INVEST in your own abilities

Clarion magazine

published by CoveComm Inc.

Clarion MAGAZINE

Search

Go

Home

COL Archives

Advertise

Reader Comments

Call for Articles

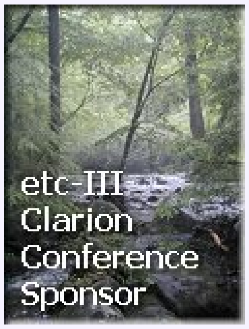
Information
Log In
Membership/ Subscriptions
FAQ
Privacy Policy

Contact Us

Downloads
PDFs
Freebies

Open Source

Site Index



Enter the **Clarion Magazine Sweepstakes** and you could win a **Compaq iPAQ** or an **ETC conference registration**! Other prizes too! No purchase required.

Normal Files, Normal Display

One of the basic concepts of normalization is that every record in a file or row in a SQL table should be uniquely identified. Steve Parker addresses two aspects of unique IDs: the generic motivation for using them, and the difficulties they can present when displayed to the end user.

Posted Monday, September 10, 2001

Designing Crosstab Reports In Clarion (Part 1)

One of the more common and useful reports is the crosstab report, which can be defined as any report that summarizes data in a two dimensional grid, such as breaking down sales data into monthly or quarterly sales. Other report writing tools often supply a wizard to guide the user through the process of creating this type of report. Clarion, however leaves the programmer to his or her own devices to create this standard summary. Part 1 of 2.

Posted Tuesday, September 11, 2001

Finding Unused Variables

The Clarion compiler has a wonderful pragma called Variable Declared But Never Used: wdnu. Andrew Guidroz shows how to use this pragma to clean up your old embedded source.

Posted Wednesday, September 12, 2001

Self-Upgrading Applications

Upgrading networked applications can take a lot of administrative effort, unless your applications know how and when to

Win $50 From Gitano Software

OE Message Converter Demo Updated

xDataBackup Manager v1.0

ExpressFlash OE Integration

xDigitalClock v1.1

PD Browse Button, Drop Combo, File Drop Lookups Updated

G-Sec Special Extended

New RUpdate Template Demo

RInstall Template Updated

G-Sec Update

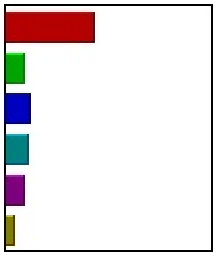
Release Candidate Available For Third Party Testing

ReportDAT! Is Back

RInstall Now Supports Legacy Templates

SURVEY

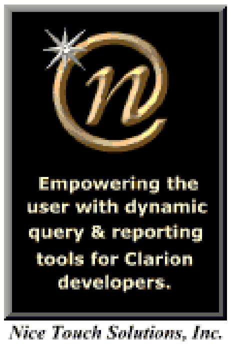
When do you plan to add XML capability to your applications?



🟥 No Plans (45%)
🟩 Developing now (10%)
🟦 In 3 months (13%)
🟦 In 6 months (13%)
🟪 In 1 year (10%)
🟫 More than 1 year away (6%)

229 Total Votes

Previous Surveys


Empowering the user with dynamic query & reporting tools for Clarion developers.
Nice Touch Solutions, Inc.

upgrade themselves. Jeff Berlinghoff explains his approach to making Clarion applications self-upgrading.
Posted Thursday, September 13, 2001

When a Datum is a Function

Long time Clarion users distinguish two types of procedures. One does not return a value and is referred to as a "procedure." The other returns a value and is referred to a "function." By this definition, reasons Dr. Parker a datum implicitly *is* a function.
Posted Thursday, September 13, 2001

Designing Crosstab Reports In Clarion (Part 2)

One of the more common and useful reports is the crosstab report, which can be defined as any report that summarizes data in a two dimensional grid, such as breaking down sales data into monthly or quarterly sales. Other report writing tools often supply a wizard to guide the user through the process of creating this type of report. Clarion, however leaves the programmer to his or her own devices to create this standard summary. Part 2 of 2.
Posted Thursday, September 13, 2001

Remembering September 11, 2001

On this international day of mourning, Winnipeggers, visitors, and a number of still-stranded American travelers gathered at City Hall at noon to remember the events of Tuesday, September 11. The service was moderated by Mayor Glen Murray, and included speakers on behalf of all three levels of government as well as various faith groups.
Posted Friday, September 14, 2001

No Articles This Week

There will be no articles published this week in Clarion Magazine - normal publication will resume next week.
Posted Monday, September 17, 2001

PDF for September 9-12, 2001

All Clarion Magazine articles for September 9-12, 2001 in PDF format (part 1 of a double issue).
Posted Monday, September 17, 2001

PDF for September 13-15, 2001

All Clarion Magazine articles for September

ProDomus ClarioNET Status

SealSoft xPictureBrowse v2.0 Released

PDF-XChange Template Fix/Update Available

xAppWallpaper Manager v1.3 released

RInstall Update And Release 1 Notification

VCRFlash Released

ARCO Word Reporter Holiday Schedule

xWindow Settings v2.0.8 Released

Windows XP Design Guidelines

OpenClose 1.2 Now Available

Metabase Version 5 Released

13-15, 2001 in PDF format (part 2 of a double issue).

Posted Monday, September 17, 2001

How To Make Unplanned Service Improvements

The last few days have been very entertaining here at Clarion Magazine. My plan for a seamless migration to a new IP block went quickly awry when an apparent routing problem related to the migration more or less took ClarionMag offline. Here's what happened...

Posted Friday, September 21, 2001

Report Redirection

This week Steve Parker takes on report redirection. Although there are excellent third party products available (and Dr. Parker isn't shy about making recommendations), there's something to be learned by reinventing the wheel.

Posted Tuesday, September 25, 2001

Do Not Adjust Your Browser

At approximately 5:22 p.m. Central Daylight Savings Time today (or 10:22 p.m. GMT, I think) my ISP (Group Telecom) switched the ClarionMag servers to a new network. The site was unavailable for approximately 9 minutes during this time, slightly less than the allotted 15 minute window.

Posted Tuesday, September 25, 2001

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

In this three part series Matt Grossmith shows how to use the ubiquitous WinInet DLL to transfer files with FTP, delete and rename files on an FTP server, and install callback procedures to monitor file transfer progress.

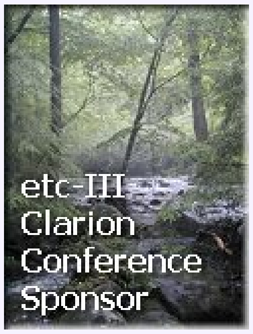
Posted Wednesday, September 26, 2001

Interview: ClarioNET's Michael Brooks

One of the more exciting Clarion developments in recent years is ClarioNET, a thin client adaptation of the Internet Connect technology. ClarioNET was initially developed by Michael Brooks, who is now working together with SoftVelocity to bring this product to market.

Posted Friday, September 28, 2001

Clarion MAGAZINE



Normal Files, Normal Display

by Steven Parker

Published 2001-09-10

One of the basic concepts of normalization is that every record in a file or row in a SQL table should be uniquely identified. The effort necessary to ensure unique identification has practical and immediate benefits in some cases: browses on SQL tables without row level unique identification can behave bizarrely in Clarion (for example, rows may show up multiple times, or errors can be thrown on an attempted update).

Since I am still developing against flat files, unique identifiers preventing spurious duplicate browse lines doesn't motivate me a whole lot. To me, the real utility of unique identification lies in the facilitation of relations and lookups. This is true for both flat files and SQL.

I want to address two aspects of unique IDs: the generic motivation for using them, and the difficulties they can present when displayed to the end user.

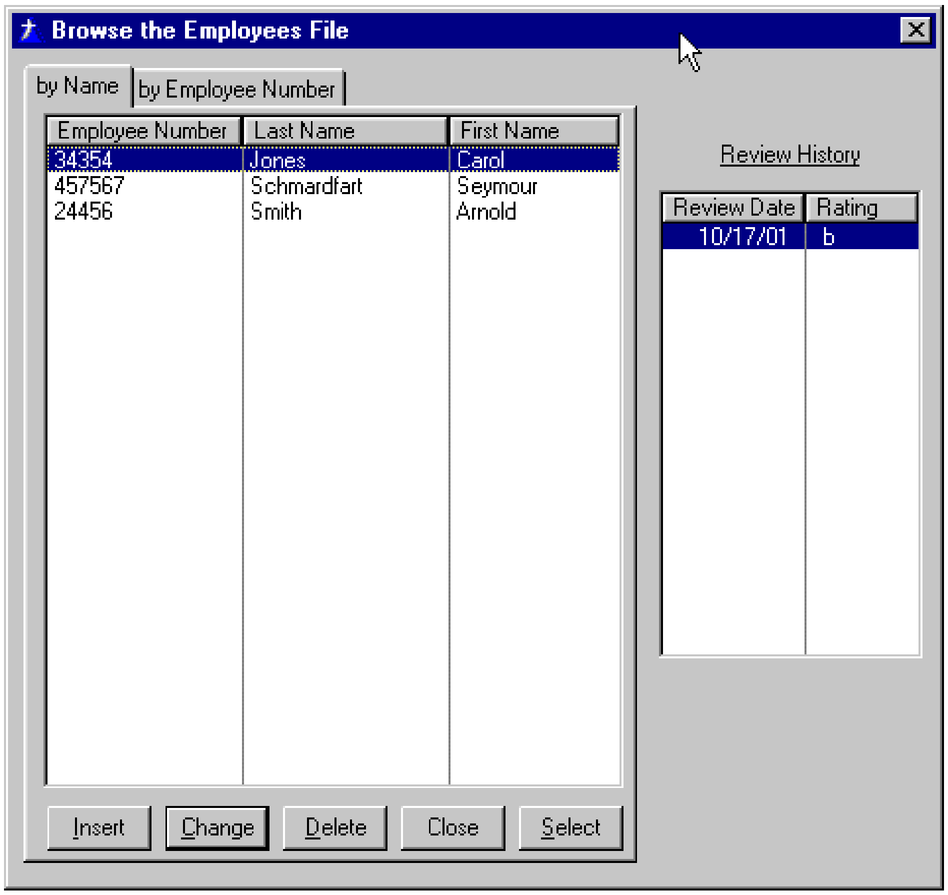
Typically, Clarion developers call this unique identifier "SysID" or something similar, and create it by template-controlled autonumbering.

How does autonumbering work?

Imagine a human resources application. One of the files (or tables) will almost certainly contain basic demographics on employees (possibly, this file will be called "EMPLOYEES"). Another might contain employee reviews (I suppose it *might* be called "REVIEWS").

If each employee has a SysID assigned (a/k/a primary key) then, if that number is also contained in REVIEWS records (a/k/a foreign key), it is easy to create code to retrieve all the reviews for a given employee (and only that employee) with a few mouse clicks.

The point is not that the same datum must be in both files/tables in order to link records; that should be fairly obvious. The point is that this manner of linking makes it *easy* to do things in our apps that make them appealing to users. For example, using this kind of link, it is (even) possible to "pair" the parent record with the child records in a single browse, as shown in Figure 1.



Employee Number	Last Name	First Name
34354	Jones	Carol
457567	Schmardfart	Seymour
24456	Smith	Arnold

Review Date	Rating
10/17/01	b

Figure 1. Review table "follows" employee table

In the dictionary for the demo application, you will notice that I have both a SysID and an Employee Number field. This is because an Employee Number is not sufficiently reliable as a linking datum. An Employee Number can change. Of course, *I* would never allow that without adequate referential integrity constraints; I would configure things so that changes would always cascade. Always.

But, suppose *you* write an application against these data files ....

Now suppose a DBA changes a table layout. Suppose there is a data corruption. Suppose a data entry clerk, in another application that does not check for duplicates, accidentally re-uses an existing Employee Number.

So much data, so many sources of mashed links.

What I need is a datum that uniquely identifies a master (parent) record, is included in each related record (RI constraints) and *is not changeable* by a user or, even a DBA (i.e., is under no one's control but mine).

An autonumbered key fits this bill to a "T." Indeed, many developers use such a field and never display it to the end user at all.

And that is why there is a SysID *and* an Employee Number.

Ok, unless you're completely new to data-driven applications, this is not news and it is not rocket science. Neither is it news that if data is used in many places, is should physically exist in only one place. From the one place where the data physically exists, it is looked up everywhere else it is needed or is useful. And, in all such cases, the data is identified by a SysID-type field. For example, an employee's name, address and phone number are in EMPLOYEES but anywhere else I need to see or print that data, I look it up from the central location using EMP:SysID.

Now, consider the simple Employee-Review structure described above. But, consider it from the point of view of the REVIEWS file (entry of performance reviews will not always be through the employee master file, so access directly to REVIEWS is necessary).

A "Wizard-ed" browse would first appear empty, expecting an employee to be selected, only then completing (as in Figure 2).

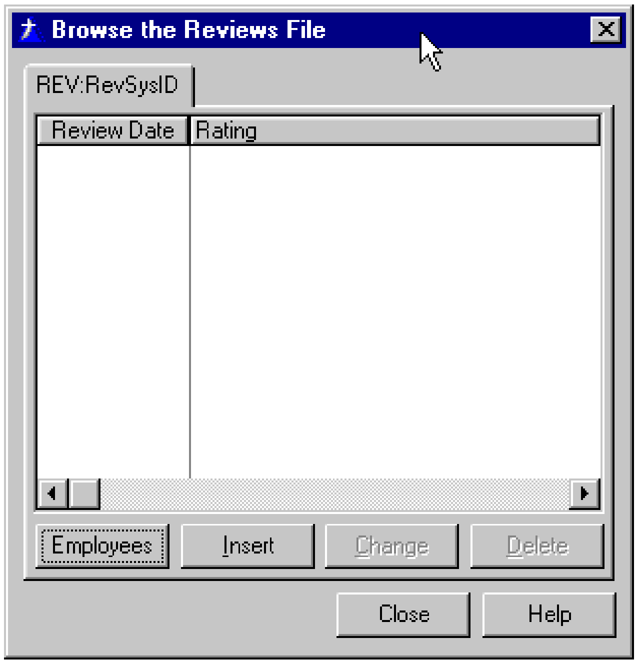
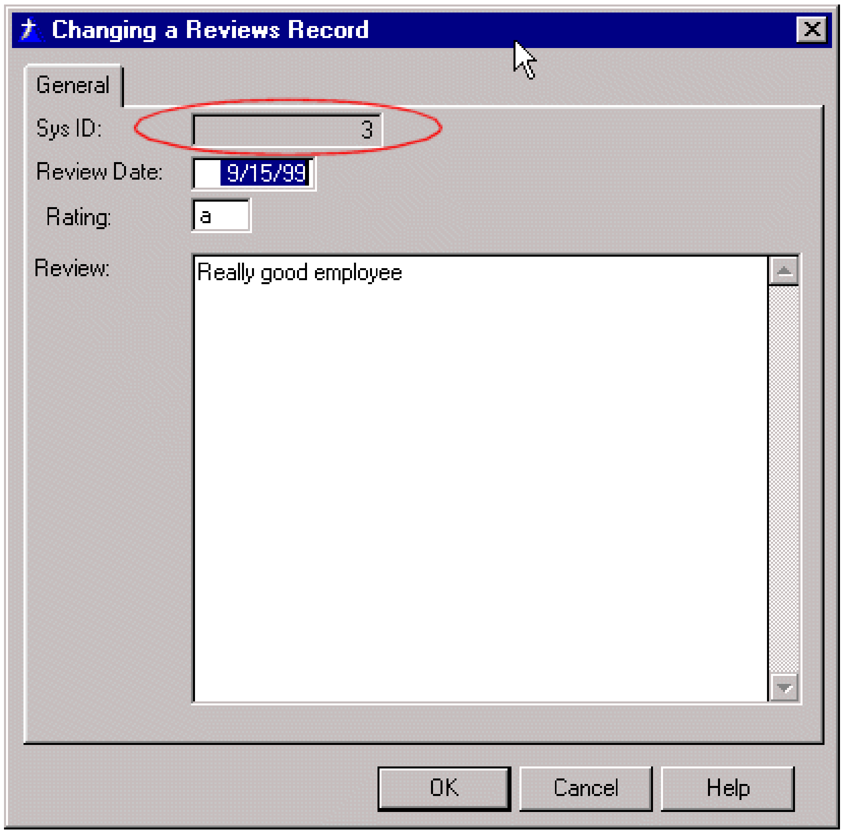


Figure 2. "Wizarded" child browse after parent selection

Not very user friendly. The typical update form (Figure 3) is no better:

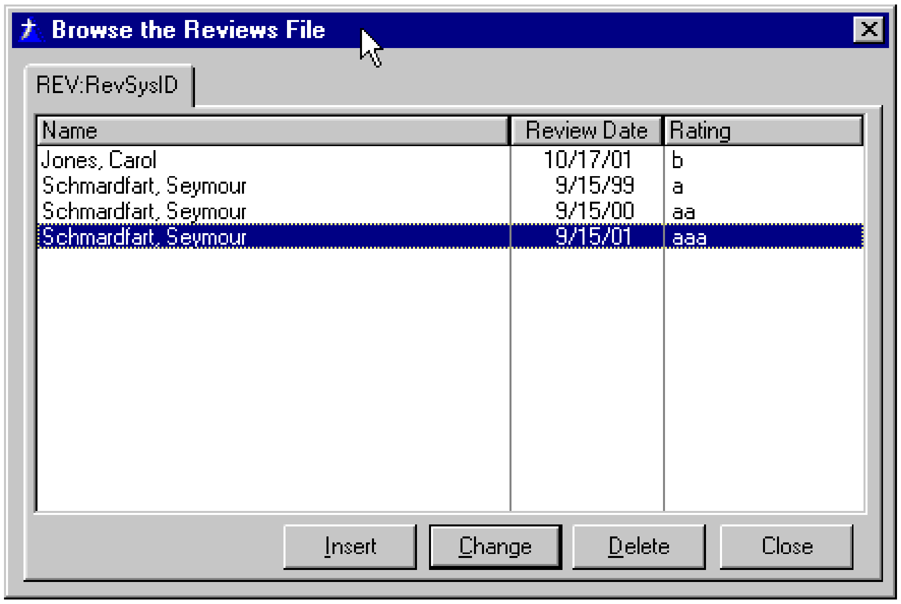


Sys ID: 3
Review Date: 9/15/99
Rating: a
Review: Really good employee

Figure 3. Child update form

It is no better because, while I may or may not know Seymour, the employee whose review is shown in Figure 3, I have no idea what a "3" is (the data files are included in the demo app so you can see that "3" is indeed Seymour Schmardfart[1]).

