Clarion Magazine

Subscription Special Ends Oct 9!

Clarion 6 is almost here, and to celebrate we're having a **pre-Clarion 6 subscription sale**! You can get a **one year** subscription for **only $79.95** (regular $95), and a **two year** subscription for **only $139.95** (regular $170). **Subscribe or renew now!** This special ends Thursday, October 9, 2003.

Your previous subscription has expired! Renew now!

A Basic Editor For Text Fields: Introduction

Clarion's TEXT control is versatile and useful, but has next to no word processing capabilities. But as Tim Phillips shows, it's relatively easy to add capabilities like search/replace, block reformatting, and more. Part 1 of 2.

Posted Thursday, September 04, 2003

A Calendar For Date-Limited Browses

One of the most-used features of Clarion is page-loaded browse boxes. But are these really a good idea? Veronica Chapman suggests otherwise, and illustrates this with a calendar you can use to date-limit a file loaded browse box.

Posted Friday, September 05, 2003

Check Please: Managing Conditional Browse Relationships

Here's a tip from Carroll Jolly on how to use a checkbox to alternate between showing all related records on a child browse, or all records whether related or not.

Posted Friday, September 05, 2003

Articles: XML

News: XML

SoftVelocity

News

Clarion Magazine Subscription Sale

Clarion Developers Challenge Week 4 Winner

xRuntimeStyle Manager v1.2 At ClarionShop

Cover Your Ass(ets) v.1.22

Gitano Software C6 releases

SURVEY

When will you add XML capabilities to your application(s)?

Already have
12.4%

In 3 months
11.8%

In 6 months
9.8%

In 1 year
2.6%

Someday
24.8%

No plans
38.6%

153 responses

Previous Surveys

PDF For August, 2003

All Clarion Magazine articles for August, 2003 in PDF format.
Posted Monday, September 08, 2003

Weekly PDF For September 1-6, 2003

All articles for September 1-6, 2003 in PDF format.
Posted Monday, September 08, 2003

Clarion 6 Gold Candidate Release 2

Clarion 6 Gold Candidate Release 2 is now available for download to C6 EA program participants.
Posted Tuesday, September 09, 2003

A Basic Editor For Text Fields: Conclusion

Clarion's TEXT control is versatile and useful, but has next to no word processing capabilities. But as Tim Phillips shows, it's relatively easy to add capabilities like search/replace, block reformatting, and more. Part 2 of 2.
Posted Thursday, September 11, 2003

Translation, Clarion Style: Currencies And Measurement

Nardus Swanevelder continues his series on translation with a template to convert currencies and measurements, and a brief look at dynamically created text, controls, and windows.
Posted Friday, September 12, 2003

Translation, Clarion Style: Selected Topics

Nardus Swanevelder continued his translation series with a look at screen design issues, third party applications, and changing languages at runtime.
Posted Thursday, September 18, 2003

Translation, Clarion Style: Runtime Issues

Clarion Jobs Page

CPCS C6 Beta For CR-4

UltraTree Viewlets/C6 News

ProcedureNotes New Year Sale

Freeware Template Bulk Loads Oracle Tables

xRuntimeStyle Manager v1.2

Icetips And AceIcons On Vacation

Fenix v1.1 Adds C# Support

CWPlus 2.00 Released

Network Troubleshooting Page

CapeSoft Has Moved

SealSoft xWhatsNew v1.2

Ace Icons New Look, New Product

Super Templates and Clarion 6

RPM And PNet For C6 CR-3

CapeSoft Accessories For C6

CPCS v6.00 (Beta) For C6 CR-3

Simplified Software Special

One Year Ago In CM

PDF for September, 2002

How Not To Ignore The Form Template

A Class For Printing Addresses

How To Ignore The Form Template

The Clarion Challenge - ContainsMatch

How To Ignore A Template

Advertising Feature: XP Menu - A Simple And Extendable Clarion User Interface

Data Structures And Algorithms Part IX - Are You Getting Too Tall?

A Template For Exporting Classes

Two Years Ago In CM

Weekly PDF for September 30-October 6, 2001

WinInet.DLL: Transferring Files

Nardus Swanevelderl complete the runtime translation discussion with examples of loading translation strings from a database, and applying this translation to a multi-DLL application.

Posted Thursday, September 18, 2003

Product Review: NetTalk, Part 1

Over the years, Tom Hebenstreit has used various methods to add Internet functionality to his programs, ranging from the built-in Windows WinInet libraries to the SocketTool libraries from Catalyst Corporation. Each had its strengths and, as usual, some weaknesses as well, so Tommy's always on the lookout for simpler, more Clarion-friendly ways of accomplishing those tasks. Enter CapeSoft's NetTalk, a veritable jack of all Internet trades. Part 1 of 2.

Posted Friday, September 19, 2003

Clarion 6 Candidate Release 4

Clarion 6 gold release candidate number four is now available for download to EA program participants.

Posted Thursday, September 25, 2003

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.

New Images From 1st Logo Design

MAV Direct ODBC Beta

ClarionForge

gReg For C6

AdiColormanager C6 Compatibility

Outlook EmailReport Template 6.0

ClarionMag Contributor Konrad Byers: 1968-2003

Nice Touch Products Compatible With C6 Candidate Releases

Subject: CPCS C6 Beta available for C6 CR-2

Icon Clearing Source Code

xWhatsNew Class v1.0

Clarion Developers Challenge Week 1 Winner

Free Template Blocks Update Buttons

RPM & PNet CR2 Installs Available

Report Wizard C6-CR1 Beta

ProcedureNotes Template Sale Ends Sept. 15

With FTP (Part 2)

Review: LogFlash 2.4 from Sterling Data

Clarion News - October 2001

PDF for September 23-29, 2001

Interview: ClarioNET's Michael Brooks

WinInet.DLL: Transferring Files With FTP (Part 1)

Do Not Adjust Your Browser

Report Redirection

Three Years Ago In CM

Using CHOOSE() To Concatenate Data

The Nuts And Bolts Of Passing Parameters: Part 2

Five Rules for Managing Complexity: Part 4

Five Rules for Managing Complexity: Part 5

Clarion News -

EasyResizeAndSplit 2.00

Gorman To Speak At MetaData Symposium 2003

PDF-XChange & Tools C6 Ready

EasyListPrint 1.05

Clarionfoundry Server Downtime

C6 Icon Removal Example

gQ For C6 Available

EasyExcel 3.00 Demo

Gitano Utilities For C6

Jesus Moreno Receives Microangelo Artist Of The Month Award

ProcedureNotes Web Site Updated

Code Generation Network

Search the news archive

September 2000

The Nuts And Bolts Of Passing Parameters: Part 1

Five Rules for Managing Complexity: Part 3

Outlook Menu Templates

The Clarion Advisor: Better Debugging With DebugView

Four Years Ago In CM

DevCon Details: Welcome And Keynote Address

DevCon Details: Web Edition 2 and iBuild@TopSpeed

Seen And Heard At DevCon

TopSpeed Headed For Java

TopSpeed's New Direction: The Web

DevCon '99 Monday Overview

DevCon Weekend Edition

Technology Poll Results

Industry Trends: How Important Is Java?

Clarion Magazine

A Basic Editor For Text Fields: Introduction

by Tim Phillips

Published 2003-09-04

The Clarion TEXT field/control is a powerful if limited feature. If you create a table with a TEXT field and several other fields to hold metadata, you have a powerful text database. It can be used to store Work To Do items, code snippets, lists of DVDs and CDs, fan fiction and – most importantly – jokes culled from the SoftVelocity Chat forum. You can load the TEXT field into a STRING and manipulate it using nothing more sophisticated than string-slicing and an ASCII character chart. Using a TEXT control leads to a database that is more compact and easier to manipulate than if you take advantage of the Clarion RTF control.

However, the basic TEXT control has a lot of limitations. Inserting tabs is a pain. There is no search or search/replace functionality. Keyboard shortcuts are atypical. And so on. Basically, all the features present in even a rudimentary word processor like the Clarion RTF control are missing.

Fortunately, you can add functional (if a little rough) versions of the common tools with just few select chunks of source code. I call the concept Basic Editor.

The Basic Editor

My Basic Editor revolves around a ViewMemo procedure that displays any TEXT passed into it. The procedure is "hot-rodded" with hot-keys and a custom popup menu to provide:

- "correct" keyboard mapping for common functions like cut/copy/paste, TAB and Select All
- Search
- Repeat Search
- Search and Replace
- Launch a highlighted file or URL using the default program on your computer
- Block reformatting of text to "clean up" text pasted from web pages
- Provide a definition of a highlighted word from a vocabulary table
- Inserting common phrases from a lookup table
- Export the text into an external word processor and then re-import it when you are done manipulating it

Follow along in this article and the attached example APP and see just how easy it is to make a Basic Editor for text using Clarion. Note that the editor makes use of several freely available templates - the links are at the end of this article.

The ViewMemo procedure

The ViewMemo procedure is built using the **Window - Generic Window Handler** procedure template. To pass

the memo to be edited into and out of the procedure, I set the **Prototype** to `(*string)` and the **Parameters** to `(IOStringToEdit)`. In the **Data** area I declare a local variable called `StringToEdit` that is a string of 40,000 characters. I will assign the incoming `IOStringToEdit` to this local variable and display the local variable. This will permit me to offer a Cancel feature to undo mistakes in editing if wanted by the user (**NOTE**: this also means that the `ViewMemo` procedure can not handle a memo that has more than 40K worth of characters in it – this is adequate for my purposes, but it may limit some readers). Most of my applications are MDI, so the MDI Child window is the proper window structure. Since part of the idea of the `ViewMemo` procedure is to make it easy for a user to read/edit, I leave the **Frame Type** set at resizable and drag the "normal" window size up to be something I find comfortable to work on as the default. I set the text caption of the window itself to a generic phrase like "View Memo".

Experience with resizable windows has shown me that I get the best results when I have the buttons along the top of the window and something like a text field below them. So the four buttons I need will be added across the top of the window. They will be built as:

1. the CLOSE control template
2. a generic button control with the text of Cancel and the **Use** set to `?Cancel`.
3. a generic button control with the text of BE (for Basic Editor), a Use variable of `?BE` and the text color set to `Color:Red`
4. a generic button control with the text of Help and a Use variable of `?Help`.

The BE and Help buttons are both used to call Help windows. The BE button should call a help window that describes the features of the Basic Editor to the user. The Help button should display help about the `ViewMemo` window itself. I personally use the InternalHelp concept that I've written about previously, but everyone has their own favorite way of doing Help. The sample application that can be downloaded for this article just uses a procedure to display ASCII text files for these two buttons. The two supplied text files are the text from my own InternalHelp Topics. I'd suggest stealing my typing and reformatting it to form whatever Help technique yotu prefer for the Display Text window.

Below these buttons, I add a text field using the **Populate Control Without Control Template** option and the Use variable set to the local variable of `StringToEdit`. I also activate the vertical scroll bar for this control so the user can scroll within it. Before I leave this, I also need to define a pile of hot-keys on this control that will be used later: TabKey, AltN, CtrlA, MouseRight, CtrlF, F3Key. The code for making these hot-keys actually do anything will be defined later on.

I then add the **WindowResize** template from the Class ABC templates. I go to the **Override Control Strategies Authorities** and set the Close, Cancel, BE and Help buttons to be locked in width and height and fixed to the left and top. The `StringToEdit` control gets set with a **Horizontal Resize Strategy** of **Resize**, a **Vertical Resize Strategy** of **Constant Bottom Border**, a **Horizontal Position Strategy** of **Fix Left** and a **Vertical Position Strategy** of **Fix Top**.

These resizing options should keep the controls from running into each other when I want to resize the entire window to see as much of the `StringToEdit` at one time as is possible.

A little time now needs to be spent in organizing and resizing controls so the `ViewMemo` window "looks pretty". Tips need to be added to various controls to explain them to the user. The tabbing order of the window needs to be set to a good default order. The `SKIP` attribute needs to be activated on the BE and HELP buttons.

With that done, there is a small amount of source code that needs to be added to correctly pass `IOStringToEdit` to the `StringToEdit` when the procedure starts and then optionally return `StringToEdit` to `IOStringToEdit` when the procedure shuts down.

Adding source code for passing IOStringToEdit

Code needs to be added to three embedded code points to control activity involving IOStringToEdit.

1. At Local Objects ->ABC Objects ->Window Manager ->Init ->just after Initialize the Procedure, add:

   ```
   StringToEdit=IOStringToEdit
   ```

   This will assign the incoming value of the `IOStringToEdit` to the local value that is actually displayed on the window

2. At Control Events ->?Close ->Accepted above the generated code I add:

   ```
   IOStringToEdit=StringToEdit
   ```

   This will assign the local StringToEdit back to the input string (which is also returned to the calling procedure) so that any edits are returned to the called procedure.

