Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS tm

Clarion ASP Review Part 3: The 1.1 Update

The big news of the past week was the formal release of the 1.1 version of Clarion/ASP. In this third installment in his extensive review, Tom Hebenstreit, Clarion Magazine's reviews editor, installs the 1.1 patch and digs a little deeper into Clarion/ASP's capabilties.

Posted Tuesday, August 06, 2002

Reports: OOP, ABC and Ignoring Templates (Part 2)

While preparing his ETC presentation, "Reports: Paying the Price of Flexibility," Steve Parker had an epiphany. He suddenly realized that he uses templates, well, *strangely*. Part 2 of 2.

Posted Thursday, August 08, 2002

Finding Field Names

Ever find yourself looking through generated file declarations for certain fields? Here's a quick template by Andrew Guidroz that will help.

Posted Wednesday, August 14, 2002

Class Wrapper Templates The Easy Way

In earlier releases of ABC, class wrapper templates were difficult to create. After creating his first class, Lee White, inveterate hand coder and template writer, went searching for a way to create a template wrapper for that class. What he found was a pleasant surprise.

Posted Thursday, August 15, 2002

Adding Page Of Pages To A Clarion Report

For many years now it has been difficult to add a Page of Pages variable to a Clarion report. It has also been difficult to print more than one copy of a report without clicking on the print/report button twice, or using the printer properties to specify multiple copies. Inspired by Bruce Johnson's ABC book, and some conversations at ETC III, Nardus Swanevelder finds a solution to both problems.

Posted Friday, August 16, 2002

Understanding Template Symbols

In this article Steffen Rasmussen introduces his preferred workflow for tackling more advanced template programming. He demonstrates his five

News

- UltraTree TreeWizard
- Beta Testers Wanted
- FirebirdSQL Open Source ODBC Driver
- TPS.repair Templates 1.5 Released
- xQuickFilter v2.12 Released
- RichReport 1.1 Released
- New Imaging SDK And Image-XChange Demo
- Icetips Magic Buttons 1.1
- Icetips Cowboy SQL Templates
- Icetips Magic Entries New Release
- INN Bio And News For 21-Aug-2002
- New MailMerge Templates And Free Re-sort Template
- Update to Simsoft Templates
- Virtual EIP 1.3 Update
- Gitano Look Good Package Sale
- Gitano Summer Sale
- EasyAnimation 1.01 Released
- New Version Of DOS Printer
- New Clarion Book Coming
- IMPEX Screenshot Pages
- LogFlash Screenshot Pages
- EasyMultiTag 2.02 Released
- chSTD Library 2.62 Released
- ImageEx 1.3 Released
- Clarion Memory Leak Checker
- Clarion Application Skins

step approach with a template for Dave Harms' many-to-many checkbox class.

Posted Tuesday, August 27, 2002

A Limerick Contest!

There's been a spate of rhyming in the SoftVelocity chat newsgroup, ranging from doggerel to limericks to haiku. Some range! Favoring the limericking crowd, Richard Rogers has proposed a limerick contest, hosted and judged by Clarion Magazine.

Posted Thursday, August 29, 2002

Data Structures and Algorithms Part VII - Up a Tree

In this installment of this series on data structures and algorithms, Alison Neal talks about the Tree data structure (which is not the same thing as a Clarion tree control).

Posted Thursday, August 29, 2002

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.

Capesoft File Manager 3 Beta

Free Browseless Form Template

New Addition To Simshape Templates

Whitemarsh Metabase Version 0505

ImageBrowsing Bundle Available

ExcelBond Version 1.3 Released

ClarionSearch.com Temporarily Unavailable

UltraTree Version 8 Announced

Product Scope 32 PRO, Version 4.5a Interim Release

SQT SQL Conversion Template Beta

gFileFind Update Available

Crystal RDC Wrapper

SetupBuilder 4.02

0-HaZzle Price Reduction

xTransparent Window v1.2 Released

IconsXP Update Available

0-OverlapZ Resize Extension

ImageEx 1.2 Released

Latest PublicPIM Version

chSTD library 2.61 Released

Search the news archive

Clarion Magazine

Clarion ASP Review Part 3: The 1.1 Update

by Tom Hebenstreit

Published 2002-08-06

The big news of the past week was the formal release of the 1.1 version of Clarion/ASP. Briefly, some of the major changes are:

- Bug fixes. According to the SoftVelocity website "every single bug report has been addressed and corrected."
- More embed points.
- Runtime support for including external HTML files into the generated documents. Include files can be specified globally or on a procedure-by-procedure basis.
- A new Source Code procedure template.
- You can now range limit "select" style browses. You can also do this dynamically based on the contents of other fields of the data entry form.
- More control over database access. You can now override SQL `SELECT` and `DELETE` statements on a procedure-by-procedure basis.

The one downside of the v1.1 upgrade is that it orphans v1.0 embed code, necessitating some extra work to get things back to where they were before.

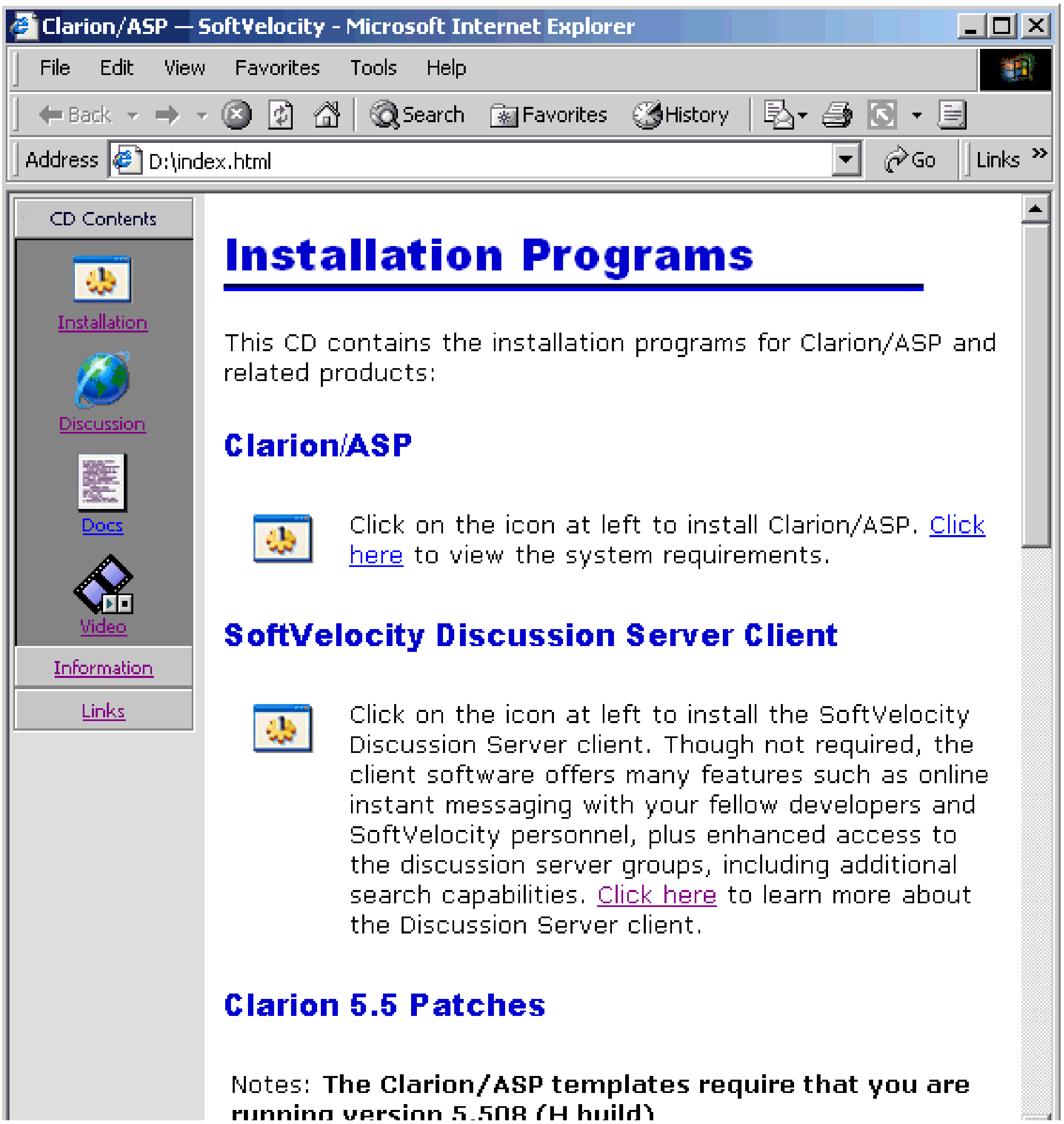
Patching Clarion/ASP 1.0 to 1.1

The update for v1.1 is available as a downloadable patch. It updates the templates and online help file to the new version. The two PDF manuals for Clarion/ASP have also been updated to reflect the v1.1 changes, but you have to download these (and the updated example applications) separately.

The patch/update process worked smoothly for me with one exception. Clarion/ASP uses a file called `blank.htm` as a design-time template for generating all of the Clarion/ASP HTML pages (more on this below). Clarion/ASP v1.1 uses a more complex `blank.htm` than v1.0, in that it has more Clarion/ASP tokens to support the new "includes" functionality. If your application has an existing v1.0 version, though, the new one won't replace it. So, following the instructions in the patch readme file, I deleted the existing `blank.htm`. This, however, resulted in a ton of generation errors as the templates looked for but couldn't find the blank.htm file. I finally ended up just creating a new file by hand. Fortunately, the readme file listed the contents of a v1.1 `blank.htm` file so it was a simple matter of cut and paste. After that, everything worked fine.

Installing Clarion/ASP v1.1

To my surprise, I received a brand new Clarion/ASP v1.1 disk within a couple days of the patch release. Upon placing it into my CD-ROM drive, I was greeted by the window shown in Figure 1.


Clarion/ASP — SoftVelocity - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address D:\index.html Go Links
CD Contents
Installation
Discussion
Docs
Video
Information
Links

Installation Programs

This CD contains the installation programs for Clarion/ASP and related products:

Clarion/ASP

Click on the icon at left to install Clarion/ASP. Click here to view the system requirements.

SoftVelocity Discussion Server Client

Click on the icon at left to install the SoftVelocity Discussion Server client. Though not required, the client software offers many features such as online instant messaging with your fellow developers and SoftVelocity personnel, plus enhanced access to the discussion server groups, including additional search capabilities. Click here to learn more about the Discussion Server client.

Clarion 5.5 Patches

Notes: **The Clarion/ASP templates require that you are running version 5.508 (H build)**

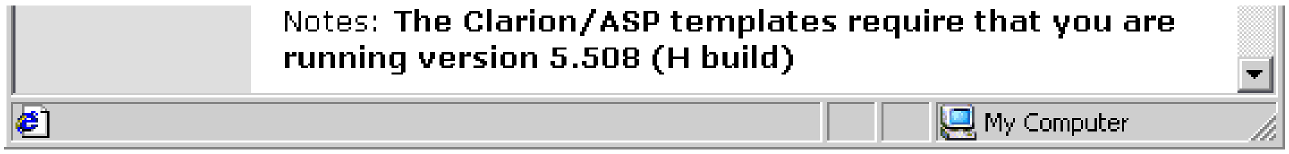

Notes: The Clarion/ASP templates require that you are running version 5.508 (H build)

Figure 1. The v1.1 install disk startup screen

Wow – talk about a world of difference between the 1.0 and 1.1 disks! Where v1.0 was Spartan and uninformative, the new disk is a gem. All options for the disk are easily accessible, there is a lot of information right up front to help you get oriented and they've even included a number of getting started videos. Not only that, the v1.1 disk also included all of the Clarion 5.5 patches to help ensure that you can update to the version required by Clarion/ASP with a minimum of hassles.

This is what an install disk should look like – kudos to SoftVelocity for doing such a stellar job on the new disk.

Going back to school

To help people get started with Clarion/ASP, SoftVelocity has developed a full online instruction course. For early adopters, the course is being bundled for free with Clarion/ASP, along with six months of their Clarion/ASP Online Support package. By the way, as of this writing that special offer has been extended to August 31, 2002. The main components of the class are:

- 9 basic lessons. These can be gone through at any time and at your own speed, although the live class/chat sessions do follow the lessons. All include multiple web pages, audio sections, and most include one or more video segments.
- 14 30-minute live class/chat sessions where you are online with an instructor and other students. The usual format was the instructor going over a topic and then students could ask questions, etc.
- Access to the Clarion/ASP Online Support news groups and chat lounges, using the SoftVelocity Discussions client (described in an earlier portion of this review.)
- Additional 'bonus' videos that provide a basic visual reference guide to most of the Clarion/ASP template prompts.

As with any new venture, there were some rough edges when people first started accessing the lessons. For most people, the primary problems had to do with playing back audio and video files. On my machine, the audio clips wouldn't play correctly, if at all, although video worked fine from the start. Within a couple days, though, SoftVelocity had ironed out the major kinks and both types of content worked well. The only other real problem seemed to be with people behind corporate firewalls that were not letting them connect to the video clips. Even though that really isn't their fault, SoftVelocity did try to help them and has posted FAQ items to help students to solve these types of problems. Finally, it helps to have a reasonably fast Internet connection, but SoftVelocity has made a lot of effort to minimize the bandwidth requirements for the video segments.

As mentioned above, the class includes a series of live 30-minute class/chat sessions. In this first pass, SoftVelocity had them scheduled three times a week, staggering the start times to try and accommodate the fact that people were going to be tuning in from all over the world. As you'd expect, not everyone was satisfied with that. For some people the times were too early, for others they were too late. I missed a couple of classes myself thinking that it was at one time when it was really at another, earlier time (yeah, yeah – I could have printed out the schedule!) or when I had prior commitments. I have to point out that missing a class/chat session does not mean you missed the content of the class – SoftVelocity posts transcripts of every class/chat session for download, usually within 15 minutes of its conclusion. What you miss is the chance to interactively ask questions of the instructor.

I don't envy SoftVelocity for having to figure how to schedule all of this, as they are trying to serve two conflicting constituencies: some developers have the time to immerse themselves into Clarion/ASP and want to learn as quickly as possible; others already have full-time commitments and are trying to squeeze their Clarion/ASP learning in whenever they can (yours truly being among the latter.) High marks go SoftVelocity for effort, though, as they have already tweaked the schedule for the next round of classes (starting August 13 by the way) based on student feedback at the end of the first round. The new classes will be at more consistent times, twice a week.

SoftVelocity has also recently announced that an advanced Clarion/ASP course is being developed, with the first class scheduled for the latter half of August. Check their web site for more information on class content and cost.

I would suggest one improvement for the chat/class sessions, though, and that would be for the instructor to end each class with suggestions of what the students should look at/try out before the next chat session. When new concepts were presented for the first time via the chat, people didn't tend to have as many questions since they were still trying to absorb/understand what had just been said.

All in all, the class was very informative, reasonably priced (especially right now – free!) and covered a lot of ground. I look forward to future classes on not only Clarion/ASP, but on Clarion itself.

Design time and run time files

Two concepts are fundamental to understanding Clarion/ASP. They are:

- Clarion/ASP is, in a large sense, a like a very sophisticated mail-merge system. Files containing Clarion/ASP tokens are read and those tokens are replaced with the other values. This process occurs twice: First, at generation time when the Clarion IDE and Clarion/ASP templates are creating HTML and ASP pages for your site, and secondly, at run time, when the resulting pages are actually processed by IIS on your web site.
- Clarion/ASP separates the visual portion of your interface (the HTML pages that interact with the user) from the functional portion (ASP pages that contain the business and application logic, e.g., code to manipulate your database and so on.)

Generating files

For every procedure with a Clarion/ASP extension template on it, two (or more) web site files are created. One is an HTML file, has the `.htm` extension and is the visual interface for the user. The other has the `.asp` extension and contains the application and database logic for the procedure.

The HTML file (the visual interface) is generated based on a design time HTML file on your local machine. At generation time, as the Clarion/ASP template reads the design time file, it looks for a special token (`@Clarion/ASP@` - all Clarion/ASP tokens have the `@` character at the start and end) and, when it finds it, it substitutes the actual HTML code for the token. For a browse, you get list code wherever the token was. For a form, you get update form code.

The beauty of this is that you can modify the design time HTML templates to contain common elements of your site such as banners, images, navigation links, etc., and all of the generated pages will automatically have these elements included. For further flexibility, you can specify different design time templates for global, list, select and form procedure types at the global level. You can also override the global settings and specify the design time HTML template to use on a per procedure basis.

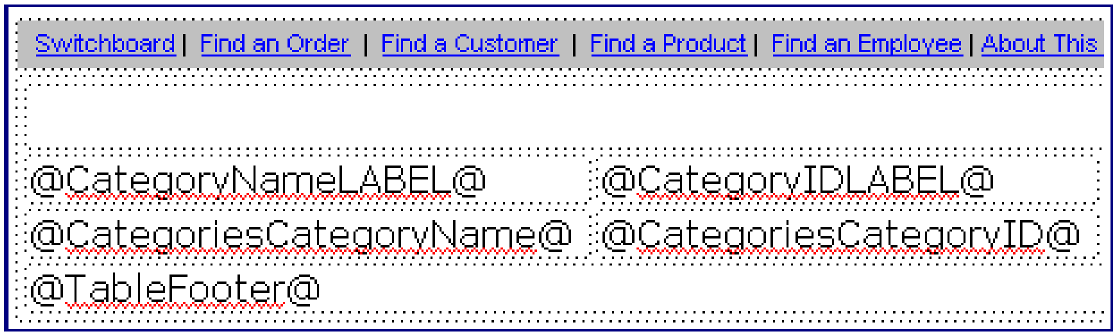
The default design time HTML file, by the way, is named `blank.htm` and is pretty much an empty page with nothing but the `@Clarion/ASP@` token in it. In a basic application, every procedure might be based on that same default design time HTML file.

