```

Take a look at the prototype of the GetFirstName function. It shows a `*STRING FullName` parameter coming into the function and a `*STRING` being returned. The `FullName` parameter is what Clarion likes to call a *variable parameter*. Since, in this case, the parameter is of `*STRING` type, the function receives a memory address and a data size. No matter how large the original `FullName` string is, only eight bytes of data will have to travel from the calling program to this function. The return value in the function (`ReturnString`) is an `&STRING`; notice that the `ReturnString` reference assignment is to a string slice (i.e. character array) of the `*STRING` parameter. `ReturnString` is now referring to just a part of the `FullName` parameter, and that is what is returned to the calling program. Eight bytes coming in and eight bytes going out. This is an order of magnitude or better than the previous version.

The Clarion documentation says that reference return types are only valid when calling external functions, and not valid for Clarion language procedures. Luckily, this is an apparent error in the documentation. Also luckily, Clarion returns the entire string reference containing memory address and size. A true external function that returns a `*STRING` would likely only return a memory address pointer. The size component of the reference would be missing. The `*STRING` would know where it is in memory, but not how much memory has been allocated. You can also return numeric * types like `*BYTE`, `*LONG`, and `*SHORT` into their corresponding reference variable types `&BYTE`, `&LONG`, and `&SHORT`.

In order to return a reference variable from a procedure you must make sure that the referred memory will be available upon return. This requires you to either `NEW` the returning memory, or have the reference point to all or part of the incoming data. If you `NEW` the memory and `RETURN` you should be aware that you will have to `DISPOSE` the reference variable later in your program or you will get a memory leak.

Also note the test for a valid `AFirstName` parameter in the calling program. Since you are returning a reference variable from the function you need to test it's "`NULL`-ity" instead of its length. You could also test on its `SIZE()`, which would tell you the amount of memory that is being referenced. Of course, `LEN()` would work, but it might lead a reader of your source code to assume that a valid zero-length string is available when, in fact, a `NULL` string was returned.

LEN May Not Be SIZE

I suggested earlier that a `*STRING` returned from an external function may have a length but not a size. As a review of `LEN()` and `SIZE()` consider the following program:

```
! Example 4
     PROGRAM


    MAP
    END !MAP


MyString        STRING('This is a Fixed length String.')
BiggerString    STRING(80)
BiggerCString   CSTRING(80)
AStringRef      &STRING
ACStringRef     &CSTRING

  CODE
  BiggerString = MyString
  BiggerCString = MyString
  AStringRef &= BiggerString
  ACstringRef &= BiggerCString
  MESSAGE('LEN(BiggerString) = ' & LEN(BiggerString) &    |
      ' and SIZE(BiggerString) = ' & SIZE(BiggerString))
  MESSAGE('LEN(BiggerCString) = ' & LEN(BiggerCString) &  |
      ' and SIZE(BiggerCString) = ' & SIZE(BiggerCString))
  MESSAGE('LEN(AStringRef) = ' & LEN(AStringRef) &  |
      ' and SIZE(AStringRef) = ' & SIZE(AStringRef))
  MESSAGE('LEN(ACStringRef) = ' & LEN(ACStringRef) &      |
      ' and SIZE(ACStringRef) = ' & SIZE(ACStringRef))
```

If you run this program you will find that the `LEN()` and `SIZE()` of `STRING`s are the same. This is because there is no way of getting the "used" length of the `STRING`. The run-time library gives you the allocated size of the string when you ask for its length. An `&STRING` gets its size and length information from the allocated, referred `STRING`. Please note, however, that the `&STRING` reference stores its own copy of the size when the reference assignment is made.

`CSTRING`s, and their reference types, are a bit different than `STRING`s in that they (should) contain an embedded ASCII zero character that lets the run-time library calculate the true "used" length of the allocated `CSTRING`. Allocated size should always be at least one character larger that the greatest length of the string. Not having an ASCII zero within the bounds of the allocated `CSTRING` may lead to the infamous "buffer over-run" condition.

All of this should be standard review information for you, and you are probably bored out of your mind right now. Just bear with me for a second.

Casting

What may not be as obvious is the fact that, since `CSTRING`s and `&CSTRING`s get their length calculated, there are situations where a `&CSTRING` will have a length but no size. To illustrate, consider the following example, noting the changes to the reference assignments:

```
! Example 5
PROGRAM


  MAP
  END !MAP


MyString        STRING('This is a Fixed length String.')
BiggerString    STRING(80)
BiggerCString   CSTRING(80)
AStringRef      &STRING
ACStringRef     &CSTRING