3. At Control Events ->?Cancel ->Accepted above the generated code I add

   ```
   StringToEdit=IOStringToEdit
   Post(Event:Accepted,?Close)
   ```

   In the event that the user choses to cancel their edits on the `ViewMemo` window, this will cause the same string that was passed into the procedure to be passed back, by assigning the passed in value to the local value and then posting an event that causes the Close button to be activated.

With this code installed, the `ViewMemo` procedure can now have a memo passed in and out of it for editing, and it has help available on several buttons.

Now it is time to define files, local variables and procedures that will support the Basic Editor code.

Basic editor supporting files

Two "helper" files need to be defined in the Data Dictionary for your application.

One stores snippets of text for insertion into the text field. It is called `TextSnippets` with a prefix of `TS`. It has two fields, a memo field called `Description`, and a string of 60 characters called `Name`. The only key for the file is a unique key on `Name` called `ByName`.

The second file stores vocabulary words and definitions. It is called `Vocabulary` with a prefix of `VOC`. It has two fields, a memo field called `Description`, and a string of 60 characters called `WordPhrase`. The only key for the file is a unique key on `WordPhrase` called `ByWordPhrase`.

Basic editor supporting local variables

The `ViewMemo` procedure needs some local variables defined to support the code it will shortly contain.

- `LocalQueue`. A queue with a prefix of `LQ`. It should then contain a single field called `String1` which is a string of 1 character.

- `NumberOfSpacesInARow` is defined as a short
- `ReplaceText` is a string of 60 characters

Basic editor supporting procedures

Seven "helper" procedures need to be defined to support the Basic Editor tools.

Four of these are for the `TextSnippets` and `Vocabulary` files. Normal Clarion browse/form combinations are needed to provide update and display functionality. The Clarion Wizards do the brunt of the work of making these, with only a little tweaking necessary to make them look good.

These procedures – for the purposes of this article – are called `BrowseTextSnippets`, `UpdateTextSnippets`, `BrowseVocabulary` and `UpdateVocabulary`.

The fifth procedure is `UseExternalEditorForMemo`. This procedure will take a string as input and write it out into an ASCII file and then load that file into NotePad so the user can edit it. The procedure will wait until the user has exited NotePad and pull the modified ASCII file back in to create a string again and pass the updated string to the procedure that called `UseExternalEditorForMemo`.

The sixth procedure is a window to get a Search value from the user. It is called `SearchWindow` and returns a string.

The seventh procedure is a window that returns a Search For and Replace With value from the user. It is called `SearchReplaceWindow`.

Basic editor code

The Basic Editor itself is driven by a main block of code in the `ViewMemo` procedure, and three separate routines to support specific functionalities.

The main block of code is added at the **Control Events ->?StringToEdit ->AlertKey** embed:

```
case KeyCode() ! what key has been pressed?
   of TabKey ! if TabKey pressed
      ! use the Ctrl-I combination to insert a tab into
      ! the text field
      PressKey(CtrlI)
   of AltN ! if Al-N pressed
     ! select the next field in tabbing order
        !after the text field
      Select(?StringToEdit+1)
   of CtrlA
      ! select all the text in the control
      Select(?StringToEdit,1,Len(StringToEdit))
   of CtrlF
      ! do a search from the top of the control
      do InitialSearch
   of F3Key
      ! do a repeat search
      do RepeatSearch
   of MouseRight
```

```
        ! user has right-moused on the text control - display a custom
        !popup menu of options they can use
        EXECUTE Popup('Cut -Ctrl-X|Copy - Ctrl-C|Paste ' |
                & ' - Ctrl-V|Select All|Find ' |
                & ' - Ctrl-F|Repeat Find ' |
                & ' - F3|Search And Replace ' |
                & ' - Ctrl-R|Launch High-Lighted|ReFormat Text|Lookup  ' |
                & 'Vocabulary|Insert Text Snippet - ' |
                & 'Ctrl-T|Use External Editor')
          PressKey(CtrlX) ! cut text to the clipboard
          PressKey(CtrlC) ! copy test to the clipboard
          PressKey(CtrlV) ! paste text from the clipboard
          Select(?StringToEdit,1,Len(StringToEdit)) ! select all text
          do InitialSearch ! search from the top of the control
          do RepeatSearch ! do a repeat search
          Begin
            ! do search/replace for text
            SearchReplaceWindow(SearchForText,ReplaceText)
            freplace(StringToEdit,CLIP(SearchForText),CLIP(ReplaceText),0)
            Display(?StringToEdit)
          End
          Begin
            ! launch file/URL (as if you had double-clicked on it)
            GLO:ShellExFileName=StringToEdit[?StringToEdit{PROP:SelStart} : |
                      ?StringToEdit{PROP:SelEnd}] ! get currently selected text
            GLO:ShellExFileName=Left(GLO:ShellExFileName) ! left it
            IMPURLHandler(0{PROP:handle},CLIP(Glo:ShellExFileName)) !launch
          End
          do ParagraphBreaksAtLine ! reformat text
          Begin
            ! display vocabulary description if one exits
            VOC:WordPhrase=StringToEdit[?StringToEdit{PROP:SelStart} : |
                      ?StringToEdit{PROP:SelEnd}]
            VOC:WordPhrase=CLIP(Left(VOC:WordPhrase))
            if Access:Vocabulary.TryFetch(VOC:ByWordPhrase)=Level:Benign
              Message(CLIP(VOC:WordPhrase) & ' = ' |
                          & CLIP(VOC:Description))
            else
              Message('Sorry, ' &  ' |
                 & StringToEdit[?StringToEdit{PROP:SelStart} |
                       : ?StringToEdit{PROP:SelEnd}]   |
                 & ' is not defined in the vocabulary for this program')
            End
          End
          Begin
            ! insert text snippet from a lookup table into the text control
            GlobalRequest=SelectRecord
            BrowseTextSnippets
            if GlobalResponse=RequestCompleted
              SetClipboard(TS:Description)
              PressKey(CtrlV)
            End
          End
```

```
    Begin
      ! export the memo into a file and edit it using an external
      ! editor
      UseExternalEditorForMemo(StringToEdit) ! call external editor
      Display(?StringToEdit) ! refresh display of text string
      END
  END
End
```

Following are the explanations of the code for each `OF` in the `CASE KeyCode()`, in order.

Tab means tab

The first Basic Editor feature I want to add is to make the TAB key work "properly" within the `StringToEdit` control. I want TAB to move the cursor one tab to the right instead of the present "jump to the next control on the window" functionality.

This is not hard to do as the Ctrl-I key combination has the effect I want for a TAB. The Clarion `PressKey()` command simulates pressing any key combination, and can be used to issue a Ctrl-I to insert a tab when the TAB button is pressed.

Since I've now effectively "disabled" the TAB key from its normal behavior, I should also create a way of moving the cursor via keyboard out of the text control to the next control in the tabbing order on the window. I think ALT-N is a good mnemonic for "tab to Next field". Moving the cursor to the next field is as simple as using the `Select()` command. `Select(?StringToEdit+1)` will cause the focus of the window to shift to the control following the `StringToEdit` in the tabbing order. **NOTE:** Be careful of any controls that you have hidden on a window. They are still in the tabbing sequence and the `Select(?StringToEdit+1)` statement could shift the cursor to a hidden control if it's next in the tabbing sequence. The cursor will disappear on you when this happens. The fix is easy - just put a visible control immediately after the text field in the tabbing order on the window.

Ctrl-a means select all

I want the keyboard shortcut Ctrl-A to be understood as Select All the text in the text control. Another usage of the `SELECT()` command can handle this easily.

```
Select(?StringToEdit,1,Len(StringToEdit))
```

This code will cause all the text in text field from the first character to the end of the field to be selected.

That's all for this week. Next week I'll finish discussing the code.

Download the source

Download the ABC Free templates

Download the ShellEx template

Tim Phillips began programming Clarion with Version 3.0 for DOS. He currently works exclusively with Clarion 5.5 ABC and is anxiously waiting for C6 to go gold. His preferred programming technique involves a lot of bass rock guitar on his cordless headset and vigorous application of the Keep It Simple Stupid principle. When not programming, he can be found trying to write fiction, "building Legos" with his two nieces, or being obsessive about television shows that have gone off the air.

Reader Comments

Add a comment

Clarion Magazine

A Calendar For Date-Limited Browses

by Veronica Chapman

Published 2003-09-05

In this article I will discuss the presentation of information to a user, for which a List Box is often used, and I will argue that there really is no place for that most Clarion of controls, the page loaded list box.

There are two relevant aspects to the presentation of information in a list box. They are (in order of importance): (a) the user does not wish to be overloaded with information, and generally wants to ‘find something out’, and (b) your application should not be overloaded with information, and generally must provide the means of enabling the user to ‘find that something out’.

I propose to discuss how these two requirements are *best* achieved.

The first thing to consider is that there will *always* be a limiting factor, or some limiting factors, determined by the user’s initial knowledge. He/she will think the company’s name begins with a "C", or know that the information lies sometime between two dates, or be fairly sure the person’s address is in Brighton, and so on.

I’m not going to dwell on these specific examples. I will say that in the first case the user needs some means of selecting a database view of all company name entries beginning with "C", in the second case a view dependent on a means of specifying a from-to date range, and in the third case a view likewise dependent on an address selection.

However these examples are not the point itself. The point was (I repeat and submit) is that *there will always be a limiting factor to determine a suitable view for the user* – but now I will add a corollary, which is *whatever that factor is, it will also be suitable for your Application*.

Now I would like to submit one final corollary: *A suitable view for the user will* never *be a page-loaded list*. I invite you to think about that last statement, remembering your own requirements when locating information from a database. I would be interested to hear from any reader who can give me a sound reason for ‘page-loading’ a database view - I (seriously) cannot think of one – but that does not mean to say that there isn’t one. So my corollary holds

until argued down.

> **NOTE:** It is for this reason that, almost immediately after purchasing Clarion many years ago, I threw away all the Clarion Templates (which were largely based on page-loading) ... and I have never used one since. I use a three templates of my own for everything. The main one being an empty Window, an empty Data Section and an empty Code Section. Nevertheless, I believe my rapidity of application development is as fast as anyone using SV Templates. This is largely because (a)The Clarion Language and Workbench are so well designed, it only takes a minute or so to produce a working window, and (b) I retain absolute and complete control over the composition of every function/procedure I create.

The techniques for providing restricted database views are fairly well established in most cases. You can use selection criteria based on typed initial letters (generally alphanumerics), or auto-lookups by the initial sequence of alphanumerics, and you can use sheet's tabs to produce alternative SORTs of the view selected. (And so on).

However many applications seem to gloss over one very important selection criteria that has already been mentioned here. This is the selection by means of a from-to date range. All too often one sees the following implementation:

1. Two date entry boxes are offered, one labeled "From" and the other labeled "To". These are quite often spin boxes;
2. The user sets the dates he or she ‘thinks might be suitable’ – and the Application responds;
3. The response "Invalid date" can occur;
4. The response "Invalid date range" can occur ("From-To" the wrong way round, etc);
5. The response "No database information available for that range" can occur;
6. When setting the range in the first place the user is often guessing (or mentally calculating) a day of the week.

I suggest that the above-described implementation is quite common, and not very professional. It is not very professional because there is no need for most of it.

First of all the ‘two spin or entry box’ implementation is clumsy and does not provide the user with any feedback until the application has processed the selection. There is no feedback as the selection is being set. Whereas if a calendar-type selection is used, then there is feedback. Furthermore invalid dates are prevented with a calendar mechanism.

Secondly there is no reason why the user should be offered settings which are outside the scope of the database. If dates are an important selection, then the database will (must) have a key on the date field (among other keys, of course).

Consequently, before offering the range, an application can inspect the database, finding the earliest-dated record (suitable, of course, in respect of other selection criteria), and likewise the

latest-dated record. If these limits are imposed on the calendar input mechanism, *and if that mechanism always returns any selection the 'right way round'*, then every problem with the implementation described above is eliminated.

There is a standard calendar-type input mechanism built into the Windows Operating System, and it can be accessed as an OLE Window Control. Its limitation is that it will only accept one date (at a time). This, if you imagine one calendar appearing on the screen, quickly followed by another almost identical one, would probably cause more problems than it was worth.

My suggested solution to all of this is embodied in a freeware package I wrote some time ago. It is available for download at the end of this article, and includes a readme file of Clarion application notes.

Included in this package is (what I call) my Date Dialogue Function, a tester for it, and a help file that contains the Interface Reference. I have used this component myself in a number of applications.

Briefly (because it is fully documented by its help file) the Date Dialogue Function is in a .DLL that can be called from any Windows-compatible application written in a suitable programming language. This is because, unlike a Clarion .APP-constructed .DLL, it is a .PRJ-constructed .DLL with (minimal) Windows API compatibility.