A look at runtime files

Run time HTML and ASP files are the end result of this Clarion/ASP generation process, and they are the files that actually get placed on your web site. Within the visual interface HTML code generated for each procedure (the `.htm` file), Clarion/ASP places other tokens that represent where your data and variables are to go. For example, the HTML for a portion of a browse might look like this:

```
<!--BEGIN RowData-->
<tr>
    <td class="@CategoriesCategoryNameSTYLE@" align="Left" width="">↵
          @CategoriesCategoryName@ </td>
    <td class="@CategoriesCategoryIDSTYLE@" align="Right" width="">↵
          @CategoriesCategoryID@ </td>
    <td class="@CategoriesDescriptionSTYLE@" align="Left" width=""↵
          >@CategoriesDescription@ </td>
    <td class="@CategoriesAutomaticDetailLinkSTYLE@" align="Center">↵
          @CategoriesAutomaticDetailLink@</td>
</tr>
<!--END RowData-->
```

For each column in this browse (i.e., every `<td>` to `</td>` pair), you see a token for the style to use and a token for the data value. And if you looked at the file directly in a browser, it would appear something like this:


Switchboard | Find an Order | Find a Customer | Find a Product | Find an Employee | About This

@CategoryNameLABEL@ @CategoryIDLABEL@
@CategoriesCategoryName@ @CategoriesCategoryID@
@TableFooter@

Figure 2. The tokens for the various portions of a list box (column labels, the data rows and a footer for the table). Note that this browse used a design time HTML file that had a navigation bar at the top.

This leads to the reason that Clarion/ASP separates the application logic and run time HTML file. Because the application logic is safely isolated off in another page, a web designer can take the generated run time HTML pages and customize them as much as they want. And as long as they don't alter the actual tokens, the Clarion/ASP application logic will process the page just fine. Note that if run time HTML pages are modified by hand, you should turn off HTML generation for those procedures in the Clarion/ASP templates, or else the manual changes would be overwritten the next time you regenerate the application.

Putting Them All Together

So how do these files work together? Remember, I said that Clarion/ASP is very similar to a mail merge process. Taking the browse as an example, the user requests the browse ASP page – the page with the application logic; as IIS processes the instructions on the page, it creates a connection to the database and fetches the data records (i.e. the result set) to display. Once it has that, it calls a function like this:

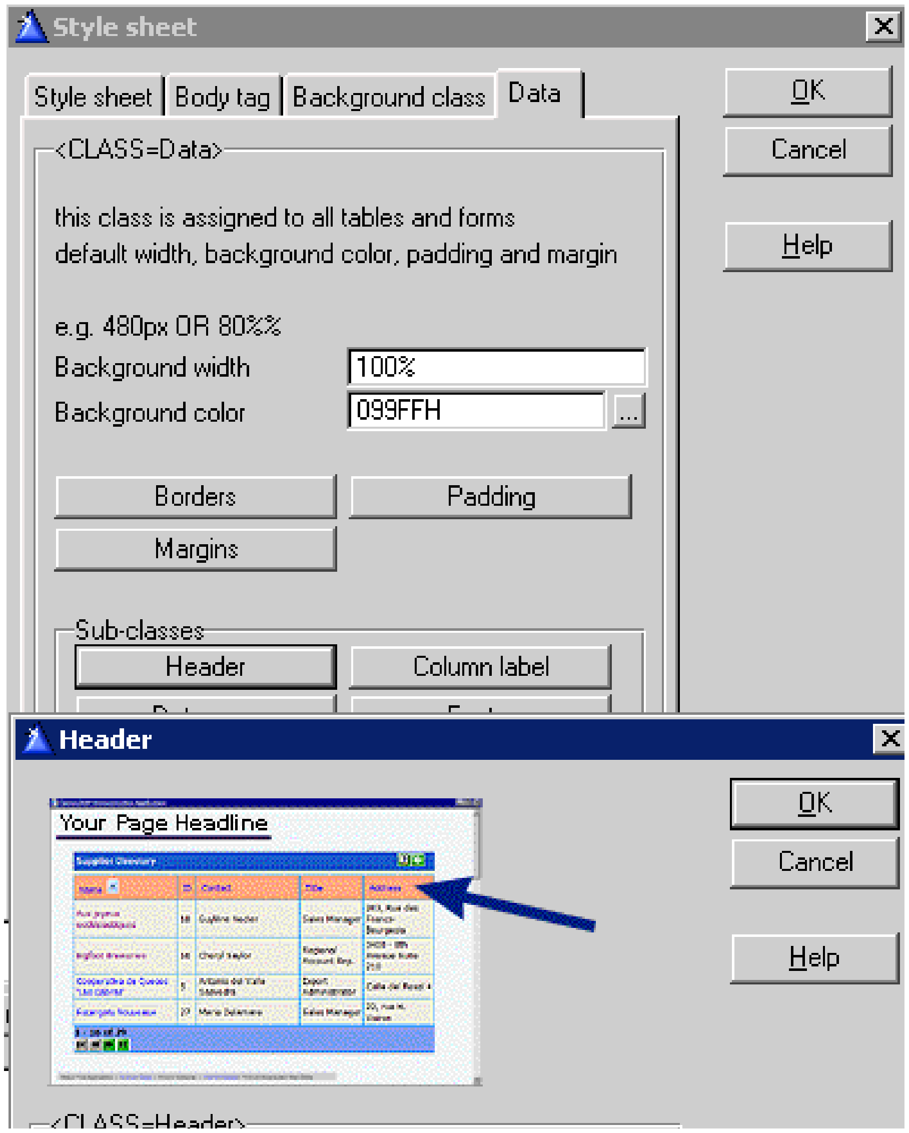
```
Call MergeBrowseCategoriesListTemplate("HTML/BrowseCategoriesList.htm")
```

This function takes the run time HTML page (the visual interface) and merges it together with the result set, replacing all of those tokens with actual data values, style names and so forth. After the merge is complete, it then sends the results of the merge back to the user as the actual web page they will view and interact with. As the user pages through the browse, the same process is repeated again and again, with only the data values changing as the user moves through the result set.

Digging back into the templates

Remember the simple app I created in Part 2? That one took about 15 minutes to assemble and generate once the basic database and IIS settings were in place. It used all default settings for the browses and forms, and the results, while functional, were not very pretty.

Continuing along with the exercises in the Annotated Examples manual, I went through the fourth example step by step. This example shows you how to create a custom style sheet and also has a lot more functionality turned on via the templates. The following figures demonstrate a few of the template screens I was filling in.



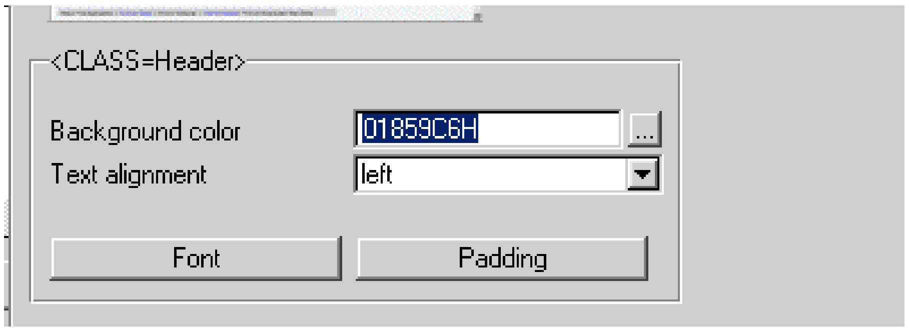
<CLASS=Data>

this class is assigned to all tables and forms
default width, background color, padding and margin

e.g. 480px OR 80%%

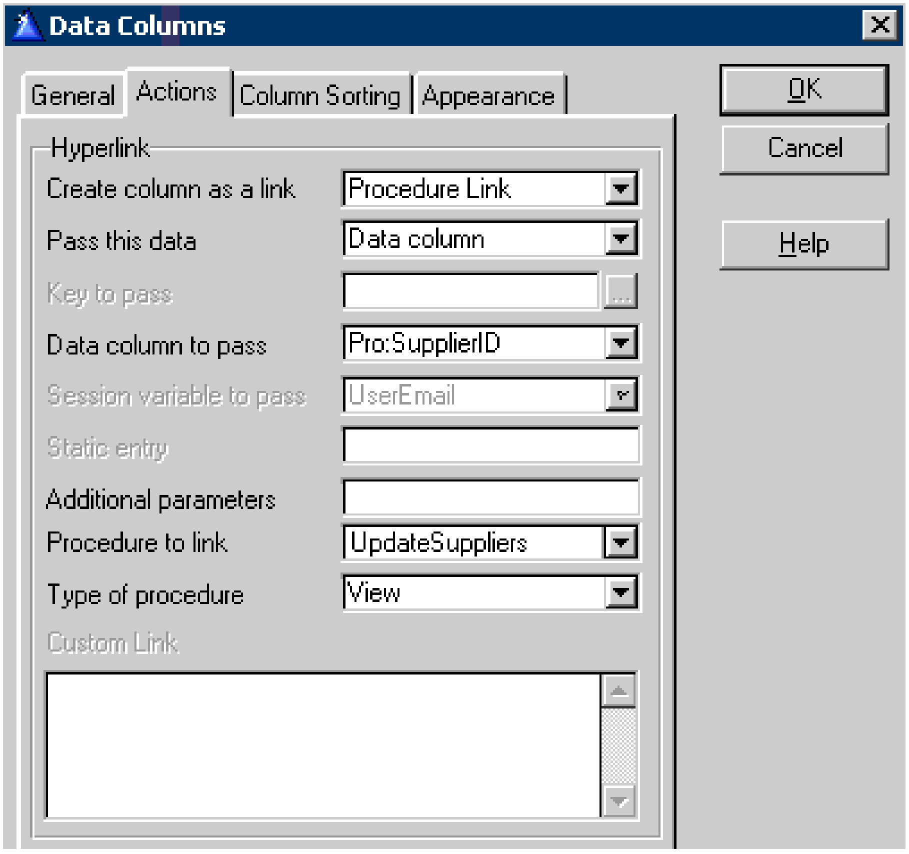
Background width	100%
Background color	099FFH

<CLASS=Header>



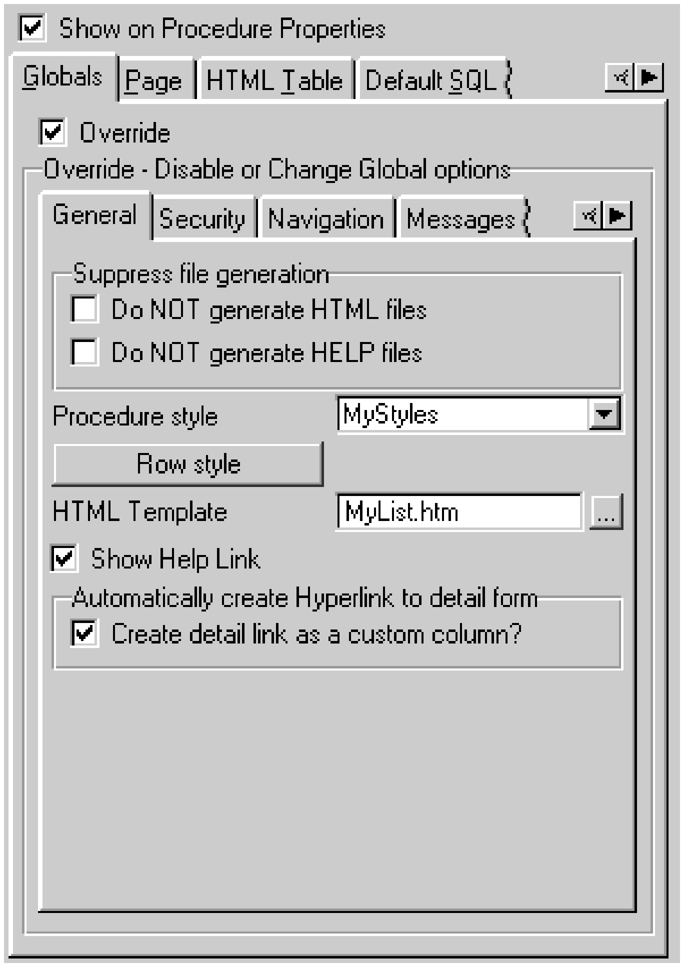
Background color	01859C6H
Text alignment	left

Figure 3. Creating a custom style sheet. Note the nice touch of having images and arrows on some of the template dialogs to let you see exactly what elements you are affecting.



Create column as a link	Procedure Link
Pass this data	Data column
Key to pass	
Data column to pass	Pro:SupplierID
Session variable to pass	UserEmail
Static entry	
Additional parameters	
Procedure to link	UpdateSuppliers
Type of procedure	View
Custom Link	

Figure 4. Creating a link from a browse/list column to pop up an update form. Links can be based on a key, any unique data column or you can enter custom script code to dynamically create links based on values in the current row.


[✓] Show on Procedure Properties

Globals | Page | HTML Table | Default SQL

[✓] Override

Override - Disable or Change Global options

General | Security | Navigation | Messages

Suppress file generation

[ ] Do NOT generate HTML files

[ ] Do NOT generate HELP files

Procedure style: MyStyles

Row style

HTML Template: MyList.htm

[✓] Show Help Link

Automatically create Hyperlink to detail form

[✓] Create detail link as a custom column?

Figure 5. Overriding the global defaults on a single procedure. These options give you complete control of how this procedure will be generated, what design time HTML file will be used, which style sheet, text for prompts and much more.

All told, going through this exercise took about three hours. It was not difficult, merely tedious as I went through each of the procedures to customize their behavior. One thing you quickly discover when using Clarion/ASP is that these templates are *very* deep. By that, I mean that they have many, many levels of detail that you can drill down into to specify exactly how the HTML and ASP pages will look and function. Thankfully, you don't *have* to do this, but these are easily some of the most comprehensive Clarion templates I have ever seen. Which is logical when you think about it, since Clarion/ASP has no Window formatter, etc. for visual input (imagine if every bit of the visual interface in your Clarion language programs had to be specified via template prompt…)

What do the results look like? Well, SoftVelocity has this application already running on their web site in the Clarion/ASP examples section, which is better than me putting in a bunch of static screen shots right here. Actually, there are three variations using different style sheets to change the look and feel of the same application. I'd recommend going and playing around with them a bit (the entire Clarion/ASP section of the SoftVelocity site is very good. Click on the Example link near the top of the page. The particular application I have been talking about is the Northwind Traders sample.

Things I've learned this week

- Any time you add or suppress data columns, set range limits or change sort settings you should always click on the Regenerate SQL button to ensure that your changes are reflected in the SQL statements passed to your database.
- On the other hand, if you have manually edited your SQL statements, pressing the Regenerate SQL button will overwrite your changes. You would need to either re-apply your edits or manually update the SQL to match your data column changes (adding or removing columns from the `SELECT` statement, for example.)
- On the other other hand, the templates won't automatically clear a `WHERE` clause when you regenerate SQL, so if you make a change that would remove the need for that clause you also need to manually clear out the old statement.
- It would be nice to have the Regenerate SQL button on the same tab as the SQL statements for a couple of reasons. One is that you can see if you have made manual changes (before you accidentally blow them off) and the other is that you could see the results of the regenerate right away (which would help remind you of a `WHERE` clause you may need to manually clear).
- If the application you are using is purely for Clarion/ASP, it is handy to remove the usual Clarion browse and form template prompts from the procedure properties and substitute the Clarion/ASP ones. Just uncheck the 'Show on Procedure Properties' box for the standard templates and check the box on the Clarion/ASP extension. Note that the changes won't take effect until you save the procedure and re-enter it.

Well, that does it for this installment. The final installment will cover writing embed code, and integrating Clarion/ASP into existing sites. I'll also summarize the journey. See you then!

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

Thanks for doing this. We are extremely interested in...
JC, I agree that an article comparing Clarion/ASP with...

Clarion Magazine

Reports: OOP, ABC and Ignoring Templates (Part 2)

by Steven Parker

Published 2002-08-08

In the last exciting episode, I made what I hope was an extremely provocative assertion. I asserted that there are only four things genuinely important in a template-based procedure:

- a data section
- a formatter (window and/or report)
- a View (where appropriate)

and

- a processing loop with lots of places for inserting my own code.

Further, I claimed that it was possible and sometimes desirable to ignore the rest of what a template provides.

It is time to defend that statement and I'll do it using reports. This time, however, instead of approaching it as "how do I do this (or that)?", my approach will be "this is what is available in the template and how it can be used."

Reports Again

I've written about reports before, more than once. In fact, it was pointed out at ETC that over 10% of my articles have been on reports (little did I suspect).

And they keep coming back.

Perhaps the subject of reports keeps coming back because the Report Template is the one that seems to do the least for you, where it seems like you most often have to go into the embed tree.

Perhaps it is because the Report Template is the most obvious validation of my approach to using the formatter, the view engine and selected embeds for direct Clarion language statements (and ignoring the rest of the template's features).

It certainly isn't any special expertise on my part.

The truth is that I like writing about reports because reports occupy a very special place in my heart. They were my entrée into the world of PC programming notoriety.

I used to work for a multi-campus community college. I was brought in to automate student services, particularly service tracking and reporting. There was only one line position available in the school, Placement Services, so I had to run the Placement Office also.

We thought we'd be using a mini and I had to write the grant to get that mini. Actually, I wrote a number of grants for them. Every single one got funded – except the one that would have gotten us the mini.

In the meantime, I had acquired a micro computer. It was a second hand DECMate, running CPM and sported a 10 MB hard drive: I was the envy of my peers. Those who had computers at all had to make do with dual floppies.

Because I (nominally) ran the Placement Office, I was suddenly called upon to report my activity (students served, how much service, that sort of thing). It took me three days to collect and collate the data.

Right.

Condor III was the DMBS of choice at the college. So, I went to work. The next time a report was requested, it took five minutes. That's more like it.

This experience made me realized that I didn't need a mini to track student services.

I had acquired Clarion Personal Developer in the meantime. So I created a fairly simple app. The main data entry form captured the student ID, the course for which the student had received tutoring, the type of service, the amount of service, the person providing service, the date of service and the department providing the service.

Student ID was looked up from a master student file. Service date defaulted to `Today()`, so users didn't have to keep re-entering it and Department was captured at user log-on. Service type was also a lookup.