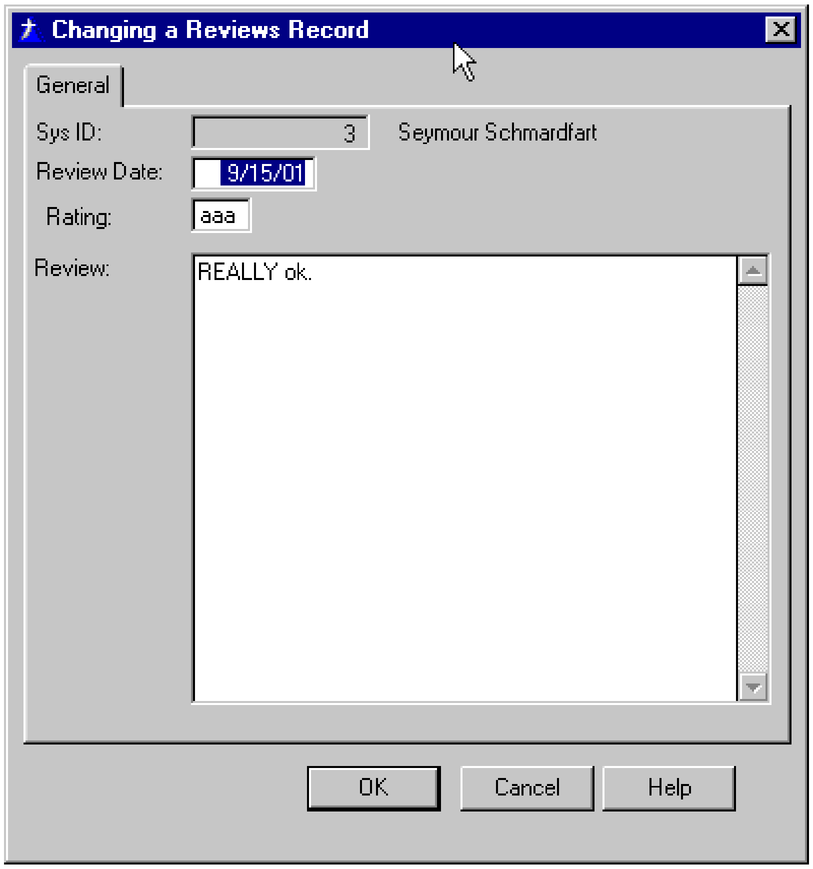
It is easy enough to create a browse of all employee reviews (i.e., without using the range limits that the wizard defaults to) and, using the relationship between EMPLOYEES and REVIEWS, display the employee name. Because related data is projected into the browse's view, the related file's fields can be used just as if they were in the primary file, including concatenating first and last name fields into a single name display. This makes a much friendlier browse:



Name	Review Date	Rating
Jones, Carol	10/17/01	b
Schmardfart, Seymour	9/15/99	a
Schmardfart, Seymour	9/15/00	aa
Schmardfart, Seymour	9/15/01	aaa

Figure 4. Child browse with parent data

I can even utilize this in the update form to display the employee name (Figure 5).



Sys ID: 3 Seymour Schmardfart
Review Date: 9/15/01
Rating: aaa
Review: REALLY ok.

Figure 5. Name display added to SysID

A digression

By the way, I've discovered a marvelous little trick for creating and displaying the name from the lookup into the EMPLOYEES file.

Previously, I would press the "Field Priming on Insert" button and prime FullName to "''" which forces it to be blank when inserting a new record.

After looking up the employee (in SysID, Accepted – see Figure 6), I let the templates handle retrieval of the employee's name for me. If I want a full name display, I concatenate the data after the lookup.

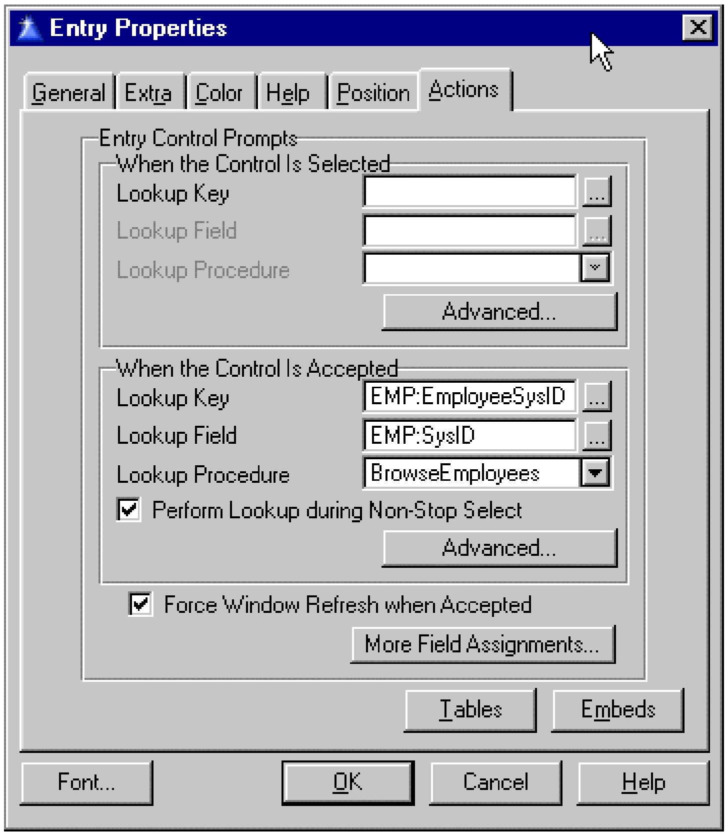


Figure 6. Standard validation

To display the name when editing an existing record, I check whether the form is being called to edit with `ThisWindow.Request = ChangeRecord` and, if so, I manually do the lookup in INIT, after opening the files.

However, I have since found that the following code, after the files are open eliminates the need to prime-on-insert to ensure a blank FullName:

```
EMP:SysID = REV:SysID
Access:Employees.Fetch(EMP:EmployeeSysIDKey)
FullName = Clip(EMP:FirstName) & ' ' & EMP:LastName
```

Because the FETCH method clears the buffer when it cannot find the record, which will happen on an Insert because there is no value in REV:SysID, the two name fields are blank and, therefore, the FullName will also be blank. But, if called to edit an existing record, the FETCH will succeed.

Back to my story

The fact of the matter is that the update form, shown in Figure 5, is pretty clunky looking. It is even clunkier to use.

In fact, it is unusable.

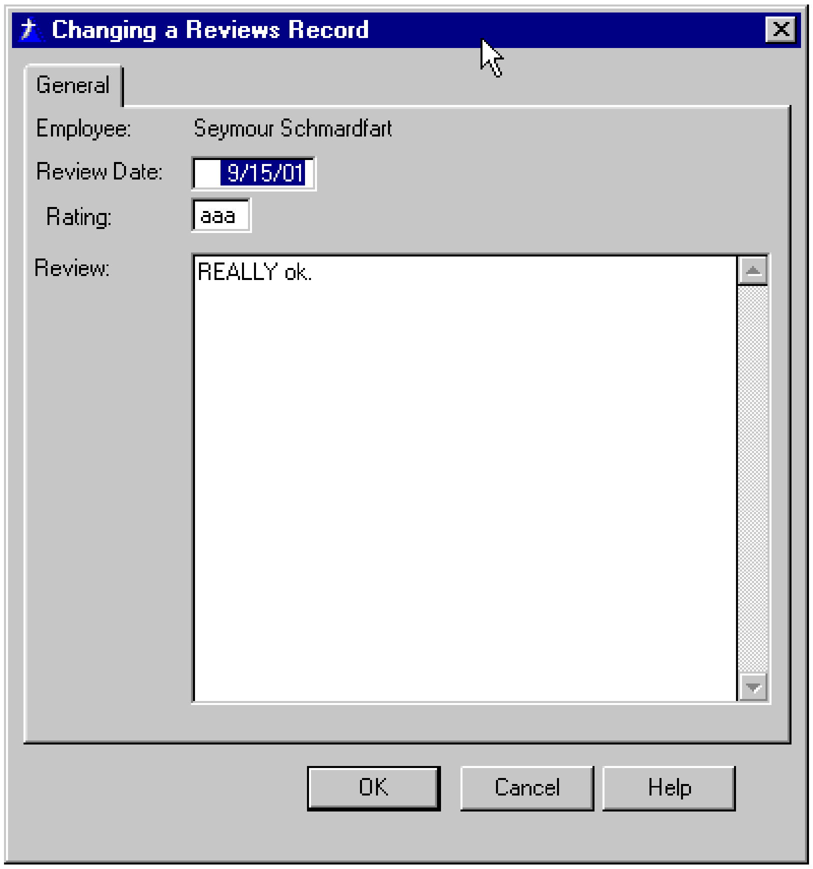
On its face, it looks like I am expecting the user to enter a SysID, not an employee name, not an employee number – either of which I might reasonably expect an end user to know or to have access to – but this utterly mysterious "SysID" thingee.

Furthermore, in the demo app, I've made the SysID field skip and read-only. How could any one ever enter the required datum? What if I'd followed common practice of not displaying it at all?

Even if I made it enterable, its virtue to me, as a designer, is that it is meaningless and this very virtue now plays against me when the user needs to complete it. So, I have to ask the user to enter a field that isn't there, or a field that can't be written to or a field that makes no sense to the user.

Right.

What I want would look more like Figure 7:



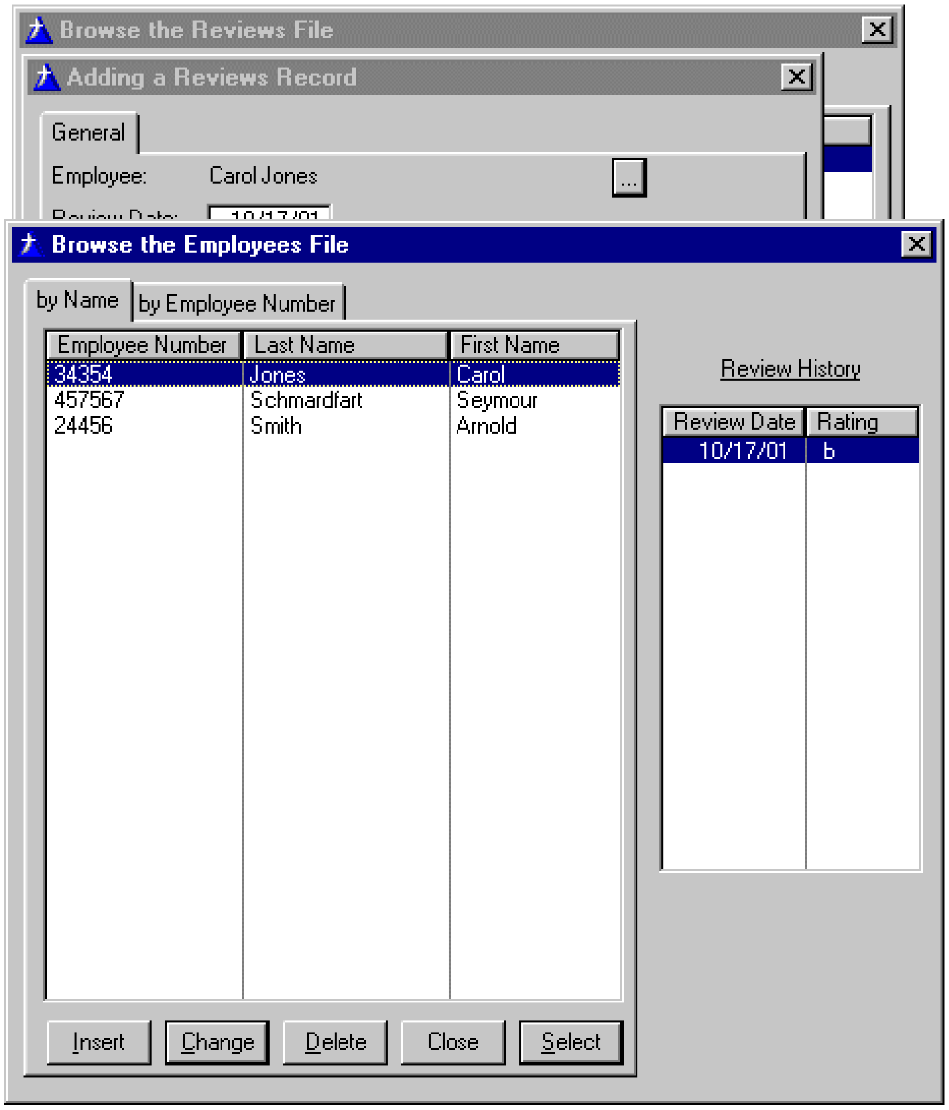
Employee: Seymour Schmardfart
Review Date: 9/15/01
Rating: aaa
Review: REALLY ok.

Figure 7. Form without "SysID"

This is easily achieved by changing the prompt, hiding the SysID field and moving the name field to the left. But I still can't enter a new record.

The conundrum is that I do not want the users touching (sometimes not even seeing) a datum that I *require* them to enter. And the field that the user can see and make sense of doesn't really exist anywhere. Do I sense a consistency problem here?

The FieldLookupButton Control template, of course, resolves all the problems (and even makes the consistency issue moot). This template requires the label of a control with a lookup (as in Figure 6, above). It does not require that the field be visible. So, if I have a lookup on REV:SysID, even though I've hidden that control, the lookup button will still work. Figure 8 shows what happens when pressing the lookup button when the SysID field is hidden.



Employee Number	Last Name	First Name
34354	Jones	Carol
457567	Schmardfart	Seymour
24456	Smith	Arnold

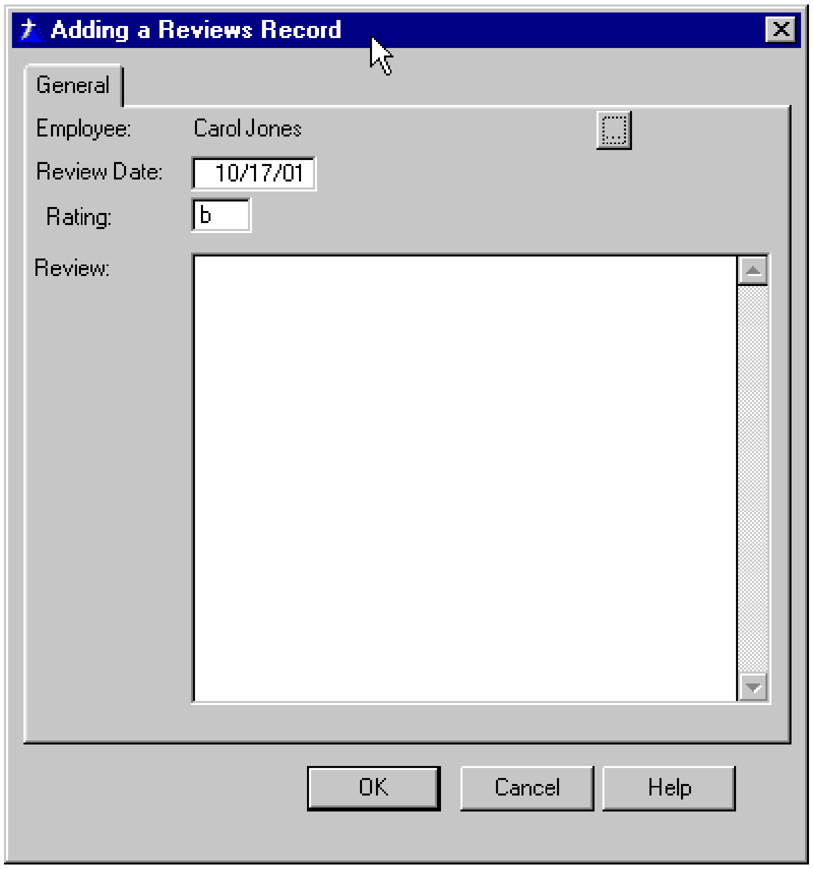
Review Date	Rating
10/17/01	b

Figure 8. Pressing "..." with SysID field hidden

Voilà. I only need to embed this code:

```
FullName = Clip(EMP:FirstName) & ' ' & EMP:LastName
```

in the button's Accepted embed to ensure that the form displays as I wish:



Employee: Carol Jones
Review Date: 10/17/01
Rating: b
Review:

Figure 9. A more user-friendly form

It is also possible to trick the users by having them look up the last name (there is a procedure in the demo app constructed this way). In this case, the lookup button would refer to the EMP:LastName field and the "Additional Assignments" button would assign:

```
REV:SysID = EMP:SysID
```

Either method will work. It's a matter of taste.

Summary

Normalization is a two edged sword. It makes relations, relational views and lookups much easier. It allows me to provide a lot of eye candy with a few mouse clicks. It allows me to give a hierarchical file structure a flat (or flatter) appearance to the user (by simply adding update buttons to a child browse.

On the other hand, normalization makes entering and displaying records in child files much more of a challenge for the typical user. With a judicious use of the tools at your disposal, however, you can lead the user down the primrose path to do what the file structure requires without creating any great intellectual challenge ... at least, not more than is absolutely necessary.