You must call the function as if it were a Windows API call, using the PASCAL attribute. *In the case of this ('Clarionized') version only*, do *not* use the `RAW` attribute. When called, with suitable parameters, it will offer the user a calendar. The user can freely move between Months/Years, provided that these remain within the limits determined by the caller (who has determined these parameters from the database, as described above).

A suitable 'generalised' Clarion `MAP` Statement for the `DateDialogue` is:

```
ateDialogue(LONG FunctionCode,|
 LONG CalendarCode,|
 LONG TODAYSample,|
 LONG HelpContextNumber,|
*CSTRING UserHelpFileName,|
 LONG HelpCommandCode,|
*CSTRING DialogueWindowAlternativeCaption,|
 LONG LowerLimitDate,|
 LONG UpperLimitDate,|
*CSTRING DayOfTheWeekTemplate,|
*LONG LowerRangeDate,|
*LONG UpperRangeDate),LONG,|
PASCAL,NAME('DateDialogue')
```

If you have access to my `WINAPIX` kit (see the download links below), a suitable `MAP` statement would be:

```
DateDialogue(sint FunctionCode,|
 sint CalendarCode,|
 LONG TODAYSample,|
 sint HelpContextNumber,|
*ASCIIsz UserHelpFileName,|
 sint HelpCommandCode,|
*ASCIIsz DialogueWindowAlternativeCaption,|
 LONG LowerLimitDate,|
 LONG UpperLimitDate,|
*ASCIIsz DayOfTheWeekTemplate,|
*LONG LowerRangeDate,|
*LONG UpperRangeDate),BOOL,|
PASCAL,NAME('DateDialogue')
```

In either case you would need to include PRODAT32.LIB (from DATEDIAL.ZIP) in your .APP's Library, object and resource files section.

The user can click-pick a date, which is suitably highlighted. He/she can then click-pick a second date. Immediately all dates in between the two are highlighted, showing the selection made. If this selection is wrong, then a further click-pick will set the calendar back to that single date selection, and allow a subsequent click-pick to set the range.

An **Apply** button is available for when the selection looks suitable.

My Date Dialogue will select single dates as well as ranges, and has parameters for limiting choices even further. For example it is possible to arrange that a user's selection must start on (say) a Monday, and end on (say) a Friday, etc.

The Tester in the package (TSTDAT32.EXE) enables every available facility to be tested and thoroughly investigated such that you can, by this method, determine each and every parameter value. Figure 1 shows a screenshot of the Tester:

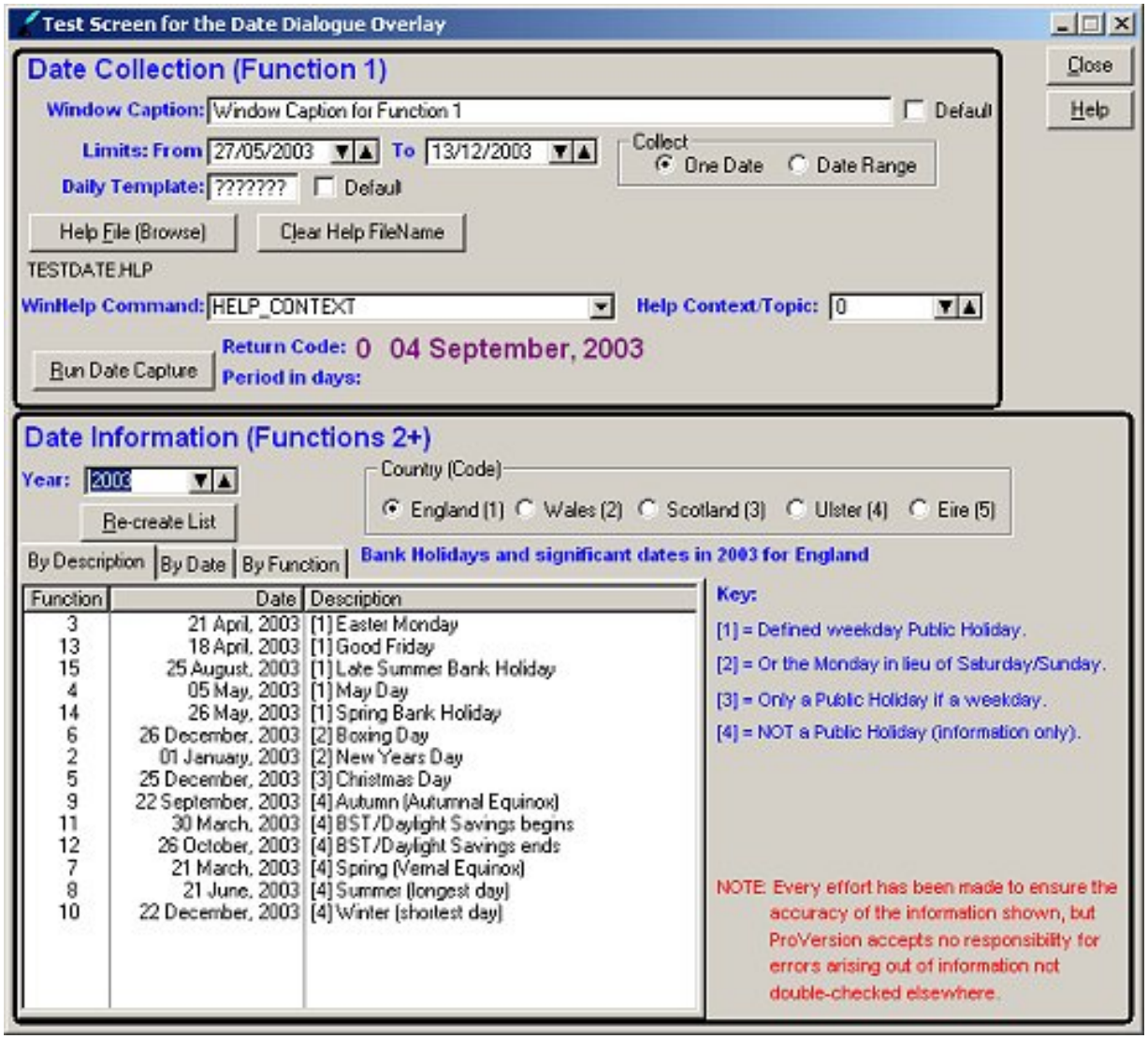

Figure 1. The DateDialogue tester (click on image for full sized version)

Figure 2 shows a screenshot of how the DateDialogue window appears when a "From-To" range of dates is being captured.

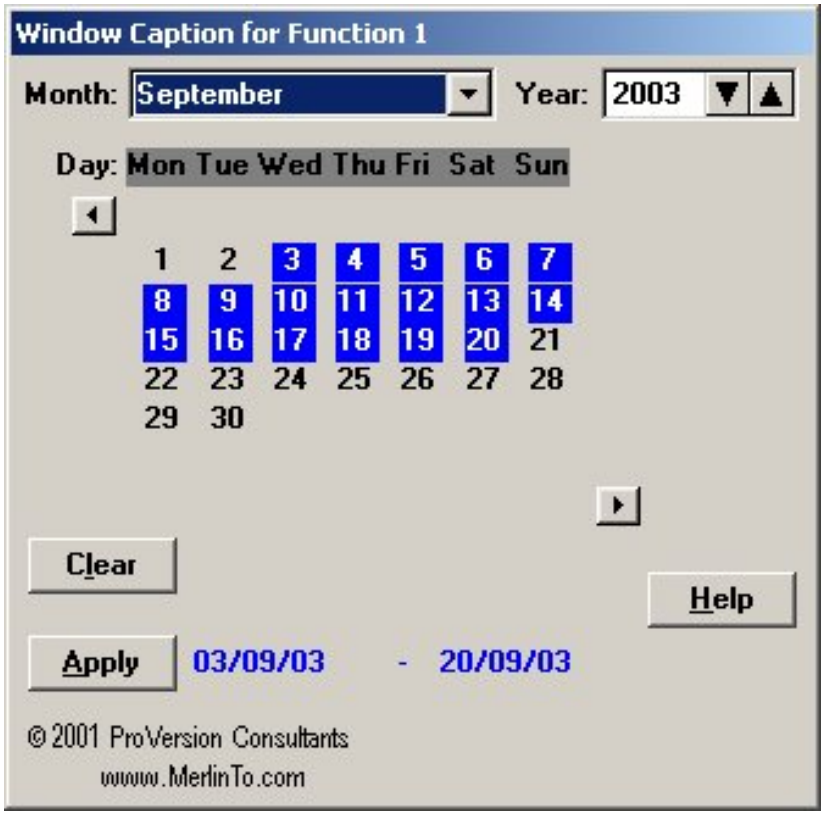

Figure 2. The DateDialogue window

This component (the DateDialogue) is free to anyone who wants to download and use it.

I would be more than interested to hear from any Reader who can argue that this method is not the most user friendly, and most "professional" method of allowing a choice of date or range of dates.

And, finally, here is a little conundrum: Can any Reader tell me what is missing from my Date Dialogue Capture Screen?

Download the source at ClarionMag

Download the source at MerlinTo

Download the WINAPIX kit at ClarionMag

Download the WINAPIX kit at MerlinTo

Veronica Chapman earned a B.Tech in Electronics & Electrical Engineering from Brunel University in 1968, and subsequently embarked on a career as a Programmer/Analyst, first writing code at machine level, and shortly thereafter working with real time systems and communications. By the mid '70s she was using languages such as COBOL and FORTRAN. Never a fan of BASIC or the C language, she discovered Clarion in the mid-90s and, ever since, has used it to create applications for the 16-bit (Win 3.x) and 32-bit Windows

Platforms. An assembly code programmer from way back, Veronica discovered a cornucopia of very useful functions in the Windows API, and set about making these functions available to Clarion applications.

Reader Comments

Add a comment

It's a nice concept but the attached file has no source...
Since the window is inside a DLL there's also no...
In defence of the Page Loaded Browse.... Hey you asked...

Clarion Magazine

Check Please: Managing Conditional Browse Relationships

by Carroll Jolly

Published 2003-09-05

I have an application which is a teacher's scheduling guide, and the data is in a set of parent/child files. The parent file contains the date, instructor, and room number. The child file includes the date and student (classes are held every Tuesday. The two files are linked by date; I display the data from the files as two browses on a window. The problem I have is that I sometimes want to show all of the potential students, and sometimes only the students associated with the selected parent (date) record.

I tried various means of record selection to show either all child records or just the linked child records. It seemed that each solution I tried would only give me one option or the other, but not both. I then tried to create two tabs; one tab with all of the records and the second tab with a subset of records. This gave me problems with trying to update the child record, plus I really only wanted one list box for the children.

The solution

The answer was pretty simple once I looked at the source code created by Clarion for the various filter conditions. First, I set up the range limit field to the child key field and the range limit type to single value, and I set the range limit value to the parent key field (which is a date field same as the child file). That met the requirement of only displaying the linked records.

To optionally display all records, I created a variable in the local data file called allrecords and used the check box control. For the unchecked value, I used 0. For the checked value , I used 1. I then entered the following code in the record filter entry field:

```
allrecords=1) or (allrecords=0
```

Now when I check the box, only those records that are related to the parent record in the parent box are shown. When the checkbox is unchecked, it shows all of the records.

Here's why this works. If I enter just `allrecords=1` as the filter value, Clarion will generate the following code:

```
BRWx::View:Browse{Prop:Filter} = |
'child:date = Parent:date AND (allrecords=1)
```

With the above code, if the `allrecords` field was checked, then I get only the related records. If the box was unchecked, then I get no records since the criteria wasn't met. I circumvented this by adding `" ) OR (allrecords=0"`, which caused Clarion to generate this code:

```
BRWx::View:Browse{Prop:Filter} = |
  'child:date = Parent:date AND (allrecords=1) OR (allrecords=0)
```

The `OR` clause is not part of the `AND` clause – it's on its own. So if it evaluates to true, so will the filter. Now when the window is open and the box is unchecked, all of the records show up because no other condition needs to be met other than allrecords is a 0. If you check the box, then only the records related to the parent box appear.

To finish the code properly, add the code to refresh the window to the checkbox so it updates the browses right away. In the control event , after generated code, accepted type

```
Forcerefresh=true
Do refreshwindow
```

Now you can check or uncheck the `allrecords` control to view all of the records or only the selected records.

Good luck

Carroll Jolly has been in computers since the TRS-DOS days of the Tandy Model 4 (circa 1983), and cut his programming teeth in BASIC. His first Clarion product was Personal Developer and he has been hooked every since. Carroll uses Focus on the mainframe for company data processing, and Clarion for day-to-day tasks. Carroll is married with three children; his hobbies include programming, golf (the other four letter word), and building computers.