The file layout in Figure 1 shows fairly well normalized data:

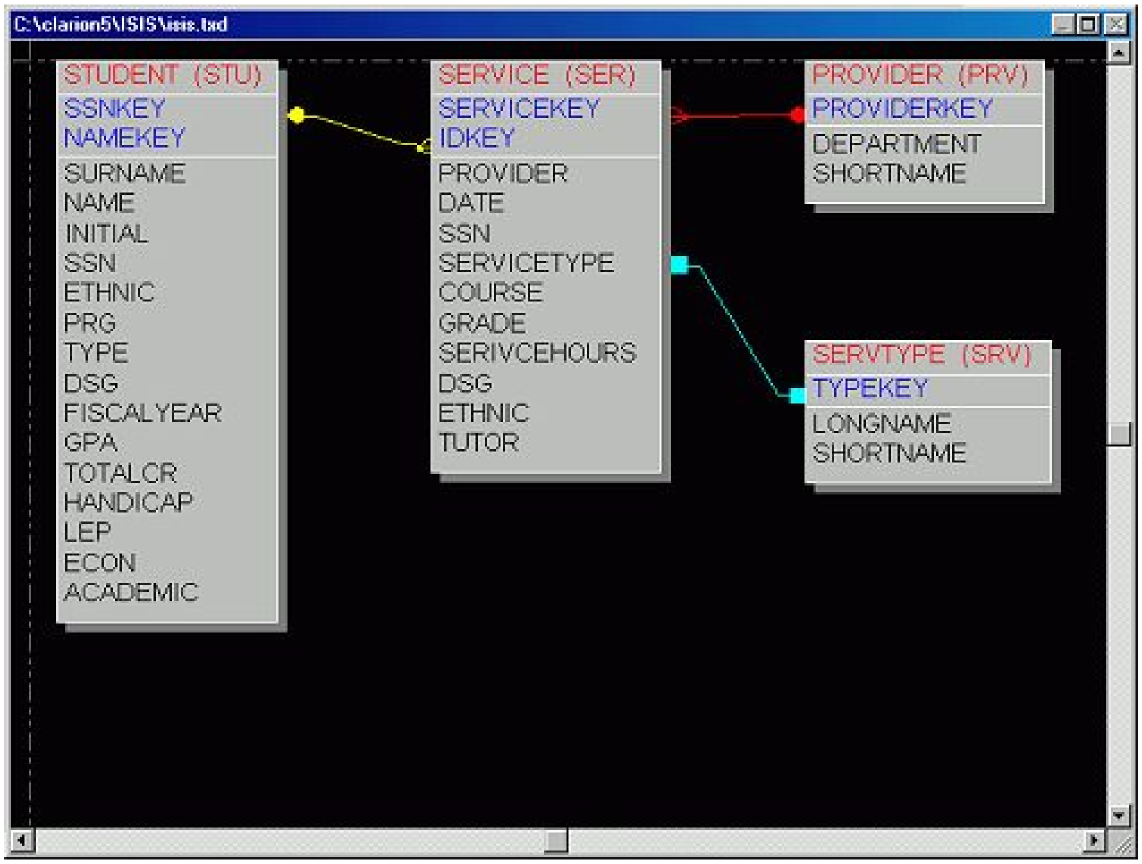


Figure 1. Student services file layout.

(Note: Figure 1 is from a Windows version, the actual application was originally deployed using CPD2.0.)

Some of the reports I had to do were:

- Students served (alpha listing)
- Students served by Special Needs Category
- Students served by ethnic code
- Students served by major

There were a total of about 50 reports that were required at various times during the year (or when the central office decided to test our "rapid response systems").

At the end of the year, representatives from the eight campuses were called together to discuss their reports. Seven were sent home to redo theirs. We were not. Our executive summaries and narrative were, it turns out, quite inaccurate but my reports simply blew the powers that were away.

After that, I had very little trouble getting funding for upgrades for Clarion.

However, what is unique about many of the reports (and all of the reports in the list above) is that they require sorting on fields not in the Service file. All of the sort fields for the reports listed above are in the master student roster, not the Service file.

To create the reports, I used the first release of Report Writer. Report Writer 2.0 had a primitive dictionary and allowed creating links between files, something new in the Clarion world. You could also define sort orders.

Report Writer used this information to create a new, temporary file layout and populated the records from the component files. After the temporary file was created, the report itself ran. I remember a number of reports that ran for 10 or 15 minutes but had taken 8 or more hours for the underlying file to be created. In effect, I could create a new file, with what amounted to *ad hoc* relations and whatever keys I wanted. This allowed me to "sort" Services by student name, major, ethnic code or anything else I could relate to it.

In retrospect, it seems clear that this is the beginning of what ultimately developed into the View engine and it is the View engine, not the Report Template that makes reports in Clarion for Windows so flexible.

What does the View engine do? It allows me to use dictionary relations *or* create *ad hoc* relations (custom joins – see Figure 2).

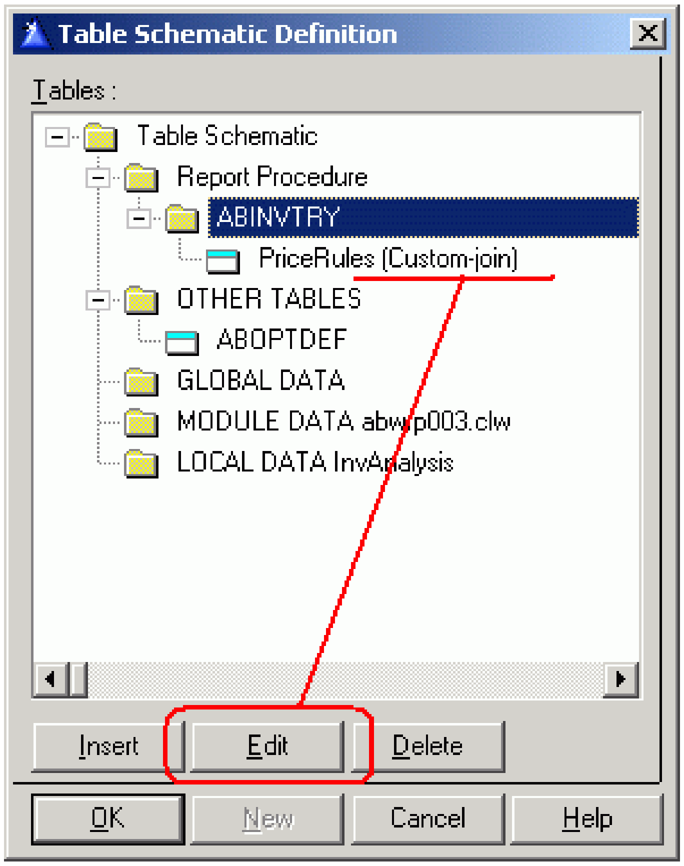


Figure 2.*Ad hoc* relation in a report procedure

The View engine allows me to choose sort orders: I can use keys defined in the Dictionary, I can modify Dictionary key using the Additional Sort Fields prompt or I can create completely *ad hoc*, including runtime defined, orders.

Finally, I can filter to my heart's content. Filter and order are properties of a view much as a key is a property of a file.

I'll give more details on the specifics of the appropriate View methods in a moment, but I want to remind you that I am preparing an infrastructure for a report and I haven't mentioned anything about the Report Template at all. "Use what it gives you, ignore the rest."

You also need to understand basic branching logic and the `PRINT` statement.

You also need to master a few embeds.

That's all.

What is a Report?

I have described a report as a `PRINT` in a loop. I have also described it as a Process with a `PRINT`.

That means that report processing is simple data handling in a loop, simple batch processing.

While I stand behind that characterization, it is just a bit over-simplified. There are, in fact, four logical steps in a report. Each corresponds to a well defined set of Clarion statements:

1. Declaring, opening and manipulating a view
2. Opening the data structure
3. Loop/Print
4. Post-loop processing, if any

Each of these also has a set of well defined ABC methods allowing developer manipulation.

Let's take a look at each step.

Create and Open a View

The template creates and opens the view for me, populating it based on the procedure's file schematic (primary file plus files related to it). As I stated earlier, this is one of the things that the template can be relied on for.

There are, however, a number of useful methods to manipulate the view and none have anything to do with the Report Template (and, therefore, can be used with browses and processes):

AddSortOrder

The `AddSortOrder` method takes a key parameter. So, it can only be used with keys. From the online help:

> The **AddSortOrder** method specifies a sort order for the `ViewManager` object and returns a number identifying the sequence in which the sort order was added.

`AddSortOrder` is most often associated with multiple sort orders on browses. While that may be the normal association, it is not the only valid one. Suppose I create my report without a key. At runtime, I present the user with a data collection window and one of the data I collect is the desired sort order, as in Figure 3.

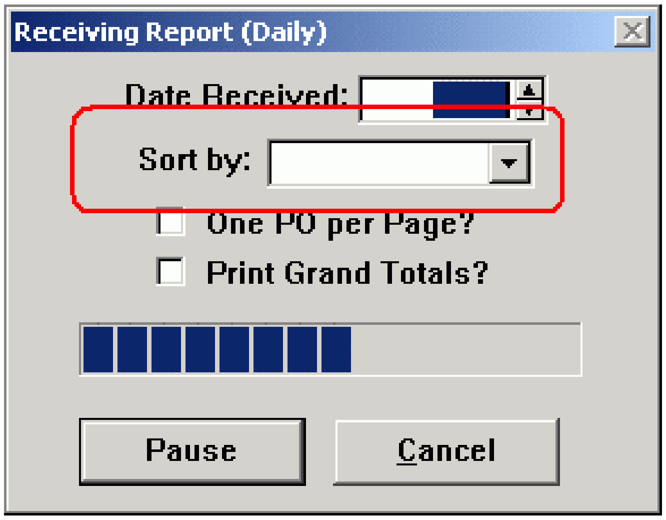


Figure 3. Collecting the sort order

In this case the drop box offers five choices, all corresponding to keys that already exist for this file. The following code ensures that my report sorts the way the user wants it to be sorted:

```
Case SortBy
Of 'UPC Code'
  ThisReport.AddSortOrder('INV:UPC_Key')
Of 'Part Number'
  ThisReport.AddSortOrder('INV:PartNumberKey')
Of 'User Sort'
  ThisReport.AddSortOrder('INV:UserKey')
Of 'Vendor'
  ThisReport.AddSortOrder('INV:VendorKey')
Of 'Dept'
  ThisReport.AddSortOrder('INV:DepartmentKey')
End
```

By examining where the Report Template sets the key, I determined that `INIT`, Priority 8501 works quite nicely. And, without bothering with the template, I have five reports in one.

AppendOrder

The LRM has this to say about `AppendOrder`:

> The **AppendOrder** method refines or extends the active sort order for the `ViewManager` object.

In other words, if a sort order is already set, I can add nodes to it (not necessarily key-based, as I
will show you momentarily).

Above, I showed how `AddSortOrder` can select a key to set a sort order at runtime. Now, suppose I want to sub-sort whichever key the user selected by Purchase Order Number (inventory and purchase orders are both in the view, linked on UPC Code).

A single additional line of code, after the `CASE` structure, does it:

```
ThisReport.AppendOrder('REC:PO_Number')
```

In fact, this is exactly what you get if you use the template's "Additional Sort Fields" prompt:

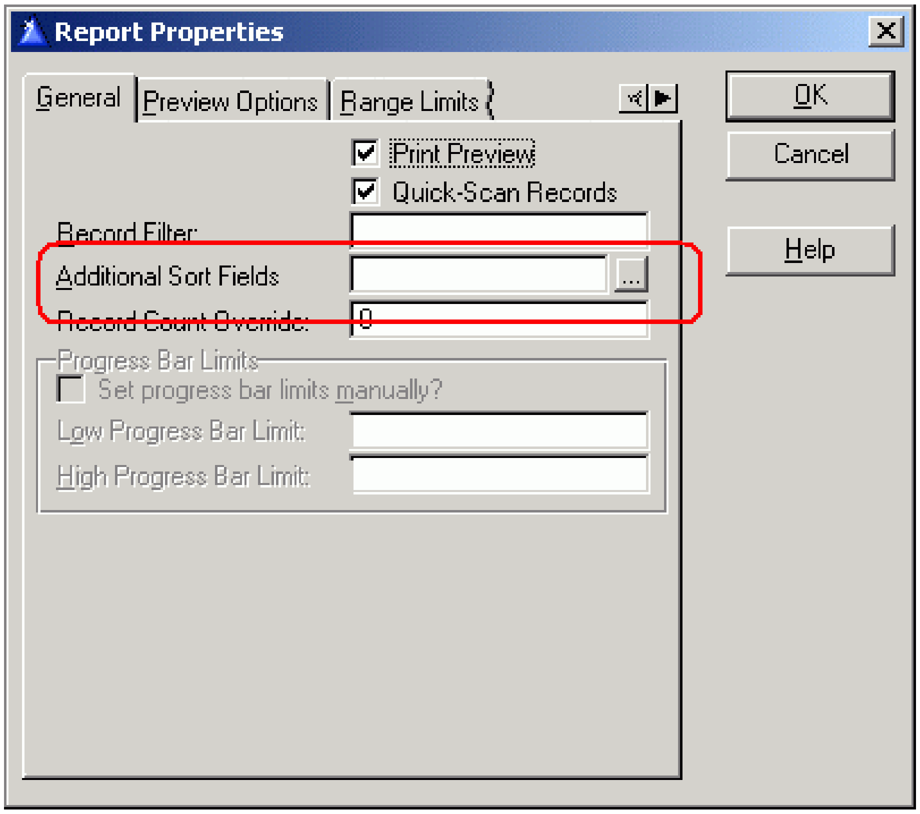


Figure 4. Adding a node using the template

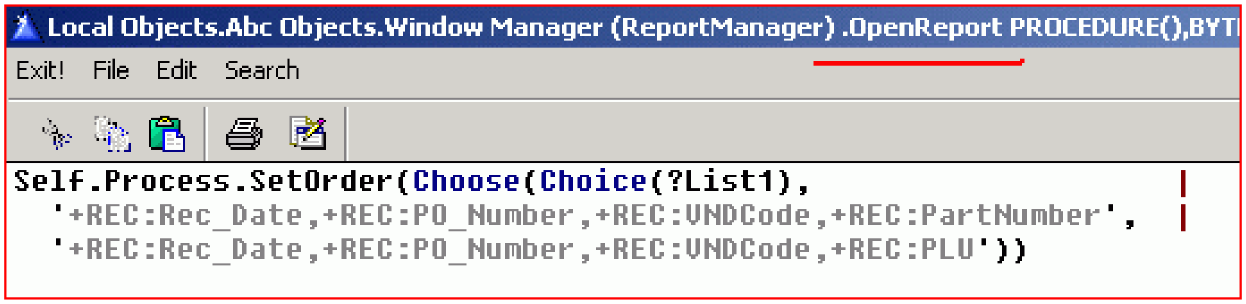
Of course, doing the `AppendOrder` in code does allow a runtime selection of sub-sorting. Also, using the template prompt with manual selection of the key, like that shown above, would `AppendOrder` before the key was set – manually setting the additional nodes *is* recommended.

SetOrder()

According to the LRM: "The **SetOrder** method replaces the active sort order for the `ViewManager` object." `SetOrder` takes a "string constant, variable, EQUATE, or expression that contains an ORDER attribute expression list."

In other words, when no existing key will do, when no modification of an existing key will give the desired result, just roll your own: all it takes is a comma separated list of field labels.

I find `SetOrder` very useful when I would have used a Dynamic Index in CDD - the difference is I don’t have to `Build` the index as I did in CDD.


Local Objects.Abc Objects.Window Manager (ReportManager) .OpenReport PROCEDURE(),BYT
Exit!
File
Edit
Search

```
Self.Process.SetOrder(Choose(Choice(?List1),
  '+REC:Rec_Date,+REC:PO_Number,+REC:VNDCode,+REC:PartNumber',
  '+REC:Rec_Date,+REC:PO_Number,+REC:VNDCode,+REC:PLU'))
```

Figure 5. Report order using SetOrder

SetFilter

The `SetFilter` method allows one procedure to "create" multiple reports and, more generally, dynamic runtime filters.

By passing in a parameter, for example, I can produce a zero-or-less-on-hand report, an exactly-zero-on-hand report or a less-than-zero-on hand report from a single report procedure:

```
Case Upper(pCallType)
Of 'ZERO'
  ThisReport.SetFilter('INV:OnHand = 0')
  Title = 'Zero On Hand'
Of 'NEG'
  ThisReport.SetFilter('INV:OnHand < 0')
  Title = 'Less than Zero On Hand'
Else
  ThisReport.SetFilter('INV:OnHand <= 0')
  Title = 'Zero or Less On Hand'
End
```

Suppose I have more comprehensive data collection form like that in Figure 6.

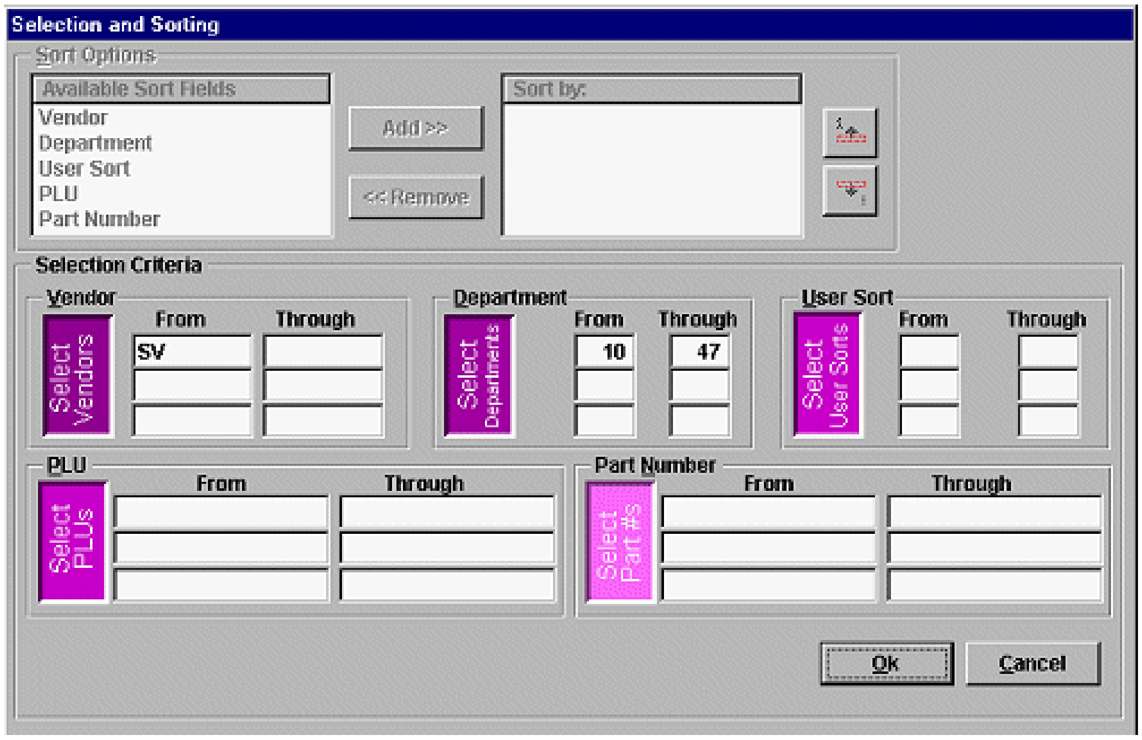


Figure 6: A comprehensive data collection form

The end user may complete no filter selection, or up to 15 selections (note, this form creates a series of strings for each block of possible filter fields). At runtime, I have no idea what the user has done but `SetFilter` will ignore blank strings and concatenate strings that do have values in them. You know what? I feel I've already done enough work getting and formatting the user's input into useable strings. So I'll let the ABC classes do a little of the work. This code does it *all* for me:

```
ThisReport.SetFilter(PLUFilter,'p5')
ThisReport.SetFilter(VNDFilter,'p4')
ThisReport.SetFilter(DeptFilter,'p3')
ThisReport.SetFilter(USRFilter,'p2')
ThisReport.SetFilter(PartFilter,'p1')
```

So far, all ABC and no template.