  CODE
  BiggerString = MyString
  BiggerCString = MyString
  AStringRef &= ( ADDRESS(BiggerString) )
  ACstringRef &= ( ADDRESS(BiggerCString) )
  MESSAGE('LEN(BiggerString) = ' & LEN(BiggerString) &    |
      ' and SIZE(BiggerString) = ' & SIZE(BiggerString))
  MESSAGE('LEN(BiggerCString) = ' & LEN(BiggerCString) &  |
      ' and SIZE(BiggerCString) = ' & SIZE(BiggerCString))
  MESSAGE('LEN(AStringRef) = ' & LEN(AStringRef) &  |
      ' and SIZE(AStringRef) = ' & SIZE(AStringRef))
  MESSAGE('LEN(ACStringRef) = ' & LEN(ACStringRef) &      |
      ' and SIZE(ACStringRef) = ' & SIZE(ACStringRef))
```

If you run this program you will see that the &STRING reference SIZE() and LEN() are now zero instead of 80. The &CSTRING SIZE() is zero instead of 80 as well. Yet the LEN() of the &CSTRING is still 30. So what is going on here? The parentheses around the ADDRESS(BiggerString) and ADDRESS(BiggerCString) makes the compiler think you are calling a function and returning a valid *STRING or *CSTRING for the reference assignment. In this case, however, instead of having both address and size information for the reference variable, the returning data only holds the address, as if you are returning from an external function that only returns addresses. Since the receiving reference variable is not getting the size, then any tests on size return zero, and the test on length for the &STRING is zero because LEN() equals SIZE() for STRINGs. The test on length returns a positive value for the &CSTRING because, as stated before, length is calculated by looking for the terminating ASCII zero.

Since the size is zero, any string slicing into the &STRING or &CSTRING that has a size of zero will result in an "array out of bounds" error. If you absolutely have to do the slice on an &STRING with no size you can, but you will have to turn off the array bounds checking debug check for your program. That is something you may not want to do during development. If there is a bug in your code you could easily trash memory.

You can do a SUB() function on the &CSTRING to pull out parts of the CSTRING up to the valid length, but this will not work on the &STRING reference assigned by address because the LEN() equals zero. You should also note, because of the zero size, CLEAR() will do nothing on an address assigned &STRING and &CSTRING. If you return a *STRING of zero size from a function, the receiving reference variable will also have a size of zero. Interestingly, returning a *CSTRING of zero size will cause the receiving reference variable to record a size that equals the length of the &CSTRING. Of course, this is one less byte that the actual allocated size since the terminating zero is not included in the length-to-size recalculation.

As an aside, GROUPs are very much like STRINGs in their memory allocation. You can cast an address to a GROUP-type reference variables just as easily as to &STRING or &CSTRING. Doing this will allow you to work with the group's individual fields at that particular address. However, once again, the size component of the reference is not set. If you CLEAR() a group reference that has no size you will GPF your program no matter the debug settings. If you want to use CLEAR() on the group you can place a GROUP structure OVER the &STRING and reference assign the &STRING by address. Here's an example of this technique:

```
    ! DATA
SomeMemoryAddress  LONG
RefVar             &STRING
AGroup             GROUP,OVER(RefVar)
AName                STRING(40)
ANumber              DECIMAL(14,2)
                   END !GROUP
   ! CODE
   RefVar &= ( SomeMemoryAddress )    ! Cast the address.
   AGroup.AName = 'Whatever you want' ! Work with the Group
   Agroup.Anumber = 42.00             ! structure at that address.
```

This is a very useful technique; using it, you can `CLEAR` the `GROUP` structure without a problem. Unfortunately, the only way to get this to work is, again, to turn off array bounds checking.

So be warned, if you treat a reference variable as a pointer (i.e. address only without size information) you are going to have to be careful what you do. With that caveat in mind, in the interest of speed and memory usage, using existing data in place and not making memory copies to static variables is the exactly right thing to do.

Next week I'll bring some of these ideas together in another example program, and reveal a few more string reference variable tricks.

Download the source

After receiving a Mechanical Engineering degree from the Georgia Institute of Technology, Jim Gambon decided that he enjoyed working with computers too much to actually get a job in the engineering field. He began programming banking and small business applications in Clarion 2.1 and has kept at it ever since. Jim and his wife Nicole own JNData, LLC in Dawsonville, Georgia.

Reader Comments

Add a comment

In the example above - is the payoff worth the risk? ...
I like squeezing every drop I can to get efficient and...

Clarion Magazine

Clarion, COM, Soap, and HTTP

by Shane Vincent

Published 2003-07-10

How I learned to stop worrying and love the COM

Over the years, I have had to deal with a lot of different ways of handling credit card payments and authorizations. Most involved some contortions due to the nature of Clarion.

Recently I wanted to add a no-cost solution to the credit card authorization process, instead of requiring my customer to buy some equipment or license some software from someone else.

I found a solution at www.richsolutions.com that fit the bill, but unfortunately (or fortunately) it used the Simple Object Access Protocol (SOAP) to pass the information back and forth (actually this product has other possible transport mechanisms, but the SOAP method seemed to be the best).

There are lots of ways to do SOAP; I settled on the PocketSoap and PocketHTTP open source COM components by Simon Fell. For the Clarion COM layer I used Jim Kane's solution as described in his excellent article Interfaces Everywhere (which I highly recommend).

Unfortunately in the version of Clarion 6 that I used for this project, the PocketSoap HTTP transport did not work, but using the PocketHTTP interface directly (which is what the PocketSoap uses in any case) resolved that problem. I have since tested this combination of PocketSoap and PocketHTTP in Clarion 5, 5.5 and 6EA and it works well in all three versions.

Getting down with COM

It's incredibly easy to use these COM components, thanks to Jim Kane. After downloading and installing the free PocketSoap and PocketHTTP (which supports SSL) packages, I edited the lines at the beginning of Jim's rtlib.clw to the following:

```
dumpfilename='psoap.inc'
tlibname='C:\Program Files\SimonFell\PocketSOAP\pSOAP32.dll'
```

All that I needed to do after that was make and run rtlib.prj, and RTLib defined the interfaces in a nice and easy to use file. I used RTLib to generate the PocketHTTP interfaces as well.

Next I added the generated includes (and Jim Kane's stdcom2.inc) to the program's global includes embed:

```
include('stdcom2.inc')
include('pSoap.inc')
include('pHttp.inc')
```

I then created a class to handle the PocketSoap COM object (the PocketHTTP class is much smaller). Here's the declaration:

```
pSoap class(StdCom2ClType)
Envelope        &ISOAPEnvelopetype
Parameters      &ISOAPNodestype
Item            &ISOAPNodetype
Node            &ISOAPNodestype
AddNode         PROCEDURE(String Name,String Val)!,long,proc
init            PROCEDURE(byte pDebugmode=1),byte
GetValue        PROCEDURE(LONG MyNode=0)
SetAddress      PROCEDURE(Long),Long
Send            PROCEDURE()
NameSpace       CSTRING(50)
lpURI           LONG
SetNode         PROCEDURE()
    END
```

I won't reproduce all of the method code here – you can find it in the source zip, in soap5.clw.

Deceptively simple

The basic program logic for a PocketSOAP session is quite simple, and is handled by `psoap.AddNodes` (adds data to the package to be sent), `psoap.GetValue` (shows the data in the COM object, can be used to check the return values), `psoap.Send` (formats, sends and receives the SOAP messages), `pSoap.ShowHeaders` (shows the HTTP headers - a debugging aid only).

```
if pSoap.init(eDebugon) AND h.init()
   pSoap.AddNodes()
   pSoap.GetValue() ! for debugging only
   pSoap.send()     ! also gets the response
   pSoap.GetValue() ! for debugging only
   h.ShowHeaders()  ! for debugging only
END
```

If you take out the debugging code, there are really just four steps: initialize the COM components, create the data to send, send the data, and receive the result.

This example deals with getting a credit card authorized using RichSolutions free SDK; this is a specific application, but it should be easy to adapt the code to any other SOAP task.

Initializing the COM objects

The initialization is based on the discussion in Interfaces Everywhere, but I made some additions specific to this example, as follows:

```
SELF.initcom(pDebugMode)
SELF.Envelope&=SELF.getinterface(address(CLSID:CoEnvelope)|
  ,address(IID:ISOAPEnvelope))
if SELF.Envelope&=NULL then do procret.
Self.Envelope.GetParameters(lpbstr1)
Self.Parameters &= (lpbstr1)
Self.NameSpace='http://RichSolutions.com/RichPayments/'
Self.tobstr(Self.NameSpace,Self.lpURI)
bstr1=''
bstr2='ProcessCreditCard'
Self.toBstr(Bstr1, lpbstr1)
Self.toBstr(Bstr2, lpbstr2)
Self.Envelope.PutEncodingStyle(lpBstr1)
self.Envelope.SetMethod(lpBstr2,Self.lpURI)
```

In this example you’ll see a lot of code referencing "nodes." Nodes are equivalent to a queue of any type of data, including other queues. The call to Self.`Envelope.GetParameters` gets the memory address of the nodes that represent the parameters of the SOAP request. Where the nodes (equivalent to a queue of any type of data, including other queues), `Self.Parameters` is then assigned by reference to that address.

In this example the encoding style and the method are set in the initialization module for this example, though it could be abstracted even more. The next item is to initialize the HTTP transport; if both the `h` class and the `psoap` class are initialized, then the program will continue, otherwise it exits.