Reader Comments

Add a comment

Clarion Magazine

A Basic Editor For Text Fields: Conclusion

by Tim Phillips

Published 2003-09-11

Last week I introduced the Basic Editor, an enhanced TEXT control that adds some basic word processing functionality. This week I'll finish commenting on the source code. To refresh your memory, here's the event handing source code that either handles or initiates the editing features (I've already discussed the features up to and including CtrlA – Select All):

```
case KeyCode() ! what key has been pressed?
   of TabKey ! if TabKey pressed
      ! use the Ctrl-I combination to insert a tab into
      ! the text field
      PressKey(CtrlI)
   of AltN ! if Al-N pressed
     ! select the next field in tabbing order
        !after the text field
      Select(?StringToEdit+1)
   of CtrlA
      ! select all the text in the control
      Select(?StringToEdit,1,Len(StringToEdit))
   of CtrlF
      ! do a search from the top of the control
      do InitialSearch
   of F3Key
      ! do a repeat search
      do RepeatSearch
   of MouseRight
      ! user has right-moused on the text control - display a custom
      !popup menu of options they can use
      EXECUTE Popup('Cut -Ctrl-X|Copy - Ctrl-C|Paste ' |
               & ' - Ctrl-V|Select All|Find ' |
               & ' - Ctrl-F|Repeat Find ' |
               & ' - F3|Search And Replace ' |
               & ' - Ctrl-R|Launch High-Lighted|ReFormat Text|Lookup  ' |
               & 'Vocabulary|Insert Text Snippet - ' |
               & 'Ctrl-T|Use External Editor')
        PressKey(CtrlX) ! cut text to the clipboard
        PressKey(CtrlC) ! copy test to the clipboard
        PressKey(CtrlV) ! paste text from the clipboard
        Select(?StringToEdit,1,Len(StringToEdit)) ! select all text
```

```
      do InitialSearch ! search from the top of the control
      do RepeatSearch ! do a repeat search
      Begin
        ! do search/replace for text
        SearchReplaceWindow(SearchForText,ReplaceText)
        freplace(StringToEdit,CLIP(SearchForText),CLIP(ReplaceText),0)
        Display(?StringToEdit)
      End
      Begin
        ! launch file/URL (as if you had double-clicked on it)
        GLO:ShellExFileName=StringToEdit[?StringToEdit{PROP:SelStart} : |
                  ?StringToEdit{PROP:SelEnd}] ! get currently selected text
        GLO:ShellExFileName=Left(GLO:ShellExFileName) ! left it
        IMPURLHandler(0{PROP:handle},CLIP(Glo:ShellExFileName)) !launch
      End
      do ParagraphBreaksAtLine ! reformat text
      Begin
        ! display vocabulary description if one exits
        VOC:WordPhrase=StringToEdit[?StringToEdit{PROP:SelStart} : |
                  ?StringToEdit{PROP:SelEnd}]
        VOC:WordPhrase=CLIP(Left(VOC:WordPhrase))
        if Access:Vocabulary.TryFetch(VOC:ByWordPhrase)=Level:Benign
          Message(CLIP(VOC:WordPhrase) & ' = ' |
                      & CLIP(VOC:Description))
        else
          Message('Sorry, ' &  ' |
             & StringToEdit[?StringToEdit{PROP:SelStart} |
                   : ?StringToEdit{PROP:SelEnd}]   |
             & ' is not defined in the vocabulary for this program')
        End
      End
      Begin
        ! insert text snippet from a lookup table into the text control
        GlobalRequest=SelectRecord
        BrowseTextSnippets
        if GlobalResponse=RequestCompleted
          SetClipboard(TS:Description)
          PressKey(CtrlV)
        End
      End
      Begin
        ! export the memo into a file and edit it using an external
        ! editor
        UseExternalEditorForMemo(StringToEdit) ! call external editor
        Display(?StringToEdit) ! refresh display of text string
        END
    END
  End
```

Ctrl-f means Find

Adding a FIND feature to a text control takes some fancy footwork. There is no existing feature that can be

"stolen" and reused by another key combination. And the limited control that a programmer has over the text control itself is a challenge. A text control can be thought of as a queue. Each line of text is a record in that queue. You can use `PROP:LineCount` to know how many "record lines" are in your text control "queue" and `PROP:Line` to read any given record in the queue. You can use `Select(?StringToEdit,50,65)` to cause a range of text (i.e. characters 50-65) to be high-lighted.

Programatically, that is about all you can do with a text control. Fortunately, it is enough – if you accept some crudeness in how the FIND feature works.

Conceptually, I will use the `SearchWindow` procedure to get a string of text that the user wants to search for. I will then loop through all the records in the "queue" that is the text control and examine each record for the presence of the string. As I step down through the records, I will move the cursor vertically down line-by-line using `Presskey(DownKey)`. When I find the text of interest in a line, I will use `PressKey(RightKey)` to shift the cursor to the right character-by-character until I reach the beginning of the string I am hunting for.

Since I want to use this logic in two places, it is best written as the routine listed below:

```
InitialSearch Routine
  ! get the text to look for
  SearchForText=SearchWindow()
  ! if the user entered anything, execute a search for it
  if CLIP(SearchForText)>''
    ! select the text control
    Select(?StringToEdit)
        ! put the cursor at the topmost,leftmost corner
    ! of the text control
    PressKey(CtrlHome)
        ! force a display so the user can see the new
    ! location of the cursor
    Display(?StringToEdit)
        ! get how many lines are in the text control
    LineCount = ?StringToEdit{PROP:LineCount}
        ! loop through all the lines in the control
    LOOP I = 1 TO LineCount
      if INstring(CLIP(SearchForText),|
            (?StringToEdit{PROP:Line,I}),1,1)>0
      ! if the string to search for is present
              ! get how many characters in from the left it is
              NumberOfRightKeys=INstring(CLIP(SearchForText),|
                (?StringToEdit{PROP:Line,I}),1,1)
      ! move the cursor to the right until it reaches
              ! the start of the string to search for
      ! move the cursor to the right until it comes to the beginning
      Loop J=1 to NumberOfRightKeys-1
        PressKey(RightKey)
      End
      break
    else
      ! the string to search for is not in this line, so step the
      ! cursor down to the next line in the control
      PressKey(DownKey)
    End
```

```
    End
    ! store the line in the text control where the
    ! cursor currently is
    SearchLineNumber=I
  End
```

Six variables local to the `ViewMemo` procedure need to be declared to use this code:

1. `SearchForText` as a string of 60
2. `LineCount` as a Long
3. `I` as a long
4. `J` as a byte
5. `NumberOfRightKeys` as a byte
6. `SearchLineNumber` as a long

Now, some may call this FIND feature crude, but it does work. The cursor slides down and right and stops on the first character of the string to search for. I remember seeing FIND features work like this in the old DOS days. It is not crude… it is "inspired by a classically timeless design!"

F3key means Repeat Search

Now, what about a repeat search? Well, I can make that work, after a fashion. If I assume that the user has not moved the cursor from the line that the last hit was found on, I can write code to search from the current line downward to the next hit.

Since I again want to use this logic in two places, it is best written as the routine listed below:

```
RepeatSearch Routine
  ! find how many lines are in the text control
  LineCount = ?StringToEdit{PROP:LineCount}
  ! if the text to search for is identified and the line where the
  ! program last hit is not the last line in the control
  if CLIP(SearchForText)>'' and SearchLineNumber<LineCount
    ! loop through the remaining lines in the control
    LOOP I = SearchLineNumber+1 TO LineCount
      ! step down a line
      PressKey(DownKey)
      ! go to the first character on the line
      PressKey(HomeKey)
      if instring(CLIP(SearchForText),|
            (?StringToEdit{PROP:Line,I}),1,1)>0
        ! if the string to search for is present
        ! get how many characters in it is
              NumberOfRightKeys=INstring(CLIP(SearchForText),|
                (?StringToEdit{PROP:Line,I}),1,1)
        ! move the cursor to the right until it reaches the
        ! start of the string to search for
        Loop J=1 to NumberOfRightKeys-1
          PressKey(RightKey)
        End
        break
      End
```

```
    End
    ! store the line in the text control where the
    ! cursor currently is
    SearchLineNumber=I
    ! if to the end of the control
    if SearchLineNumber>=LineCount
      Message('End of Text Field')
    End
  End
```

I'd be the first to say that this is a primitive way to do a Repeat Search. I wouldn't advocate using this sort of tool with the general population of end-users, but it's fine for an in-house utility with knowledgeable people running it; the code is functional. And, more importantly, it's better than no repeat search at all.

Mouse-right popup menu

The right-mouse click has become a fundamental way of interacting with objects on windows since Windows95 came into being. The miniature menus created by right-mousing can be created and empowered via combining the Clarion `Popup()` statement with the `Execute` statement.

The `Popup()` statement works very simply. You feed it a string that is your popup menu entries separated by a `|` character, and it returns a number (1,2,3...) that represents which entry on the popup menu was selected by the user. You can then use the `EXECUTE` statement to cause the appropriate command to be "executed" based upon the listed order between the `EXECUTE` and its partnered `END` statement.

Cut/copy/paste with the "correct" key-combinations

Why the standard popup menu on a text control lists keyboard shortcuts for cut/copy/paste that are not the windows standards of Ctrl-X, Ctrl-C and Ctrl-V, I do not know. Fortunately, it is easy to fix. The correct shortcuts themselves are already understood by the Clarion text control; it is just a matter of correctly listing them on the new popup menu, and then using `PressKey()` to feed the commands in if the user selects them from the popup menu.

Select all, search, repeat search

All these options are easy to empower for the popup menu. They are mostly just a reuse of the routines already defined earlier for use with HotKeys on the text control.

Search/replace

This option uses the procedure of `SearchReplaceWindow` to get a value to look for and a value to replace with from the user.

The freplace() function from the ABC Free standard functions template is then used to scan the `StringToEdit` and replace any instance of the `SearchForText` with the `ReplaceText`.

Finally, a call to `Display()` forces a refresh of the window so any alterations made to the text in `StringToEdit` are visible to the user.

Launch high-lighted file/URL

I think launching a file or URL is a particularly cool feature given how little code is necessary to implement it. I wanted to be able to high-light a file path name or URL written out in the text control and then either open that file or go to that URL with just a single click of my mouse.

All the code to do this is located between the `BEGIN-END` combination (which tells `EXECUTE` to treat the block of code as a single selectable element – in this case, the eighth item on the popup menu).

A function is needed that can enable the `ShellExecute()` API call. This API provides the capability you see in Windows Explorer if you double-click on a file, or if you click on a hyperlink in something like an email. The computer's operating system determines the correct software to view the file/URL with and launches it automatically for you.

You can hand-code the prototype and library for the `ShellExecute()`, call or you can use one of the freebie templates kicking around. I personally like the ShellEx template from Sterling Software – and it is what I've used here.

ShellEX declares a global variable called `GLO:ShellExFileName` and a procedure called `IMPURLHandler()`. You simply set `GLO:ShellExFileName` to whatever you want to "launch" and then use the `IMPURLHandler()` to call the operating system and pass it `GLO:ShellExFileName`.

Bulk reformat text

One of the things that got me started into this Basic Editor adventure was the desire to store and easily read fan fiction I've found on the Internet. Frequently, such information is lightly formatted via HTML in ways that I find distracting when pasted into a Text control.

Studying the normal ways the fan fiction I find is formatted - and some experimentation with "fixes" - led to the following `ParagraphBreaksAtLine` routine that can reformat all the text in a text control to match how I want text formatted for easy reading.

```
ParagraphBreaksAtLine Routine
 ! save double carriage returns to assign them to uncommon characters
 freplace(StringToEdit,'<13><10><13><10>','||||',0)
 ! remove any remaining carriage returns by turning them into spaces
 freplace(StringToEdit,'<13><10>',' ',0)
 ! remove duplicates spaces that are side-by-side so 2 spaces in
 ! a row becomes a single space.
 Free(LocalQueue)
 NumberOfSpacesInARow=0
 Loop I = 1 to Len(StringToEdit)
  case StringToEdit[I]
   of ' '
      ! if we have found a space, increment the counter
      NumberOfSpacesInARow=NumberOfSpacesInARow+1
   else
    case NumberOfSpacesInARow
     of 0
         ! do nothing
     of 1
        case StringToEdit[I]
         of '<13>'
```

```
            ! do nothing
          else
           LQ:String1=' '
           add(LocalQueue)
         End
      else
       LQ:String1=' '
       add(LocalQueue)
     End
     NumberOfSpacesInARow=0
     LQ:String1=StringToEdit[I]
     add(LocalQueue)
  End
 End
 StringToEdit=''
 Loop I = 1 to Records(LocalQueue)
  Get(LocalQueue,I)
  StringToEdit[I]=LQ:String1
 End
 Free(LocalQueue)
 ! add back in appropriate double spaces behind
 ! normal punctuation marks
 freplace(StringToEdit,'. ','.  ',0)
 freplace(StringToEdit,'? ','?  ',0)
 freplace(StringToEdit,'! ','!  ',0)
 ! restore all the double carriage returns
 freplace(StringToEdit,'||||','<13><10><9>',0)
 ! display the string again so the changes are visible
 Display(StringToEdit)
```

Display vocabulary definition

One of the features I wanted is the ability to highlight a word and receive a corresponding vocabulary description. If the `ViewMemo` feature was used in my InternalHelp design, it could prove very useful to comprehension to have basic definitions of words only a highlight and mouse-click away.