Open the Report

This is normally handled by the template. It is one of those cases where I believe the Report Template functionality is perfectly adequate.

It *is* perfectly adequate because the template uses the `OpenReport` method and, in fact, the template's opening of the report is just a wrapper for `OpenReport`. But if you know what the `OpenReport` method is doing, you gain a few options.

Examine the method code (it's in abreport.clw – in the template, this would be called by

`OpenReport, Parent call`):

```
ReportManager.OpenReport  PROCEDURE
RVal BYTE,AUTO
  CODE
    SELF.Process.Reset() ! Needed for 're-read' case
    RVal = SELF.Next()
    SELF.DeferOpenReport = 0
    IF RVal
      SELF.TakeNoRecords
    ELSE
      SELF.OpenFailed = 0
      IF ~SELF.Report&=NULL
        OPEN(SELF.Report)
        IF ~SELF.Preview &= NULL
          SELF.Report{PROP:Preview} = SELF.PreviewQueue.Filename
        END
      END
    END
    RETURN RVal
```

At the point where the `OpenReport` method is called, all files and the view are open. Filters and sort orders have been applied to the view. So the critical move is `RVal = SELF.Next()`. Like most ABC access methods, `SELF.Next` returns zero if there was no problem (i.e., the `Next()`succeeded), otherwise it returns a numeric value.

Remember that this is the first attempt to access the view, so if `Rval` contains something (i.e. a value other than level:benign), there is no first record. That is, there are no records. In this case, `TakeNoRecord` is immediately called. Otherwise the report data structure is opened and processing continues (though nothing has printed yet).

So, if there are no records and you want to do something about *that*, the `TakeNoRecord` embed is the place to be. More important is the fact that if `OpenReport` returns zero, the report will print. But no page or group header (if any) or detail has printed. Yet.

If I want to print a title page or a band that prints only once, after the parent call is the place to do it:

```
IF ~ReturnValue
  PRINT(RPT:TitlePage)
END
```

If I place the `PageAfter` attribute on the `TitlePage` band, it will be a title page. Otherwise it will simply print (but after the page header – it seems "Page Header" means something special to the Report engine). This is also the point at which I can make final adjustments to report properties, such as whether to form feed after group breaks or not:

```
SetTarget(Report)
If OnePOPerPage
  ?break5{Prop:PageAfter} = True
Else
  ?break5{Prop:PageAfter} = False
End
SetTarget
```

Still pure ABC.

Loop/Print

If the report opened, i.e., there are records qualifying for inclusion in the report, I am now working inside the main processing loop.

The foundation behavior is:

```
Loop
  Next(file)
   If ErrorCode()
     Break
  End
  Print(RPT:Detail)
End
```

If there is another record, `Next()` succeeds and the detail is printed. Otherwise, processing continues after the loop structure.

As in a Process, this is exactly what the `TakeRecord` method does. `TakeRecord` is, logically, nothing more than the `Next ... If ErrorCode()` check; i.e., it is where the record is read.

The template is doing nothing special, nothing I wouldn’t be doing myself if I were hand coding the loop. In other words, there’s nothing mysterious about `TakeRecord`, I fully understand what it is going on here.

If I am in `TakeRecord`, it follows that a record has been read and is in access. I can also verify whether any field’s value has changed and print a band only when such a change occurs. I can manually detect and act on group breaks.

Priority 5001 is immediately after the record is read and before the default `PRINT` statement. So, if a user has indicated that they want to sort on, say, vendor and if I’ve stored the previous value of "vendor," then:

```
If FirstLoop
 SAV:Vendor = INV:Vendor
 FirstLoop = False
End
If SAV:Vendor <> INV:Vendor
  Print(RPT:VendorTotals)
  SAV:Vendor = INV:Vendor
  TTL:Vendor = ExtPrice
Else
  TTL:Vendor += ExtPrint
End
```

The `FirstLoop` condition is necessary because when the first record is read, `SAV:Vendor` is blank. The vendor "footer" would print because `SAV:Vendor <> INV:Vendor` would be true.

If any one of several variables could be used for this dynamic break (say I had given the user several choices, storing them for internal use):

```
Loop I# = 1 to 3
  Case ParmString[i#]  !Check runtime for "key" changes
  Of 'U'
    Do CheckUserSort   !Did User Sort change?
  Of 'V'
    Do CheckVendor     !Did Vendor change?
  Of 'D'
    Do CheckDepartment !Did Department change?
  End
End
```

If each of the routines called (local methods would work as well) looks like the previous snippet, viola, instant dynamic, user selected runtime breaks and subtotals.

If you are not subtotaling, remove the `FirstLoop` check (and the accumulators) and you have "headers" to separate the sections.

There’s nothing especially difficult in this. It is something virtually everyone has done at one time or another in a hand coded loop. So, why not do exactly the same thing in the loop provided by the Report Template? In a Process, the same will do quite nicely. It’s a matter of not taking the "report" in "Report Template" seriously and remembering that it’s just a loop.

To reiterate: there’s nothing magical about the `PRINT` statement after `TakeRecord, Parent call`. There is utility in thinking of it simply as a loop.

Also, there is nothing *requiring* that the default detail actually be printed. Set its filter to `False`, create a second band (`False` also) with different fields in it and let the user decide at runtime what they want:

```
If DetailRequested
```

```
  Print(RPT:detail3)
Else
  Print(RPT:detail4)
End
```

or:

```
If IsManager !management
  Print(RPT:detail3)
Else               !working stiff
  Print(RPT:detail4)
End
```

As I said, a template provides a data section, a formatter, a view and a loop. The formatter determines appearances; the view determines what is/is not included; the loop determines when something is included. The template has nothing to do with it.

Post-Loop Processing

All good things, they say, must come to an end. And so it is with loops (usually).

After all records have be printed, all bands banded, all breaks broken, it is time for end-of-report data. These include grand totals and other report footers.

But there is no "After Processing Loop embed." Or, at least, nothing called that.

In fact, `AskPreview, Before Parent call` is the "After Processing Loop" embed. And, even if you have set `SkipPreview`, either in the template prompt by turning off preview or in code, `AskPreview` *is* called. (In a Process, `ThisWindow.Kill, Call Close file method` serves the same function.)

Redux

I started by expressing my conviction that the only important parts of a template are:

- the data section
- the formatter (window or report)
- the View (where appropriate)
- the processing loop with lots of places for inserting my own code.

The Report Template is the template with the worst reputation, the template that provides the fewest options, bells, whistles and geegaws. But, approaching it as a data section with formatter, a view and a few embeds in a loop, frees me to do almost anything I want.

Ignoring the template, I free myself to look at the ABC methods affecting views and loops.

With this freedom, I can sort, break, subtotal and filter on any field or combination of fields in the files composing the view. I can let the end user decide whether or not to subtotal and I can use a single report for both the detailed and summary version. And, not once do I use anything other than ABC methods.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

Part 2 does demonstrate validity of the "extremely...

Clarion Magazine

Finding Field Names

by Andrew Guidroz II

Published 2002-08-14

I tend to name my fields very specifically. I always use "UniqueNumber" for autoincrementing unique fields, and I use FileName + "UniqueNumber" for linking fields. And from time to time, I would like to easily see every place I am linked to a certain file. Or, sometimes, I'm interested in all fields with "Date" in the name.

To help me find the fields I'm interested in I created this quick little template utility. As with all template utilities, you can run it from the IDE under the Application Menu, Template Utility, or by pressing Ctrl-U.

When you run the template it will prompt you for a search string. It will then list all of the fields in your current dictionary that contain that search field, and put the result in a file called FilesSearch.TXT in the current directory. You can then peruse this file, copy and paste information for coding, etc. Here's the template code (formatted to fit on this page):

```
#UTILITY (ListSearchForFields,'List Searched for ↵
   Dictionary Fields in FilesSearch.TXT'),WIZARD
#SHEET
  #TAB('Print Fields from Dictionary Wizard')
    #DISPLAY('This wizard will generate a report ↵
      of the fields of your current dictionary.')
    #DISPLAY('To specify which fields you want on ↵
      the report, click on the Next button.')
  #ENDTAB
  #TAB('Print Fields from Dictionary Wizard - ↵
      Select Fields'),FINISH(1)
    #PROMPT('Please enter the string to search ↵
      for in the field name:',@S100),%FieldSearchString
  #ENDTAB
#ENDSHEET
```

```
#MESSAGE ('FilesSearch.Txt', 3)
#DECLARE (%BuildFilesTxt)
#SET (%BuildFilesTxt, 'FilesSearch.TXT')
#CREATE (%BuildFilesTxt)
#FOR(%File)
  #FOR(%Field)
    #IF(INSTRING(UPPER(%FieldSearchString),UPPER(%Field),1,1))
 %[45]Field
    #ENDIF
  #ENDFOR
#ENDFOR
#CLOSE(%BuildFilesTxt)
```

To use this utility download the TPL file below and register it (in the Clarion IDE, choose Setup|Template Registry, and click on Register). You can also paste just the UTILITY template portion into your own template chain if you prefer.

Download the source

Andrew Guidroz II, when he isn't traveling around the countryside watching his 2001 SEC Champion LSU Fighting Tigers, writes software for all facets of the insurance industry. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the U.S. Andrew's Cajun website is www.coonass.com.

Reader Comments

Add a comment

Intesting use of templates. I use Carl Barnes' Clarion...

Clarion Magazine

Home COL Archives

Topics > Templates > Templates, writing

Class Wrapper Templates The Easy Way

by Lee White

Published 2002-08-15

As a tried and true procedural hand coder and template writer I could never wrap my mind, much less a template, around a class. Occasionally the proverbial light at the end of the tunnel would flicker, but it never seemed to stay on for long, until now! So, with a modest amount of light, I jumped in and wrote my first class, yes my first class, only to be confronted with the daunting task of writing a wrapper for that class.

Having read articles from the COL archives on class wrappers, as well as information provided by other developers, I found myself asking why class wrappers seemed so complicated. It is, at least from my understanding, the purpose of templates and classes to make things simpler, not to complicate them. So I began to dig, and I found that many facets of wrapper design have been simplified , in the fullness of time, by the ABC templates.

The current (Clarion 5.5) incarnation of the ABC templates makes the process of creating a wrapper quite easy and straightforward, without the overload of source previously required. To this end I give you the results of many hours of digging and dogged testing with many trials and a multitude of errors. But hey, it's what we do for a living, right?!

Slimming down the templates

The bulk of template code previously required was for the generation of embeds and derived class procedures. Well, this is no longer the case. All that code has been replaced with a single call into the existing ABC templates.

ABGROUP.TPW contains the `%GenerateVirtuals` group. This group is the key to eliminating the majority of the source you would otherwise need to write to create your class wrappers. This group takes four parameters and handles the vast majority of the drudgery for you.

In order, the parameters are: the class name, embed tree description, the name of a locally defined group (that I'll cover later), and an omittable parameter used specifically for global object generation. This omittable parameter, which defaults to `%FALSE`, changes the embed prototype for new methods derived from your base class by eliminating the use of `%ActiveTemplateInstance`. For those wondering, `%ActiveTemplateInstance` is a multi-valued template symbol that has a unique value for each instance of your template. This provides a mechanism for referencing each individual template instance within the generated source and with other related templates.

That's it! I think you'll agree that this is a far cry from the previous glut of required code.

The source

The first line of code in my class wrapper template includes an ABC template `#GROUP`:

```
#INSERT(%OOPPrompts(ABC))
```

This adds hidden prompts required by calls within the template. I could get really involved here and explain in detail what is included, but I won't. If you want to discover this on your own you can find the group in ABGROUP.TPW. Suffice it to say, it's required — don't leave it out!

The `PREPARE` section includes calls that must be performed to correctly display the prompts in your template when accessed within your application (please note that the following source has line break characters not present in the original - for the exact source see the download at the end of this article).

```
#PREPARE
  #CALL(%ReadABCFiles(ABC))
  #CALL(%SetClassItem(ABC),↵
    'MyClass'&%ActiveTemplateInstance)
  #CALL(%SetOOPDefaults(ABC),'MyClass'↵
    '&%ActiveTemplateInstance,'cMyClass')
#ENDPREPARE
```

The `ATSTART` section, albeit a copy of the `PREPARE` section, performs the same calls but, unlike the `#PREPARE` section that is only used while accessing the template prompts, this section is processed prior to actual source code generation.

```
#ATSTART
  #CALL(%ReadABCFiles(ABC))
  #CALL(%SetClassItem(ABC),↵
    ''MyClass'&%ActiveTemplateInstance)
  #CALL(%SetOOPDefaults(ABC),'MyClass'↵
    '&%ActiveTemplateInstance,'cMyClass')
#ENDAT
```

`#CALL(%ReadABCFiles(ABC))`, simply put, purges a bunch of template symbols which are then reloaded through a `#SERVICE` call into C55TPLS.DLL. What happens in there is anyone's guess! `#CALL(%SetClassItem(ABC)` verifies the instance of the class referenced by the template and `#CALL(%SetOOPDefaults(ABC)` sets the default values for the base class and the object name. That's probably more than you really needed to know, but it's too late now.

The `#SHEET` section displays your class's template prompts, including class name, source (ABC or other), method and data prompts and more, all automatically generated via the ABC `#GROUP`:

```
#SHEET
  #TAB('Setup')
    #DISPLAY('Your template prompts')
  #ENDTAB
  #TAB('Class')
    #WITH(%ClassItem,'MyClass'&%ActiveTemplateInstance)
      #INSERT(%ClassPrompts(ABC))
    #ENDWITH
  #ENDTAB
#ENDSHEET
```

Granted, you most likely will have a few prompts for setting up options within your class, but for the sake of simplicity I did not include any above. The second `#TAB` includes the requisite template source to include the standard Class prompts. No rocket science needed here!

So far these snippets of template code are fairly straightforward and not very different, if at all, from previous discussions of class wrappers. Just wait, the real difference is almost here. Look at the following code:

```
#AT(%GatherObjects)
  #CALL(%ReadABCFiles(ABC))
  #CALL(%SetClassItem(ABC),↵
    ''MyClass'&%ActiveTemplateInstance)
  #ADD(%ObjectList,%ThisObjectName)
  #SET(%ObjectListType,'cMyClass')
#ENDAT
```

Well, this code section should be easy to figure out. It "gathers objects" used for source code generation. The next section makes the call that writes the definition for your class into the procedure:

```
#AT(%LocalDataClasses)
#CALL(%SetClassItem(ABC),↵
    ''MyClass'&%ActiveTemplateInstance)
#INSERT(%GenerateClassDefinition(ABC),%ClassLines)
#ENDAT
```

If you include any embedded source in one or more of your virtual methods, you will also discover that this call writes those specific methods and prototypes following the class definition. If no methods include embedded source they will not be included in the class definition. Once again, as with all calls into the ABC templates, you don’t have to worry about the internals; someone else already has.

Ok, so where’s the good stuff? Right here:

```
#AT(%ProcedureRoutines)
#CALL(%GenerateVirtuals(ABC),↵
    ''MyClass'&%ActiveTemplateInstance,↵
    ''Local Objects|Abc Objects|MyClass Description',↵
    ''%EmbedVirtuals(MyTemplateSet)')
#ENDAT
```

It might not look like much, but that’s the beauty of this wrapper, it doesn’t take a huge chunk of template code to get where you want to go. Ignoring the obligatory `#AT` and `#ENDAT`, a single line of template code handles the generation of *all* derived methods. That’s it, one line to replace a whole mess of code necessary in earlier versions of ABC! Again, you can get a better idea what lives behind this single line of code by taking a tour of ABGROUP.TPW.

That leaves just three more sections for the entire wrapper. The first is common to all the wrappers I’ve seen. The last two, though, are not.

```
#AT(%MyClassMethodCodeSection,%ActiveTemplateInstance)↵
    ',PRIORITY(5000),DESCRIPTION('Parent Call')↵
    ',WHERE(%ParentCallValid())
  #CALL(%GenerateParentCall(ABC))
#ENDAT
```

That’s the common section. This is responsible for generating the `PARENT.Method` calls within the generated source. This is not something you want to forget unless you *never* write any embed source!

The last two sections are the only local groups you need in your wrapper:

```
#GROUP(%EmbedVirtuals,%TreeText,↵
    '%DataText,%CodeText)
#EMBED(%MyClassMethodDataSection, ↵
    ''MyClass Description Method Data Section'),↵
    '%ActiveTemplateInstance,%pClassMethod,↵
    '%pClassMethodPrototype,LABEL,DATA,↵
    'TREE(%TreeText&%DataText)
```

```
  #?CODE
  #EMBED(%MyClassMethodCodeSection, ↵
    ''MyClass Description Method Code Section'),↵
    '%ActiveTemplateInstance,%pClassMethod,↵
    '%pClassMethodPrototype,TREE(%TreeText&%CodeText)
```

Those four lines define the group called by `%GenerateVirtuals` with `%EmbedVirtuals(MyTemplateSet)`. This is an important piece of the puzzle since this group generates embeds for your methods!

And finally, this is the group called by the section that generates the `PARENT.Method` calls:

```
#GROUP(%ParentCallValid),AUTO
#DECLARE(%RVal)
#CALL(%ParentCallValid(ABC)),%RVal
#RETURN(%RVal)
```

By using a local group that can be called directly from a `WHERE()` clause you can dispense with several other groups otherwise required in your wrapper. Since `WHERE()`,in this instance, cannot include the required `(set)` parameter to call a group that resides in a separate template set, such as ABCHAIN.TPL, your only alternative is to duplicate this external group, and all subsequent groups it calls, or utilize a local group with `#CALL`, since `#CALL` *can* use this parameter.