Download the source

Notes

[1] Seymour Schmardfart, according to Jerome Singer, is the brunt of every weird example in Social Psychology. He has served me faithfully through the years in a role expanding beyond the narrow confines of mere inter-personal behavior studies.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reader Comments

Add a comment

Clarion MAGAZINE

Search
Go

Home

COL Archives

Advertise

Reader Comments

Call for Articles

Information
Log In
Membership/ Subscriptions
FAQ
Privacy Policy

Contact Us

Downloads
PDFs
Freebies

Open Source

Site Index

Designing Crosstab Reports In Clarion (Part 1)

by David Potter

Published 2001-09-11

One of the more common and useful reports your clients may ask you to create is the crosstab report, which can be defined as any report that summarizes data in a two dimensional grid, such as breaking down sales data into monthly or quarterly sales. Other report writing tools often supply a wizard to guide the user through the process of creating this type of report. Clarion, however leaves the programmer to his or her own devices to create this standard summary. This often leads the Clarion programmer to rely on third party report writers (such as Crystal Reports) or third party templates to create crosstab reports.

Although it might not be easy for the novice programmer to see, it is actually very simple to create a crosstab report using the standard report template supplied with Clarion. You just need to know a few tricks.

Data declarations

To illustrate my point I will create a crosstab report summarizing some hypothetical sales data into quarterly results. To make it a little more interesting I'll break the results down by sales location as well. First I'll create a simple `Sales` table to hold the data, consisting of the following structure.

```
SalesRecId          LONG
SaleDate            LONG
SaleLocation        STRING(20)
SalesTotal          DECIMAL(7,2)
```

As I don't have any actual sales figures I'll use the Data Modeller to populate this table with some random data.

The report I want to produce will have this general layout:

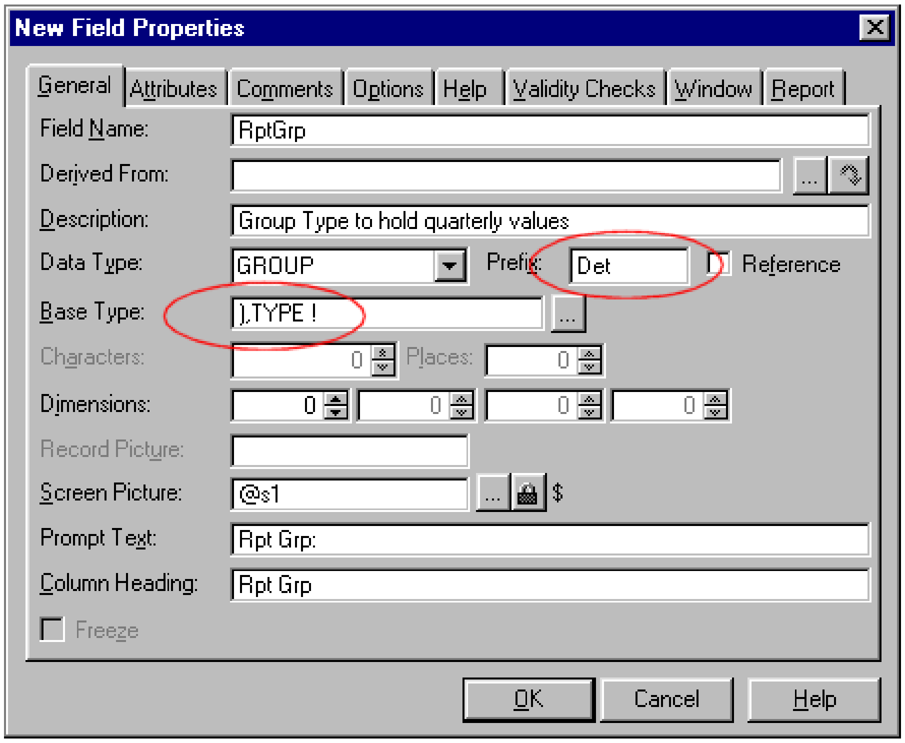
```
Sales
Location    Year   Qtr1   Qtr2   Qtr3   Qtr4  Total
XXXXXX      ####   #      #      #      #     #
```

A good place to start is a report procedure with the Sales table as the primary table. I also need to declare a data structure to hold and print the accumulated data that I get from the sales table. Depending on the particular implementation, I can use either a Group or a Queue structure to accomplish this. I will start off by using a group structure. Later on I will show how to use a queue to make the report more versatile.

Declaring the group as a type can save a lot of work in the future: I can use the type to declare similar groups to accumulate totals or subtotals as needed. For demonstration purposes I'll just declare a GROUP called RptGroup type here. In the data formatter, I just add the code

```
),TYPE !
```

in the Base Type field of the data entry screen, as shown in Figure 1.



Field Name:	RptGrp
Derived From:	
Description:	Group Type to hold quarterly values
Data Type:	GROUP
Prefix:	Det
Reference	☐
Base Type:	),TYPE !
Characters:	0
Places:	0
Dimensions:	0 0 0 0
Record Picture:	
Screen Picture:	@s1
Prompt Text:	Rpt Grp:
Column Heading:	Rpt Grp
Freeze	☐

Figure 1. Creating a TYPEd GROUP

Normally a Base Type would be another group, and the generated code would look something like this:

```
RptGroup    GROUP(ParentGroup)
            END
```

When I add the code in Figure 1, the compiler discards everything after the ! character. Instead of a base type, I'm just adding a closing parenthesis and a TYPE attribute.

```
RptGroup    GROUP(),TYPE !)
            END
```

You can make this declaration either at the global or the procedural level. As I do not plan on using the group type anywhere else in the program I'll just declare it at the procedural level. I need five fields in the type definition: Qtr1Sales, Qtr2Sales, Qtr3Sales, Qtr4sales will accumulate sales for each quarter, and TotalSales will hold the yearly total. Notice that I have included the prefix Det in the prefix definition of the type declaration. This comes after the comment I entered into the Base Type, and will not be seen by the compiler. The data declaration in code will look like this:

```
RptGrp       GROUP(),TYPE !), PRE(Det)
QtrnSales      LONG
...
TotalSales     LONG
             END
```

I can later use the prefix to my advantage when populating the fields in the report formatter. For now I'll declare a group of RptGrp type to use to populate the report detail. First I declare a group called DetailGrp in the data formatter. I declare the Data Type as GROUP and place the entry RptGrp in the Base Type field. Here I will also declare the Prefix as Det. The compiler will see this prefix and use it as the actual prefix for the DetailGrp. The code declaring the DetailGrp will look like this:

```
DetailGrp    GROUP(RptGrp), PRE(Det)
             END
```

Now I need some variables to hold the Location and Year values. These variables will be used to monitor printing the report detail as well. In order to keep things clean and be consistent I will put these variables in a group as well. I will only need one instance of this group so I won't declare it as a TYPE. The group definition will look as follows:

```
SavGrp        Group(),PRE(Sav)
Location       STRING(20)
Year           SHORT
              END
```

Populating the report

Next I can populate the main detail of my report using the data structures I declared here. For the sales location I will use the Sav:Location field, and Sav:Year field for the sales year. The Det:Qtr*n*Sales fields will be used to fill in the rest of the report.

Here is where placing the prefix Det in the type definition comes in handy. Because the DetailGroup is declared as a RptGroup type it does not have any fields in the definition. So I cannot populate these fields using the local data file schematic. However since I put the same prefix in the RptGroup definition, I can select those fields from local data to place them in the report formatter. The fields will be given the `Det` prefix as they are added. This way I can fool the compiler to see these fields as coming from the Detail group.

I want to control the printing of the detail myself, so it is necessary to suppress the template from generating a print statement. If you do not know how to do this, Clarion's help can probably explain it better than I can.

1. Select Contents from the Help menu

2. select the button **How Do I ... ?**.

3. Select Reports

4. Go to the FAQ section in the Clarion online help, find the article *How to suppress printing a Detail Band until explicitly called to print*, and follow the directions.

Defining a key

Steve Parker, in his Clarion Magazine article *Completely Dynamic Report Orders and Breaks*, defined a Clarion report as nothing more than "a Process with a `Print()` statement and access to the Report Formatter." In keeping with this concept, I can use the process loop to accumulate the totals in my group structure. In order to accomplish this efficiently I need to process the records from my Sales table in the correct order, so I declare a key as follows and use it as the primary key for the report.

```
LocationKey KEY(SAL:SaleLocation,SAL:SaleDate)↵
              ,DUP,OPT
```

Summarizing the data

Now I can use the `ThisReport.TakeRecord` embed to accumulate the data. First I'll need to calculate the proper quarter from the `SaleDate`. It is easy to derive a new method in the `ThisReport` process class to do this.

1. Select Report Properties
2. Go to the Classes tab
3. Check the Derive? checkbox
4. Select the New Class Methods button.
5. Press the Insert button
6. Enter `GetQuarterFromDate` in the New Method Name
7. Enter `(LONG SalesDate),BYTE` in the New Class Prototype entry field.
8. Press the Code Embed button to access the correct embed to enter the code for the function.
9. In this case the code consists of one line

   ```
   RETURN INT((MONTH(SalesDate)+2)/3)
   ```

I can now use the new method to sort each record from the table into the proper quarter in my detail group:

```
Det:TotalSales += SAL:SalesTotal
CASE SELF.GetQuarterFromDate(SAL:SaleDate)
OF 1
  Det:Qtr1Sales += SAL:SalesTotal
OF 2
  Det:Qtr2Sales += SAL:SalesTotal
...
END
```

Printing the data

So far I have a report that accumulates all of the data into one record and doesn't print anything. The obvious question is where and when to print the record? From the primary key, I know the records in the file will be processed in order of `Location`, then `Date`. I want to print the summarized data by location and year. By monitoring the records as they are processed I'll know to print the record whenever the `Location` or `Year` changes.

This is where the `SavGrp` comes in. The general plan is to set the `Location` and `Year` fields in the `SavGrp` equal to the values read from the primary file as I process the record. Then after getting the next record, I compare the new values with the saved values. If they are the same I continue accumulating my totals. If the values have changed it is time to:

1. Print the accumulated values as they stand before adding the new record.
2. Clear the `Detail` group to start accumulating the next summary record.
3. Reset my `SavGrp` fields equal to those of the record.

The following code will do just that:

```
IF Sav:Location <> SAL:SaleLocation |
   OR Sav:Year <> YEAR(SAL:SaleDate)
    IF Sav:Location
      PRINT(Rpt:Detail)
    END
    CLEAR(DetailGrp)
    Sav:Location = SAL:SaleLocation
    Sav:Year = YEAR(SAL:SaleDate)
END
```

Before printing the detail I make sure there is a value in `Sav:Location`. If there is not, I know it is the first pass through the process loop, and therefore there is nothing to print. It is important that all the variables are cleared beforehand to make sure they don't contain any stray values to throw the logic off. I usually do this in the `ThisReport.OpenReport` embed. It can be done in the `ThisWindow.Init` embed, but if the `PauseButton` template is used this will not clear the variables if the report is rerun.

There is one final gotcha that I must take care of before I am finished. When the last record is processed the loop is exited and the final `Detail` is left in memory and will not be printed. To correct this I must place another `Print` statement outside of the process loop. The `ThisWindow.AskPreview` embed is the best place to do this, before the generated code.

Report totals

Now that I have the detail printing under control I can declare report breaks, headers and footers and have the report engine calculate the totals for me as in any other Clarion report. This works great as long as there are no calculated fields in the detail footer.

Yes, folks, there is one major weakness in the Clarion report template! There is no easy way to calculate a value in the group footer and get the correct answer. At one point I thought I had found a clever way by using the `Using` entry field for the variable in the group footer, only to find that the last record before the break was not included in the equation, giving erroneous results. After much experimentation I found the best way was to avoid the use of group footers when they included any calculations on total fields. This is not as daunting as it sounds, particularly when using Group type definitions. (Remember, a report group and a `GROUP` data structure are two completely different things!)

In order to illustrate this I will make a few minor changes in the report design. Let's say that the customer wants to see the percentage of sales for each quarter and wants it broken down by Year and location. The first thing I must do is add four fields to my `RptGrp` type to hold the result of these calculations, and place them in the detail band. I will name these `Qtr`*`n`*`Pct`. I could calculate these values right after accumulating my totals in the `TakeRecord` embed. However it is more efficient to do the calculations right before printing the detail.

Now I need a way to accumulate the values for the grand totals and subtotals. To accomplish this I need only to declare two more groups of the type `RptGrp`. I'll call them `SubGrp` and `TotalGrp` and give them the prefixes `Sub` and `Tot` respectively. I declare two new report details and set them up not to print as with the primary detail. I can call these `SubDtl` and `TotDtl`. By changing the prefix in the `RptGrp` I can populate the variables much as I did for the report detail. The most efficient place to accumulate the totals would be right after printing the detail group. It is also necessary to make minor changes in order to print the `SubTotals` and accumulate the grand totals. The code in the `TakeRecord` embed point at this point looks similar to this:

```
IF Sav:Location <> SAL:SaleLocation |
 OR Sav:Year <> YEAR(SAL:SaleDate)
  IF Sav:Location
    !Calculate percentages
    Det:QtrnPct = |
      (Det:QtrnSales/Det:TotalSales) * 100
    ...
    PRINT(Rpt:Detail)
    !Accumulate Sub Totals
    Sub:QtrnSales += Det:QtrnSales
    ...
    IF Sav:Location <> SAL:SaleLocation
      Sub:QtrnPct = |
        (Sub:QtrnSales/ Sub:TotalSales) * 100
      ...
      PRINT(RPT:SubDtl)
      !Accumulate  grand totals
      Tot:QtrnSales += Sub:QtrnSales
      ...
      CLEAR(SubGrp)
    END
  END
END
...
```

The last thing that needs to be done is to finish printing the final detail and footers after the records have finished processing. I do this in the Ask Preview embed:

```
!Calculate percentages
Det:QtrnPct = |
  (Det:QtrnSales/Det:TotalSales) * 100
...
PRINT(Rpt:Detail)
!Accumulate Sub Totals
Sub:QtrnSales += Det:QtrnSales
...
Sub:QtrnPct = |
  (Sub:QtrnSales/ Sub:TotalSales) * 100
...
PRINT(RPT:SubDtl)
!Accumulate  grand totals
Tot:QtrnSales += Sub:QtrnSales
...
Sub:QtrnPct = |
  (Sub:QtrnSales/ Sub:TotalSales) * 100
...
PRINT(RPT:Totals)
```

Summary

That's pretty much all there is to creating a crosstab report. Essentially all I have done is take advantage of the process loop provided by the report template to accumulate and summarize the data. By monitoring when the data changes I can determine when to print the data in the format I want. This is the same logic that would be used to hand code a report with headers and footers. However by using the report template I can avoid a lot of the work. I also gain easy access to some of the nice frills the template provides, such as the report previewer and progress window. In Part 2 I will show how to use a queue to add further versatility to this technique.

Download the source (Clarion 5.5)

David Potter start his professional career as Chief Assayer at a gold mine. While buried under a mountain of lab results, he was given a copy of Clarion Personal Developer to help track the assay and metallurgical data. David found Clarion easy to use and soon upgraded to Clarion Professional Developer version 2. He has been developing in Clarion ever since. When the gold ran out David decided to go back to school and become a full time programmer. He is currently working in the MIS department of a plastic manufacturing plant in Washington State, and uses Clarion to develop applications on Oracle and AS400/DB2 databases.

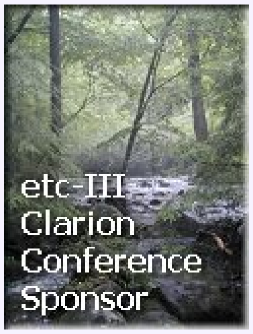
Reader Comments

Add a comment

Excellent article. Very clever extension of resources.
I'm not sure that all of your fooling around with the...
I'm not sure all the "fooling around" I did with the totals...

Reborn Free CLARION online
published by CoveComm Inc.

Clarion MAGAZINE



Finding Unused Variables

by Andrew Guidroz II

Published 2001-09-12

I had this brilliant idea come to me in the middle of coding a procedure in one of my customers' apps. And, in the middle of coding that idea, I realized that it just wasn't going to work. I attacked the problem from a totally different angle, the light bulb *really* went off this time, and the completed app met the client's approval. But this article isn't about my great idea; it's about how, six months later, modifications to that same procedure revealed to me the fallout from that failed first attempt.

It is pretty easy to trim out bad code or code you are replacing. Often times, you forget that other part of the trimming: getting rid of the variables you added that have since been abandoned.

Luckily, the Clarion compiler has a wonderful pragma called Variable Declared But Never Used: `wdnu`. There is a description of it in the Programmer's Guide:

> `#pragma warn(wdnu => on | off | err) cp`
>
> **Variable declared but never used**. When on or err, the compiler checks whether a local variable has been declared but never used in the function. The default setting is on.

The `wdnu` pragma is really easy to use. Just go to the Project Settings for your app, click Properties, and go to the Defines tab. There should already be some defines in there. For example, these are the defines from the ClubMgr app in the examples directory:

```
_ABCDllMode_=>0
_ABCLinkMode_=>1
```

Just add another line like this:

```
wdnu
_ABCDllMode_=>0
_ABCLinkMode_=>1
```

and you’re off to the races.

In order to really get the most out of using wdnu, the first thing you need to do is repopulate your application with only one procedure per module. The compiler cannot tell that you haven’t used a variable until it reaches the end of a procedure. It is easier to figure out which procedure is the offending one if you can attack them one at a time.

Add the wdnu pragma to the ClubMgr app, redistributing the procedures at one per module, and compile. The compiler will give you a host of wonderful warnings. Here are some of the warnings from the UpdateLetters procedure:

```
(clubm001.clw 182,1) Warning: Label not used: FILESOPENED
(clubm001.clw 182,1) Warning: Label not used: LOCALREQUEST
(clubm001.clw 182,1) Warning: Label not used: FIELDCOLORQUEUE
(clubm001.clw 182,1) Warning: Label not used: CURRENTTAB
(clubm001.clw 182,1) Warning: Label not used: RECORDCHANGED
(clubm001.clw 182,1) Warning: Label not used: CURCTRLFEQ
```

The FilesOpened warning is a classic. That was a flag that the legacy templates used to track if the files used by the procedure had been successfully opened. In ABC, this flag has been moved into the FileManager class so it is no longer needed. You can safely delete it.