The code to do this is very simple. The `VOC:WordPhrase` value is primed with whatever text is currently high-lighted in the text control. A `FETCH()` is then used to look for a match in the list of vocabulary words. If a match is found, the user is showed a message that has the Description field from the vocabulary record that has been found. If no match is found, the user is told that the word high-lighted is not defined in the vocabulary list for the program.

Insert text snippet

The ability to insert generic chunks of text from a lookup table is very useful when consistency is important (or you can get away with reusing the same generic phrases and don't want to bother with retyping everything).

The code for this is also quite simple.

The `GlobalRequest` variable is set to `SelectRecord` and the browse of text snippets is called. When the user returns from `BrowseTextSnippets`, the `GlobalResponse` variable is checked to see if the user completed the request by selecting a `TextSnippets` record. If this was done, the Windows clipboard is filled

with the contents of the `TextSnippets` description field and `PressKey()` is used to issue a "paste from clipboard" command. This will paste the contents of the clipboard wherever the cursor currently is within the text control.

Use external editor for memo

I wanted the ability to use a more powerful editor than even the Basic Editor upon demand of the user. Tools like NotePad, WordPad and TextPad are common, easy-to-use and quite powerful. Critically, they will accept the name of the file to be edited as a command-line parameter. This makes it possible to use the Clarion `RUN()` command to start the external word processor and wait until that editor has been shutdown so the application knows when to re-import the edited ASCII file.

The code for using an External Editor is contained within the UseExternalEditorForMemo() procedure, and does the following:.

- exports a string passed to it into a temporary ASCII file
- calls NotePad to edit the ASCII file and waits until the user has finished editing the text and closed NotePad
- re-imports the ASCII File and assigns whatever is in that file back to a string that will be returned to the procedure that called `UseExternalEditorForMemo()`

- The `Display()` after the call to UseExternalEditorForMemo() will cause a refresh of the `StringToEdit` text control so any edits made by the user will be visible.

  NOTE: A user could use an external wordprocessor to create a file of greater than 40,000 characters. When the program attempts to re-import that file, it will end up truncated. The `UseExternalEditorForMemo()` procedure I've built for usage here makes no attempt to find this condition and warn the user that they are about to lose information that they have typed.

Conclusion

With all this code I now have a powerful `ViewMemo` procedure.

It has basic - but functional – tools for editing and working with text, including the ability to export the text into a more powerful wordprocessor upon user-request.

All I need to do to edit any memo or large string field is pass it into the `ViewMemo` procedure as a parameter and put a `Display(?fieldname)` below the `ViewMemo` call so the text will visibly update when the user returns from `ViewMemo`.

Maybe Clarion 7 will introduce a text control with more hooks to make it easier to manipulate (and with a "real" search and replace mechanism) but until then, you can use Basic Editor to help provide features you really want and need.

Download the source

Download the ABC Free templates

Download the ShellEx template

Tim Phillips began programming Clarion with Version 3.0 for DOS. He currently works exclusively with Clarion 5.5 ABC and is anxiously waiting for C6 to go gold. His preferred programming technique involves a lot of bass rock guitar on his cordless headset and vigorous application of the Keep It Simple Stupid principle. When not programming, he can be found trying to write fiction, "building Legos" with his two nieces, or being obsessive about television shows that have gone off the air.

Reader Comments

Add a comment

"Why the standard popup menu on a text control lists...
One thing you may have noticed is that when you display...

Clarion Magazine

Translation, Clarion Style: Currencies And Measurement

by Nardus Swanevelder

Published 2003-09-12

In my previous translation article I looked at how to use the Clarion template prompts, environment files, a new derived error class and a couple of TRN files to translate an application into a different language.

In this three part article I will address the following remaining questions:

- What about currency indicators and measurement formats?
- What about dynamically created text, controls and windows?
- What about dynamically created reports?
- What about screen design?
- What about third party applications?
- How do I change the language at runtime because the solutions given so far mostly give me only an option of one language per EXE?

To recap what was covered in the previous article, lets look at how to implement the `Errorclass`, ABError.Trn file and the Afrikaans.Env file.

If you haven't done so yet, copy the MyError.inc and MyError.clw files to the c55\libsrc folder. You should copy the International.tpl file to the c55\thirdParty\template folder and register the template in Clarion.

Open the AutoLogSingle.Inc file to follow the steps that are going to be discussed next.

1. Click on the **Global** button, and if **Enable Run-time Translation** is not selected, please select it now.
2. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`. If it is not available in the drop down list, go one step back and click on the Refresh Application Builder Class Information button, then click on the **General** button and change the **Error Class** to MyErrorClass.
3. Click on the **Configure** button under **Run-time Translator**, click on **Additional Translator Groups**, and add a new group if it is not there already. Specify the **Source** file and the **Group** name. In the example file I used AutoLog-Afr.Trn as a source name and `Autolog` as a group name.
4. While still in the global properties, click on the **Embed** button and in the **After Global Includes Embed** add the following:

```
Language BYTE
```

This variable will be used to store the preferred language. In the **Global Objects, ABC objects, Error Manager (MyErrorClass), Init Procedure, After the Parent call** embed add the following code:

```
INIMgr.Init('AutoLog.INI')
Language = INIMgr.Fetch('Preferences','Language')
If Language = 0
   Language = 1
.
GlobalErrors.MyInit(Language)
If Language = 1
   LOCALE('Afrikaans.ENV')
.
```

First you have to retrieve the preferred language from an INI file (or whatever method you prefer), then you initialize the `ErrorClass` with the preferred language and set the Environment file by using the `Locale` statement.

5. Go to the **UpdateVehicles** form and add the **International Local Declaration Template** to the form. This will change the money value controls to use the South African Rand indicator (R).
6. Compile the app and run the app. The application will now be in Afrikaans and it will use the Rand indicators on the **Update Vehicle** form. Also see the change in the locale, for example am is now vm (voor middag – direct translation: before mid day).

NOTE: The Translator is case and space sensitive. Each of the following lines of text is unique to the Translator:

```
Enter the students ID number:
 Enter the students ID number:
Enter the students ID number :
Enter The Students ID Number:
```

Now that you're up to speed with what I covered in the previous article, it's time to continue with the outstanding questions.

Currency indicators, measurements etc.

I looked at various options for currency indicators and measurements, but the only one that seemed viable was to look at a template to override these settings. The idea is to go through the window controls, find the control's picture and then change it as and if necessary. The following table lists some of the template language statements that will be used in the template:

`%Control`	The field "equate" labels of all controls in the window. Multi-valued. Dependent on `%Window`.
`%ControlUse`	The control's `USE` variable (not field equate). Dependent on `%Control`.
`%ControlType`	The type of control (`MENU`, `ITEM`, `ENTRY`, `BUTTON`, etc.). Dependent on `%Control`.

%ControlField	All fields populated into the LIST, COMBO, or SPIN control. Multi-valued. Dependent on %Control.
%ControlFieldPicture	Contains the picture token of the field in the LIST or COMBO control. Dependent on %ControlField.
%ControlFieldFormat	Contains the portion of the FORMAT attribute string that applies to the field in the LIST or COMBO control. Dependent on %ControlField.

The biggest problem that I had to resolve was something that in the end turned out not to be a problem at all. I was under the impression that a control's display picture is not available at runtime. In fact, PROP:Text returns the picture quite nicely, but I didn't know that. I did know that in a list box it is possible to get the display picture using the template statement %Control{PROP:Format}.

So how did I solve the picture problem for form controls *without* using PROP:Text? I used the template language to examine the database. If you've made a lot of picture changes in your application and your dictionary pictures are out of date then this won't work, but you should really be making your changes in the dictionary anyway and then propagating them to the application. In any case, although I could have used PROP:Text to get the picture at runtime, the following explanation will still go the template route.

Getting the picture

To access the picture in the dictionary you use the following template statements: %File, %FilePrefix, %Field, %FieldPicture and %FieldID. %Primary, %Secondary and %OtherFiles are multi-valued variables and contain all the primary, secondary and other files used in the procedure.

At the beginning of the procedure, you run through the procedure's files with the #FIX(%File,%Primary), #FIX(%File, %Secondary) and #FIX(%File,%OtherFiles) statements. Then you use the #For(%Field) statement to get the individual fields. Then you can get the field picture through the %FieldPicture statement.

I used two Instring statements to determine if a field has a decimal picture. If the picture starts with N it is a numerical field, and using the second Instring I determine if it is a decimal picture by searching for a ".". If both of the Instring statements are true I add the field's use variable and picture to a queue using the following template statements:

```
Q1:FieldID      = Upper('%FilePrefix:%FieldID')
Q1:FieldPicture = '%FieldPicture'
Add(FieldQ)
```

Thus the queue only contains fields that have decimal pictures. You could refine this to cater to your specific needs, e.g. if your currency picture starts with a $ you could search for that rather than looking for decimal values, or you could look for the indicator for centimeter and change it to the indicator for inches.

At this point you have run through all the files in the procedure and have populated a queue with all the fields that match your picture criteria. As this needs to be done for each procedure, the template that you create should be a local extension template. Here's the start of the template

```
#EXTENSION(InternationalLocalD,'International Local Declarations')
!************************************************************
```

The following sheet section is currently empty as no prompts are needed for this template. This could change depending on your requirements. You could add prompts asking for the search string and replace string.

```
#SHEET
  #Display('No parameters for this Template')
#ENDSHEET
```

Use the `#AT(%DataSection)` statement to declare the queue that will hold all the fields that met the picture requirement.

```
#AT(%DataSection),PRIORITY(3000)
!#Declare Queue to hold fieldnames and pictures
FieldQ    Queue,PRE()
Q1:FieldID       String(50)
Q1:FieldPicture  String(20)
          .
LCL:PictureString  CString(20)
#ENDAT
```

At the `%WindowManagerMethodCodeSection` run through all the files, check the picture and populate the queue. The replace string (`LCL:PictureString`) is hard coded but it could be changed by adding a prompt to the template. The replace string I used is replacing the currency indicator with the South African Rand indicator.

```
#!*********************************************************
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(8000)
 LCL:PictureString = '@N~R~-14.2'
 Free(FieldQ)
     #FIX(%File,%Primary)
     #FOR(%Field)
       If Instring('@N',Upper('%FieldPicture'),1,1)
          If Instring('.',Upper('%FieldPicture'),1,1)
             Q1:FieldID        = Upper('%FilePrefix:%FieldID')
             Q1:FieldPicture   = '%FieldPicture'
             Add(FieldQ)
          End
       End
     #ENDFOR
     #FIX(%File,%Secondary)
     #FOR(%Field)
     …
     #ENDFOR
#FOR(%OtherFiles)                       #! Open Other files
    #FIX(%File,%OtherFiles)
    #FOR(%Field)
     …
    #ENDFOR
#ENDFOR
```

Again, if you're using `PROP:Text` you don't actually need the above code.

Next, run through the controls on the screen and check to see if each control exists in the queue. If it exists in the queue you know what the picture is and it can be change if needed.

There are other methods that you can use as well, but this was the quickest I could find that met my requirements. It would be possible to improve this template to cater to different search strings like `@D` or `@T`. You could also enhance the template by adding the capability not to change the picture of certain controls.

One thing to keep in mind is that the declaration of `LCL:PictureString` must be a `CString`. Don't ask why, just believe it!