That’s it. That’s the entire class wrapper.

You can download an annotated copy of the wrapper below. I recommend you use this as a starting point for your own class wrappers.

A few tidbits

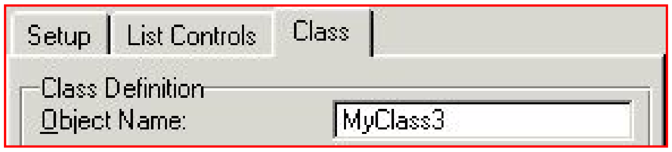
This class wrapper will *not* create embeds for non-virtual methods unless you utilize yet another oddity of the Clarion template language.

If you look around in the base class includes you will notice the occasional occurrence of `!,EXTENDS` following a method prototype. These are *not* comments; they are flags to the IDE to include non-virtual methods using the current Clarion templates and, obviously, this wrapper.

So, if you have non-virtual methods to which you need access within the generated source, add `!,EXTENDS` somewhere following the method prototype. This text does not need to be aligned in any special fashion. It simply needs to be on the same line as the prototype. For example:

```
MethodName PROCEDURE()  !,EXTENDS   and any other comments if desired
```

Here’s another good point to remember as you are writing you class wrapper templates. Whenever you define a template section that generates source code *outside* of a derived class, you need to `#FIX %ClassItem` and use `%ThisObjectName` for calls to class methods and references to class properties. For example, assume you have a class as shown in Figure 1:



Object Name: MyClass3

Figure 1. Naming a class object

You need to `#FIX` the %ClassItem variable:

```
#FIX(%ClassItem,'MyClass'&%ActiveTemplateInstance)
```

In your template, you might reference the object this way:

%ThisObjectName.Kill

The generated source will look like this:

```
MyClass3.Kill
```

Using this combination will result in the correct object name, as entered on the Class tab, being used in your generated source. Failure to use this technique will eventually get you in trouble if you ever change the object name from the default. So don’t get lazy — it’s just one more line of template source that will save you hours of grief.

Now go forth and wrap those classes!

Download the source

Lee White's introduction to computer languages began during his sophomore year in high school, through an open class at the University of Alabama, c1969. His first language was Fortran 4 with Watfor. Introduced to Clarion 2.1 in 1990, he has remained a steadfast Clarion user and supporter. He is often better known for his contributions to the etc conferences and third party products.

Reader Comments

Add a comment

Thank you for this long needed article. While too late for...
Great article Lee... I'm for anything that simplifies.
Am I right that this only works with ABC complient...
Bruce, if you're referring to !ABCIncludeFile then, yes,...

Clarion Magazine

For marketing your Applications and Developer Accessories or to purchase other 3rd Party Tools . . .
Developer PLUS
tm

Adding Page Of Pages To A Clarion Report

by Nardus Swanevelder

Published 2002-08-16

For many years now it has been difficult to add a "page of pages" variable to a Clarion report. It has also been difficult to print more than one copy of a report without clicking on the print/report button twice, or using the printer properties to specify multiple copies. There are templates available which do all this and more (RPM and CPCS spring to mind), but being from South Africa I have always found most third party software to be on the expensive side.

After attending the ETC III conference and speaking to Bruce Johnson (Capesoft), Lee White (RPM) and Michael Brooks (Clarionet), I started to get the idea that it was possible to implement these features without that much work. Then, when I was reading Bruce's Programming in Clarion's ABC book, the light bulb came on.

Before I jump into the code, here's some background.

Clarion's Preview function

I decided early on that the only way it was possible to implement the two new features was to use Clarion's preview function. Preview creates a Windows Meta File (WMF) per page that is printed. WMF files are really binary files containing source code that tells the printer exactly what to print. A Clarion application stores the file path to each of these files in a queue, and when the user chooses to print, the application runs through the queue and send the files to the printer. After the print job has finished, the application removes the files and clears the queue.

This is really a neat feature because it is possible to manipulate the queue and files to do wonderful things with your reports. For example, if you delete a record out of the queue, the corresponding page will not be printed anymore (remember to remove the WMF file manually as Clarion cannot remove it automatically anymore).

You can also add a record for a previously stored WMF file to the queue, and in this way add a cover page to all your reports. I haven't done this myself, but if you try it, watch out for your application possibly removing the file.

Adding to the ABC classes

ABC applications use the `PrintPreviewClass` to enable you to use the preview function, and this is a good place to start to implement the new features. You are not going to change the AbReport.inc and AbReport.Clw file, but rather extend the `PrintPreviewClass` to ensure that the code is not overwritten with an updated version of Clarion.

You will need some knowledge of the way Clarion implement the classes; I rely heavily on information from Bruce's ABC programming book.

A class is made up of two parts. An INC file and a CLW file. Both these files belong in the `\c55\libsrc` folder. The INC files contain the class definitions, so to create a derived class you first create an INC file. You can call your new file PreviewNew.Inc. Here's the source:

```
!ABCIncludefile
  Omit('_EndOfInclude_',_BetterPreview_)
_BetterPreview_     Equate(1)
  Include('Abreport.inc')
BetterPreviewer CLASS(PrintPreviewClass),TYPE|
   ,MODULE('PREVIEWNEW.CLW')|
,LINK('PREVIEWNEW.CLW',_ABCLinkMode_),DLL(ABCDllMode_)
ChangePagesG    Byte(0)
Ask               PROCEDURE(),DERIVED
SetChangePages    PROCEDURE(Byte ChangePages=0),BYTE,VIRTUAL,PROC
                END
_EndOfInclude_
```

The comment in the beginning of the file marks this as a member of the ABC library. The `Omit`, `Equate`, and `_EndOfInclude_` lines make up a compiler directive which prevents the INC file from being included more than once.

You need to include the AbReport.Inc file, because the new class is based on `PrintPreviewClass`, which is defined in the AbReport.Inc file.

Next you have to declare the new class, starting with the name of the new class (`BetterPreviewer`), declared as a child of the parent (`PrintPreviewClass`). The rest of the attributes tell the compiler and linker how to treat the class source.

It is also possible to declare data that will be available to the class as a whole, hence the class variable `ChangePagesG`. More on this later.

You have to specify the method(s) that you want to override; in this case you need to override the `Ask` method, as well as create a new method that will be used to supply information to the derived `Ask` procedure. Remember that because you are inheriting you only have to specify the differences between this class and the parent class; all the other methods you get by default.

The next file you have to look at is PreviewNew.Clw. Here's a skeleton source file (there's more procedure code to come):

```
Member
  Map
  End
  Include('PreviewNew.inc')
BetterPreviewer.Ask PROCEDURE()
Code
Parent.Ask

BetterPreviewer.SetChangePages PROCEDURE(Byte ChangePages)
RETVAL  Byte,Auto
  Code
  Return Retva
```

The `MEMBER` statement specifies that this is a generic source module (not bound to any one application). The `MAP` is necessary if you want to use any of the Clarion internal functions. And the `INCLUDE` points back to the class definition. Note that all methods that were defined in the INC file have to be coded in the CLW file. Important to note is the Parent.Ask call in the `BetterPreviewer.Ask` method; this ensures that the original code gets called as well as your own code.

Refreshing the ABC Classes

Before adding code to the new method, make sure the IDE can find your derived class. To save time Clarion only loads the ABC classes once it starts. If you add your own ABC classes, and you want the IDE to see them, do one of the following:

- Close the Clarion IDE and re-open it, or
- Click on the Global Button, tab across to the Classes tab and click on the Refresh ABC Information button

Using BetterPreviewer in your application

The next step is to tell your application to use the new `BetterPreviewer` class. To do this:

1. Open your application file, or, in a multi-DLL environment, the global data file.
2. Click on the Global button and select the Classes tab.
3. Click on the Process and Reports button.

4. Go to the Print Previewer droplist and select `BetterPreviewer` from the list.
5. Remember that if you are using a multi-DLL application that you have to compile the global data app as well as the apps where you use the Previewer.

Adding the “Copies” functionality

To add the number of copies functionality you need to do two things: firstly you have to add a control on the current preview window that enables the user to specify the number of copies they want to print; secondly, you need to add the code to manipulate the WMF queue to print the additional copies.

Here’s how you create the copies control. You have to specify a couple of variables to be able to create the new spin control (variable for number of copies), as well as its corresponding prompt. Update the `BetterPreviewer.Ask` procedure with the following variables:

```
BetterPreviewer.Ask PROCEDURE()
CopiesPrompt    long    !Variable for Prompt
CopiesSpin      long    !Variable for Spin control
Nr_Copies       byte    !Variable name to keep number of copies
Findtoolbar     long    !Find equate for toolbar
InstrVar        long    !instring variable
LoopCount       Long    !Count number of times looping files
NumberRecords   Long    !Number of records in preview queue
```

To add the control at first seemed like an easy enough task, but as most of us have experienced at one time or another, the job was a lot harder than it looked. The biggest problem here is that most of the preview screen is defined as either private or protected, and if something is protected you cannot reference it in the derived class. So using `GetPosition(?ZoomList, X,Y,Width,Height)` is not an option, nor is `SetTarget(PreviewWindow)`.

After I’d tried various things, Bruce Johnson came to the rescue with the following solution:

```
  !Find equate for the toolbar
  Findtoolbar = 0
  loop
    Findtoolbar = target{prop:nextfield,Findtoolbar}
    if Findtoolbar = 0 then break.
    if Findtoolbar{prop:type} = create:toolbar then break.
  end
```

This enables you to specify the parent when creating the control. It works by looping through the fields on the currently open window (represented by the system variable `TARGET`) until it finds the toolbar. You then specify the toolbar as the parent of the `PROMPT` and `SPIN` controls.

```
!set default copies to 1
Nr_Copies = 1
! create copies spin box and
! use the toolbar as the parent
CopiesSpin                 = |
  create(0,CREATE:Spin,Findtoolbar)
CopiesSpin{prop:text}      = '@n3'
CopiesSpin{prop:use}       = Nr_Copies
CopiesSpin{prop:rangehigh} = '10'
CopiesSpin{prop:rangelow}  = '1'
CopiesSpin{prop:step}      = '1'
CopiesSpin{prop:msg}       = |
    'Specify number of copies to print'
Setposition(CopiesSpin,325,8,20,9)
!Unhide the control
CopiesSpin{prop:hide}      = 0
```

You have created the necessary controls; now how do you print the additional copies?

You have to call the `Parent.Ask` first to display the Preview screen and get the input from the user in regards to the number of copies. Then you manipulate the queue to print the additional copies. The easiest method is to determine the original number of records in the queue, and then add a new record for each original record to the value of the number of copies minus 1.

Assume you have a report that consists of two pages. The queue will look something like this:

```
C:\temp\clatmp1.wmf
C:\temp\clatmp2.wmf
```

If the user wants to print two copies you need to manipulate the queue to look like this:

```
C:\temp\clatmp1.wmf
C:\temp\clatmp2.wmf
C:\temp\clatmp1.wmf
C:\temp\clatmp2.wmf
```

This will print each WMF file twice in the correct sequence without duplicating the WMF file on the user’s machine. Here’s the code:

```
  Parent.Ask ! Open Preview Window and get Nr_Copies
  !If user specified more than 1 copies
  If Nr_copies > 1
     !Get no records original queue
     NumberRecords = Records(SELF.ImageQueue)
     !Add filenames to queue -(copies-1) times
     Loop NI# = 1 to (Nr_Copies - 1)
        Loop LoopCount = 1 to NumberRecords
           Get(SELF.ImageQueue,LoopCount)
           Add(Self.ImageQueue)
        END
     END
     !set initial value of PagesToPrint property
     Self.SetDefaultPages
  END
```

When you call the `Self.SetDefaultPages` procedure it resets the total number of pages that will be printed. The initial value is `1-n`, where n is equal to the total number of pages in the report. It is important to note that if you are using the preview queue for other purposes as well that you must make sure that you do not add the whole queue but only the original number of records. For example, if you use the preview queue to create a Word document (insert the WMF files into a Word document) you do not want to create a document from the whole queue, especially if the user specified 20 copies.

Add the following code to the Local Objects, ABC Objects, Previewer (BetterPreviewer), Ask, Priority 4500.

```
!Get the number of records before the user specifies copies
LCL:Nr_of_records_in_PrintQueue = records(SELF.ImageQueue)
```

Adding the “Page of Pages”

To add the page of pages functionality to your report you have to do a couple of things. As I said earlier, a WMF file is really a binary file containing source code that tells the printer exactly what to print, and this is the solution to the problem.

Here are the steps that you need to take to add page of pages to your report:

1. Add a string to the report where you want to print the page of pages. You can use, for example, the string `_Pages_`.
2. Print the whole report and get the total number of pages in the report. Do not send the report to the printer just yet.
3. Go through all the WMF files in the queue, starting with the first one, and in each file change the `_Pages_` string to a string indicating the actual number of pages.
4. Let Clarion print the queue as normal

The first thing to do is to declare a DOS file to handle the WMF file:

```
BetterPreviewer.Ask PROCEDURE()
InstrVar        Long
IName           STRING(255),AUTO,STATIC
WMFFile               FILE,DRIVER('DOS'),NAME(IName)
                     RECORD
Buffer                 STRING(5000)
                     ..
```

To add the DOS driver to your application, click on Project, go to Database driver libraries and add the DOS driver.

All that remains is the code to change the WMF files. The only real catch here is that if you are replacing `_Pages_`, which consists of seven characters, you must replace it with seven characters. Using this method you can print reports up to 9999 pages, because the first three characters are the word "of" and a space. You can play around with the code but the way I have done it is to add the following to the report:

```
 Page <<# $$$$$$$
```

Page	String control on the report
<<#	?ReportPageNumber – this is the default Clarion page number
$$$$$$$	String(7) – variable set to "_Pages_"

This means that if your report has three pages in total, and you have not executed the new code, the first page will have this at the bottom of the WMF file:

```
Page 1 _Pages_
```

After the code has executed it will look like this:

```
Page 1 of 3
```

If you do not add the spaces to keep the total replacement to 7 characters you will end up with a blank page.

Here's the source code to modify the WMFs, which goes in the `Ask` method:

```
NumberRecords = Records(SELF.ImageQueue)
Loop LoopCount = 1 to NumberRecords
  Get(SELF.ImageQueue,LoopCount)
  IName  = SELF.ImageQueue.Filename
  OPEN(WMFFile)
  If error()
    Message('Error: ' & Error() |
      & '|Could not open page:' & Loopcount,'Preview')
  end
  Set(WMFfile)
  Loop
    Previous(WMFfile)
    If error()
```

```
      Set(WMFfile)
      Next(WMFfile)
      If error()
        Message('Error: ' & Error() |
           & '|Could not access page:' & Loopcount,'Preview')
        break
      End
    End
    InstrVar = INSTRING('_Pages_',WMFFile.buffer,1,1)
    if InstrVar <> 0
      If NumberRecords < 10
        WMFFile.buffer = |
          Sub(WMFFile.buffer,1,InstrVar-1) |
          & 'of ' & NumberRecords & '   ' |
          & sub(WMFFile.buffer,InstrVar+7,len(WMFFile.buffer))
      ElsIf NumberRecords < 100
        WMFFile.buffer = Sub(WMFFile.buffer,1,InstrVar-1)|
          & 'of ' & NumberRecords & '  ' |
          & sub(WMFFile.buffer,InstrVar+7,len(WMFFile.buffer))
      ElsIf NumberRecords < 1000
        WMFFile.buffer = Sub(WMFFile.buffer,1,InstrVar-1)|
          & 'of ' & NumberRecords & ' ' |
          & sub(WMFFile.buffer,InstrVar+7,len(WMFFile.buffer))
      Else
        WMFFile.buffer = Sub(WMFFile.buffer,1,InstrVar-1) |
          & 'of ' & NumberRecords |
          & sub(WMFFile.buffer,InstrVar+7,len(WMFFile.buffer))
      End
      Put(WMFFile)
      If error()
        Message('Error: ' & Error() |
          & 'Could not update page:' & Loopcount,'Preview')
      End
      break
    End
  End
  CLOSE(WMFFile)
End
```

If this code goes in the `Ask` method, why do you need `BetterPreviewer.SetChangePages`?

What happens if you want to disable the page of pages option? There are two ways to do this. The first is to change the `BetterPreviewer` class back to the standard `PrintPreviewClass`, but then you lose the Copies functionality as well. The second option is to switch the page of pages code on and off from your application. That's what `SetChangePages` lets you do:

```
BetterPreviewer.SetChangePages PROCEDURE(Byte ChangePages)
RETVAL  Byte,Auto
  Code
  Self.ChangePagesG = ChangePages
  Retval = ChangePages
  Return Retval
```

This is where you make use of the variable that is available to the class as a whole. You are going to pass a value to the method, the method will set the class variable, and that will be used in the `BetterPreview.Ask` method.

```
If Self.ChangePagesG = 1
   !Normal Code for BetterPreviewer -
   ! Page of Pages goes here
End
```

All you have to do now in the app is to embed the following code at Local Objects, ABC Objects, Previewer (BetterPreviewer), Ask, Priority 4500, or in other words before the parent call:

```
!Set the checking for _Pages_ = on
Previewer.SetChangePages(true)
```

That is it. There may be better ways do to this, and if you know of one I would be happy to hear from you. One possible improvement is a template to set the class properties. Feel free to email me or post your suggestions as reader comments.

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

It's corresponding to a real need But i use Legacy...
Hi I have not worked with legacy for years. When I have...
Very Nice - I'll certainly use it.

Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
DeveloperPLUS
tm

Home COL Archives

Topics > Templates > Templates, writing

Understanding Template Symbols

by steffen Rasmussen

Published 2002-08-27

Over the years Clarion Magazine has published a number of articles about template writing. Many of these articles are great for starting to write basic templates, but if you want something a bit more advanced, you’re on your own.

In this article I’ll introduce you to my preferred workflow in tackling more advanced template programming. I’m not going to show you every command possible in template writing, or how they are used. What I am going to do is show you a way to determine how you can use the different template commands.

There are five primary stages which I follow in any template development cycle:

1. Create working source code
2. Convert the embed points to template #AT statements
3. Make the #AT statements independent
4. Create Template prompts
5. Make the source code independent

Each stage is followed by a test which the code must pass before I proceed to the next stage.