LocalRequest is another flag that has gone the way of the dinosaur. The new ABC classes use the FileManager’s Self.Request property instead.

CurrentTab seems to have no references any more either.

RecordChanged has been replaced by a host of ABC classes that compare record buffers, so once again it can be deleted from every procedure in every ABC app you have.

The last two variables that have warnings associated with them are FieldColorQueue and CurCtrlFEQ. These variables are used to support field coloring during field validation. I don’t use field coloring so these variable are never used in my apps. There aren’t accessible from the Data button on the procedure properties window either and will be regenerated by the templates. You just have to live with the warning on those two.

Deleting those four variables won’t make your EXE any more efficient if you are compiling in release mode. If you do not check off "Build Release System" on the Global Options window of the project properties (i.e. build in debug mode), you will use additional memory for those variables.

A little further along in the compile, in the fifth module and the BrowseTransactions procedure, you’ll see an unused ABC declaration:

```
(clubm005.clw 206,1) Warning: Label not used: BRW1::SORT0:STEPCLASS
```

This warning appears because the templates still generate a locator class even though the browse is flagged as having no locator. That one is safe to ignore also.

In module 16, the HelpAbout Window has plenty of Legacy variables that can be done away with.

In module 20, `PrintCommitteeMembers` has a variable called `SkipDetails`. This one is always generated by the templates, and is used for report detail line filters. It is useful and safe to leave (not that you have any choice).

In module 24, `PrintLabels` has a few local variables that are no longer being used by the application. This procedure is a good example of what you will likely find in your own evolving applications.

Even the ABC classes themselves suffer from variables being abandoned. You will see a warning for the file abdrops.clw, which contains the Drop List classes. There is a variable there called `QM` that is no longer being used. You may also see warning about not an unused `ProgessMgr` class in those reports and processes that do not have a selected index or key.

Not all warnings are valid. For instance, Consider the following `QUEUE` definition:

```
MyQueue QUEUE,PRE(MyQ)
Item1     LONG
Item2     LONG
        END
```

Clarion supports two different ways of referencing the queue elements. One is with the prefix:

```
MyQ:Item1
```

and the other is with the fully qualified dot syntax name:

```
MyQueue.Item1
```

Queue or group items declared with a prefix will generate `wdnu` warnings even if you have referenced to them by their dot syntax name. I hope the good folks at SoftVelocity get this one fixed.

Third party templates may generate lots of these types of warnings. As well, many third party developers provide local variables which you can use to tailor the behavior of the products, and not all of these variables will always be used.

I hope you find this information as useful in your apps as it has been in mine. Although the compiler won’t include unreferenced variables in the final EXE/DLL (provided debug is off), removing old unused code makes apps much easier to maintain.

Andrew Guidroz II, when he isn't handfeeding the tufted titmouse, 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

Very good tip! I learned something new Russ
Good article! Never knew this existed. App_Ref has a...

Self-Upgrading Applications

by Jeff Berlinghoff

Published 2001-09-13

In previous jobs I have been responsible for network administration as well as programming. I often pursued programmatic solutions to network administration, especially as the size and number of networks grew, and the number of support positions never was able to keep up. Part of this work involved updating networked applications, where the program files were stored on workstations. Eventually I developed a way for my Clarion applications to upgrade themselves over the network, and in this article I'll show how you can do the same. But first, a little background.

Very early on, each of the workstations I supported only had one hard drive, and since I always knew the applications were installed on `C:` I could modify the network login script to check the EXE date at login to determine if it was time to automatically run an upgrade program.

With each new network operating system, login script languages changed. When I started in computer networking, we were using Novell Netware 286 version 2.10, and it had pretty decent login script commands – of course that was in DOS and Windows 3.1 days, and the requirements were pretty low. When we finally switched to MS Windows NT, the only login script processor was a DOS batch file, which was pretty limiting compared to Novell's scripting language. Finally we found other utilities to supplement the lack of a built-in feature rich login script processor. KixTart was an early NT script processor, and later VBScript started becoming more popular.

In time, the number of applications grew as did the number of hard drives in some computers, and there was no absolutely predictable place to look for the application and determine when to force an upgrade. In addition, since there was (and is) no guarantee to which network operating system or login script process may be in use at a client's site, it was a challenge, as the developer, to recommend how the administrator might modify login scripts to support an automatic application upgrade.

The easy way out was to place the application in a shared directory and require each workstation to simply have a shortcut to the application. This, however, meant the app took unnecessary network bandwidth when loading. With large cheap hard drives everywhere, there was no sense in clogging up the network and increasing application load time. Still, when a new release of an in-house application was ready, the last thing I wanted to do was to touch every computer that had the program installed. Some people rarely used the programs, while others would use them frequently; as a result I'd often concentrate on updating the main users' workstations. Then, often much later, someone else would try running an old version of the program, and I'd be interrupted and have to upgrade that workstation.

Part of the problem, of course, was not really knowing who the main users of an application were... people come, people go, some get reassigned or only run an app when the main users are in a data entry crunch or unavailable.

It also wasn't ideal to dump a lot of extra work on the network administrators who supported our applications. They probably have better things to do than assist in application upgrades, or modifying their login scripts. Some administrators have end-user sticks that they like to beat on boxes when "they" do stupid things. Sayings like "the network would be fine if it wasn't for the users" abound. End-users don't understand the restrictions that may be placed upon them, usually for their own good – to keep them from "doing *that* again!"

My ideal solution is one that does not require network administrator rights or abilities (after all, not all programmers are network administrators). Actually, as a previous network administrator, I'd be cautious of letting someone who was not an administrator on my network tamper with the login scripts. These were the kinds of problems that were the driving force behind my concept of making an application that could know when it was time to be upgraded.

What version am I?

My approach has two goals: 1) keep track of which workstations need a software upgrade, and 2) make the application run the setup program. In order to do the former the currently executing program needs to know which version it is, and have access to a table containing the version number of the most recent release of the program. This check should happen when the program starts; if the program is not current then it should run the upgrade program and terminate itself.

The table that holds the version information has an additional role. Sometimes I'll release one or two updates to a program in a week's time. If a user has a dialup connection to the network, this can mean a lot of time spent downloading the update, particularly if it's a full install. Now some releases will require a full install, others just a small patch. And someone who is upgrading from the previous version may only need the patch, while someone else upgrading from a very old version will almost certainly need a full install. In order to accommodate different upgrade paths, I use a second table that stores different upgrade paths and install program names.

Tables

There are three tables required to support this concept. When I originally developed this auto-upgrading concept, I was designing for MSSQL, but the concept is the same regardless of driver. To reach the maximum audience, I've coded it here for TPS files, but you can easily adapt this same table structure to your environment.

```
Applications     FILE,DRIVER('TOPSPEED'),|
                      NAME('Applications'),PRE(APP),|
                      CREATE,BINDABLE,THREAD
ID_Key              KEY(APP:ID),NOCASE,OPT,PRIMARY
Name_Key            KEY(APP:Name),NOCASE,OPT
Record              RECORD,PRE()
ID                     LONG
Name                   CSTRING(31)
CurrentVersion         CSTRING(11)
                    END
                 END
AppVersions      FILE,DRIVER('TOPSPEED'),|
                      NAME('AppVersions'),PRE(Ver),|
                      CREATE,BINDABLE,THREAD
ID_Key              KEY(Ver:ID),NOCASE,OPT,PRIMARY
AppID_Key           KEY(Ver:AppID,Ver:OldVersion,|
                      Ver:NewVersion),NOCASE,OPT
Record              RECORD,PRE()
ID                     LONG
AppID                  LONG
OldVersion             CSTRING(11)
NewVersion             CSTRING(11)
UpgradePath            CSTRING(256)
                    END
                 END
```

In order to know which old versions may still be out there, I need a table to hold the workstation names, the application version last used, when it last ran, and who the user was. I call this table AppVersionsInUse.

```
AppVersionsInUse FILE,DRIVER('TOPSPEED'),|
                      NAME('AppVersionsInUse'),|
                      PRE(AIU),CREATE,BINDABLE,THREAD
ID_Key              KEY(AIU:ID),NOCASE,|
                      OPT,PRIMARY
Workstation_Key     KEY(AIU:AppID,AIU:WorkStation),|
                      NOCASE,OPT
AppID_AppVer_Key    KEY(AIU:AppID,AIU:AppVer,|
                      AIU:WorkStation),DUP,NOCASE,OPT
```

```
Record                RECORD,PRE()
ID                       LONG
AppID                    LONG
AppVer                   CSTRING(11)
WorkStation              CSTRING(17)
UserName                 CSTRING(17)
LastRun_Date             LONG
                      END
                   END
```

One important note about these files: if you use them as TPS, I recommend you make the names a variable to hold the complete file name including path to these tables. All versions of the program will have to be able to locate these tables, so you will need a little code to initialize the paths. If you are in a multi-user environment with the data tables in a central location and the applications running from the workstation, chances are that you are already doing this.

API calls

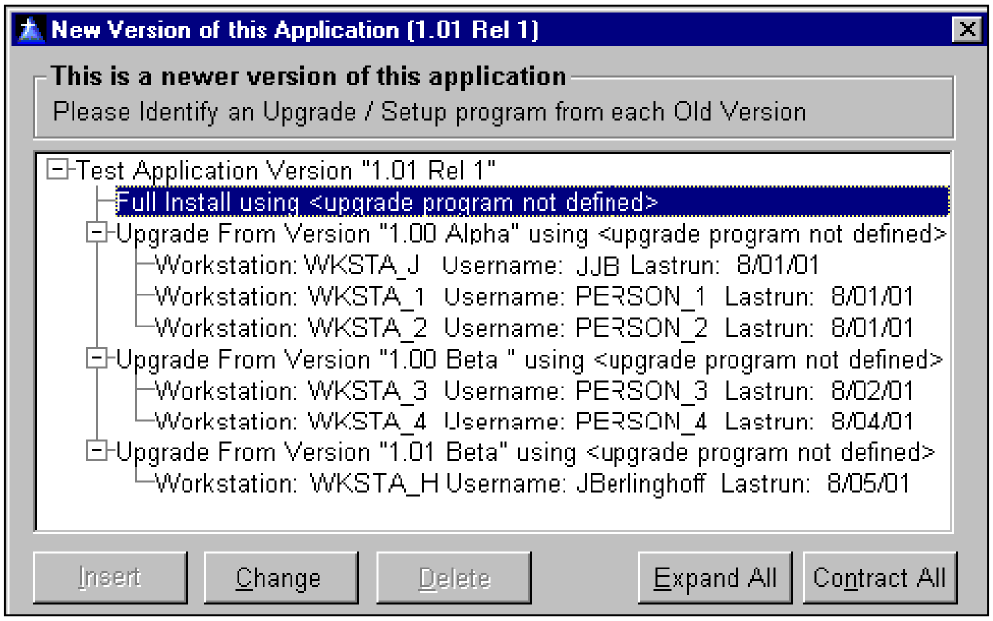
There are two Windows API calls you must prototype: one to get the User ID, and another to get the Workstation ID:

```
MODULE('WINAPI')
  WNetGetUser(*CSTRING,*USHORT),|
    USHORT,RAW,PASCAL
  GetComputerName(*CSTRING,*ULONG)|
    ,signed,raw,pascal,NAME('GetComputerNameA')
END
```

Putting it all together

When the program starts it passes the Application Name and Version to the AppCurrent function. If the application is not the most current version, the program prompts the user to run the upgrade, and terminates. If the application is a more recent version than any listed in the Applications table, AppCurrent creates new entries in Applications and AppVersions. You'll then have to fill in the paths to the upgrade programs so that all future uses will trigger an upgrade.

I am making the assumption that the first person to actually run the new version will be responsible for copying the upgrade programs to a common location for all to run. Figure 1 shows an example of the window that appears when a new version of the test application has been found. This window also shows three previous versions in use that may need paths to upgrade programs.

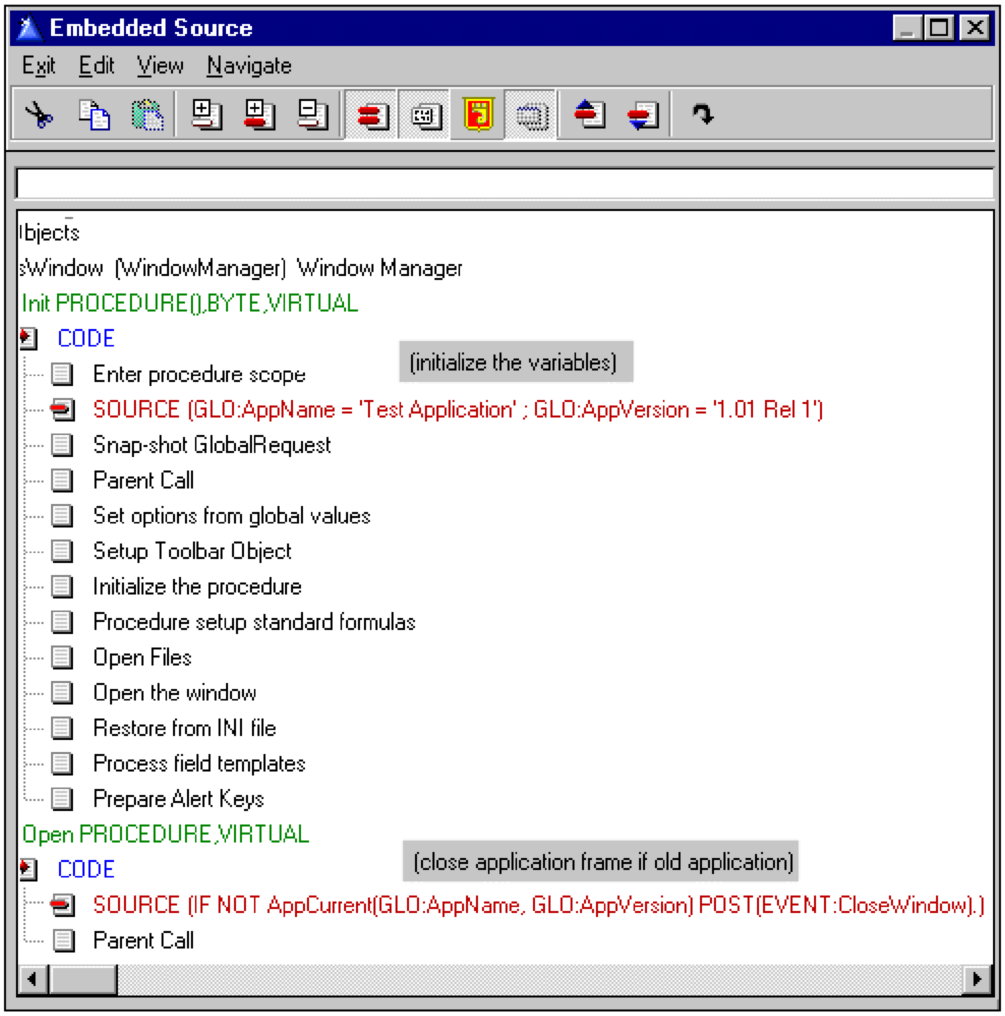


```
New Version of this Application (1.01 Rel 1)

This is a newer version of this application
Please Identify an Upgrade / Setup program from each Old Version

⊟-Test Application Version "1.01 Rel 1"
   ├Full Install using <upgrade program not defined>
   ⊟-Upgrade From Version "1.00 Alpha" using <upgrade program not defined>
   │  ├Workstation: WKSTA_J   Username: JJB Lastrun: 8/01/01
   │  ├Workstation: WKSTA_1  Username: PERSON_1  Lastrun: 8/01/01
   │  └Workstation: WKSTA_2  Username: PERSON_2  Lastrun: 8/01/01
   ⊟-Upgrade From Version "1.00 Beta " using <upgrade program not defined>
   │  ├Workstation: WKSTA_3  Username: PERSON_3  Lastrun: 8/02/01
   │  └Workstation: WKSTA_4  Username: PERSON_4  Lastrun: 8/04/01
   ⊟-Upgrade From Version "1.01 Beta" using <upgrade program not defined>
      └Workstation: WKSTA_H Username: JBerlinghoff  Lastrun: 8/05/01
```

Figure 1. New Version Identification Window

The sample application and dictionary are available for download below. The embeds in the Main procedure serve as guidelines to calling the AppCurrent function, which in turn calls AppVersionsTree and UpdateAppVersions. The relationships defined in the dictionary are mainly there to support the tree template used in AppVersionsTree.

In order to make this concept work you need to define current application name and version variables. These variables should be set and then passed as parameters to the AppCurrent function. In the sample application I initialize the variables in the main window INIT method. Since I want the application to close when an old version is run, I close the window in the OPEN method when the call to AppCurrent fails.