The above code needs to be added to the bottom of the previous code in the `%WindowManagerMethodCodeSection`, as follow:

```
 #FOR(%Control)
     #If(%ControlUse <> '')
        Q1:FieldID = Upper('%ControlUse')
        Get(FieldQ,Q1:FieldID)
        If not error()
           ?%ControlUse{PROP:Text} = LCL:PictureString
        End
     #ENDIF
 #ENDFOR
#ENDAT
```

I am sure I am forgetting something… Oh yes, how list boxes are handled! For list boxes you need to change the above code slightly. You need to check if the control is an Entry control or a list box control so the above code needs to change to the following:

```
#FOR(%Control)
     #IF(%ControlType='ENTRY')
        #If(%ControlUse <> '')
           Q1:FieldID = Upper('%ControlUse')
           Get(FieldQ,Q1:FieldID)
           If not error()
              ?%ControlUse{PROP:Text} = LCL:PictureString
           End
        #ENDIF
```

Remember that the picture of a control in a list box start with `@` and ends with `@,` for example: `@n~$~-10.2@`. It is also necessary to check the string in a loop because there might be more than one column that needs changing. What I have done is to find the `@N` and then find the corresponding end `@`. If this end `@` is a `.2@` I replace the picture with the picture in `LCL:PictureString`. I continue to search for `@N` and `.2@` until the instring statement returns zero, and then I break out of the loop. Here is the code to implement this:

```
     #ELSIF(%ControlType='LIST')
        I# = 1
        Loop
           !First find the "@N"
```

```
            I# = Instring('@N',Upper(%Control{PROP:Format}),1,I#)
            !Find end of picture string by finding the next "@"
            K# = Instring('@',Upper(%Control{PROP:Format}),1,I#+1)
            !See if picture string ends in ".2@"
            J# = Instring('.2@',Upper(%Control{PROP:Format}), 1, I#+1)
            !if ending "@" & ".2@" part of same picture change
            If K# = J#+2
               LCL:Temp_Picture = Sub(%Control{PROP:Format},1,I#-1) |
                           & Clip(LCL:PictureString) & '@' & |
                 Sub(%Control{PROP:Format},J#+3,Len(%Control{PROP:Format}))
               %Control{PROP:format} = LCL:Temp_Picture
            End
            !If no more "@N" or ".2@" left in format string break
            If I# = 0 or J# = 0 then break.
            !found ".2@" but is further in string, continue search
            If J# > K#
               I# = K# + 1
               Cycle
            End
            I# += 1              !Increase counter to continue search
         End
      #ENDIF
  #ENDFOR
#ENDAT
```

That's the code using the queue to keep the picture strings, but what will the code look like using `PROP:Text`? Here's another version with that code:

```
#EXTENSION(InternationalLocalD,'International Local Declarations')
!************************************************************

#SHEET
  #Display('No parameters for this Template')
#ENDSHEET

#AT(%DataSection),PRIORITY(3000)
!#Declare Queue to hold fieldnames and pictures
LCL:PictureString  CString(20)
#ENDAT

#!************************************************************
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(8000)
 LCL:PictureString = '@N~R~-14.2'
#FOR(%Control)
     #IF(%ControlType='ENTRY')
       If Instring('@N',Upper(%Control{PROP:Text}),1,1)
          If Instring('.',Upper(%Control{PROP:Text}),1,1)
              ?%ControlUse{PROP:Text} = LCL:PictureString
          End
       End
     #ELSIF(%ControlType='LIST')
```

```
         I# = 1
         Loop
            !First find the "@N"
            I# = Instring('@N',Upper(%Control{PROP:Format}), 1, I#)
            !Find end of picture string by finding the next "@"
            K# = Instring('@',Upper(%Control{PROP:Format}),1,I#+1)
            !See if picture string ends in ".2@"
            J# = Instring('.2@',Upper(%Control{PROP:Format}), 1,I#+1)
            !if ending "@" & ".2@" part of same picture change
            If K# = J#+2
               LCL:Temp_Picture = Sub(%Control{PROP:Format},1,I#-1) |
                           & Clip(LCL:PictureString) & '@' & |
                 Sub(%Control{PROP:Format},J#+3,Len(%Control{PROP:Format}))
               %Control{PROP:format} = LCL:Temp_Picture
            end
            !If no more "@N" or ".2@" left in format string break
            If I# = 0 or J# = 0 then break.
            !found ".2@" but is further in string, continue search
            If J# > K#
               I# = K# + 1
               Cycle
            End
            I# += 1              !Increase counter to continue search
         End
      #ENDIF
  #ENDFOR
#ENDAT
```

Dynamically created text

If you create text that will be displayed via a use variable, you can translate the text with the following statement:

```
MyVar=Translator.TranslateString(MyVar)
```

Remember that you have to add this after the window has been opened.

Dynamically created controls

If you are creating controls dynamically you can translate them as well. The following statements have to be used after the window has been opened:

```
Translator.TranslateControl(?Sound)
Translator.TranslateControl(?VolumePrompt)
```

You can also use the statement in such a way that it will translate all the controls starting at a certain control and ending at another control. The following statement will translate all the controls starting at `?Sound` and stopping at `?VolumePrompt`.

```
Translator.TranslateControls(?Sound,?VolumePrompt)
```

Dynamically created windows

If you create windows dynamically you just have to add one statement after the window has been opened:

```
Translator.TranslateWindow
```

So far I've covered currency indicators and measurement formats, and dynamically created controls, windows and reports. Next week I'll look at screen design issues, third party applications, and in the following week at changing languages at runtime.

Download the source

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

Clarion Magazine

Translation, Clarion Style: Selected Topics

by Nardus Swanevelder

Published 2003-09-18

Last week I discussed some of the remaining questions in my discussion of translation, specifically currency indicators and measurement formats, and dynamically created controls, windows and reports. In this article I'll look at screen design issues and third party applications, and I'll begin a discussion of changing languages at runtime.

Screen design

Although the scope of this article is not to address screen design for multiple-languages there are some simple rules you can follow:

1. The replace text should be the same length or shorter than the search text, otherwise the screen might have to be reformatted. English text tends to be shorter than the equivalent text in other languages so leave some extra space.
2. Some designers advocate using right justification and not left justification for prompts. For instance, this:

Enter the student's Full Christian Name:
Enter the student's Surname:
Enter the student's ID:

becomes this:

Enter the student's Full Christian Name:
Enter the student's Surname:
Enter the student's ID:

Chinese characters

The subject of Chinese characters (and other double byte characters) comes up regularly in the Clarion newsgroups. Here's a suggestion from Kelvin Chua, posted in comp.lang.clarion (quoted with

permission):

1. *Add a text box.*
2. *Make sure that there is no FONT set.*
3. *Set FONT Size to 11, it is more presentable.*
4. *Run any Chinese processor.*
5. *You should be able to enter the Chinese characters.*

All my applications allow customers to key in Chinese into the entry form.

Tom Kam, also in comp.lang.clarion, has this to say about Chinese characters in Windows XP (quoted with permission):

> *In WinXP, we don't need external Chinese System (e.g.: RichWin, NJStar) to input/display Chinese characters. WinXP already supports it. But, I believe we have to configure it properly in Settings/Control Panel/Regional and Language Options*
>
> 1. *In Languages tab, check the "Install files for East Asian languages". XP will ask for the WinXP installion CD. A few Chinese fonts will be copied onto the hard disk.*
> 2. *On the Advanced tab, "Select a language to match the language version of the non-Unicode program you want to use", select Chinese(PRC) or Chinese(Taiwan) as needed. Then tick the associated code pages below it (defaulted). Will need to restart the machine after the change.*
>
> *Now it should be able to display Chinese characters properly. This should also fix the original problem that Chinese characters changed into '?' after they were selected.*
>
> *In my Clarion experience, Chinese characters can display in any type of controls (e.g.: Text, String, Entry, List, Combo...), but only Text control can take Chinese characters input. And I don't need to use special Chinese font - MS Sans Serif and Courier are just fine.*

Hand coded reports

If translation is enabled Clarion will generate the following code for a report:

```
Previewer.AddItem(Translator)
Translator.TranslateWindow(SELF.Report)
```

If you are using a source only procedure you can use the code above or you can use something like this:

```
SetTarget(Report)
Translator.TranslateWindow
SetTarget(PreviewWindow)
Translator.TranslateWindow
```

I have not tested the above in a source report so it might require some playing to get the exact code correct.

Third party software/templates

Third party translation will depend on the product that you use. I mainly use products from Capesoft, IceTips and Ingasoftplus. Capesoft's products and the IceTips Cowboy SQL Templates support translation, and the products I use from Ingasoft do not require translation.

Single EXE ASCII translation (AutoLogRunTime – Ascii.App)

To enable runtime translation from an ASCII file you have to dig deeper into the translator class. If you look at the `TranslatorClass.Init` method (ABUtil.clw) you can see that the class instantiates a Translator queue, and when you look at the `TranslatorClass.AddTranslation` method you will see you require a search string and a replace string. This shows that it is necessary to populate the Translator queue with the search and replace strings that are stored somewhere. The next question is where in the app should the search and replace strings be added to the queue? The obvious answer is in the `TranslatorClass.Init` method, after the parent call.

Here's how to add search and replace strings from an ASCII file to the translator. First, open the AutoLogRuntime-Ini.app file and follow these steps:

1. Click on the **Global** button. If **Enable Run-time Translation** is not selected, please select it now.
2. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`.
3. Click on the **Configure** button under Run-**time Translator**, click on **Additional Translator Groups**, and make sure there are no Translator groups added. This will be done at runtime.
4. While still in the global properties, click on the **Embed** button and in the **After Global Includes** embed add the following (could also add it under the **Data** button):

```
Language          BYTE            !Language Flag
LanguageFile      STRING(255)     !Language FileName
From_Pos          Long            !Read from this postition
Recs_read         Long            !Number of recs read
LCL:Search        STRING(255)     !Search String
LCL:Replace       STRING(255)     !Replace String
LCL:Count         Long            !Counter
LCL:Count1        Ushort          !Counter
LCL:Count2        Long            !Counter
```

In the **Global Objects, ABC objects, Error Manager (MyErrorClass), Init Procedure, After the Parent call** add the following code:

```
INIMgr.Init('AutoLog.INI')
Language = IniMgr.Fetch('Preferences','Language')
If Language = 0
   Language = 1
End
```

```
GlobalErrors.MyInit(Language)
If Language = 1
   LOCALE('Afrikaans.ENV')
End
```

The code above retrieves the preferred language from an ASCII file (or whatever method you prefer), then initializes the Error Class with the preferred language and set the Environment file by using the `Locale` statement.

5. Because the search and replace strings are saved in an ASCII file it is necessary to add the file declaration at the **Global Data** embed.

```
AsciiFile  FILE,DRIVER('ASCII'),NAME(LanguageFile),PRE(A1),THREAD
RECORD       RECORD,PRE()
Line           STRING(255)
             END
           END
```

6. The next step is to override the `TranslatorClass.Init` method and to do that the following code is added in the Gl**obal Objects, ABC objects, Runtime Translator (TranslatorClass), Init Procedure** embed, after the parent call.

```
LanguageFile = INIMgr.Fetch('Preferences','LanguageFile')
If LanguageFile = ''
   LanguageFile = 'AutoLog-Afr.trn'
End
Close(AsciiFile)
Open(AsciiFile)
If error() then stop(error()).
From_pos = 1
Get(AsciiFile,From_pos)
!Find size of file
Recs_Read = Bytes(AsciiFile)
From_Pos += Recs_Read
!Read first record
Get(AsciiFile,From_pos)
LCL:Count1 = Instring('(',Clip(A1:Line),1,1)
!Find the number of groups in the file
LCL:Count2 = Sub(Clip(A1:Line),LCL:Count1+1,|
  Len(Clip(A1:Line))-2)
!Loop through number of groups
Loop LCL:Count = 1 to LCL:Count2
  Recs_Read = Bytes(AsciiFile)
  From_Pos += Recs_Read
  !Read search string
  Get(AsciiFile,From_pos)
  If Error() then Break.
```

```
    If LCL:Count = 1
      LCL:Count1 = Instring('(',Clip(A1:Line),1,1)
    End
    LCL:Search = Sub(Clip(A1:Line),LCL:Count1+2,|
      Len(Clip(A1:Line))-LCL:Count1-3)
    Recs_Read  = Bytes(AsciiFile)
    From_Pos += Recs_Read
    !Read replace string
    Get(AsciiFile,From_pos)
    If Error() then Break.
    LCL:Replace = Sub(Clip(A1:Line),LCL:Count1+2,|
      Len(Clip(A1:Line))-LCL:Count1-3)
    !Add search & replace string to translator queue
    Translator.AddTranslation(Clip(LCL:Search),Clip(LCL:Replace))
    If error() then stop(error()).
  End
  Close(AsciiFile)
```

7. Go to the **UpdateVehicles** form and add the **International Local Declaration Template** to the form. This will change the money value controls to use the South African Rand indicator (R).
8. Compile the app and compare the different language replace strings to test the translations. If you are using AutoLog-Afr.Trn, the application will be in Afrikaans.

It would also be possible to use a normal INI file and use `IniMgr.Fetch` to retrieve the search and replace strings.

To improve the ASCII process the next step would be to implement the AnyAscii class as discussed in ASCIIing For More

Summary

In this article I continued with my discussion of the remaining "unanswered questions" about translation, with a look at screen design issues, third party applications, and changing languages at runtime. Next time I'll finish up the runtime translation discussion with some advanced techniques.

Download the source

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

Clarion Magazine

Reborn Free CLARION online

Translation, Clarion Style: Runtime Issues

by Nardus Swanevelder

Published 2003-09-18

So far in this series I've discussed a number of translation issues including currency indicators and measurement formats, dynamically created text, controls and windows, dynamically created reports, screen design, and third party applications. I also began discussing how to change translation languages at runtime.

In this article I'll complete the runtime translation discussion with examples of loading translation strings from a database, and applying this translation to a multi-DLL application.