Stage 1 – Create working source code

Before even considering writing any template I always write source code in the application. If the code is something that I am going to use over and over again, then it is a prime candidate for a template. For this article I’ll use the code written by Dave Harms in his article Checkboxes For Many-to-Many Relationships, including the most recent update, The Checkbox Class Performance Upgrade. I’m not going to provide any further explanation of the checkbox source code; for that you will have to read Dave’s articles.

Stage 2 – Convert the embed points to template #AT statements

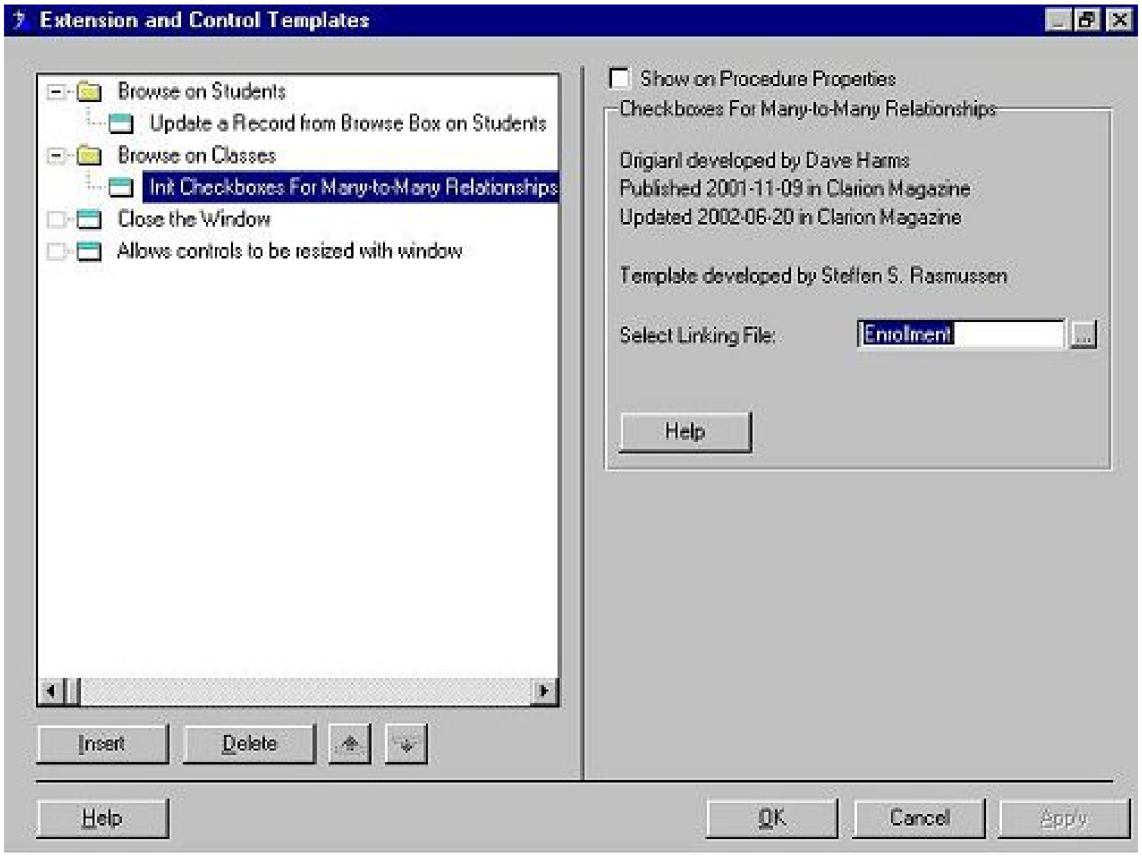
Converting source embed points to template #AT statements is the first part of creating a template, and is part of the basic template development. For a detailed approach see Andrew Guidroz II's article Creating #AT Statements The Easy Way.

In Dave's updated example program there are three places where he has embedded code. Use Andrew's embed template to create the corresponding three template #AT statements. For this particularly example an extension template is the most appropriate, so you will end up with:

```
#EXTENSION(BrowseManyToMany,'Init Checkboxes For Many-to-Many↵
   | Relationships'),WINDOW,PROCEDURE,REQ(BrowseBox(ABC))
#AT(%DataSection)
        !Code
#ENDAT
#AT(%BrowserMethodCodeSection,'1','TakeNewSelection','()')
        !Code
#ENDAT
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE')
        !Code
#ENDAT
```

Copy Dave's code into all the places with `!Code`. Now you have an extension template that can substitute all the embed code in the procedure.

Notice the `REQ(BrowseBox(ABC)` attribute - this indicates this template requires another template. In this case it requires that you select a browse box in the Extension and Control Templates window. For this particular code another requirement is that you have to select the "right side" browse box (see the first checkbox article for a discussion of "left" and "right" browses).


Extension and Control Templates
Browse on Students
Update a Record from Browse Box on Students
Browse on Classes
Init Checkboxes For Many-to-Many Relationships
Close the Window
Allows controls to be resized with window
Insert
Delete
Show on Procedure Properties
Checkboxes For Many-to-Many Relationships
Help
OK
Cancel
Apply

Origianl developed by Dave Harms
Published 2001-11-09 in Clarion Magazine
Updated 2002-06-20 in Clarion Magazine

Template developed by Steffen S. Rasmussen

Select Linking File: Enrollment

Figure 1. The checkbox extention template prompts

The next step is to prepare the template for test. First, comment all the embedded source so it is put out of action. Now implement the new extension template, compile, run and test to see if the program behaves as it should. You should also examine the generated source to see that the template code appears at the same location as the original embedded source. If the source matches, and everything works properly, it is safe to delete all the original commented code.

Stage 3 - Make the #AT statements independent

When using Andrew's template or any other `#AT` statement template generator, there are always some hard coded variables which have to be changed in order to reuse the template in different procedures. In this particularly code example there is only one `#AT` statement that has to be changed, namely the `#AT(%BrowserMethodCodeSection,'1','TakeNewSelection','()')`. Here the number `1` is hard coded. If you look at the definition of the `#AT` statement you see that the second parameter is the instance number. The browse is a control; therefore the instance number is represented by the built in template symbol %ControlInstance which you have to substitute with the "`1`".

After making that change to the extension template, all you have to do is compile, run and test to confirm that the program behaves as it should.

Stage 4 – Creating Template Prompts

Now it is time to create a template dialog containing prompts for every variable in the code. These variables are:

1. `Enrollment` - Linking FileManager
2. `ENR:StuSeq` - Linking File key
3. `ENR:StudentNumber` - Linking File left field
4. `ENR:ClassNumber` - Linking file right field
5. `STU:Number` - Left file primary key field
6. `CLA:ClassNumber` - Right file primary key field
7. `Queue:Browse` - View queue
8. `InClass_Icon` - Icon field
9. `CLA:ClassNumber` - Queue right field

In this example you will have to have nine template prompts:

```
#PROMPT('Linking FileManager: ',FILE),%LinkingFileManager
#PROMPT('Linking File key: ',KEY),%LinkingFileKey
#PROMPT('Linking File left field: ',FIELD),%LinkingFileLeftField
#PROMPT('Linking file right field: ',FIELD),%LinkingFileRightField
#PROMPT('Left file primary key field: ',FIELD),%LeftFilePrimaryKeyField
#PROMPT('Right file primary key field: ',FIELD),%RightFilePrimaryKeyField
#PROMPT(View queue: ', @s20),%ViewQueue
#PROMPT('Icon field: ',FIELD),%IconField
#PROMPT('Queue right field: ',FIELD),%QueRightField
```

The next step is to substitute the nine variables with the above created template symbols:

```
#AT(%WindowManagerMethodCodeSection,'Init', '(),BYTE')
ClassesBrowse.ActiveInvisible = true
 InitParams.LinkFM &= access:%LinkingFileManager
 InitParams.LinkKey &= %LinkingFileKey
 InitParams.LinkLeftField &= %LinkingFileLeftField
 InitParams.LinkRightField &= %LinkingFileRightField
 InitParams.LeftPrimaryID &= %LeftFilePrimaryKeyField
 InitParams.RightPrimaryID &= %RightFilePrimaryKeyField
 InitParams.ViewQ &= %ViewQueue
 InitParams.ViewQIconField &= %ViewQueue.%IconField_Icon
 InitParams.ViewQRightField &= %ViewQueue.%QueRightField
 ClassesBrowse.Init(InitParams)
#ENDAT
```

Apart from couple of variables that still need to be changed, you have a full working template. I have used this template for quite a while, but every time I had to use it I never got the variable selection right the first time. Of course, it should be possible to make the template determine a lot of these variables by itself, and this is where you take that extra step into a more advanced part of template writing.

Which template symbol to use

Before you can start to use the different template symbols you have to understand what I call the templates dependencies view, or just the template view. In other words what template symbols can you use? The best way to illustrate this is with a compressed version of the Symbol Hierarchy Overview from the Programmer's Guide:

```
%Application
   %Procedure
      %Window
         %Control
   %DictionaryFile
      %File
        %Field
        %Relation
```

As you can see the symbol hierarchy starts with `%Application`. This means that you can use all the built-in symbols that are dependent on `%Application`. Moving one level down the tree you will get all the symbols that are dependent on `%Procedure` and `%DictionaryFile`, and so forth. The further you move down the tree, the more built-in symbols will be available to you.

How does this apply to this template? You know it is part of the `%Application` as well as a `%Procedure`. You also know that the template is part of a list box because that was a requirement (`REQ(BrowseBox(ABC))`) for the template. A list box consist of a `%Control`, which is part of the `%Window`, and a `%DictionaryFile`. As you can see you have quite a few symbols to work with in this template.

Now it is just a matter of finding the right template symbols to use, and this in itself isn't always that easy. The short explanation for each template symbol give you an idea of its use. The trick is then to find out how to use it. So how do you do that?

Personally I use trial and error, placing comments into the template code containing the template symbols I think are usable. For example, if I needed the application name a pretty good guess would be to use the `%Application`, so my comment would look like:

```
!Application = %Application
```

After the program has been compiled I right click on the procedure where the template is used, select module and search for `!Application`. If the result is `!Application = enrol`, the template symbol can be used, but if it is `!Application =` it can't be used.

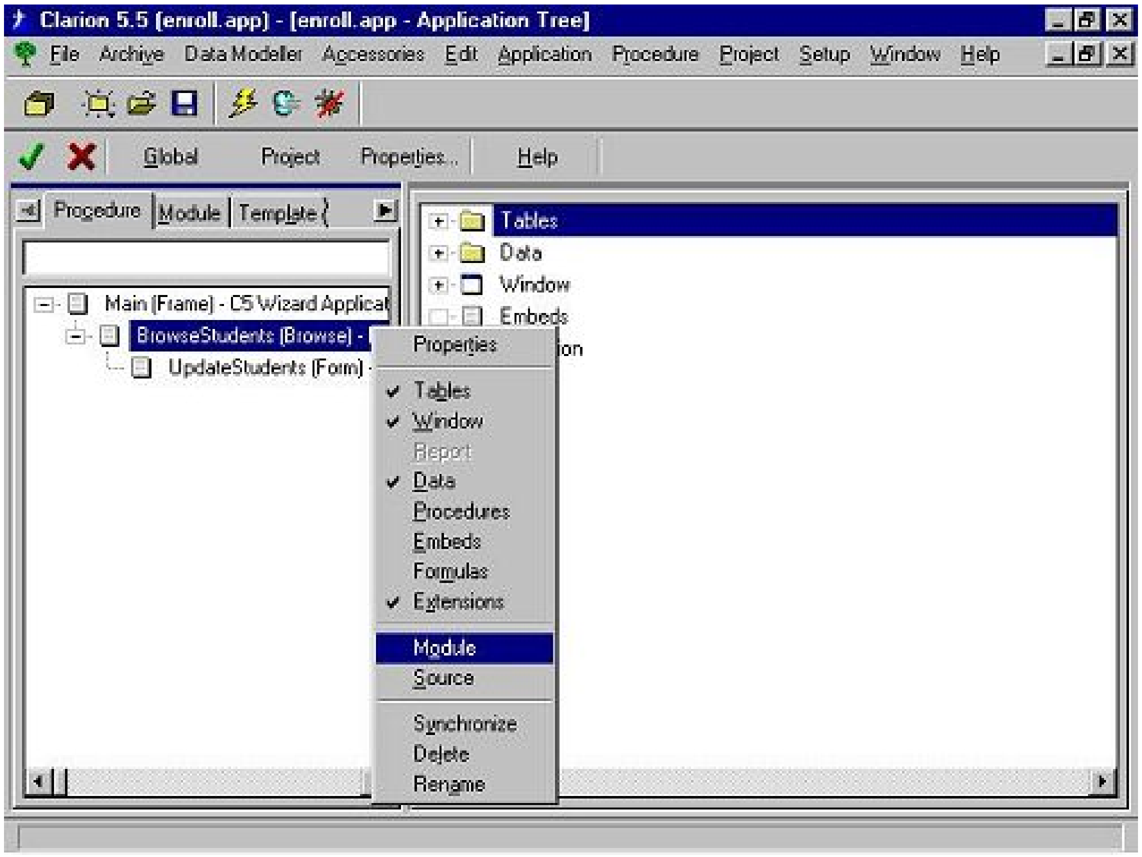

Clarion 5.5 (enroll.app) - [enroll.app - Application Tree]
File Archive Data Modeller Accessories Edit Application Procedure Project Setup Window Help
Global Project Properties... Help
Procedure Module Template
Main (Frame) - C5 Wizard Applicat
BrowseStudents (Browse) -
UpdateStudents (Form) -
Tables
Data
Window
Embeds
Properties
Tables
Window
Report
Data
Procedures
Embeds
Formulas
Extensions
Module
Source
Synchronize
Delete
Rename

Figure 2: Select Module

Testing symbol values is straightforward, that is, until you come across a multi-valued symbol. A multi-valued symbol is a symbol which is dependent on another symbol. For example, if you want to use the `%File` symbol, you can see in the symbol hierarchy overview above that it is dependent on the `%DictionaryFile`, which is part of this template's view. To show all the `%File`s you can use the following template statement:

```
#FOR(%File)
  !File = %File
#ENDFOR
```

In this example the result of the above code is:

```
!File = Students
!File = Teachers
!File = Classes
!File = Enrollment
!File = Courses
!File = Majors
```

Stage 5 – Make the source code application independent

In order to make the source code independent you have to determine which variables are in the templates dependencies view. Since the template only "knows" the list box control and the

Classes file (don't confuse this file, which is part of a student tracking application, with Clarion classes), the only variables that are in view are:

```
Queue:Browse      - View queue
InClass_Icon      - Icon field
CLA:ClassNumber   - Right file primary key field
CLA:ClassNumber   - Queue right field
```

Queue:Browse is the queue of the list box, and therefore the symbol %ControlFrom is a good place to start. So instead of using %ViewQueue use %ControlFrom:

```
InitParams.ViewQ &= %ControlFrom        !Queue:Browse
```

Observe the comment !Queue:Browse. Whenever you substitute a variable with a template symbol it is a good idea to always keep the variable as a comment. After you have compiled the program you can look in the Module code and confirm that the symbol is the same as the original variable. When you have a confirmation you can delete the comment.

That's one variable out of the way, and now you can delete the View Queue template prompt.

The icon field is part of the list box and therefore part of the multi-valued %ControlField, so you have to determining which field contains the Icon field. Fortunately Dave has made this pretty easy since the requirement for his class is that the Icon field is in the first column of the browse. To select a specific column you can use the #Select statement to fix the %ControlField to a particular instance number, which in this particular case is the same as the column number:

```
#SELECT(%ControlField,1)
```

Because %ControlField has been fixed to one value you don't need to use a #FOR loop:

```
InitParams.ViewQIconField&=%ControlFrom.%ControlField_Icon
```

The Right file's primary key field and the Queue right field contains the same value, so with out studying Dave's class more thoroughly I am going to assume that these two values are always going to be the same. In order to get the right fields primary key you have to be able to select the Classes file from the multi valued %File symbol. Because this template is related to the Classes file list box, the primary file is Classes. So by using the #FIX statement you can reference Classes outside the #FOR loop just like the #SELECT statement above. The code required for these two variables is a bit more extensive:

```
#FIX(%File,%Primary)
#FOR(%Key),WHERE(%KeyPrimary=1)
#SELECT(%KeyField,1)
    InitParams.RightPrimaryID &= %KeyField
    InitParams.ViewQRightField &= %ControlFrom.%KeyField
#ENDFOR
```

That's four template prompts out of the way, five to go. Again look at the template view. The `Classes` file is related to the `Enrollment` file in a one-to-many relationship. The problem is that `Classes` can also relate to other files, and there is no way to know which relationship to use. So there has to be a prompt for selecting the linking file in the many-to-many file relationship.

```
#PROMPT('Select Linking File: ',FILE),%SelectLinkingFile
```

Now you are down to at least one prompt and four variables:

1. `ENR:StuSeq` - Linking File key
2. `ENR:StudentNumber` - Linking File left field
3. `ENR:ClassNumber` - Linking file right field
4. `STU:Number` - Left file primary key field

Because of that one prompt, the template "view" has increased so it now contains both the right field as well as the linking field. For the linking field I am going to make another assumption, namely that the linking field can only contain two relations: one to the right field and one to the left field. If this wasn't the case another prompt must be included, but for this example I will leave it out. Now it is possible to loop through all the files and select the one that matches the linking file:

```
#FOR(%FILE),WHERE(%FILE=%SelectLinkingFile)
```

Next you will have to loop through all the relations in the linking file. You know there are two files – the left and the right file, but you have only told the template about the right file. So you are going to assume that if it isn't the right file it is the left. In other words you will have two `#FOR` statements:

```
!-------Left file--------
#FOR(%Relation), WHERE(%Relation<>%Primary)
#ENDFOR
!------Right file--------
#FOR(%Relation), WHERE(%Relation=%Primary)
#ENDFOR
```

I won't go into any further detail on how to find the last four variables – I'll leave that as a reader exercise. You may also want to read the section "Symbols Dependent on %Relation" in the Programmer's Guide.

Using user defined symbols from other templates

There is still one variable that I haven't mentioned yet, and that is the Class Definition Object name for the right list box. Clarion assigns a default name that Dave changed to `ClassesBrowse` (remember that Classes here refers to the table containing classes a student may take, and not Clarion CLASSes). The object name isn't part of the built-in symbols so there is no help here. You could create an extra prompt for the programmer to type the object name, but that isn't very satisfactory. There is nothing more annoying than having to type in a value two times. Since it has already been typed in once in the browse box template, which has created a list box control, it is possible to find the user defined symbol used to store the object name in the source code of the browse box.