Embedded Source

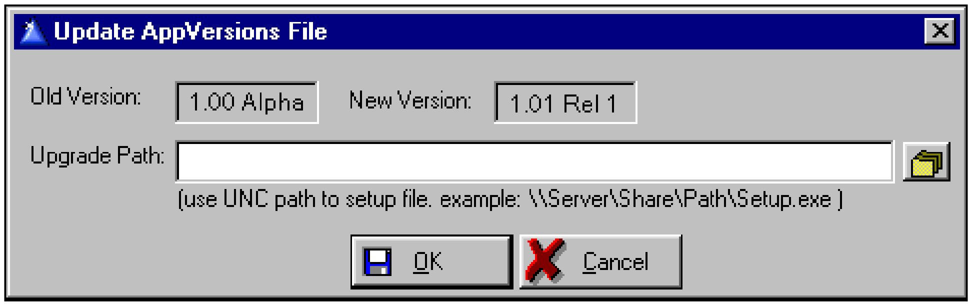
Exit Edit View Navigate

```
Ibjects
sWindow  (WindowManager)  Window Manager
Init PROCEDURE(),BYTE,VIRTUAL
  CODE                                        (initialize the variables)
    Enter procedure scope
    SOURCE (GLO:AppName = 'Test Application' ; GLO:AppVersion = '1.01 Rel 1')
    Snap-shot GlobalRequest
    Parent Call
    Set options from global values
    Setup Toolbar Object
    Initialize the procedure
    Procedure setup standard formulas
    Open Files
    Open the window
    Restore from INI file
    Process field templates
    Prepare Alert Keys
Open PROCEDURE,VIRTUAL
  CODE                                        (close application frame if old application)
    SOURCE (IF NOT AppCurrent(GLO:AppName, GLO:AppVersion) POST(EVENT:CloseWindow).)
    Parent Call
```

Figure 2. Application Frame Embedded Code

If a newer version has already been defined, the user is prompted to upgrade prior to closing the current application. When the new version is run the first time, the AppVersionsTree procedure runs (Figure 1 above). All workstations that have been logged by the AppCurrent procedure have version information associated with them, and AppVersionsTree groups all known workstations by the last known version they ran.

The *only* upgrade path that must be defined is the full install path. If you choose to ship separate upgrade applications from older versions they can also be defined. If there is not an upgrade defined from the workstation's version to the newest version then the full install will run.



Update AppVersions File

Old Version: 1.00 Alpha   New Version: 1.01 Rel 1

Upgrade Path:

(use UNC path to setup file. example: \\Server\Share\Path\Setup.exe)

Figure 3. Defining an Upgrade Path

The most important thing to note about defining the path to an installation or upgrade program is the note about using a UNC path. Do *not* specify a drive letter in the path unless you can absolutely guarantee that *all* the end users will have that drive letter mapped to the same UNC path. If you don't use a UNC path, the application will not be able to be upgraded on some workstations, and since you're closing the old application when it runs the user will be dead in the water. The AppCurrent procedure will report an invalid path to an upgrade program, but this error can be avoided by entering the paths correctly.

When someone has indeed incorrectly entered a path, or if someone moved or deleted an installation program, there must be a way for the end-user to be able to correct this path. The sample program has a File menu option called "Application Upgrade Paths." This calls AppVersionsTree so the paths can be edited. I don't recommend making this menu option available to everybody since some people will go in there and muck about. Trust me, I know all about end users – they can't be trusted. I recommend hiding or disabling this menu option to people who aren't application supervisors or who haven't shown themselves responsible.

Summary

This self-upgrading approach has made it easier for me to keep my applications current on end users workstations while removing the need for network login script modifications. I can be certain that all the users in any given location are running the same version and will have seamless upgrades as new versions become available. Hopefully you and your clients can benefit from this too. I'm certain there will be some interesting functions that could grow out of this self-upgrading concept.

Download the source

Jeff Berlinghoff lives in Tallahassee, Florida and is a programmer for WindowBook Inc. Jeff has been writing in Clarion since 1993, starting with CPD 3.0 and then developing with each of the Windows versions, with an emphasis on MSSQL over the past three years. Jeff enjoys spending his spare time with his wife Donna, and their three children Andrew, Zachary and Jenna.

Reader Comments

Add a comment

Reborn Free CLARION online published by CoveComm Inc.

Clarion MAGAZINE

Clarion News

Win $50 From Gitano Software

Sign up for Gitano's mailing list, and you could win $50.
Posted Friday, September 28, 2001

OE Message Converter Demo Updated

Sterling Data has a new demo showing the improved ExpressFlash templates. Message conversion is faster, as is parsing of some types of messages. The user interface has also been updated.
Posted Wednesday, September 26, 2001

---

xDataBackup Manager v1.0

New from SealSoft is xDataBackup Manager, a library with extension template which lets end users back up files at runtime. All data files are packed into ZIP file which is 100% compatible with PKZIP® 1.10, 2.04g and 2.x archives. xDataBackup Manager uses Linder Software's data compression library. Features include: automatic backup of data at start/end end of program; "smart" backup (if data has changed); data change indicator on program toolbar; flexible archive file name setup. Demo available now; install available soon.
Posted Tuesday, September 25, 2001

ExpressFlash OE Integration

Sterling Data's ExpressFlash lets you import messages from

Outlook Express into your app's database. You can use ExpressFlash for many tasks, including workflow management and automated order email processing. For developers who like to dive in and work at the source code level there is an option to convert Outlook Express mailboxes to raw text data which you can then parse out as you want. Now in final beta, ExpressFlash is $195. Compatible with Clarion 5.5 ABC and OE 5/6. Demo available.
Posted Friday, September 21, 2001

---

xDigitalClock v1.1

New in SeakSoft's xDigitalClock v1.1: ability to set time offset to handle different time zones; set resize strategy; compile in five digital sets. A new install and demo are available.
Posted Friday, September 21, 2001

PD Browse Button, Drop Combo, File Drop Lookups Updated

New C55 installs for ProDomus Better Lookups are now available for download. These support ABC and Legacy apps with and without ClarioNET.
Posted Friday, September 21, 2001

G-Sec Special Extended

G-Sec introductory price has been extended until Monday, September 17th, 2001.
Posted Monday, September 17, 2001

New RUpdate Template Demo

RUpdate is a Clarion template that allows a developer to implement the following measures into Applications created with the ABC (and soon to be, Legacy) template chain: let the end user to check the Web for application version status; provide for download the new version file(s); run the version update file(s). RUpdate will allow files to be downloaded up to 4MB per file. All file types except *.exe, *.pdf and *.com files can be downloaded (i.e. all files not registered with IE to run as executable binaries in the browser). This should not pose a problem as most update files are distributed as *.zip files to minimize size and download

time. RUpdate will soon be available as a BETA release.
Posted Monday, September 17, 2001

RInstall Template Updated

The RInstall template set has been updated; changes include corrections to the file limiting during the evaluation period as well as support for C5 ABC and all Legacy template versions.
Posted Monday, September 17, 2001

G-Sec Update

A new G-Sec build and demo are now available for download.
Posted Wednesday, September 12, 2001

Release Candidate Available For Third Party Testing

The Clarion C5.506 Service Release (f) is available for 3rd party compatibility testing. If you are a 3rd party developer and are interested in testing this release candidate with your product please email Bob Brooker directly.
Posted Tuesday, September 11, 2001

ReportDAT! Is Back

A new release of ReportDAT!, version 5.5, is now available. ReportDAT! is a heavy-duty crosstab report template for Clarion that will summarize your data in one or two dimensions. Through a special arrangement with Grebar Systems, Clarion Central has updated ReportDAT! to work with Clarion version 5 and 5.5, both Legacy AND ABC templates! Save $50 through the month of September! Upgrade pricing is $99 (regular price $149), and the full version is $199 (regular price $199). Demo available.
Posted Tuesday, September 11, 2001

RInstall Now Supports Legacy Templates

RInstall has been updated to support Legacy templates as well as ABC. Due to the api calls incorporated, only 32 bit is supported. RInstall (Legacy) have been tested with Clarion versions as far back as CW2.003. The same password purchased from Clarionshop for the previous update is still in use. RInstal will exit

the beta stage ($59.00) and enter Release 1 ($79.00) on 17 September 2001.
Posted Tuesday, September 11, 2001

ProDomus ClarioNET Status

ProDomus has released the following information on ClarioNET compatibility: Translator Plus is fully compatible, with several modifications in the latest release. Changes include code to change the Clarion environment on the user's machine and the modifications of the hook functions to allow translation of message, stop, and halt statements without source code modification. PD Translator is also compatible except that its message function handling and changes to the Clarion user's environment are not quite as easy. PD Lookup's browse button is being tested. This has the same problem as a standard Clarion lookup where if you make an invalid entry and click the lookup button (or another tab), the browse displays and then, when it closes, the app freezes. This appears to be a ClarioNET issue. The drops have not been tested but there are reports that they do work. PD 1-Touch Date Tools have had ClarioNET code added. Fixes to the ClarioNET handling of alerted keys are needed for this to be fully functional. Otherwise, it is close.
Posted Tuesday, September 11, 2001

SealSoft xPictureBrowse v2.0 Released

New in xPictureBrowse v2.0: there are now two classes, xPBrowseClass and xPViewClass, which work together; the template contains two control extensions, one to preview and select graphic files from directory, and the other to display an image control from graphic files from file or TPS blob. You can also set your own caption for the main xPictureBrowse window. New install and demo available. For xPictureBrowse v2.0 the new price is $49. Registered users of the previous version will receive the new version free of charge.
Posted Tuesday, September 11, 2001

PDF-XChange Template Fix/Update Available

An update to PDF-XChange is now available; this release provides a temporary fix for a problem with creating a multi-DLL apps. Also included is a multi-DLL example. A more refined fix will be

available later.
Posted Monday, September 10, 2001

---

xAppWallpaper Manager v1.3 released

New in SealSoft's xAppWallpaper Manager: a new SetCaption method which changes the caption of the main xApWallpaper Manager window and all messages at runtime. Updated demo and install available.
Posted Monday, September 10, 2001

RInstall Update And Release 1 Notification

RInstall has been updated 6 Sept 2001. It now allows you to choose if you want to enable or disable record locking for files during the evaluation period. A new feature is the ability to hide or disable controls during the evaluation period. This is useful if users are not allowed to perform certain actions or access areas of applications during the evaluation period. Except for bug fixes, this is the last addition of functionality for the version 1 of Rinstaller. During the beta the price is $59; as of September 17, 2001, the Release 1 price will be $79.
Posted Thursday, September 06, 2001

VCRFlash Released

Sterling Data's VCRFlash turns forms into browses; users can scroll through records without exiting the form back to the browse. The template is easy to use; just drop the buttons on to a form and fill in a few template prompts. The form will "browse" according to the range and/or filter set up on the calling browse. Other features include: multi-language support; two different default button sizes - but you can use your own icons if you want; legacy and ABC versions; compatibility with Clarion CW2.x through C5.5. All source code supplied. No black boxes. No runtime royalties. Demo available. VCRFlash is $99 during the beta period.
Posted Thursday, September 06, 2001

ARCO Word Reporter Holiday Schedule

Hanspeter Stuts is on holidays until Tuesday, September 18th. Your emails will be forwarded Nick Freitag and to Hanspeter's

laptop (which doesn't get checked very often). Please be patient.
Posted Thursday, September 06, 2001

[xWindow Settings v2.0.8 Released]

New in SealSoft's xWindow Settings v2.0.8: the new method SetCaption(STRING NormalMode, STRING EditMode) will change the caption of main xWinSet window at runtime; you can set window titles, both for Normal Mode, and for Edit Mode; in the template you can assign the caption using literal text or a variable. New install and demo available.
Posted Wednesday, September 05, 2001

[Windows XP Design Guidelines]

These guidelines from Microsoft will help developers and designers adopt the new look and feel of the Microsoft® Windows® XP operating system. This collection of style elements and controls provides a base of design specifications and implementation details for using Windows XP themeing in an application.
Posted Wednesday, September 05, 2001

[OpenClose 1.2 Now Available]

Ron Eisner has updated his free OpenClose utility, which allows you to open any file in any access mode. You then run and test other programs against the open file. You can run multiple copies of the program to open multiple files. The utility's purpose is to make it easy to simulate various access modes your program may encounter, so you can see how your program reacts to files previously opened in various modes. New features include: more informative debug output; built-in event history; instance identifier (generated or specified); hot keys for most actions; droplists for easy setting of access mode. The program safely closes and releases any open file upon exit.
Posted Wednesday, September 05, 2001

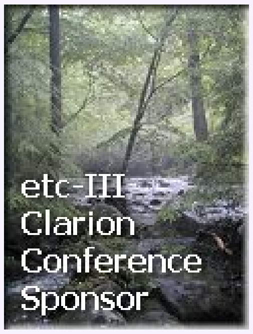
[Metabase Version 5 Released]

Whitemarsh Information Systems has released version 5 of the Metabase product. This is a significant enhancement over the version released in the Spring of 2000. For example, the

Metabase now generates basic SQL (tested with MS SQL) from the Specified, Implemented, and Operational data models.

Posted Monday, September 03, 2001

Clarion MAGAZINE



When a Datum is a Function

by Steven Parker

Published 2001-09-13

Long time Clarion users distinguish two types of procedures. One does not return a value and is referred to as a "procedure." The other returns a value and is referred to a "function."

It used to be easy to distinguish procedures and function when looking in source files. Procedures were prototyped with the Procedure key word and functions with the Function key word. But, with C5, that distinction began to blur. The C5 Programmer's Guide says:

> A PROCEDURE can be prototyped to RETURN a value, and therefore may be called as part of an expression or parameter list. A PROCEDURE which does not RETURN a value may only be called explicitly as a separate program statement – it may not be used as part of an expression or parameter list.
>
> A PROCEDURE prototyped to RETURN a value may also be called explicitly as a separate program statement when the returned value is unimportant ... [using] the PROC attribute.

A function I recently created generated the following prototype:

```
GetDayOfWeek PROCEDURE (Long pDate)
```

But this same function shows up in the Inc file as:

```
GetDayOfWeek FUNCTION(Long pDate),String
```

Typically, a function, `GetDayOfWeek` for example, would be used as follows:

```
LOC:PrintDate = GetDayOfWeek(EOD:Date)
```

Because a function returns a value and may be used in an expression, the code

```
If Records(myFile) > 0
```

and the code

```
If Records(myFile)
```

are equivalent. As the LRM points out, "Logical expressions evaluate true-false conditions in IF, LOOP UNTIL, and LOOP WHILE control structures." Therefore, if there are 17 records in myFile, these statements evaluate as:

```
If 17 > 0
```

and

```
If 17
```

respectively.

The first, `If 17 > 0`, is clear enough. `If 17` evaluates identically because "true" in Clarion is any non-zero and "17" is clearly non-zero.

Okay, "function" is clear.

A little known fact is that every variable in Clarion can be treated like a function. And this can be a very useful piece of information.

Suppose I have a customer file with a prefix of `CUS` and a column labeled "`Name`." The most common use for `CUS:Name` is in assignments such as:

```
CUS:Name = ‘Valued Clarion Mag Customer’
```

But `CUS:Name` can also be used in expressions. For example,

```
If CUS:Name = ‘Seymour’
```

or

```
If Instring(‘Sey’,CUS:Name,1,1)
```

If variables can be called in expressions then they must be functions, given the Programmer’s Guide statement above, at least implicitly.

And, if so, then it follows that

```
If CUS:Name = ''
```

and

```
If ~CUS:Name
```

and

```
If NOT CUS:Name
```

are equivalent. And they are.

Assume then that a variable is an implicit function. If the variable has no value (numerics 0 or strings blank), calling that variable in an expression will return False. If a variable is an implicit function then if the variable has a value (numerics non-0 or strings non-blank), calling that variable in an expression will return True.

And this is exactly what happens.

```
If ~CUS:Name
  Message('You must enter a name',|
    'Don''t Do That!',Icon:Exclamation)
  Select(?CUS:Name)
  Cycle
End
```

(I have seen the above rendered as `If Len(Clip(CUS:Name)) = 0`, truly I have, on more than one occasion.)

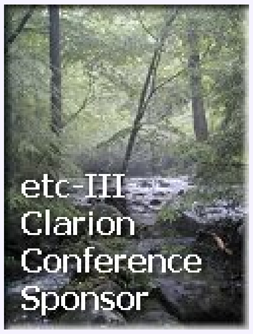
Knowing that a variable can behave like a function provides an easy way to test whether it has been filled in or whether it has a certain value.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reader Comments

Add a comment

Clarion MAGAZINE



Designing Crosstab Reports In Clarion (Part 2)

by David Potter

Published 2001-09-13

In Part 1 I demonstrated how to create a simple crosstab report using the standard Clarion report template. I described how to use a group data structure to sort and accumulate the data from a hypothetical sales table. This week I will show how to amend that procedure to use a queue in addition to the group to add some versatility to the report.

Data declarations

First I declare a group type similar to the one used in the earlier report. The only change will be the addition of two fields to the group to save the Location and Year values.

```
Location STRING(20)
Year     SHORT
```

Now I can declare a Queue in the data formatter. I call it RptQueue with the prefix Rpq. The queue contains only one data member, a group structure of the type RptGrp. I will leave the prefix field blank in the group declaration so all of the members of the group will share the prefix Rpq as well. To allow me to populate the report I give the RptGrp type definition the prefix Rpq as well (see Part 1 for a description of that technique). I'll declare a TotGrp and SubGrp structure of type RptQueue outside of the queue definition to use for accumulating totals and subtotals in the report. I need a SavGrp as well just like in the earlier article.

Filling the queue

The rest of the report is set up much the same as the first report using the group structure. The primary file and key remain the same. The report structure consists of three details in which the printing has been suppressed. Even the code in the ThisReport.TakeRecord embed remains remarkably similar:

```
IF Sav:Location <> Sal:Location    |
  OR Sav:Year <> YEAR(Sal:Date)
  IF Sav:Location
     Rpq:Location = Sav:Location
     Rpq:Year = Sav:Year
     Self.CalculateGroupPct(Rpq:DetailGrp)
     ADD(RptQueue)
  END
  Sav:Location = Sal:Location
  Sav:Year = YEAR(Sal:Date)
END
```

If you comparing this code to the code presented in last week's article, you'll easily see the primary difference. Instead of printing the detail when the parameters change, I now add the detail's data to the queue. I've also made a minor change to clean up the code. Where in the last report I calculated the percentages individually for each quarter, I have now derived a method in the `ThisReport` class to do this for me. The method takes a `RptGrp` type as its only parameter. This method call eliminates some duplicate code.