Single EXE translation from a database table (AutoLogRunTime – Language.App)

I found the use of an ASCII or INI file for storing the translation strings (as discussed last week) cumbersome. Where do you store the ASCII/INI file, on the user's local drive? On the network drive? If it is saved on the network all the ASCII `GET` statements are done over the network. If it is saved on the user's local drive, then how do you do maintain all the ASCII/INI files?

Is there another solution? Yes: Use a database table. The advantages of this are that it is centrally stored and to develop a browse and update from for the language table is easy. It is possible to do the same for an ASCII/INI file but it is a lot more difficult. I have been told that a language table is more difficult to maintain than an ASCII/INI file, but I disagree with this statement.

What columns should the table have? For this article `Language`, `SearchString` and `ReplaceString` columns should be sufficient. Using a database table means that the table can be a TopSpeed table, MS SQL table etc.

Follow these steps to get runtime translation to work in a single EXE application.

1. Open the dictionary and add a table with the columns as mention above. The dictionary that comes with the source for this article contains the required table already.

2. Open the AutoLogRuntime – Language.app
3. Click on the **Global** button, if **Enable Run-time Translation** is not selected, please select it now.
4. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`.
5. Click on the **Configure** button under **Run-time Translator**, click on **Additional Translator Groups**, and make sure there are no translator groups added. This will be done at runtime.
6. While still in the global properties, click on the **Embed** button and in the **After Global Includes Embed** add the following:

```
INIMgr.Init('AutoLog.INI')
!get language preference
GLB:Language = INIMgr.Fetch('Preferences','Language')
If GLB:Language = 0
  GLB:Language = 1
  LAN:Language = 'AFR'
End
```

7. The next step is to override the `TranslatorClass.Init` method and to do that the following code is added in the **Global Objects, ABC objects, Runtime Translator (TranslatorClass), Init Procedure** embed, after the parent call.

```
Access:Language.Open()
Access:Language.Usefile()
Set(LAN:Language_Key,LAN:Language_Key)
Loop until Access:Language.Next()
  If LAN:Language <> 'AFR' then break.
  Translator.AddTranslation(Clip(LAN:Search_Text),|
    Clip(LAN:Replace_Text))
  If error() then stop(error()).
End
Access:Language.Close()
```

8. The last step is to add a browse and update form for the new language table.
9. Compile the app and compare the different language replace strings to test the translations. If you use the language.tps file that ships with the source the following strings will be translated:

Update Vehicle	
Registration No.	Registrasie Nr
Make	Maak
Update Expenses (under Update Vehicle)	
Service Date	Diens Datum

Description Beskrywing

Multi DLL translation (AutoLogDict.App, AutoLogProc.App and AutoLogRun.App)

One thing that very seldom gets enough attention in Clarion articles (from my point of view) is how to implement the described feature in a multi-DLL environment. So here is my explanation of how to do runtime translation in a multi DLL.

Before looking at what is different in a multi-DLL app in regards to implementing the translator class lets look at how I have split the AutoLog app. I have created a Data DLL which contains the global data and table definitions (AutoLogDict), I have created an app which contains some of the procedures (AutoLogProc), and I have created an app which contains the Main Frame and some procedures (AutLogRun). I know this is not the correct way to split an app but for the purposes of demonstrating the working of translation in a multi-DLL app I though this to be the easiest.

The next thing to understand is the sequence in which the EXE and DLLs are executed. When you execute the EXE the following happens:

First the program file AutoLogRun.exe is loaded, and the operating system finds a table of data in the file which states, effectively, "this program uses the following list of functions from the DLL AutoLogDict.DLL and AutoLogProc.DLL". This is called a list of *imports* or *imported functions* from the DLLs AutoLogDict.DLL and AutoLogProc.DLL in the program AutoLogRun.exe.

The loader code next searches for the files AutoLogDict.DLL and AutoLogProc.DLL, and if it finds them then the files are loaded. Inside the AutoLogDict.DLL and AutoLogProc.DLL files is another list. This list, called the *export list*, gives the address inside the DLL file of each of the functions which the DLL allows other programs to access.

Then the loader goes through both lists and fills in a table inside AutoLogRun.exe (actually the copy of that table in memory) with the addresses of all those functions in AutoLogDict.DLL and AutoLogProc.DLL (based on where the DLLs were loaded in memory by the OS). Now, whenever the program wants to call a function in the DLL it simply uses the address stored in the appropriate table entry by the loader. (If you want more info on the use of the addresses please see Carl Barnes article on Optimizing DLL Loading - Introduction to Rebasing)

After all of that, what is the sequence in which the DLLs will be loaded? You can put a stop in the code to check this, but AutoLogDict is executed first, then AutoLogProc, and after that AutoLogRun. Remember that I am talking about the sequence in which the translation code will be executed in the apps.

Another thing to keep in mind is that the global `ErrorClass` is initialized in the AutoLogRun app, and in the two DLLs a local `ErrorClass` is initialized and then is assigned to the global `ErrorClass` reference.

```
LocalErrors.Init
GlobalErrors &= LocalErrors
```

In the first article I wrote that in a multi-DLL environment you had to add the translation groups to each DLL as well. In a translation at runtime environment it means that the translation pairs need to be added to the translation queue in each app.

AutoLogDict.app

1. Open the AutoLogDict.app
2. Click on the **Global** button, if **Enable Run-time Translation** is not selected, please select it now.
3. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`.
4. Click on the **Configure** button under Run**-time Translator**, click on **Additional Translator Groups**, and make sure there are no translator groups added. This will be done at runtime.
5. While in the global properties add a global queue to keep the search and replace strings.

```
GLB:Language            BYTE !Language Flag
Language_Queue          QUEUE,PRE(LQ)
Search_Text               STRING(255)
Replace_Text              STRING(255)
                        END
```

6. Add the following code to **Program Setup** embed

```
INIMgr.Init('AutoLog.Ini')
GLB:Language = INIMgr.Fetch('Preferences','Language')
If GLB:Language = 0
   GLB:Language = 1
End
If GLB:Language = 1
   LOCALE('Afrikaans.ENV')
   LAN:Language = 'AFR'
End
Access:Language.Open()
Access:Language.UseFile()
Free(Language_Queue)
Set(LAN:Language_Key,LAN:Language_Key)
Loop until Access:Language.Next()
  If LAN:Language <> 'AFR' then break.
  Translator.AddTranslation(Clip(LAN:Search_Text), |
    Clip(LAN:Replace_Text))
  LQ:Search_Text  = Clip(LAN:Search_Text)
  LQ:Replace_Text = Clip(LAN:Replace_Text)
```

```
   Add(Language_Queue)
   If error() then stop(error()).
End
Access:Language.Close()
```

The code above runs through the language file and adds the search and replace strings to the queue. This queue will be used in the other apps to add the translation pairs to the translator queue.

7. Compile this app first. Remember that due to the naming convention used in this example that you might have to do a **Generate all** and a recompile to clear all the unresolved errors.

AutoLogProc.app

1. Open the AutoLogProc.app
2. Click on the **Global** button, if **Enable Run-time Translation** is not selected, please select it now.
3. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`.
4. Click on the **Configure** button under Run-**time Translator**, click on **Additional Translator Groups**, and make sure there are no translator groups added. This will be done at runtime.
5. While in the global properties add a global queue to keep the search and replace strings.

```
GLB:Language              BYTE !Language Flag
Language_Queue            QUEUE,PRE(LQ)
Search_Text                 STRING(255)
Replace_Text                STRING(255)
                          END
```

Remember to set the **Storage Class** to **External DLL** for `GLB:Language` and `Language_Queue`.

6. Add the following code to **Global Objects, ABC objects, Runtime Translator (TranslatorClass), Init Procedure** embed, after the parent call:

```
Loop I# = 1 to Records(Language_Queue)
  Get(Language_Queue,i#)
  Translator.AddTranslation(Clip(LQ:Search_Text),|
    Clip(LQ:Replace_Text))
End
```

7. Check that the Browse and form update for language file are created.
8. Compile this app second. Remember that due to the naming convention used in this example that you might have to do a **Generate all** to clear all the unresolved errors

AutoLogRun.app

1. Open the AutoLogDict.app
2. Click on the **Global** button, if **Enable Run-time Translation** is not selected, please select it now.
3. Scroll to the **Classes** tab, click on the **General** button, change the **Error Class** to `MyErrorClass`.
4. Click on the **Configure** button under **Run-time Translator**, click on **Additional Translator Groups**, and make sure there are no translator groups added. This will be done at runtime.
5. While in the global properties add a global queue to keep the search and replace strings.

```
GLB:Language            BYTE !Language Flag
Language_Queue          QUEUE,PRE(LQ)
Search_Text               STRING(255)
Replace_Text              STRING(255)
                        END
```

   Remember to set the **Storage Class** to **External DLL** for `GLB:Language` and `Language_Queue`.

6. In the **Global Objects, ABC objects, Error Manager (MyErrorClass), Init Procedure, After the Parent call** embed add the following code:

```
GlobalErrors.MyInit(GLB:Language)
If GLB:Language = 1
   LOCALE('Afrikaans.ENV')
End
```

   The above code needs to be added in this app as this is the place where the global `ErrorClass` is initiated.

7. Add the following code to **Global Objects, ABC objects, Runtime Translator (TranslatorClass), Init Procedure** embed, after the parent call:

```
Loop I# = 1 to Records(Language_Queue)
  Get(Language_Queue,i#)
  Translator.AddTranslation(Clip(LQ:Search_Text), |
    Clip(LQ:Replace_Text))
End
Free(Language_Queue)
```

   In the above code remember to free the memory that is allocated to the language queue, as shown.

8. Compile the app and compare the different language replace strings to test the translations. If you use the language.tps file that ships with the source the following strings would have been translated:

Update Vehicle	
Registration No.	Registrasie Nr
Make	Maak
Update Expenses (under Update Vehicle)	
Service Date	Diens Datum
Description	Beskrywing

What about speed?

If your screen contains a huge number of prompts, strings, controls etc then there is an increase in the time it takes to translate all the controls on the screen. It can lead to a situation where the old text is displayed first, and then it is replaced with the new text.

What about the size of the EXE and or DLLs and what about memory usage?

Your EXE or DLL will increase in size equal to the size of your TRN file (this is if you are using the single TRN file method. The multi-DLL runtime example will use more memory initially due to the extra queue that keeps the translation pairs. This memory is freed in the main EXE.

Summary

Now you know how to add translation to your application using the standard Clarion template prompts, as well as how to add runtime translation to your app using an ASCII/INI file or using a database table. You also have a good starting point on how to change currency and measurement indicators at runtime using the international template. I hope this will assist you in learning about translation. Feel free to use any of the code in this article, but please let me know if you fix any bugs or make improvements.

Download the source

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

Clarion Magazine

Product Review: NetTalk, Part 1

by Tom Hebenstreit

Published 2003-09-19

More and more often, customers, clients and users now take it for granted that programs should be Internet and/or network aware. Whether it is uploading files, sending tech support emails, checking for updates, accessing web sites, refreshing browses when data changes or any of the myriad other ways that programs can connect with one another or the world at large, it is my experience that the isolated standalone program is becoming a rather rare beastie.

I've used several methods in the past to add Internet functionality to my programs, ranging from the built-in Windows WinInet libraries to the SocketTool libraries from Catalyst Corporation. Each had its strengths and, as usual, some weaknesses as well, so I've always been on the lookout for simpler, more Clarion-friendly ways of accomplishing those tasks. Enter CapeSoft's NetTalk, a veritable jack of all Internet trades, and from a long-time Clarion vendor to boot.

Ok, NetTalk, it's your turn under the microscope!

Major features

At the highest level, NetTalk offers three broad areas of functionality for use with any network that uses TCP/IP (the networking protocol that the Internet uses and that most LANs now use.) They are:

NetSimple Objects – These include most standard Internet connectivity functions, including:

- FTP – File Transfer Protocol, for uploading and downloading files from ftp sites.
- HTTP – Hyper Text Transport Protocol, for communicating with web servers.
- SMTP – Simple Mail Transport Protocol covers sending email and attachments.
- POP3 – Post Office Protocol is for receiving email and attachments.
- NNTP – Network news, i.e. posting and retrieving messages from newsgroups.
- SNMP – Simple Network Management Protocol. Commonly used to communicate with devices.

NetTalk Objects – These utilize the special NetTalk protocol to communicate with other applications using NetTalk. Some of the highlights are:

- Automatic error-recovery and optional 168-bit DES packet encryption.
- Can communicate with other NetTalk application on the same machine, a local area network (LAN), a wide area network (WAN) or the Internet.
- Support for TCP and UDP (if you know what those are, you'll know what I mean.)
- Includes control templates for functions like chat, automatic browse refreshes when network data changes, remote shut down of your applications, support for connecting to machines with dynamic IP addresses.