Start by going into the Template Registry and open the Class ABC folder, as well as the underlying Control Templates folder. Find the BrowseBox template and select the Edit Definition button. Now you have to search for an unknown user-defined symbol. I usually start by searching for the input prompt text, which, in this case, is 'Object Name'. If this doesn't give any result try with one of the words, for example "Object", which has several hits. Now you just have to study the code for each of the hits and see if any of them can be used. The first user-defined symbol I came across was `%ThisObjectName`, but when using it in my own template the result, `BRW5::Sort0:Locator`, isn't quite what I expected, and certainly not something that can be directly used here. A bit further down I found the code `#SET(%ManagerName,%ThisObjectName)`. Maybe `%ManagerName` can be used? After a quick test it turned out that `%ManagerName` was just what I was looking for to substitute for `ClassesBrowse` in the template.

The last thing to do is to run over the template code and remove all unnecessary comments. That's it, now you have a template for the cciBrowseClassB class, as well as a new approach for developing your own templates.

Download the source

Steffen S. Rasmussen has graduated in Computer Science from Copenhagen Business College. Since then he has worked as a programmer, system technician and network administrator, and is currently IT manager. Clarion is a quite a new language to Steffen since his only been working with it since January 2000. But what better way to learn it than by trying to teach others! Steffen has also set up a web site to collect as many examples of different user interfaces as possible to inspire Clarion developers.

Reader Comments

Add a comment

For Step 2 finding the #AT's you could also use the...

Clarion Magazine

Clarion News

Published 2001-11-21

UltraTree TreeWizard

Paragon Design & Development has released TreeWizard for UltraTree Platinum Version 8, any Edition. The TreeWizard generates a complete UltraTree procedure, using standard wizard defaults for positioning, button size, button text, etc. The TreeWizard installs a configuration tab into the IDE for specifying UltraTree-specific defaults. The TreeWizard automatically populates all related files of the selected primary file, and automatically finds a valid update form if one is available in the application. Among UltraTree features automatically generated by the wizard are section default colors (up to six color sets used in rotation), data columns, section headers, and tree folder icons. Time to generate a fully functional tree from a dictionary of related file is about three minutes.

Posted Tuesday, August 27, 2002

Beta Testers Wanted

Gary James at CapeSoft is looking for beta testers for a new product which will enable you to easily integrate Microsoft Office components and functionality into your applications. As with File Explorer, the new product (called "CapeSoft Office Inside") will be template based (easy to use), and will implement Andy Ireland's COM classes (stable / fast) to wrap the Microsoft Office COM components (lots of useful features).

Posted Tuesday, August 27, 2002

FirebirdSQL Open Source ODBC Driver

Kelvin Chua reports he's successfully tested the latest open source ODBC Driver for FirebirdSQL with some browses and update forms, with 250,000 records. Available from IBPhoenix, still in beta.

Posted Tuesday, August 27, 2002

TPS.repair Templates 1.5 Released

Version 1.5 of the TPS.repair Templates for Clarion 5.5 is now available. All registered users are entitled to download the new version free of charge. A fully functional 30 day trial version is also available. New features include: Support for TopSpeed Multi-Table superfiles; Native support for Danish, Dutch and Portuguese; Enhanced command line interface; Improved manual (22 pages now); Support for Windows XP. In September the price goes up to US$75.
Posted Tuesday, August 27, 2002

xQuickFilter v2.12 Released

xQuickFilter v2.12 is now available. New features include: Connection xQuickFilter to application was changed; Added support for Fomin Runtime Report; Wildcard filtering; Automatically removal of prefix from field name when xQuickFilter popup menu generated. New Demonstration program (with FRB) and Install Kit for Clarion 5.5 are available. Install kit for Clarion 5 will be ready soon.
Posted Tuesday, August 27, 2002

RichReport 1.1 Released

Version 1.1 of RichReport is now available from solid software. This update is free for all registered users. It's not a big update introducing new features, only minor changes have been made, including: Added the OnBeforePrinting virtual method - overriding this method allows you to fine-tune the detail before it will be printed by the RichReport runtime library; Added the OnAfterPrinting virtual method.
Posted Tuesday, August 27, 2002

New Imaging SDK And Image-XChange Demo

Tracker Software has released a beta demo of the Image-XChange product. The first beta is expected in a week or so, with the Clarion templates following shortly after. Image-XChange works across the full spectrum of 32 bit Windows operating systems, and offers both Twain and WIA (for XP) scanning support and a good deal more. Pricing will be available soon and will include a special price break for existing PDF & TIFF-XChange SDK owners.
Posted Tuesday, August 27, 2002

Icetips Magic Buttons 1.1

Icetips Software has released version 1.1 of Magic Buttons. This is a complete rewrite using classes, and is much more flexible than the original version. Magic Buttons add background images to your buttons with an extension template. You have complete control over what images are used, you can select one image for all buttons, select images for individual buttons, exclude selected buttons or include only selected buttons. You can also specify variable image names and have the users pick the images they like. The Icetips Magic Buttons come complete with a demo and a 31 page PDF manual with detailed documentation of templates and classes. Through August 31st, Icetips is offering the Magic Buttons and the Magic Entries in a special introductory price of only $99.00, saving a total of $59.00 off the combined list price. Demo available.

Posted Friday, August 23, 2002

Icetips Cowboy SQL Templates

Icetips Software has released a new-to-Icetips product, the Icetips Cowboy SQL templates. Previously known as the CCS SQL templates, Icetips Software has taken over the sales and marketing of these templates and will be working on new features and improvements. Andy Stapleton will assist Icetips with technical support when needed. The Cowboy SQL templates are available for Clarion 2-5.5 Clarion templates (Legacy) and Clarion 5.0 and Clarion 5.5 ABC.

Posted Friday, August 23, 2002

Icetips Magic Entries New Release

Icetips Software has released a new product, Icetips Magic Entries. Magic Entries replaces the old-fashioned three dimensional entries with flat entry fields. Magic Entries is compatible with Clarion 4, Clarion 5 and Clarion 5.5, ABC and Clarion templates. It comes complete with a demo and a 37 page PDF manual with detailed documentation of templates and classes. Through August 31st, Icetips is offering the Magic Entries for a special introductory price of US$49.00, or in a bundle with Icetips Magic Buttons for only $99.00, saving a total of $59.00 off the combined list price. Demo available.

Posted Friday, August 23, 2002

INN Bio And News For 21-Aug-2002

For this week's Bio, INN visits Europe again to meet a former professional (and Army) cook. Now a programmer, he's spent time working with Clarion and IBM AS400s, and building (or not?) web sites. In the News section, look for a hot tip in the "Marketing Moments" series.

Posted Friday, August 23, 2002

New MailMerge Templates And Free Re-sort Template

Solace Software has released a new template which allows easy Mail Merging in your application. The template works with both standard text fields, string fields and SoftVelocity's RTF text control. The template allows you to specify prefixes and suffixes for the tokens to be inserted into text as well as renaming fields and changing pictures for merging. It also includes an undo merge facility so users can insert tokens and test the merged data. This template is also fully compatible with RichReport from Solid Software, which allows you to print RTF text on standard Clarion reports. As the two products are complementary, Solid Software and Solace Software are offering both as a special bundle for $150. A demo of the new mailmerge template which also includes RichReport as the print engine, is available. Also available is the free Resort template, which can be added to a Browse, Report or Process and allows users to select their own sort orders. The order may be any field in the current view and does not require a key to be defined. Multiple fields can be defined and each field may be in ascending or descending order. There have also been updates to the free template set, particularly the ScrRes template which can now detect any screen resolution and automatically change it to a specified resolution. There is also a bug fix for the problem with the Start bar being left in the middle of the screen on some systems after the resolution has been changed.

Posted Friday, August 23, 2002

Update to Simsoft Templates

A free update of the Simsoft Templates is available to registered users. New features include: A new extension template for the ECALENDAR calendar lookup and Diary procedures, which will pop up a window when called setting out appointments on a given date or range of preceding dates; A code template to simplify the calling of the new procedure; A small fix to the ECALENDAR procedure (appointments deleted did not remain deleted unless the days appointments were cleared); The simcalc.clw include file which seems to have been omitted from the previous installation file; A new sample application.

Posted Friday, August 23, 2002

Virtual EIP 1.3 Update

An update to Virtual EIP, version 1.3 is now available. New features and changes include: Select Columns switch (Browse); Use VCR Request switch (Browse); Improved After/Bottom Insert strategies (Browse); Fixed XP form alignment issue (Form); Conditional VEIP (Form - Other Tab); Post Accepted to Focus() (Form - Other Tab).

Posted Friday, August 23, 2002

Gitano Look Good Package Sale

The Look Good Package includes Theme Pack 1 and gBuddy's color template. As a bonus Gitano Software will design one box shot for your product and give you 20% discount on any custom graphics work. Price $79.95, on sale for $53.57.

Posted Monday, August 19, 2002

Gitano Summer Sale

From August 16 to September 3, 2002, you can get IconsXp, Theme Pack 1 and The Look Good Package with a savings of 33% off the regular price, full versions or upgrades. IconsXP is a collection of 160 + custom icons. Price $19.95, on sale for $13.37. Theme Pack is a set of 12 themes to be used in conjunction with Icetips Wizards, Standard and Professional, and/or G-Buddy. Includes IconsXP, price $29.95, on sale for $20.07.

Posted Monday, August 19, 2002

EasyAnimation 1.01 Released

EasyAnimatiuon 1.01 is now available. Changes include: Improved support for background process; Changed methods Init (added the new parameter) and Play (changed default values); New methods IsOpen and IsPlay; New demo.

Posted Monday, August 19, 2002

New Version Of DOS Printer

Dave Beggs has released version 6.9 of DOS Printer. New features include: Doesn't take focus when printing if the app is in a dos window; Faster; More Epson codes supported.

Posted Monday, August 19, 2002

New Clarion Book Coming

Russ Eggen has a Clarion book in production, targeted at intermediate to advanced Clarion coders. Expected length is around 200 pages. Price is not yet available, and the book is expected late August or early September.

Posted Monday, August 19, 2002

IMPEX Screenshot Pages

The Sterling Data web site now contains a page showing screen shots of the LogFlash (database audit trail) product.

Posted Monday, August 19, 2002

LogFlash Screenshot Pages

The Sterling Data web site now contains a page showing screen shots of the IMPEX (import/export) product.

Posted Monday, August 19, 2002

EasyMultiTag 2.02 Released

EasyMultiTag ver 2.02 is now available. All known bugs have been fixed, and the prototypes of the UpdateBegin and UpdateEnd methods were changed.

Posted Monday, August 19, 2002

chSTD Library 2.62 Released

The chSTD library ver. 2.62 is now available. Includes a bug fix and several additional features.

Posted Monday, August 19, 2002

ImageEx 1.3 Released

Version 1.3 of ImageEx (a free update for existing users) is now available. New features include: A panoramic image viewer - views 360° panorama pictures created using stitching programs (can be in any of the supported image formats). Additionally allows you to create clickable hotspots (rectangles, ellipses or polygons) with individual tooltip, color and opacity settings; StretchFilters (Lanczos and Mitchell); Grayscale() method added to the Extended Image control; CopyToClipboard() method added to the Viewer control; Improved

FlipHorizontally, FlipVertically, RotateLeft, RotateRight and Rotate180 methods (better speed). A new demo is also available.

Posted Monday, August 19, 2002

Clarion Memory Leak Checker

Andy Ireland is expected to release his Clarion leak checker product shortly. Price is $199 (but may initially be on sale at a lower price). This product will work with any version of Clarion that supports interfaces.

Posted Monday, August 19, 2002

Clarion Application Skins

Pratik has released his iAlchemy Skin product, which lets you add skins to your Clarion applications. Demo available.

Posted Monday, August 19, 2002

Capesoft File Manager 3 Beta

CapeSoft has released File Manager 3, beta 1. This release builds on File Manager 2. New features include an SQL engine, which allows FM3 to convert SQL tables as well as TPS files. You can also convert from TPS to SQL. Beta 1 includes support for MSSQL. Oracle support is in development now, and will be followed by support for other popular SQL engines. FM3 is $199, or $149 during the beta period. Existing FM2 users cross-grade to FM3 for $99. Users who purchased FM2 after June 1, 2002 will receive FM3 free of charge. NOTE: This is not an upgrade to FM2, but a new product. FM2 will continue to be supported.

Posted Tuesday, August 13, 2002

Free Browseless Form Template

Tabajara Consultoria has released a free no-browse form template. Additional free templates are available.

Posted Tuesday, August 13, 2002

New Addition To Simshape Templates

New in the Simshape Tempaltes is a control template that allows the button images to be assigned at runtime and easily changed whenever needed. The upgrade is free to registered users and is obtainable through ClarionShop.

Posted Tuesday, August 13, 2002

Whitemarsh Metabase Version 0505

The main addition to Whitemarsh Metabase V0505 is the ability to generate a Clarion TXD. To find out more about this new metabase release log onto www.wiscorp.com and press the What's New Button.

Posted Tuesday, August 13, 2002

ImageBrowsing Bundle Available

ImageBrowsingBundle is a bundle of three products from solid software: SysTree, SysList and ImageEx. These are the products required to build the ImageBrowser example from the ImageEx demo application. Cost is $239. Demo available.

Posted Tuesday, August 13, 2002

ExcelBond Version 1.3 Released

Sterling Data has released ExcelBond 1.3. This product creates native Excel XLS files direct from your TPS, DAT etc. files. It works as an add-on to the IMPEX templates. Demo available.

Posted Tuesday, August 13, 2002

ClarionSearch.com Temporarily Unavailable

Peck Kim Han's ClarionSearch site is down for some extended maintenance.

Posted Tuesday, August 13, 2002

UltraTree Version 8 Announced

Paragon Design & Development has announced UltraTree version 8. Now available are Lifetime Editions, which provide free updates for the life of the product, and a new Value Edition, which makes UltraTree standard features available at a price point not seen since 1997. The new TreeWizard feature is nearing completion. This wizard generates a complete UltraTree procedure containing a multilevel UltraTree. Other new features in version 8 include the View button template standard feature, and the Row and Column Rearrangement optional feature. The loading algorithm for Views was completely revamped, introducing memory caching of projected fields. This greatly improved UltraTree performance in many circumstances, particularly with the SQL and ODBC drivers.

Posted Monday, August 12, 2002

Product Scope 32 PRO, Version 4.5a Interim Release

Product Scope 32 PRO, Version 4.5a Interim release is now available. This program is used for the development and viewing of the Clarion Third Party Profile Exchange. This interim release is currently only available as a separate download - it will be incorporated into the full 4.5 install and V4 to V4.5 upgrade soon.

Posted Friday, August 09, 2002

SQT SQL Conversion Template Beta

Swiss Quality Tools has released a beta version of its SQL Conversion Templates. These templates make it possible to update SQL databases with dictionary changes. The sample application and the templates support only MS SQL at the moment. Support for additional databases is planned. Beta testers wanted. Cost and availability have yet to be determined.

Posted Friday, August 09, 2002

gFileFind Update Available

An update to gFileFind is now available with more features in the example application. Free to registered users.

Posted Thursday, August 08, 2002

Crystal RDC Wrapper

Klarisoft has released its Clarion RDC (Report Designer Component) wrapper which lets Clarion developers to take advantage of the latest features within Crystal Reports. This technology replaces old methods of integration with Crystal Reports such as Print Engine API and Crystal ActiveX control. It will enable you to take full advantage of the Crystal Report Print Engine. RDC Wrapper provides access to the Embeddable Crystal Reports Designer Control to allow your users to design and edit Crystal Reports in your application at runtime. RDC wrapper is available in full and lite versions, as source or DLL. Trial version available.

Posted Thursday, August 08, 2002

SetupBuilder 4.02

SetupBuilder 4.02 is available now. The update is available free of charge to all SetupBuilder customers. Release 4.02 contains several bug fixes and enhancements. If you have previously installed SetupBuilder 4.x it is recommended to uninstall that version prior to installing version 4.02. Please make a backup of all 4.x project files (.sb4)! SetupBuilder 4.02 costs $199.00 for a royalty-free usage license. A trial version is available.

Posted Thursday, August 08, 2002

0-HaZzle Price Reduction

Langaard Software has dropped the price of 0-HaZzle down to $69. Customers who has purchased 0-HaZzle for $199 can take out the difference in Langaard's other products (0-MeZz , 0-OverlapZ, and AnalyZe.IT) and/or in upcoming products (such as 0-BabelfiZh) when released.

Posted Thursday, August 08, 2002

xTransparent Window v1.2 Released

SealSoft's xTransparent Window v1.2 is now available. This release supports the Legacy template family.

Posted Thursday, August 08, 2002

IconsXP Update Available

IconsXP 1 update is now available - this release includes over 160 icons. Free for registered users, all others can purchase it for $19.95. Buy IconsXP today, or for a greater savings get the

Theme Pack 1 or the Look Good Package. Both of these include IconsXP.

Posted Thursday, August 08, 2002

0-OverlapZ Resize Extension

Aditech Langaard Software has released 0-OverlapZ, an extension template that makes it possible to: specify resize strategies for each control type in the window; specify resize strategies inside the window formatter; let you derive the resize strategies for one control from another control. Available at www.clarionshop.com.

Posted Thursday, August 01, 2002

ImageEx 1.2 Released

Version 1.2 of ImageEx is now available. This update is free for all registered users. New features include: support for EMF files; improved handling of transparent icons and metafiles; a method for replacing colors to the extended image and the viewer control; a method for convolution filters (3x3 pixels) to the viewer control (with several pre-defined filters like blur, soften, sharpen, edge detection); the thumbnailer can now store thumbnails directly to BLOBs; the thumbnailer can now store thumbnails directly into image list, allowing its use directly with SysTree and SysList.

Posted Thursday, August 01, 2002

Latest PublicPIM Version

The latest version of PublicPIM is now available. James Orr is still looking for someone to implement this in Clarion.

Posted Thursday, August 01, 2002

chSTD library 2.61 Released

The chSTD library ver. 2.61 is now available. Changes include: added chFB functions module (49 functions and procedures); added test example for the basic functions module; minor modifications in the source code of the templates not influencing their functionality. Basic functions include date/time processing, error reporting, multi-threading, OS procedures, runtime expression evaluation, string processing, and window processing.