Now when the report exits the process loop after reading the last record in the file, the ReportQueue will be almost filled. I say almost because it will be one record short. To get the last record into the queue I will have to add two lines of code in the ThisWindow.AskPreview embed.

```
ThisReport.CalculateGroupPct(Rpq:DetailGrp)
ADD(RptQueue)
```

Printing the report

I print the report in the AskPreview embed as well. I do this using a simple loop:

```
LOOP I = 1 to RECORDS(RptQueue)
  GET(RptQueue,I)
  PRINT(Rpt:Detail)
END
```

Totals and subtotals

The logic to print the subtotals can be placed around the `PRINT` statement here as well. Be sure to clear the `SavGrp` structure before starting the loop.

```
CLEAR(SavGrp)
LOOP I = 1 to RECORDS(RptQueue)
  GET(RptQueue,I)
  IF Sav:Location <> Rpq:Location
    IF Sav:Location
        ThisReport.CalculateGroupPct(SubGrp)
        PRINT(SubDtl)
        ThisReport.AddGroupTotals(TotGrp,SubGrp)
     END
```

```
      Sav:Location = Rpq:Location
    END
    PRINT(Rpt:Detail)
    ThisReport.AddGroupTotals(SubGrp,Rpq:DetailGrp)
  END
```

I derived a new class to add the totals and subtotals as well. This class takes two parameters, both of the `RptGrp` type, and adds the values of the second group to those of the first. After printing all of the records in the queue there will still be values hanging in the `SubGrp` and `TotGrp` that need to be printed. This can be done by adding the following code, directly after exiting the loop.

```
ThisReport.CalculateGroupPct(SubGrp)
PRINT(SubDtl)
ThisReportAddGrpTotals(TotGrp,SubGrp)
ThisReportCalculateGrouPct(TotGrp)
PRINT(Rpt:Total)
```

Advantages to using a queue

This pretty much handles how to print a simple crosstab report using a Queue structure. So far I haven't presented any advantage in this method over using a group structure. The big advantage is that the queue can be sorted almost instantaneously and it is easy to let the user choose the sort order to print the report, even if no key exists, without a performance penalty.

To illustrate this I will change the requirements again for the crosstab report. Now let's say the user wants to have the choice of printing the report by Location and Year or Year and Location. To do this efficiently I would need a key sorted by Date first then location, which I don't presently have. However if I load the queue using the LocationKey, I can add the line

```
SORT(RptQueue,Rpq:Year,Rpq:Location)
```

before the print loop to change the order I read and print the queue. The extra time necessary to print the report in this order would now be negligible, compared to reading the file without a key. It would be necessary to change some of the logic to handle printing the Subtotals in this case. I have included a sample program to demonstrate both techniques I have discussed in this article.

Using a queue can give almost unlimited flexibility in writing a report. I was once giving the assignment of duplicating a Weekly Report currently being done on a spreadsheet. The data for the report was spread out over three different platforms, DB2 on an AS400, Oracle on a Sun Server and Clarion DAT files. The information was culled from running several reports, then hand entered into the spreadsheet. To make matters worse, many of the cells in the report were dependent on cells in previous rows. This is easy to do in a spreadsheet but tricky in a data report. I was able to solve the problem by looping through the necessary files and filling up a queue with the data. By giving each record in the queue a unique identifier, I was able to extract the report from the queue by taking the following steps:

1. GETting the records from the Queue needed for any calculation
2. Temporarily holding these records in group structures
3. Perform the necessary calculation on the group fields
4. Entering the calculated values into the Queue record structure
5. Adding the new queue record back to the queue with the appropriate index

After all of the values were calculated I simply looped through the queue in the proper order to finish printing the report.

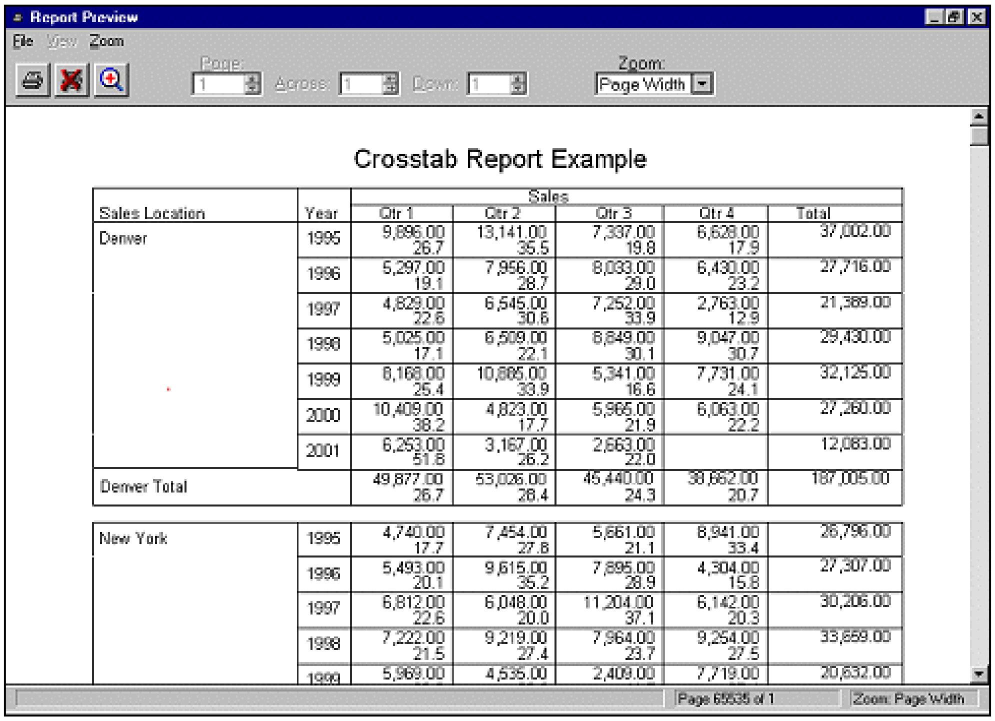
Some additional tips

When using Groups to print the totals and subtotals I find I am often creating details that are very similar in structure. Populating each detail individually can be time consuming. However by using the text editor (The ... button on the side of the report button in the template) to copy and paste the structure from the original detail to the total details I can save a lot of time (see Andrew Guidroz's article I Didn't Need That Much Detail for additional tips on manipulating the report structure directly). First I get the detail structure exactly like I want, adding lines to separate the records etc. Then I go to the text editor and

1. Copy the entire detail structure from the `Detail` declaration to the `END` statement and paste it directly after the `END` statement.
2. Change the name of the detail and the detail's use variable
3. Replace all of the prefixes of the variable with the proper prefix needed for the detail in question.
4. Edit the equate names for each control, ie. lines and boxes, to make sure they are unique. I usually do this simply by appending a different letter to each equate in each total detail. ie ?Line1 becomes ?Line1a in the Subtotal and ?Line1b in the Total.

The nice thing about this technique is all of the graphic elements, i.e. lines, will line up exactly between the detail and all of the footers.

By monitoring the changing variables you can also hide and unhide graphic controls to give the report a more pleasing appearance, as shown in Figure 2.



Crosstab Report Example

Sales Location	Year	Sales Qtr 1	Qtr 2	Qtr 3	Qtr 4	Total
Denver	1995	9,896.00 26.7	13,141.00 35.5	7,337.00 19.8	6,628.00 17.9	37,002.00
	1996	5,297.00 19.1	7,956.00 28.7	8,033.00 29.0	6,430.00 23.2	27,716.00
	1997	4,829.00 22.6	6,545.00 30.6	7,252.00 33.9	2,763.00 12.9	21,389.00
	1998	5,025.00 17.1	6,509.00 22.1	8,849.00 30.1	9,047.00 30.7	29,430.00
	1999	8,168.00 25.4	10,885.00 33.9	5,341.00 16.6	7,731.00 24.1	32,125.00
	2000	10,409.00 38.2	4,823.00 17.7	5,965.00 21.9	6,063.00 22.2	27,260.00
	2001	6,253.00 51.8	3,167.00 26.2	2,663.00 22.0		12,083.00
Denver Total		49,877.00 26.7	53,026.00 28.4	45,440.00 24.3	38,662.00 20.7	187,005.00
New York	1995	4,740.00 17.7	7,454.00 27.8	5,661.00 21.1	8,941.00 33.4	26,796.00
	1996	5,493.00 20.1	9,615.00 35.2	7,895.00 28.9	4,304.00 15.8	27,307.00
	1997	6,812.00 22.6	6,048.00 20.0	11,204.00 37.1	6,142.00 20.3	30,206.00
	1998	7,222.00 21.5	9,219.00 27.4	7,964.00 23.7	9,254.00 27.5	33,659.00
	1999	5,969.00	4,535.00	2,409.00	7,719.00	20,632.00

Figure 2. A crosstab report example

Notice how I have hidden the Sales Location variable so it will print only on the top line of each group. The example code will explain more clearly how this is done.

Summary

I hope I have given some tips that will make the Clarion report templates more flexible and easier to use. Although I have worked with other report writers I always come back to the standard Clarion report templates. The chief reason for this is the flexibility. I like the fact that I have the Clarion language to rely on, not to mention the fact that there are no additional client setup steps or licensing fees to deal with.

David Potter start his professional career as Chief Assayer at a gold mine. While buried under a mountain of lab results, he was given a copy of Clarion Personal Developer to help track the assay and metallurgical data. David found Clarion easy to use and soon upgraded to Clarion Professional Developer version 2. He has been developing in Clarion ever since. When the gold ran out David decided to go back to school and become a full time programmer. He is currently working in the MIS department of a plastic manufacturing plant in Washington State, and uses Clarion to develop applications on Oracle and AS400/DB2 databases.

Reader Comments

Add a comment

published by CoveComm Inc.

Clarion MAGAZINE

Remembering September 11, 2001

Published 2001-09-14

September 14, 2001 – Winnipeg, Canada. On this international day of mourning, Winnipeggers, visitors, and a number of still-stranded American travelers gathered at City Hall at noon to remember the events of Tuesday, September 11. The service was moderated by Mayor Glen Murray, and included speakers on behalf of all three levels of government as well as various faith groups.

My office is only several blocks from City Hall; at lunch I joined the crowd in front of the Council Building. The service began with the Canadian Mennonite University Choir singing the Isaac Watts hymn "O God Our Help In Ages Past (Our hope for years to come)." After a second number by the choir, Mayor Murray took the podium and reflected on the week's horrific events. He called for justice, and for peace. He pointed out that our city is home to people of many faiths, of numerous ethnic origins. Many of our ancestors, and some of this generation, came here fleeing persecution and oppression. Certainly this week our belief in our own peace and safety has been severely tested, but it isn't the first time this has happened. We have dark periods in our own history, where we have criminally oppressed members of our society. The risk is always there; we now have an Muslim population in our city which has been under considerable pressure since the hijackings took place.

It was a somber gathering, with little applause for the speakers. Yet after a member of the Winnipeg Muslim community came to the microphone and spoke, first in Arabic, then in halting French and English, of his people's moral and material support for those who have suffered this attack, we clapped. We applauded his courage, but I think we also clapped for ourselves, to raise up that trampled hope that we really are a global family.

We pray for peace and justice, comfort and compassion. We think of those who died helplessly, who gave their lives in rescue attempts, who suffered injury, and all their friends and families. We pray for wisdom for world leaders, and for safety for the innocent.

David Harms

Publisher, Clarion Magazine

Reader Comments

Add a comment

I am also shocked by what happened in New York and...
Thank you for this. I was in Scotland when I learned of...
I also want to express my sympathy to anyone affected by...

Custom Software for all your Business Needs & Add-on Tools for Clarion Developers — Certified Clarion Developer

published by CoveComm Inc.

Search
Go

Home

COL Archives

Advertise

Reader Comments

Call for Articles

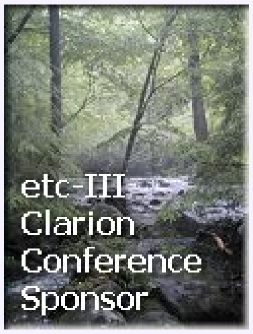
Information
Log In
Membership/ Subscriptions
FAQ
Privacy Policy

Contact Us

Downloads
PDFs
Freebies

Open Source

Site Index



Clarion Magazine's Publication Schedule

Published 2001-05-02

Clarion Magazine is a weekly magazine, published 48 times per year, with articles and news items published throughout the week. Because the immutable laws of mathematics say that 52-48=4, there are normally four weeks per year when Clarion Magazine doesn't publish.

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

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

How To Make Unplanned Service Improvements

Published 2001-09-21

The last few days have been very entertaining here at the magnificent ClarionMag corporate headquarters. On Tuesday, a day I took off to do some needed work on the house, the nimda worm hit the 'net, and caused some service interruptions to Clarion Magazine. That in itself wasn't a big problem, just another in a series of network-clogging Windows server viri that I couldn't personally do anything about (since I don't run any Windows web servers).

On Thursday morning, however, I lost the ability to query my ISP's name servers. Actually that's not completely true. I could query the name servers, but I never got any replies. I could however ping those same machines. DNS queries went out, DNS responses got dropped on the way back. At least that's how it seemed. The end result was that I couldn't surf anywhere, or get mail (without resorting to IP numbers), or do a host of other everyday tasks. Some ClarionMag users were also affected, although others could still get to the web site.

The nimda worm complicated things, although its role isn't completely clear to me yet. But the main problem was the dropped packets, and that apparently had to do with my running two subnets on the same wire. About a year ago my ISP was purchased by Group Telecom; one of the happy consequences of that acquisition was GT's plan to upgrade the local infrastructure for business Internet customers. I was to be moved to a new network layer, which meant a new block of IP addresses. I was particularly concerned that this not result in any interruption of service to ClarionMag customers, so my local GT tech suggested we temporarily run both the new and old subnets at the same time, allowing me to migrate my servers across without downtime. For full control over the process, I elected to set up my own primary DNS.

That migration would probably have taken place by now, but a while ago I ran into some unusual problems with zone transfers, and, oddly enough, DNS replies getting lost in transmission.

Whatever indication that was of the problem to come, Thursday morning everything fell apart. I spent a lot of time on the phone with my ISP, but at best I was able to get replies from their secondary DNS, but not the primary. Both were functioning normally from other vantage points, just not from my subnets. The name server I couldn’t get replies from was on the same Class B network as my new subnet, while the name server I could get replies from was on the same Class A network as my old subnet.

This seemed like a clear indication of a routing problem. I elected to change over completely to the new subnet, and have my ISP propagate the DNS changes overnight. The move to the new subnet cleared up the dropped DNS packets, and I was able to get full Internet access again.

The new IP block in itself doesn’t represent a service improvement, but is the necessary first part of moving to the new network. The actual cutover to the new layer is now just a matter of switching some wires around, and I’m assured should result in a service interruption of only 15 minutes or so. Barring any unforeseen developments, I *will* be able to give Clarion Magazine subscribers some advance notice of that event.

Thank you again for your patience.

Dave Harms

Publisher

Reader Comments

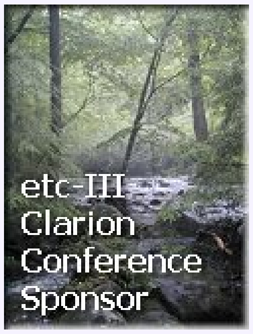
Add a comment

Dave, Look at it this way. If you *must* have a service...
Dave, S* happens. That's why they call it work. I...**
Thanks, guys. Today's cutover to the new layer should help...
Dave, After printing an article and pressing the Back...
Richard, Try empting your browser's cache and see if...

Reborn Free CLARION online

published by CoveComm Inc.

Clarion MAGAZINE



Report Redirection

by Steven Parker

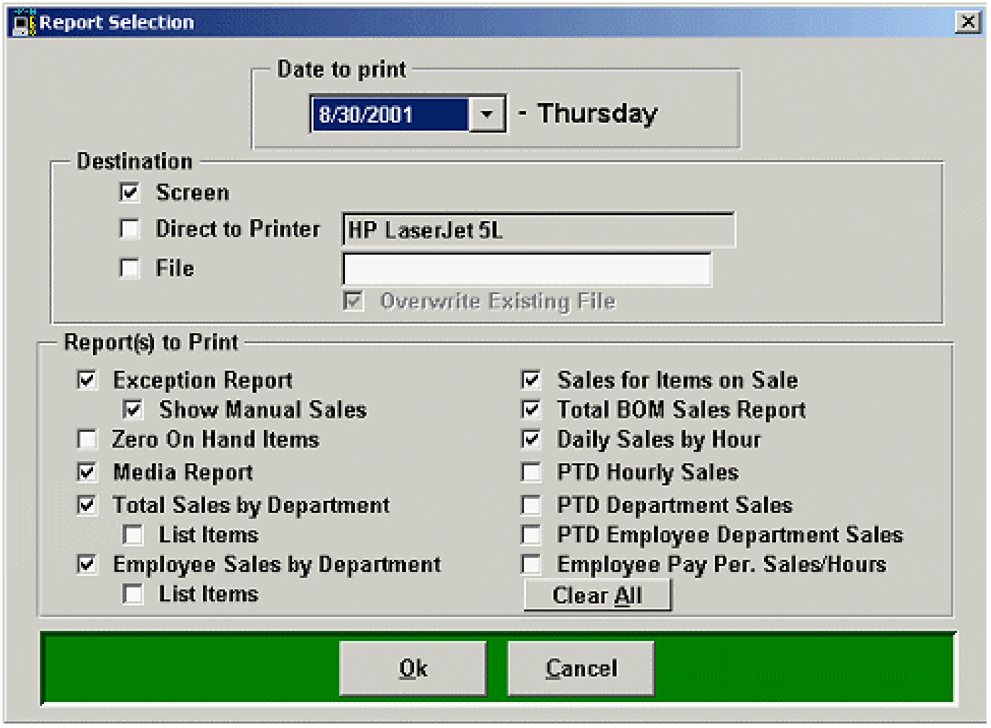
Published 2001-09-25

Everything you need to know about report redirection can be found at www.cwaddons.com/products/rpm/rpm1.html, the home of Lee White's Report and Presentation Manager. Everything. Which means that the rest of this article is superfluous.