```
self.initcom(edebugon)
self.Http &=SELF.getinterface(address(CLSID:CoPocketHTTP)|
  ,address(IID:IHttpRequest))
IF self.Http &= NULL then do procret.
Self.http.GetHeaders(lptr)
Self.Header   &= pSoap.SetAddress(lptr)
self.Method('POST')
self.AddNode('Content-Type', 'text/xml; charset=UTF-8')
```

Basically the `self.Method` call tells the server that this is a `POST` message, and the other calls just add headers so that it’s clear that the server knows what `SoapAction` is desired.

Putting data in

The next step is to add nodes to the envelope, (in this case the data consists of the credit card number, name on the card, mag stripe read, passwords and user id of the customer, and other information related to the transaction), this is done via a call to `psoap.addnodes` and `psoap.addnode`.This is a simple implementation, but the nodes do store variant types (see Jim’s article on COM fundamentals for more on variants).

Each value that is to be set has two elements, the name and the value. This is done in `addnode` by the following code:

```
variantINIT(address(vparm))
vParm.vt=vt_bstr
Self.toBstr(Bstr2, lpbstr2)
vParm.Value=lpBstr2
SELF.tobstr(bstr1,lpBstr1)
Self.VariantByValue(address(vParm),lpVparm1,|
  lpVparm2,lpVparm3,lpVparm4)
Self.Parameters.Create(lpBstr1,lpVparm1,lpVparm2,|
 lpVparm3,lpVparm4,Self.lpURI,res,res,res)
variantclear(address(vparm))
```

Basically, `pSoap.addnode` takes the name (which is a string that gets converted to a BString, which is a UNICODE character array in PocketSoap) and the value (which is a variant type that for this solution can be simplified as a BString as well) and passes those to the PocketSoap object to have them stored there.

Drawing on Jim Kane's discussion in When Clarion Com Will Not Do, the PocketSoap help, and information gleaned from examining the interfaces generated by RTLib, by it became clear to me that implementing the variant handling libraries was quite important. If you don't initialize the memory location for a variant the program will crash, because variants are passed on the stack. I added the following to the start of the program:

```
variantINIT(long),pascal,long,name('variantinit'),proc
variantclear(long),pascal,long,name('variantclear'),proc
```

`VariantInit` and `VariantClear` are described in the MSDN Platform SDK under Automation.`VariantInit` sets the *vt* field of `vparm` to `VT_EMPTY` and initializes the memory space of the address passed to it. The API call to `variantClear` frees the variant's memory; failure to do this results in a memory leak.

The calls to `pSoap.getvalue` and `h.showHeaders` are for debugging purposes so that the elements of the HTTP and envelope nodes can be viewed in a hand coded window with a simple list box.

Going all the way with SSL

The `pSoap.Send` method handles sending the transaction information and converting the response into data that is usable in Clarion. This uses the Secure Sockets Layer, or SSL – all you have to do is specify `https://` in the URL instead of `http://` (of course that also assumes the server supports SSL).

```
bstr1='https://test.richsolutions.com/richpayments/richpay.asmx?wdsl'
!Change the https to http for normal communication
IF Self.ToBstr(bstr1,lpBstr1) !OR Self.ToBstr(bstr2,lpBstr2)
END
Self.Envelope.Serialize(lpvparm1)
vParm.vt=vt_bstr
vParm.Value=lpvparm1
Self.VariantByValue(address(vParm),lpVparm1,lpVparm2,lpVparm3,lpVparm4)
h.http.GetResponse(lpBstr1,lpVparm1,lpVparm2,lpVparm3,lpVparm4,lpbstr1)
h.response &= Self.SetAddress(lpbstr1)
h.response.GetString(lpbstr1)
vParm.value=lpbstr1
vParm.vt=vt_bstr
```