Posted Thursday, August 01, 2002

Reader Comments

Add a comment

A Limerick Contest!

Published 2002-08-29

There’s been a spate of rhyming in the SoftVelocity chat newsgroup, ranging from doggerel to limericks to haiku. Some range! Favoring the limericking crowd, Richard Rogers is throwing down the following challenge, hosted and judged by Clarion Magazine.

> Here's an offer to you gals and guys.
> The Sylkie will offer a prize
> to the best loop or case
> with a limerick base
> and we'll hope that there aren't any ties!

The winning entry will receive either a license for App Init, or a cool little notes database, from Richard. The rules are as follows:

- The limerick itself must be compilable Clarion source code, in the form of an IF, LOOP, or CASE structure.
- Words to be used can be declared (apart from the verse) as equates, data labels, calls to external routines, etc.
- Points given for actual usefulness of the code, rhyme, meter, double-entendre (but please remember this is a family magazine), etc.
- Line continuation characters are considered punctuation.
- You may only use a comment on the last line.

An Example (you can do better than this, right?):

```
  DATA
IntoTheBrew  LONG(0)
Land         LONG(0)
Sea          EQUATE(100)
Water       EQUATE(5)
Tea           EQUATE(5)
Sugar            EQUATE(1)
```

```
    CODE
Loop until Land = Sea
   Land = Water + Tea
   IntoTheBrew = |
   Sugar + 2
END ! Of Loop and an exit for me
```

The contest ends midnight GMT, September 8, 2002. Send your entries to limerick@clarionmag.com.

Reader Comments

Add a comment

Clarion Magazine

Data Structures and Algorithms Part VII - Up a Tree

by Alison Neal

Published 2002-08-29

In this installment of this series on data structures and algorithms, I'll be talking about the Tree data structure. Tree data structures should, in no way, be confused with Tree Browses. The Tree browse is merely a visual representation of the relationships between various data elements or records. The Tree data structure, on the other hand, is like the Stack and the List, which are a means of maintaining data in computer memory.

People use trees quite a lot to notate relationships between entities and to show dependencies between entities. For example, genealogists use the family tree, as shown in Figure 1.

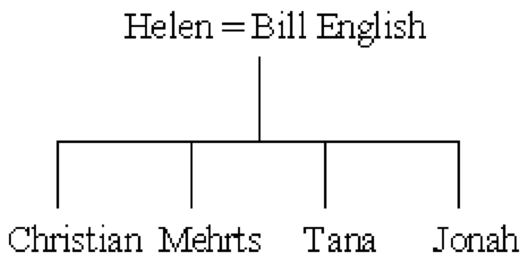


Figure 1. A family tree

Each element in the tree is known as a *node*, and the link between a parent and child is called a *branch*; each node except the root must descend from a parent via a branch. The root of the tree is known as the *ancestor* of all the other nodes in the tree. In a general Tree there are no limits on the number of children a node may have, however the Family Tree probably isn't the best example; it's overly complicated because each person has two parents and possibly half brothers and half sisters. For purposes of this discussion, I'll restrict myself to trees where each node has only one parent. Figure 2 shows a tree of a knockout sport tournament where, after each round, one competitor goes forward to the next round (e.g. in a wrestling match between Tana and Jonah, Jonah goes through to the next round):

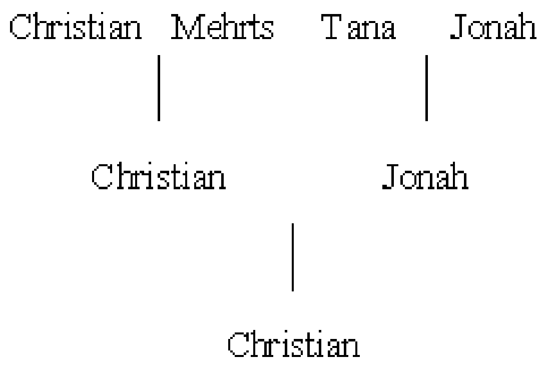


Figure 2. A knockout tournament tree

The Tree is fundamentally a hierarchical structure. For example, a company organisation chart is a tree. It defines the relationships between management and workers (Figure 3).

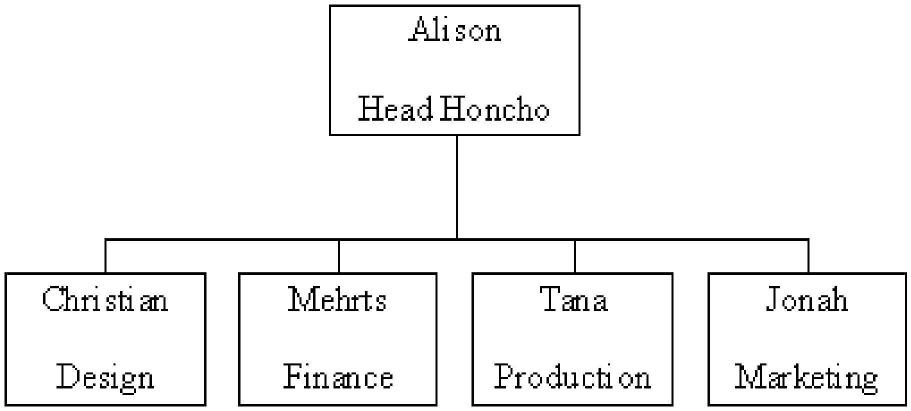


Figure 3. An organizational chart

Clarion programs can also be represented as a tree, with each procedure being a node, the calling procedures being the parent and the subordinates being the children. Some compilers actually break programs down into trees before translating them into machine code.

The elements of a tree can be of any type. If they are integers then I'd talk about a "tree of integers", if strings then a "tree of strings." I can, however, also use a "tree of lists" or a "tree of stacks" if I need to.

An important assumption with all of these trees is that the sub trees are disjoint; that is, they do not interconnect with each other; a node cannot have half-brothers or half-sisters, as in Figure 4.

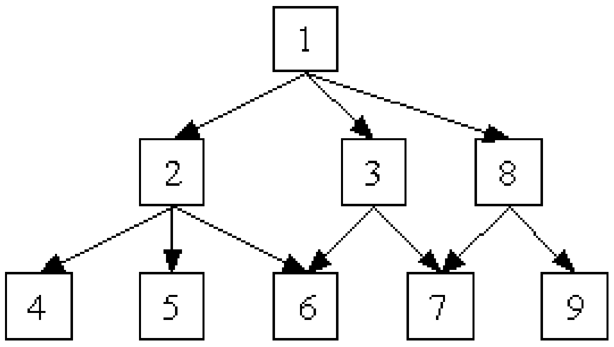


Figure 4. A Directed Acyclic Graph

The structure above is known as a *Directed Acyclic Graph* (DAG) and isn’t normally considered to be a tree. Where there is no interconnection between sub trees, any given node can itself be the root of another complete tree. Figure 5 shows the "tree" in Figure 4 with the interconnections removed.

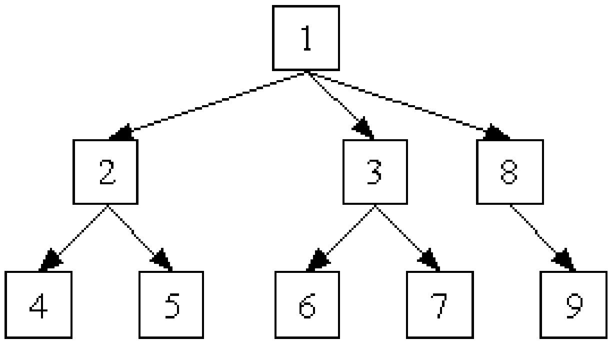


Figure 5. The tree with no interconnections

The tree in Figure 6 is also a tree in its own right.

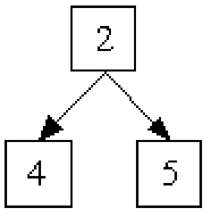


Figure 6. A tree, taken from another tree.

A Tree can therefore be defined recursively as either empty, or as a node with a number of sub trees (which may be empty). So, like the other recursive structures, the list and the stack, I have to be extremely careful in coding for terminating conditions.

Binary Trees

While a node can have an unrestricted number of children I will only be considering Binary Trees. A Binary Tree is where a node can only have two children at most. This by no means restricts the application of trees, because a General Tree can be transformed into a Binary Tree and vice versa without any loss of data. The Binary Tree is generally accepted as being the most useful type of tree structure.

There are two types of Binary Tree implementation: the linear representation, which can be achieved through the use of arrays, and the linked representation, which uses reference variables. I will only be covering the linked implementation. Binary Search Trees

The Binary Search Tree, like the ordered list, is maintained in order from the start. However, finding something in a Linked List can be slow because you have to start at the head each time. As the ordering property of the Binary Search tree puts lower values in the left sub tree and greater values in the right sub tree the number of nodes I have to visit to find the data element I want can be "effectively" halved. I say effectively because as I will show later, this will only occur under ideal circumstances.

For example if I want to add the numbers 22,99,4,12,33,9,2, and 100 to my Binary Search Tree, my tree would look like Figure 7.

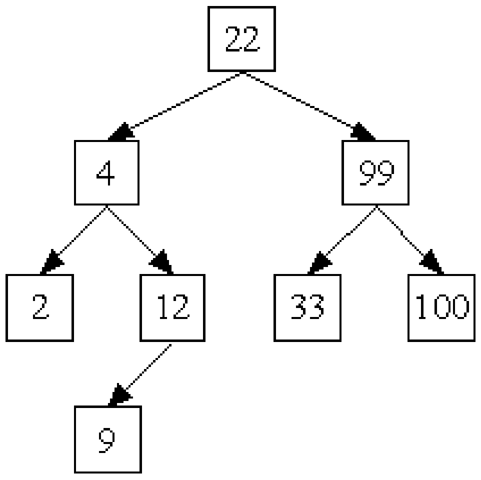


Figure 7. A binary search tree

The linked implementation of the Binary Search Tree is defined thus:

```
treeNode       CLASS,TYPE
NodeVal          ULONG
ltree            &treeNode
rtree            &treeNode
               END
root           &treeNode
```

The methods I've used for the Binary Search Tree are as follows:

Method	**Description**
`Init`	Initialise the tree
`Kill`	Dispose of the tree
`IsEmpty`	Check to see whether the tree is currently empty
`Insert`	Insert an item into the tree
`Height`	Find out the current height of the tree
`Find`	Find an element in the tree

The code for the init method is:

```
SELF.root &= NULL
```

The code for the kill method is:

```
SELF.root &= SELF.del(SELF.root)
```

This calls a recursive del function that runs through the tree deleting the left most nodes and then the right most nodes:

```
    IF ~t &= NULL
      t.ltree &= SELF.del(t.ltree)
      t.rtree &= SELF.del(t.rtree)
      SELF.curr &= t
      DISPOSE(SELF.curr)
      SELF.curr &= NULL
    END
    RETURN t
```

It is important to remember when deleting nodes that if you delete the head node first without storing the reference variables, then you will lose the ability to access the subordinate nodes so you can delete them; the result is a memory leak.

The code for the `isEmpty` method is very similar to that used for the stack and the tree. If the root is `NULL` then the tree must still be empty:

```
RETURN CHOOSE(SELF.root &= NULL,TRUE,FALSE)
```

The Insert method like the `Kill` method makes a call to another recursive function:

```
SELF.root &= SELF.Binsert(SELF.root,yourVal,i)
RETURN i
```

Note here that I've used i to represent wether the insert was successful or not. My implementation of the Binary Search Tree does not maintain duplicate data elements. It is a matter of choice whether you want to store duplicate items or not. If you do not then the add method can be slowed down as it performs a check every time to see if an element already exists, and if you do then there is an increase in utilized memory space.

So for my implementation if I try to insert something into the Tree that already exists I like to be aware of it.

```
binTree.Binsert     PROCEDURE(*treeNode t, ULONG yourVal, *BOOL flag)
    CODE
    IF ~t &= NULL AND t.nodeVal = yourVal THEN flag = FALSE.
    IF t &= NULL
      SELF.curr &= NEW(treeNode)
?     ASSERT(~SELF.curr &= NULL)
      t &= SELF.curr
      t.ltree &= NULL
      t.rtree &= NULL
      t.nodeVal = yourVal
      flag = TRUE
    END
    IF yourVal > t.nodeVal
      t.rtree &= SELF.Binsert(t.rtree,yourVal,flag)
    ELSIF yourVal < t.nodeVal
      t.ltree &= SELF.Binsert(t.ltree,yourVal,flag)
    END
    RETURN t
```

The `Binsert` function above starts by checking whether the current node value equals the new value, and if it does then the insert has been unsuccessful. Next, if the current node is `NULL` then I must be in the right place to insert the new value, and the insert is successful.

However if the new value is greater than the current value, then I need to go down a level to the right, or if the value is less than the current value then I need to go down a level to the left, and try again.

To find out the current height of the tree again I need to call a recursive function:

```
binTree.BTheight            PROCEDURE()
    CODE
    RETURN SELF.Bheight(SELF.root)
binTree.Bheight         PROCEDURE(*treeNode t)
L   ULONG
R   ULONG
    CODE
    IF t &= NULL THEN RETURN 0.
    L = SELF.Bheight(t.ltree)
    R = SELF.Bheight(t.rtree)
    RETURN CHOOSE(L > R, 1 + L, 1 + R)
```

The tallest side, or the side with the most levels decides the height of the Tree. The addition of one to either `L` or `R` is to account for the current node. So, with each recursion where the node does not equal `NULL` one is added. While the Binary Search Tree does maintain data in order is does not maintain tree "balance", which means that one side of the tree may be significantly higher than the other or one side may have more nodes allocated than the other. If a tree becomes too tall or too wide, then processing time can be significantly impacted. In my next article I will be looking at some ways of maintaining balance.

The `find` method as you can see is also going to use a recursive function:

```
binTree.BTfind          PROCEDURE(ULONG yourVal)
    CODE
    RETURN SELF.Bfind(SELF.root, yourVal)
binTree.Bfind           PROCEDURE(*treeNode t,ULONG yourVal)
    CODE
    IF t &= NULL THEN RETURN FALSE.
    IF t.nodeVal = yourVal THEN SELF.curr &= t; RETURN TRUE.
    IF t.nodeval < yourVal
      RETURN SELF.Bfind(t.rtree, yourVal)
    ELSE
      RETURN SELF.Bfind(t.ltree, yourVal)
    END
```

It’s in searching that the Binary Search Tree comes into its own. Rather than visiting every node through the head of the List, the Binary Search Tree can split the search time required by half, because the data is split left and right in logical sequence.

Summary

The Binary Search tree can be less efficient than the linked list because each node holds two reference variables, many of which may in fact be `NULL` depending on the order of insertion. If the data is entered in the tree in sorted order the tree can effectively collapse into a linked list. For example if I entered the numbers 2,4,9,12,22,33,99 and 100 in order, I would get the tree in Figure 8.

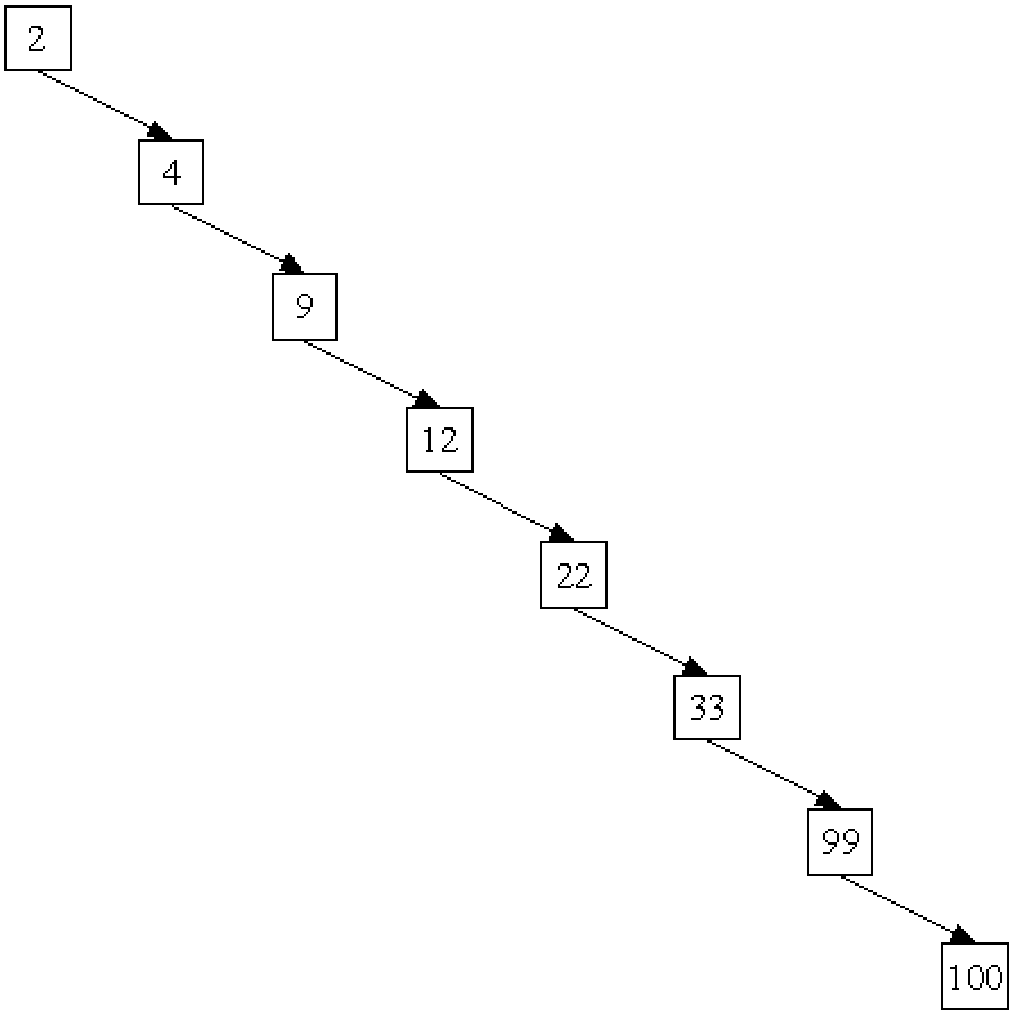


Figure 8. A one-sided tree

This is definitely not a desirable situation but Balanced Trees were developed to address this problem, and I will cover these next.

Download the source

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, *Manufacturers, Military & Government, Legal & Financial, and Retail.*

Reader Comments

Add a comment