Sort of.

Recently, I needed to configure redirection properties for a group of reports, the reports constituting the group being selected at runtime (see Figure 1). Unfortunately, I did not read the manual and, as a result, reinvented the wheel.

Figure 1. Report redirection/report selection combo



Report Selection

Date to print: 8/30/2001 - Thursday

Destination
- ☑ Screen
- ☐ Direct to Printer: HP LaserJet 5L
- ☐ File:
- ☑ Overwrite Existing File

Report(s) to Print

☑ Exception Report	☑ Sales for Items on Sale
☑ Show Manual Sales	☑ Total BOM Sales Report
☐ Zero On Hand Items	☑ Daily Sales by Hour
☑ Media Report	☐ PTD Hourly Sales
☑ Total Sales by Department	☐ PTD Department Sales
☐ List Items	☐ PTD Employee Department Sales
☑ Employee Sales by Department	☐ Employee Pay Per. Sales/Hours
☐ List Items	Clear All

Ok Cancel

In this article, I am going to show you how to do basic report redirection. But, because RPM does just about everything one could want and does it properly, I'm going to use my *sturm und drang* ("storm and stress," for those of you who did not take Psychology 101) reinventing the wheel to demonstrate something useful about the resources available within the Clarion environment, and explain some of the "gotchas" of playing with printers.

Report Redirection

In its simplest terms, report redirection is a feature that allows a user to direct a report to any of a number of destinations, which typically include printers, on-screen preview and saving to file.

One of the few genuinely positive contributions of Windows has been in the area of report redirection, though I doubt this was part of Microsoft's game plan. In days of DOS, when developers were bold and Windows wasn't "innovated," networks and network setup were a lot more complicated (at least until Artisoft's LANtastic peer-to-peer products appeared). It was not at all unusual for a PC to have a local printer (physically attached) as well as a network printer. Some had only a network printer but it would be on LPT2 (which DOS Clarion did not like at all).

This meant that at least two parallel ports had to be available for redirection. But, because of the multitude of ways networks were configured, all four possible parallel ports had to be selectable. Mastery of the `Mode` command was often required.

Many were still using serial printers (some of the best and fastest line printers were serial). Four COM ports now entered the equation. Mastery of the `Mode` command was essential. Save to file and preview on screen were the only "simple" choices.

Microsoft introduced peer-to-peer networking late in the Windows 3.*x* cycle and, in typical Microsoft fashion, got it fairly close to right a few releases later, in Windows 9x. The important part was that Microsoft transferred hardware communication from the application to the O/S. Thus, once a printer driver was installed, that printer was available, even if not actually connected to the PC (magical, isn't it?). Once a printer was selected, the O/S knew its characteristics and handled port communication. In other words, the developer is no longer concerned with what port a device is on or any of the device's characteristics.

To make peer networking work and to integrate it into the Windows interface, Windows handled connecting to remote drives in a similar manner. And, now, printers, hard drives and virtually any device with a driver can be addressed in a similar manner; you just select the device from a dialog or through an API call. Clarion provides properties, essentially interfaces to these APIs, for many of these, certainly all printer related calls.

To me, this means that I only need to provide the choices. Uninstalled printers can't be chosen (and unconnected printers usually, but not always, throw errors when you attempt to print to them). Installed but unconnected printers are the user's problem, not mine. Devices not on the network can't be chosen.

To me, this means that I only need to provide the end user with three choices: preview, print and save to file. For the sake of clarity, as shown in Figure 1, I present these as "Screen," "[send] Direct[ly] to Printer" and "File."

I could also provide a "Printer Selection" dialog to enhance the "print" option if I wanted to, perhaps with a button next to the printer name display. In this app, I do not, leaving the default "Print Setup" menu item on the frame for this purpose.

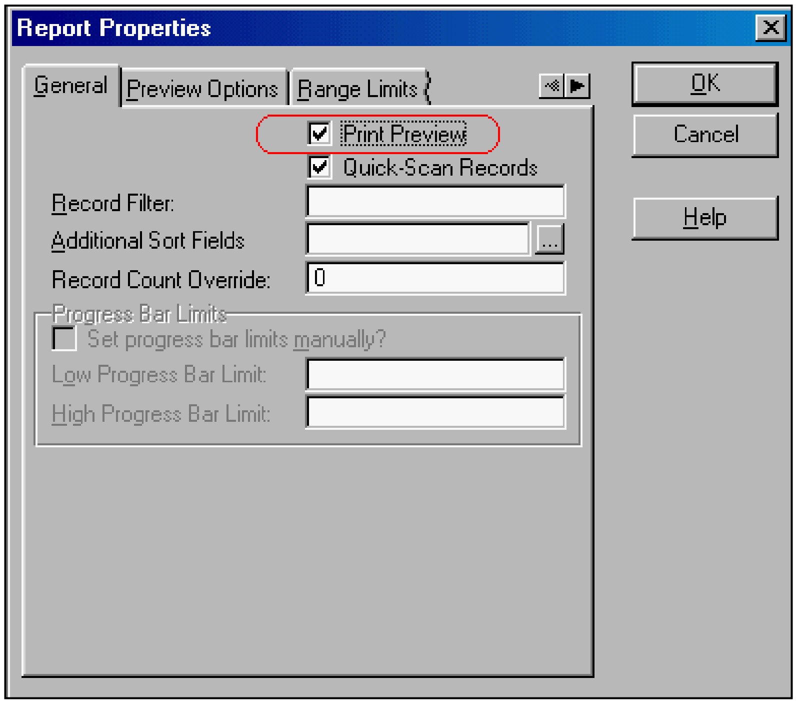
In my application, I have to offer the user the opportunity to name a target file when "file" is the report destination. Many of my users still hand off reports by sneakernet and high priced DP departments and payroll processing services don't seem to know how to use variable file names or are unable to rename files. This is fairly standard stuff.

Additionally, if an operator re-uses a file name, I have to offer the opportunity to append to the existing file or overwrite it.

Redirection resources

Clarion reports default to preview on-screen. At development time, I can turn that option off and the report will go directly to the printer (see Figure 2).

Figure 2. Standard report preview options



[✓] Print Preview
[✓] Quick-Scan Records
Record Filter:
Additional Sort Fields
Record Count Override: 0
Progress Bar Limits
[ ] Set progress bar limits manually?
Low Progress Bar Limit:
High Progress Bar Limit:

This does not qualify as redirection because the end user has no options. However, adding user selectable options is fairly straightforward using the ABC template chain.

Virtually everything you need to know to implement basic report redirection is available in the on-line help (specifically, the "FAQ" section, the number one resource).

There are three FAQs that I consider essential reading:

- "How to Print to a File"
- "How to change the printer device without calling PRINTERDIALOG"
- "How do I skip the print preview if the user has elected not to preview"

In fact, the first two are really just variations on a theme: how to change the target device (printers, disks and files, as far as Windows is concerned, are just output devices; in fact, if you want to by-pass the Hardware Abstraction Layer and talk directly to printer hardware, you actually use the `WriteFile` API call where the "file" is the printer port). The last actually links into Richard Taylor's "Making the Transition to the ABC Templates," itself worth a periodic re-read.

"How do I skip the print preview if the user has elected not to preview" is the key to report redirection. This FAQ gives the basics for providing a user choice. The best part is that I can capitalize on the basic ABC program flow: if the user selects "screen," then the default preview is used and I don't have to do anything; any other response and preview is skipped. Only if "Direct to Printer" or "File" are selected, preview is skipped, do I have to provide programmatic actions (and because of the default flow, "Direct to Printer" is nothing more than stopping preview). Knowing, from the docs, that I can let the user opt to skip preview makes everything else possible.

Skipping preview

According to the docs, there is a property, `ThisWindow.SkipPreview`, which, if `True`, allows a report to by-pass preview even though the preview option is checked (on) when creating the report.

Searching a report's generated source, however, turns up no instances of `SkipPreview`.

Searching the \LIBSRC directory will find it. `SkipPreview` is declared in `abreport.inc` and used in `ReportManager.AskPreview`:

```
IF NOT SELF.Report &= NULL
  IF NOT SELF.Preview &= NULL AND ↵
      SELF.Response = RequestCompleted
    ENDPAGE(SELF.Report)
    DoFlush = CHOOSE (NOT **SELF.SkipPreview**,|
        SELF.Preview.Display(SELF.Zoom), TRUE)
    SELF.Report {PROP:FlushPreview} = DoFlush
    FREE(SELF.Preview.ImageQueue)
  END
END
```

The only reference in the typical report procedure to `ReportManger` is:

```
ThisWindow CLASS(ReportManager)
```

which explains why the docs tell you to set `ThisWindow.SkipPreview` in order to by-pass preview.

It also explains why you will not find `SkipPreview` when searching the generated source. `ReportManager` is instantiated as `ThisWindow`, therefore, the `ThisWindow` object simply inherits all the properties (data declarations) of the `ReportManager` class.

In my redirection procedure, I use three local variables and display them as check boxes (I use Actions | Assign Values to uncheck the other two check boxes). "Screen" sets `S = True`, "Direct to Printer" sets a variable "D" and "File" corresponds to "F."

Radio buttons would work just as well; it's a matter of the desired user interface. The key is that just before calling a report, I use these check boxes to set another variable. This variable is passed to the report as a parameter:

```
If S         !Setup target parameter
  ReportTarget = 'S'
Elsif D
  ReportTarget = 'D'
Elsif F
  ReportTarget = 'F'
End
ReportByName(ReportTarget)
```

This leaves me with only one thing to check in the report procedure:

```
If pTarget <> 'S'
  ThisWindow.SkipPreview = True
End
```

The next question is:

Where?

The docs state that the `SkipPreview` property cannot be set before the object (`ThisWindow`) is initialized. So, it recommends setting and forgetting at the end of `INIT`.

In my particular case, I ask the user for the report device before calling any report(s). The recommendation is perfectly adequate for this case. In fact, I can check the parameter any time before the AskPreview method is actually called.

However, since most cases do not involve running reports in batches, you need to ask for the target device on a report-by-report basis. That is, you need to ask the destination for each report to be run.

In this case, you will embed something like:

```
Btn#=MESSAGE('', |
  'Select Report Target', ICON:Question, |
   'Preview|Print', 1, 0)
IF    Btn# = 1      ! Name: Preview  (Default)
ELSIF Btn# = 2      ! Name: Print
  ThisWindow.SkipPreview = True
END !IF
```

(This is a Message() created using Carl Barnes' CW Assistant. You can also create a window procedure to get the user's response.)

Now suppose there are no records qualifying for inclusion in the report.

If you "set and forget," the user will tell you where the report is to go only to have your designated no-records action occur. If you have left the default no-records message, the user says "Send my report to ..." and then gets the message; if you short circuit this message, the user selects a destination and gets nothing. Only if you created and print a "no records" band are user expectations entirely met.

Late in INIT may be a safe set-and-forget point but there must be a more end-user friendly way; that place is WindowManager | OpenReport a/k/a ReportManager.OpenReport.

OpenReport is so important because this where it is determined whether or not there are records for the report. Looking at the "Parent call" demonstrates why:

```
ReturnValue = PARENT.OpenReport()
```

If ReturnValue has no value (i.e., is zero), then the report opened (i.e. is already opened). If ReturnValue has a value, then there were no qualifying records ("no records" has already been called). This is slightly counterintuitive, but if you trace through the "Parent" code, it becomes clear:

```
ReportManager.OpenReport  PROCEDURE
RVal BYTE,AUTO
  CODE
    SELF.Process.Reset() ! Needed for 're-read' case
    RVal = SELF.Next()
    SELF.DeferOpenReport = 0
    IF Rval           !Rval is not 0
      SELF.TakeNoRecords
    ELSE              !Rval is 0
      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
```

It may still look a bit peculiar that if `Rval` *does* have a value, there were no records. But, again, examining the code in `ReportManager.Next()`(which is what determines the value in `Rval`) demonstrates that `Rval` actually contains the "error" level:

```
ReportManager.Next PROCEDURE
  CODE
    CASE SELF.Process.Next()
    OF Level:Notify
      IF SELF.Process.RecordsProcessed
        SELF.Response = RequestCompleted
        POST(Event:CloseWindow)
        RETURN Level:Notify
      END
    OROF Level:Fatal
      SELF.Response = RequestCancelled
      POST(Event:CloseWindow)
      RETURN Level:Fatal
    END
    RETURN Level:Benign
```

`Next()` is 0 (`Level:Benign`) when a record is read. When `Next()` fails, the code returns `Level:Notify` or `Level:Fatal`. This is, then, no more counterintuitive than any of the ABC file access methods.

If you don't want to ask your user where the report should go when there are no records for the report, place the following code in `ReportManager.OpenReport`, After Parent Call:

```
If ~ReturnValue
  Btn#=MESSAGE('', |
       'Select Report Target', ICON:Question, |
        'Preview|Print', 1, 0)
  IF    Btn# = 1      ! Name: Preview  (Default)
  ELSIF Btn# = 2      ! Name: Print
    ThisWindow.SkipPreview = True
  END !IF
END
```

Now the user will be asked for a destination only if there really is a report to send there.

While we're at it

In OpenReport, After Parent call, I know whether there is anything to report. But nothing has actually happened other than opening the report. Opening only creates the structure in memory and makes its data (bands, in this case) available.

Because nothing has been printed yet, this embed is ideal for printing a title page. You can create this as a simple band, filter set to False and, perhaps, with PageAfter set or you can create it as a print-on-first-page-only band:

```
If ~ReturnValue
  Print(RPT:TitleBand)
End
```

"Print" to file

As it stands, the following code handles two of the three circumstances needing to be handled:

```
If ~ReturnValue
  Btn#=MESSAGE('', |
       'Select Report Target', ICON:Question, |
        'Preview|Print', 1, 0)
  IF    Btn# = 1       ! Name: Preview  (Default)
  ELSIF Btn# = 2       ! Name: Print
    ThisWindow.SkipPreview = True
  END !IF
End
```

If the only choices you need to offer are "preview" and "print," you're done. If you need to offer a "file" option, this code must be expanded so that it looks like this:

```
CASE MESSAGE('Select report target:', |
     'Select',ICON:Question, |
      'Preview|Print|File', 1, 0)
OF 1  ! Name: Preview  (Default)
  !default preview, no action required
OF 2  ! Name: Print
  ThisWindow.SkipPreview = True
OF 3  ! Name: File
  ThisWindow.SkipPreview = True
  Printer{PropPrint:Device} = 'Generic / Text Only'
  Printer{PropPrint:PrintToFile} = True
  Printer{PropPrint:PrintToName} = GetFileName()
END !CASE
```

(The lines in bold are adapted from the print-to-file FAQ.)

Unfortunately the `OpenReport, After Parent call` embed will not work for the printing to file option and cannot be made to work by adding appropriate `SetTarget` statements. The FAQ recommends `ProcessManager, Open` (if you trace this all the way back, it is where the underlying view is created) but this embed will present the `Message()` twice and therefore, in my opinion, is appropriate for hardcode print-to-file only. `OpenReport, Before Parent call` does work as expected (in the sample app, I also move the saving of the default printer to `ThisWindow.INIT` and the restoration to `Kill`, which work well for the purpose) but this does leave the user in the position of requesting a destination and getting a "no records" message.

It is important to turn off print-to-file at the end of the procedure:

```
Printer{PropPrinter:PrintToFile} = False
```

The FAQ recommends `ProcessManager.Close` but `ThisWindow.Kill` also works nicely. *Any* printer properties set in the report will retain their values until either reset or the application terminates. So, if you do not reset `Printer{PropPrinter:PrintToFile}`, all subsequent reports will also go to file. Similarly, if you do not set

```
Printer{PropPrint:Device} = SavePrinter
```

all subsequent reports will go to the `Generic / Text Only` device (if "file" was chosen).

It is easy and safe to reset both of these properties at procedure end regardless of the user's choice. It may also be necessary for you to change your report font to a fixed width font, like `Courier`, otherwise the file produced may not be useable.

Summary

I stand by my initial statement: "Everything you need to know about report redirection can be found at www.cwaddons.com/products/rpm/rpm1.html. Everything." On the other hand, this exercise had not been entirely wasted. `ReportManager.OpenReport` emerged as a very important embed. For printing something on the first page only, this is the place.

Clicking the on-line help "FAQ" button and perusing the tips and tricks there emerged as a good use of time. And finally, any printer properties set will stay in effect when the procedure terminates unless explicitly reset. So, reset them.

Download the source (C5.5)

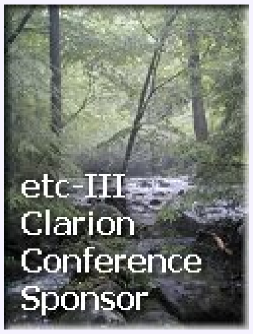
Download the source (C5.0)

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reader Comments

Add a comment

Clarion MAGAZINE



Do Not Adjust Your Browser

Published 2001-09-25

At approximately 5:22 p.m. Central Daylight Savings Time today (or 10:22 p.m. GMT, I think) my ISP (Group Telecom) switched the ClarionMag servers to a new network. The site was unavailable for approximately 9 minutes during this time, slightly less than the allotted 15 minute window.

This network cutover follows on the heels of last week's unexpectedly accelerated switch to Clarion Magazine's new IP address block. This new block of addresses is now on the new network, where Clarion Magazine has a closer connection to the backbone and less traffic on the local segment (down to 15-20 other users, from as many as 200 now).

What does the cutover to the new network really mean? More reliability, and perhaps a little more speed, although there's fiber to the building, and we're pretty close to the backbone already.