NetDUN Object – This one lets you access and utilize Windows Dial-Up Networking connections, i.e., connecting to a network or the Internet using a modem. The NetDUN object also lets you detect when a new dial-up connection has been established by another program, so that your program can then take care of any pending business like sending emails or checking web sites.

NetTalk is compatible with all versions of Clarion from 5.0 on up, both local and standalone. Both ABC and Clarion templates are supported, as are Web Builder and ClarioNET. The only real restriction is that your programs must be 32-bit (not exactly a deal breaker, I'd say.)

In part 1 of this review I'll be looking at the NetSimple objects, since they are probably the functions that first attract people to NetTalk. Part 2 will cover the NetTalk network and NetDUN (Dial-up) objects and what they do.

Before I begin, one final note about the "beta" designation for the version of NetTalk that I used for this review. CapeSoft uses a release system where periodically a stable milestone version is given an even dot number (i.e. the first number after the dot) and designated as a Gold release. This is *not* to imply that the beta releases (i.e., versions with odd dot numbers) are horribly unstable or anything like that, it just means that they are being actively updated with new features and/or fixes on what seems to be about a monthly basis. At the time of this review, the current Gold version is 2.81 and was released in July 2003, whereas the 2.92 version I tested was released in early September. I very much like the fact that CapeSoft lets you choose your own comfort level – safe and stable or latest and greatest (and still pretty darn stable.)

Installation

I downloaded the latest version of NetTalk for Clarion 5.5 from the CapeSoft web site. The install arrives packaged as a .SAF file, an encrypted file format developed by CapeSoft. In order to extract and run the actual installation exe, you must have a copy of CapeSoft's free Safe Reader utility. If you don't already have the utility from a prior CapeSoft product installation, it can also be downloaded from the CapeSoft web site.

Once you have the utility installed, it is used to validate your NetTalk unlock code and extract the NetTalk setup program from the .SAF file. From that point on, the NetTalk installation

process is very straightforward. All CapeSoft product installs follow the Clarion 3rd Party Association (C3PA) guidelines, so the install takes care of automatically detecting the location of your Clarion 5.5 installation and optionally registering the templates. It can also update your autoexec.bat and C5.5 redirection files if needed.

All example files and documentation end up in a new NetTalk folder under your \C55\3rdParty folder. The NetTalk product documentation can be accessed from either the usual Windows start menu or from the Accessories menu within the Clarion IDE itself. By the way, if you didn't have an accessories menu in Clarion before, you'll have one once the NetTalk setup is complete.

Once the setup program finishes, the NetTalk documentation is automatically displayed in your web browser to help you get oriented. Although I'll cover the documentation and examples more thoroughly in Part 2, I will say here that CapeSoft's documentation is exemplary.

Implementation

To get started with NetTalk, you must first add the NetTalk global template to your application. There is only one prompt on the template, a checkbox, and that is to *not* generate any NetTalk code at all. Huh? Well, NetTalk is ABC compliant, which means that Clarion automatically sucks it into every ABC application you write. Since not all applications will need to include NetTalk, that option lets you easily get rid of it. If, in the future, you want to activate NetTalk for the application, just uncheck the box.

In testing and using the various NetTalk NetSimple objects, I found I usually followed roughly the same sequence whenever I wanted to try out another area of NetSimple functionality. It went something like this:

1. Read the Jump Start section for the object in the NetTalk documentation.
2. Open, compile and run the NetTalk example program for that function to see it in action.
3. Copy the embed code from the Jump Start documentation or the example into my application. For some functions like filling out and sending an email, I just copied the entire demo procedure.
4. Do a few minor tweaks to the demo code and/or template prompts so that they pointed to the Internet addresses, etc., that I wanted to use.
5. Compile, run, sit back, and marvel at how the darn thing just *worked* right away.

Let's take a look at a simple example where I wanted to get a file from a web server. The file in question is not a web page in this case, but an INI-type text file that contains version information. I use this file to detect if there is a newer version of the program available. It will do nicely, though, for illustrating the basic concepts of NetTalk NetSimple object use.

To start with, I added the NetTalk global extension. I then created a Window procedure and added the NetTalk **Use a NetTalk or NetSimple Object…** procedure template to it.

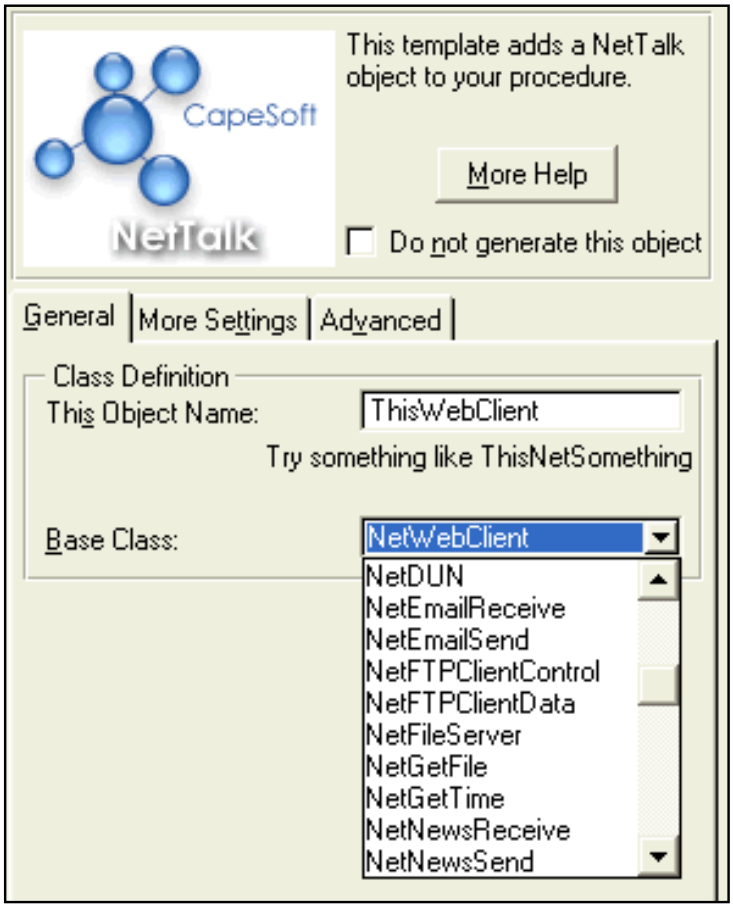

Figure 1. The template for creating a NetTalk object in a procedure

As you can see, there really isn't a whole lot to do here other than specify what you want the NetSimple object to be named and, more importantly, what the base class for the object should be. In this case, I chose the `NetWebClient` because I wanted to talk to a web server using HTTP. Depending on the base class that you choose, there may be additional options to set on the **More Settings** tab. For the web client object, the only option on that tab is a checkbox to suppress error messages that NetTalk might pop up. The **Advanced** tab lets you override and derive the NetTalk class methods and properties.

In the accepted embed for the window's **OK** button, I put the following code:

```
!--- Build default page/file, then see if have override from INI
PageURL = '192.168.100.100/Updates/Current.INI'
PageURL = GETINI('System','WebUpURL',PageURL,GLO:SystemINI)

SETCURSOR(CURSOR:Wait)
ThisWebClient.SetAllHeadersDefault()! Reset flags, etc.
ThisWebClient.CanUseProxy = 1       ! Can use a proxy
ThisWebClient.HeaderOnly  = 0       ! We want the whole page
ThisWebClient.Fetch(clip(PageURL))  !** Does all the work
IF ThisWebClient.Error
   SETCURSOR
   MESSAGE('This WebSite could not be contacted.'    & |
```

```
      '|Reason: ' & ThisWebClient.Error & ' - ' & |
      ThisWebClient.InterpretError(),'Oops!',ICON:HAND)
END
```

Okay, that's just three lines of code to set some options and one line to connect to a web server and initiate retrieving a file, plus some error checking.

The next important thing to understand here is that almost all actions in the NetSimple objects are by default asynchronous, meaning that when you call something like a `Fetch` method, control is returned to your code right away (the above check of `ThisWebClient.Error` simply detects any errors that occurred before the `Fetch` method hands control off the communications code). Meanwhile, NetTalk will continue to putter around in the background trying to do what you asked it to do.

How do you know what's happening, then? There are two methods (among many others) that are key for the web client object: `ThisWebClient.PageReceived()` and `ThisWebClient.ErrorTrap()`. As you might guess, the first method is called when the `ThisWebClient` object has successfully retrieved the page, and the second is only called if `ThisWebClient.Fetch(PageURL)` ultimately fails. Thus, all you do is place your code to process the results in the relevant embed point (these are generated by the procedure template also). For example, here is the type of code I have in the `PageReceived()` embed:

```
ResultText = ThisWebClient.Page [1 : ThisWebClient.PageLen]
Display
!-- Parse the results, first checking for a server error
TextLine = ?ResultText{PROP:Line,1}
IF ~instring('200 OK',TextLine,1,1)
   Message('The server returned a result code of:' & |
           '||' & clip(TextLine),'Bummer!',ICON:HAND)
ELSE
  !-- Received page OK
  LOOP NDX = 1 TO ?ResultText{PROP:LineCount}
    TextLine = ?ResultText{PROP:Line,ndx}
    IF UPPER(TextLine[1 : 8]) = 'VERSION='
       CheckSuccessful = True
       !... other code too boring to look at ...
    END
  END
END
```

First, I move the page that I got back from the server (currently stored in a string property named, amazingly enough, `ThisWebClient.Page`) into a `TEXT` control so that I can use `PROP:LINE` to easily parse out the page line by line. It also lets me display the returned page on the window for debugging purposes if I want to.

Next, I check to see that I got the page I wanted and not an error message from the server, like a 404 NOT FOUND or something like that. Once I get past that basic checking, I just loop

through the page looking for the line I want. Beyond all of my extra code, you should realize that it took just five lines of actual NetTalk code (including the one method call) to connect to the web server, download the page and close the connection. NetSimple, I guess!

In case you are curious, here is what the contents of `ThisWebClient.Page` actually looks like, in this case:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Cache-Control: max-age=604800
Expires: Thu, 17 Sep 2003 16:15:47 GMT
Connection: keep-alive
Date: Thu, 12 Sep 2003 16:15:47 GMT
Content-Type: application/octet-stream
Accept-Ranges: bytes
Last-Modified: Wed, 28 May 2003 19:44:22 GMT
ETag: "02f25895125c31:887"
Content-Length: 116

[CurrentVersion]
;NOTE: Summary entry is optional
Summary=Test to trigger message, no actual update!
Version=2003.06.01
```

By the way, my actual file on the web server only has the last four lines in it. The rest are all the standard HTTP headers that are created by the server when it actually sends the file. If I set the `ThisWebClient.HeaderOnly` property to `True` before calling `ThisWebClient.Fetch(PageURL)`, I'll only get the headers and not the last four lines of actual text.

Note that I could also have used NetTalk's FTP capabilities to accomplish the same basic task, using very similar code and concepts. FTP does take a bit more code, though, since FTP servers require logins and there are extra options to handle for changing folders, retrieving file listings and so on. In the case of the FTP client objects, you would be placing your code to handle the responses to those requests in the method called, for example, `ThisFTPClient.Done()`. Typically, the `Done` method is called whenever the object completes a request.

All of the NetSimple objects have many well-documented properties and methods that can be used to more finely tune and/or interact with the particular task. The nice part is that you often don't *have* to use them.

NetSimple Objects performance

All of the protocols I tried (SMTP, POP3, FTP and HTTP) worked as expected. NetTalk took about the same time uploading a large file as did my commercial ftp programs. I experienced no weird behavior in connecting to sites hosted by my company or to commercial sites. Email was sent properly.

In the real world, using the FTP and SMTP objects, I wrote a small program that runs all day long and just checks certain folders for new files. Whenever a new file appears or a specified time period has elapsed, the program connects to an external ftp server for the upload and/or to check for new files to download. If problems occur when processing downloaded files (invalid data values, for example), it emails the details to the network administrator. The program has been run continuously for a month at a time and worked like a champ.

Similarly, the HTTP example I used has been spread to hundreds of machines and has not had a problem reported yet (he says while knocking on wood.)

Wrapping up Part 1

So far, I have to say I'm very impressed with NetTalk. It's easy to use, reliable and well documented; we'll see if CapeSoft can keep it up as I dig further into the NetTalk protocol and the NetDUN object. See you next time!

A longtime Clarion user, Tom Hebenstreit is an admitted tool junkie who refuses to go straight and code without his arsenal of third party products. During those rare moments when he isn't either using or writing about Clarion, he indulges his twin passions for blues and beer by performing around Southern California in a variety of totally-obscure-but-famous-any-day-now rock and blues bands.

Reader Comments

Add a comment

Tom, Clarion will suck in all objects if any class in...
Thanks for the tip! Only problem is that the ABCInclude...
Tom, The 5.5 Help does describe this, but very poorly. ...
It's a bit of a moot point from NetTalk's point of view....