```
Self.FromBstr(lpBstr1,bstr1,0)
Self.VariantByValue(address(vParm),lpVparm1,lpVparm2,lpVparm3,lpVparm4)
Self.Envelope.Parse(lpVparm1,lpVparm2,lpVparm3,lpVparm4,0)  Self.SetNode()
```

The `Self.Envelope.Serialize` method creates the XML envelope, which you can think of as a wrapper for the data being transported. The envelope defines the data elements, includes the data, and tells the other end what is desired to be done with the data. The envelope is stored in the memory location passed to the method (`lpvparm1`) as a variant type. `Self.Envelope.Serialize` will return a pointer to a BString which is then stored in `vParm.Value`. The `EnvelopeParse` method expects a copy of the variant, that is, a variant passed by value, not by address. To do this you have to break the variant down into its components, which is four `LONG` variables. This is what `Self.variantbyvalue method` (defined in stdcom2.clw) does.

In summary, the XML envelope is passed to h.http.getresponse as the `lpVparm1`, `lpVparm2`, `lpVparm3` and `lpVparm4`. The passing of `lpbstr1` at the beginning tells the HTTP Transport where we want to find the server, and `lpbstr1` is reused to handle the return value from `getresponse` at the end of the method call.

The response is then extracted from the address returned in `lpbstr1`, and is then converted into a BString with a call to `h.response.GetString(lpbstr1)`.

Then the string is passed back to the envelope for parsing (i.e. the string gets put into all its different nodes) by a call to `Self.Envelope.Parse`; now the data returned by the SOAP server is readily accessible.

It's important to note here that after calling `Envelope.Parse`, the old pointer to the parameter collection of nodes will point to the old data. You need to make a new call to `Envelope.GetParameters` to change the reference to the *new* address at which the data is stored (for some reason the data is stored in a new location, the old location has all the old data; this had me confused for a while, as I couldn't understand why the `Envelope.Parse Method` wasn't doing what I thought it was).

The call to `Self.Node` sets the node to the first return element (the credit card response, hopefully), and in this example it will display the node.

Finally you can check the values of the nodes by looking at the code from getvalues (which can be easily modified to store the values, but this is an example, and I am keeping it simple).

```
Self.Parameters &= Self.SetAddress(lpmyNode)
Self.Parameters.GetCount(MyCount)
LOOP Pos = 0 To MyCount-1
    Self.Parameters.GetItem(Pos,lpItem)
    IF lpItem~=0
        Self.Item &= Self.SetAddress(lpItem)
        Self.Item.GetValue(varParm1)
        lpbstr2=VarParm1.Value
        IF VarParm1.Vt=vt_bstr
          Self.FromBstr(lpBstr2,bstr2)
        ELSE
          bstr2=''
        END
        Self.Item.GetName(lpBstr1)
        Self.FromBstr(lpBstr1,bstr1)
        Self.Item.GetNodes(lpItem)
        Self.Node &= Self.SetAddress(lpItem)
        Self.Node.GetCount(MC2)
        List1:Queue.F1=CLIP(bstr1) &':'& clip(bstr2) |
          &':'&varParm1.vt&':'&lpItem&':'&MC2
        ADD(List1:queue)
    END
```

The parameter can be considered to a queue which holds items. The `Self.Parameters.getCount` method can be considered to be equivalent to a call `Records(SomeQueue)`. `Self.Parameters.GetItem(Pos,lpItem)` serves the same function as the Clarion `GET(`*`Queue,Pointer`*`)`, where the return value is `lpItem`, a pointer to the memory location at which the `Item` is stored.

The important thing to remember about the parameter count is it is zero based; to get the items (`Self.Parameters.GetCount`) start from 0 and go to one less than the number of parameters. Also ensure that only BStrings are passed to the `fromBstr` function or weird things will happen.

Summary

The purpose of this article is to show you how easy it can be to implement COM in Clarion (almost as easy as VB) with Jim Kane's RTLib, and to show a real world example. While this example does not do all the work (for instance it doesn't give different options for authorizing, and the variables are coded in), it does show an example how to use SSL, HTTP, SOAP and XML in just about any version of Clarion from 5 on without having to buy anything (other than your Clarion Magazine subscription so you can learn all these great methods!).

I spent a lot of time trying to get the PocketSoap internal HTTP transport methods to work with Clarion (they worked with Visual Basic and Visual C) before I switched over to using Pocket HTTP, which worked fine. The curious thing is that there isn't much difference between the two, as the transport in PocketSoap is based on PocketHTTP, it just sets the headers and HTTP method automatically for SOAP communications.

Download the source

Shane Vincent has a degree in Engineering from the University of Illinois and is a graduate of the Naval Nuclear Power School. He used to teach computer programming at Triton College. Currently living near Chicago, he works as a freelance consult, having worked most recently with Clarion-Software.com, Tickets.com and Nu-Tech.

Reader Comments

Add a comment

***Excellent* title!**
Thanks, I hope you found the rest of the article useful as...