Dave Harms

Publisher

Reader Comments

Add a comment

Search
Go

Home

COL Archives

Advertise

Reader Comments

Call for Articles

Information
Log In
Membership/ Subscriptions
FAQ
Privacy Policy
Contact Us

Downloads
PDFs
Freebies
Open Source

Site Index

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

by Matt Grossmith

Published 2001-09-26

WinInet.DLL has been around for a while now. It is a 32bit DLL that allows the Windows developer to use File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Gopher and many other protocols without having to get into the nitty gritty of socket programming.

There are over 200 functions and procedures in the WinInet.DLL, but for this article I will only be covering the functions and procedures used to manage an Internet connection and handle FTP requirements. I'll also cover some general Windows programming issues including converting VB syntax to Clarion syntax.

Resources

You can find an extensive resource of WinInet information at the Microsoft Developer Network (MSDN) website. This page includes a variety of tutorial and other informational documents, as well as detailed information on the DLL's functions and needed structures.

As WinInet.DLL is a higher-level abstraction of the lower-level protocols, when these standards change there should be no impact on systems you have already developed; the latest version of the DLL will (in theory) take care of things for you.

Prototyping WinInet.DLL

The MSDN site gives examples of how to prototype API calls for Visual Basic (VB). This is usually a good starting point for Clarion developers, as the conversion from VB to Clarion declarations is fairly simple. Consider this VB declaration:

```
Declare Function FtpRenameFile Lib
  "wininet.dll" Alias "FtpRenameFileA"
  (ByVal hFtpSession As Long,
  ByVal lpszExisting As String,
  ByVal lpszNew As String) As Boolean
```

The corresponding Clarion declaration looks like this:

```
Module('wininet')
  FtpRenameFile(long,*cstring,*cstring),|
    long,raw,pascal,name('FtpRenameFileA')
End
```

As you can see, converting VB syntax to Clarion syntax isn't too scary after all. Here's a quick guide on how to translate these declarations

- In the VB declaration the word after "Declare" indicates whether this is a function or a procedure; this is followed by the function or procedure name.

- The word in quotes after "Lib" is the module/DLL that contains the procedure. Most of the common windows functions are in the Win32 library that ships with Clarion. If you are using a function or procedure that comes from a new DLL (like the WinInet stuff) then you will need to make a LIB file for the DLL. Use the LibMaker that comes with Clarion and load the DLL into it, then save the definitions as a LIB file and include it in your project under the "Library, object and resource files" section.

- Some VB functions and procedures have an "Alias" (this is usually a 16bit / 32bit compatibility issue). If this is the case, then the word in quotes following the alias is used as the NAME attribute in Clarion.

- If the declaration is for a function then there will be a return value. This is the last part of the VB declaration ("As Boolean" in the above example) and gets slapped on after the parameter list in Clarion, i.e. `,*cstring),long`.

- As far as the actual parameters are concerned there is usually a direct relationship between VB and Clarion when it comes to numerics. You can either declare equates, i.e. `Handle EQUATE(Ulong)`, or change the declaration to use the Clarion equivalent. Strings are usually passed as pointers to null terminated strings (the `lpszExisting` in the VB declaration means "Long Pointer to String ...."), so are prototyped as CStrings passed by reference (`*cstring`).

- The last thing to do is add the `raw,pascal` attributes to the Clarion declaration. The `raw` attribute tells Clarion to only pass the address of parameters (normally Clarion passes the address and length) and the `pascal` attribute tells Clarion to pass parameters on the stack and disable name mangling; this ensures compatibility with the Windows API calling convention.

The functions and procedures I am going to use initially are listed below, however I will be using more of the WinInet functions later in this series of articles. A complete list is available in the downloadable code at the end of this article.

Listing 1. Wininet prototypes

```
module('wininet')
  InternetOpen(*cstring,long,*cstring,|
    *cstring,long),long,raw,pascal,|
    name('InternetOpenA')
  InternetCloseHandle(long),long,raw,pascal
  InternetFindNextFile(long,|
    win32_find_data_def),long,raw,|
    pascal,name('InternetFindNextFileA')
  InternetConnect(long,*cstring,long,|
    *cstring,*cstring,long,long,long),|
    long,raw,pascal,name('InternetConnectA')
  FtpGetCurrentDirectory(long,|
    *cstring,*long),long,raw,pascal,|
    name('FtpGetCurrentDirectoryA')
  FtpSetCurrentDirectory(long,*cstring),|
    long,raw,pascal,|
    name('FtpSetCurrentDirectoryA')
  FtpCreateDirectory(long,*cstring)|
    ,long,raw,pascal,|
    name('FtpCreateDirectoryA')
  FtpRemoveDirectory(long,*cstring)|
    ,long,raw,pascal,
    name('FtpRemoveDirectoryA')
  FtpDeleteFile(long,*cstring),|
    long,raw,pascal,name('FtpDeleteFileA')
  FtpRenameFile(long,*cstring,*cstring)|
    ,long,raw,pascal,name('FtpRenameFileA')
  FtpGetFile(long,*cstring,*cstring,|
    long,long,long,long),long,raw,pascal,|
    name('FtpGetFileA')
  FtpPutFile(long,*cstring,*cstring,|
    long,long),long,raw,pascal,
    name('FtpPutFileA')
  FtpFindFirstFile(long,*cstring,|
    win32_find_data_def,long,long),|
    long,raw,pascal,name('FtpFindFirstFileA')
end
```

Using WinInet.DLL

The basic methodology of using WinInet is the same for up/downloading a file via FTP or retrieving a page via HTTP.

1) Initialize the WinInet.DLL and get a handle to the Internet

2) Use this Internet handle to obtain a handle to a connection to the server you want

3) Use this Connection handle to download from or upload to the server

4) Free the connection handle

5) Free the Internet handle

Due to the way WinInet DLL works, you can free the handle to the Internet and all the handles created after this will be freed automatically.

Declarations

From the above outline you can see there are some variables needed to manage the handles. There are also some equates that are needed for the function calls. I'll cover these as I use them, but for clarity I'll list them here. You can find a larger list of equates, covering most of the possible parameters, in the downloadable code at the end of the article.

Listing 2. Variables and equates

```
! Handles
InternetOpenHandle        long
InternetConnectionHandle  long
FTPFindFileHandle         long

! Variables
! An identifier for the application
ApplicationName          CString('MyApplication')
! A useful constant for passing
! NULL to windows APIs
NullString               CString('')
! The server URL needed for InternetConnect
URL                      CString(261)
! The user name needed for InternetConnect
UserName                 Cstring(100)
! The user password needed for InternetConnect
Password                 Cstring(100)
! The name of a file on the FTP server
FTPTXFileName            CString(261)
! The name of a file on the local machine
LocalTXFilename          CString(261)

!Equates
INTERNET_OPEN_TYPE_DIRECT EQUATE(1)
INTERNET_DEFAULT_FTP_PORT EQUATE(21)
INTERNET_SERVICE_FTP      EQUATE(1)
INTERNET_FLAG_PASSIVE     EQUATE(08000000h)
FTP_TRANSFER_TYPE_UNKNOWN EQUATE(0)
```

Initializing the DLL

Before you try to use any of the WinInet functions you will need to initialize the DLL by using the `InternetOpen` function:

```
InternetOpenHandle = InternetOpen(ApplicationName,|
    INTERNET_OPEN_TYPE_DIRECT, |
    NullString,|
    NullString,|
    0)
InternetOpen
```

`InternetOpen` expects to be passed the following parameters:

`Agent`	The name or identifier of the application using the internet functions

`AccessType`	The type of access required. This can be any one of the following:
	
	`INTERNET_OPEN_TYPE_DIRECT`
	
	Resolves all host names locally. This will ignore any proxy settings you have and try to resolve host names directly
	
	`INTERNET_OPEN_TYPE_PRECONFIG`
	
	Retrieves the proxy or direct configuration from the registry. This will use the settings you have for proxy servers but it will also run any Internet setup files – these are the files that change your homepage settings etc.You may have these if your version of Internet Explorer was supplied by an ISP
	
	`INTERNET_OPEN_TYPE_PRECONFIG _WITH_NO_AUTOPROXY`
	
	Retrieves the proxy or direct configuration from the registry and prevents the use of a start-up file.
	
	`INTERNET_OPEN_TYPE_PROXY`
	
	Passes requests to the proxy unless a proxy bypass list is supplied and the name to be resolved bypasses the proxy. If you use this setting then you will need to provide the proxy settings manually (see below). In this case, the function uses `INTERNET_OPEN_TYPE_DIRECT`.

`ProxyName`	Name of the proxy server (if required and `AccessType` is set to `INTERNET_OPEN_TYPE_PROXY`)

ProxyBypass	A list of URL's or IP addresses that should not be routed through the proxy server (if required and AccessType is set to INTERNET_OPEN_TYPE_PROXY)
Flags	Alters the function of the access and can any combination of the following: INTERNET_FLAG_ASYNC Makes only asynchronous requests on handles descended from the handle returned from this function INTERNET_FLAG_FROM_CACHE Does not make network requests. All entities are returned from the cache. If the requested item is not in the cache, a suitable error, such as ERROR_FILE_NOT_FOUND, is returned. INTERNET_FLAG_OFFLINE Identical to INTERNET_FLAG_FROM_CACHE. Does not make network requests. All entities are returned from the cache. If the requested item is not in the cache, a suitable error, such as ERROR_FILE_NOT_FOUND, is returned

InternetOpen will return a handle for the application to use in subsequent WinInet calls. If the return value is zero then the call has failed and the application will not be able to access the Internet with the supplied parameters. To determine the error you can call GetLastError ; you may also want to look at the error handling function in the sample application.

Make a connection

Now that you have access to the Internet, you need to create a connection to a server. You do this with the InternetConnect function:

```
URL = 'ftp.icetips.com'
UserName = 'guest'
Password = 'my@emailaddress.com'
InternetConnectionHandle = InternetConnect(
  InternetOpenHandle,|
  URL, |
  INTERNET_DEFAULT_FTP_PORT, |
  UserName, |
  Password, |
  INTERNET_SERVICE_FTP,
  INTERNET_FLAG_ACTIVE,
  0)
```

`InternetConnect` expects to be passed the following parameters:

InternetHandle	A handle returned from a previous call to InternetOpen
ServerName	The name of the server to connect to
Port	Which port on the server you want to connect to (usually 21 for FTP, 80 for HTTP)
Username	The name of the user to log on to the server
Password	The users password
Service	What service to access (FTP/HTTP/ Gopher)
Flags	Alters the type of connection made, and is specific to the service requested. If the service is FTP setting this flag to 8000000h causes the application to use passive FTP semantics. (For a good explanation of the difference between active and passive ftp see this SlackSite article

`InternetConnect` will return a handle to the session if the connection was successful, if the return value is zero then the connection has failed, and the application will not be able to access the service requested. To determine the error you can call `GetLastError`.

With a session handle in hand, you're ready to do a file transfer. I'll explain how to do that next week.

Download the source

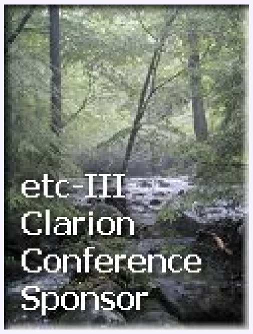
Reader Comments

Add a comment

What are your thoughts about this? Antonio: "Handytools...
Antonio, I've heard WinInet can have difficulties when...
Antonio So far we have put this FTP method into about 15...

published by CoveComm Inc.

Clarion MAGAZINE



Interview: ClarioNET's Michael Brooks

Published 2001-09-28

One of the more exciting Clarion developments in recent years is ClarioNET, a thin client adaptation of the Internet Connect technology. ClarioNET was initially developed by Michael Brooks, who is now working together with SoftVelocity to bring this product to market.

Clarion Magazine: How did you become a Clarion developer?

Michael Brooks: It all began in 1975 when I noticed a terminal in the math department of the junior college I was attending. The student sitting there said he had direct access to a "mainframe" and was writing a program in "APL". After learning what a mainframe was I knew there was opportunity there.

After Fortran, punch cards, basic, etc at a junior college I transferred to a university where I studied Civil/Structural Engineering. There was no good computer system there and so my association with programming slipped in those years. But shortly after graduation I purchased an Apple II with VisiCalc, and authored dozens of standard calculations for structural engineering design of buildings. Seeing the potential, I started a business selling Lotus 1-2-3 templates for structural engineering. I've had that engineering software business now for 17 years, and for the past 10 years have been using Clarion.

What motivated you to write ClarioNET?

I was looking for a way to bring my software business into a recurring revenue model that would also simplify distribution and license (copy) protection. The only things available that looked promising were Citrix and Clarion's Internet Connect. Citrix was out because of the cost, maintenance, and hardware requirements. Internet Connect was out because HTML could not provide a professional user interface for my clients.

So one day I realized that somehow the Clarion IBC (Internet Builder Classes) source code was able to get all the information from a running Clarion program. So I set about dissecting the IBC classes to see what was going on. After a couple of months of testing (before DevCon 1999) I realized that it was entirely possible to build a remote user interface by simply CREATEing controls and managing them through their field equate numbers.

When I had a simple system up and running, I realized that this specific solution could be a tremendous benefit to me, and provide a great thin client solution. It wasn't until much later (ETC 2000) that interest from the Clarion community really pushed me to develop it as a third party product.

How does ClarioNET directly address the problems you were facing?

We have 3,500+ users nationwide of our structural engineering product. And it's a hard sell to new users because the product is nearly $1,000 per license. In addition, with the proliferation of networks, our users are looking for solutions to workstation maintenance and do not want to bear the cost of complex systems like Citrix themselves.

So ClarioNET will be used to offer a remote usage option for our clients. They can use it anywhere, keep their files on our database server, eliminate update installs, and simply pay a small monthly fee. The flexibility is incredible, and because the bandwidth is so small and only port 80 is used it's zero effort for them to start using it.

It's a little early for most of them though. It's a new concept in software usage, and we're offering it on a testing basis now to get their feedback and refine the features.

Are there any cross-platform possibilities for ClarioNET, do you think?

Not on the server side, but I'm sure clients could be created for different platforms. It's a matter of need though. Our client base, which basically is American businesses, just sticks with Windows. Everyone knows Windows and it gets better each year. As in all business decisions, if there is a verified need that will support development costs, then we'll start planning for that product enhancement.

How is ClarioNET affected by recent concerns over IIS security? Are there any possible scenarios for server hacks? Buffer overruns?

This is not my area of expertise but I'll give it a try. IIS security is out of our hands and up to the administrator. A breach there, as we've seen with Code Red, is a system breach and the effects are far more important than just for ClarioNET.

The other two issues I'll need to defer to Soft Velocity. The application broker actually receives the incoming datastream (both initial request and routine communication) and passes it along. As long as the app broker is handling the stream limits correctly I don't think there is cause for concern. It just acts as a message router.

When the datastream gets to ClarioNET I would say that there is zero chance that anything can compromise the system. All datastreams are clipped to maximum buffer size. The streams are compressed, encrypted, and encoded and are very, very specific to ClarioNET. Even if they were compromised they could not be used to control the server - the logic would already need to be present in the server program to cause the problems.

Is it possible to use SSL with ClarioNET?

No, but we could eventually build it in if it was a make or break issue with the product’s worldwide acceptance. The current encryption is *very* strong, multi-layered, and uses continuously changing encryption seeds.

What is ClarioNET's scalability like?

Actually the first issue here is the design of the server application. Programmers need to remember the environment their programs are running in. Everything they use in their programs they are battling for: RAM, resources, disk tasks, clock cycles for loop operations, etc. You need to be very, very wise in conserving the resources available on the server. The worst example is a locally linked EXE file.

With proper design of the application you can really load the server with users and it behaves well. We've all heard from Steve Parker and Andy Stapleton about 125+ simultaneous users, but when you get the end users hammering away there are limits to perceived turnaround speed. I think Soft Velocity's guideline of 50 users per server is good for a "real" business application. I think a slight refinement to that recommendation would be 35 users per processor.

In our business we figured that of 3,500 users 33% would be on line simultaneously, so we purchased 26 dual 733 PII Dell 2450's with 512 MB ram. If there is a higher percentage we can just add servers. They're cheap and the redundancy of multiple servers is essential.

What features do you have planned for future releases?

We’re currently looking into the additional needs that the current users have asked about. There are not too many though. Because of the long beta test period it is incredible how much functionality was put into the initial release.

The biggest area that we need to look at is integration with the Clarion templates. ClarioNET must work in a particular way that is dictated by the thin-client model. Future template changes can offer edit-in-place, queries, and other items not currently supported.

It's obviously been a lot of hard work. Would you do it again?

Yes. ClarioNET was more work than I'd planned on, but I enjoy research and low-level programming. And it is a critical part of my engineering business’s future. In all humility I’m amazed that it’s complete and works as well as it does. Clarion is one amazing tool!

What's the target market for ClarioNET?

It's almost limitless. What is the market for a computer program? I'm not saying that out of ego or visions of profits. Citrix proved the concept and did $517M last year, so we know there must be some interest somewhere in the thin client model. Once we have some real case studies ready, including ours and a nice project SV is working on, ClarioNET will essentially be proven and will get even more serious attention. We'll see - it will be an exciting six months!

Are sales meeting your expectations?

Yes, absolutely. What is also heartwarming is the positive support from the newsgroup. Sure there are frustrations, but in general Clarion users are very excited [about ClarioNET].

What's the nature of your relationship with SoftVelocity?

We work together closely to ensure that the product meets the needs of the community, and we jointly research and resolve problems and research new opportunities. New properties have been added to Clarion specifically to support ClarioNET, and [SoftVelocity is] immediately responsive when tough technical issues come up. SoftVelocity has impressed me with their commitment and dedication to ClarioNET, and we are enjoying a closer relationship each week.

We both understand the possibilities here. The "thin client" world is just beginning and it’s extremely exciting, especially with development tools like Clarion and ClarioNET.

Reader Comments

Add a comment

You said locally linked EXE's were very bad for...