# Effective Reporting in a Client/Server Environment.

Bill Mell
Information Packaging Unlimited
105 Forrest Avenue - Suite 7
Narberth, PA 19072
Voice (610) 617-8820
Fax (610) 617-8825
September 22, 1997

## Reasons for Effective Reporting

- Primary means of getting data out of the database
- Well designed reports excite the customer
- They showcase the work you have put into the rest of the system
- Frequently the reports are the only part of the system that the person authorizing payment for it will see.

## Important Issues not covered

- Report layout and design
- How to use the Report Formatter
- 3rd party Reporting tools
- Report writer
- SQL - Structured Query Language

## Why C/S reporting is fun

- You can do some really cool stuff with reports in Windows.
- SQL allows you to get just what you need in the order you need it, relatively easily.
- You have the opportunity to take advantage of Clarion's most thoughtful compiler directive (OMIT)

## Let's narrow the focus a bit

- CW has native drivers available for AS400 and Oracle (for a price)
- Most Client/Server applications will be utilizing the ODBC driver
- My experience is exclusively with the ODBC driver.
- Guess what I'll be talking about.

## Benefits of ODBC driver

- The database can be directly addressed using {prop:sql} to send SQL statements designed to return data in any order or configuration.
- You can take advantage of the server's horsepower for filtering and calculations minimizing network traffic.

## Limitations of ODBC driver

- Extra processing layer for program. Our experience is that this is a non-issue existing primarily in theory - not a significant performance issue.
- Certain properties of CW views are not supported, most notably {prop:order}
  - You need to have indexes defined in the CW dictionary for any order you wish to use in standard CW views.

## Template issues

- CW Reporting templates use CW Views exclusively.
- CW Reporting templates initiate unnecessary lookups for joined tables
- Omit #1 - embed points for eliminating unnecessary lookups.
- Omit #2 - embed points for eliminating references to the CW View.

## StupidTempTable Theory

- Why do we need this theory.
- How the theory works.
- Implementing the theory.
- Report formatter implications arising from the use of this theory.

## Types of Reports - I

- Reports on a single or joined table using a defined key for presentation sequence.
  - » Standard Report with CW View works fine.
  - » Use Omit # 3 for accurate progress bar.
  - » Use Omit # 1 if it is a joined table.

## Types of Reports - II

- Reports with the same layout of data from a single table presented in different sequences or based on widely divergent filtering criteria.
- Bypass CW View - {prop:sql} on table.
  - » Use Embed #2 to bypass the View references
  - » Use Embed #3 for accurate progress bar.

## Types of Reports - III

- Complex Analytical Reports with returned server calculations or Reports using joins not defined in the CW dictionary.
  - » Use the StupidTempTable.
  - » Bypass CW View - {prop:sql} on table.
  - » Use Embed #2 to bypass the View references.
  - » Use Embed #3 for accurate progress bar.

## Reporting Strategies

1) **Reports on a Single or Joined Table using defined key order** – Standard Report with CW View works fine
   a) Use Embed #3 for setting RecordsToProcess variable.
   b) Use Embed #1 if it is a joined table
2) **Reports with the same layout of data from a single table, presented in different sequences or based on widely divergent search criteria.** – Bypass the View References to report directly on File referenced using {prop:sql}
   a) Use Embed #3 for setting RecordsToProcess variable
   b) Keep all but one file in the 'Other Files' Area
   c) Use Embed #2 to bypass the View References
   d) Use StupidTempTable if join is involved.
3) **Complex Analytical Reports with returned server calculations or Reports using joins not defined in the CW dictionary.** – Bypass the View References to report directly on File referenced using {prop:sql}
   a) Use Embed #3 for setting RecordsToProcess variable.
   b) Use StupidTempTable for all such reports.
   c) Use Queue Structure for reports needing further processing, Group Structure for straight reporting.
   d) Use Embed #2 to bypass the View References.

## Miscellaneous Tips

1) **Alternative to the Progress Bar Display** – There are times when the progress bar is more pain than it is worth, in such cases I have created an alternate display progress window in the template code that displays a changing graphic. I have it as an option in my report properties, so that the code is substituted appropriately when it is selected. You can get as creative as you wish with the graphics . . . use your imagination . . . it's adds fun to your program.
2) **Example of a 'Group By' SQL statement with calculated fields** – Suppose you wanted to report on the sum of the invoices billed to your customers each month since you started keeping records. In non-SQL reporting you would have to process through all of the invoice headers and add it up. You can get exactly the information you need with one simple SQL statement.
   a) Temptable you would return the values to:

```
TempTable  FILE,DRIVER('ODBC'),NAME('StupidTempTable'),OWNER(Glo:ConnectString),PRE(Stt)
Record      RECORD
YearRpt       LONG,NAME('X_INT')                !Server Calculated Field
Account_ID    LIKE(Acc:Provider_ID),NAME('X_INT')
MonthRpt      LONG,NAME('X_INT')                !Server Calculated Field
MonthTotl     LIKE(Inv:Total),NAME('X_MONEY')   !Server Calculated Sum
             END
            END
           END
```

   b) Select statement you would use to return the values:

```
Temptable{prop:sql} = 'select datepart(yy,i.invoice_date),'&|
'a.account_id,datepart(mm,i.invoice_date),'&|
'sum(i.invoice_total) from invoices I, accounts a where '&|
'a.account_id = i.account_id and i.status is not <39>V<39> '&|
'group by datepart(yy,i.invoice_date),a.account_id,'&|
'datepart(mm,i.invoice_date)'
```

   c) The values returned would include the year, account_id, month, and the total of the invoices for that account in that month of that year. The totals returned would depend upon the grouping you use, but it sure beats looping through all the records for the information. The return set can be looped through and reported on just as if it was a normal table.
3) **Example of a Cool Thing you can do with Windows type Reports** – Suppose you were writing some sort of summary report listing detailed information using report breaks on different things with the fields in the detail section being totaled in the break footer. You could give your client a report setup screen that allowed them to set a variable specifying whether or not they wanted to print all the detail or just the totals.
   a) To show just the totals after the report opens set the Height property of the detail and it's fields to 0 and hide the detail fields and let the report print as normal.
   b) The same report can be a summary or a detail depending upon the variable set. Pretty cool!

# Embeds You Can Use

1. **Omits to eliminate extraneous lookups for joined tables**
   a) "**Before Lookups**" – `OMIT('**EndOmit**')`
   b) "**After Lookups**" - `**EndOmit**` *This will allow the compiler to ignore the extraneous lookups generated by the template.*
2. **Omits to use Table instead of View in Report**
   a) "**Declaration Section**" - *After initial code generation (before compile) copy the local variables up to but not including the View Declaration from the source file. Paste the information here, with an* `OMIT('**EndOmit**')` beneath the last line.
   b) "**Data Section, Before Report Declaration**" - `**EndOmit**` *(this will cause the compiler to ignore the view declaration generated by the template).*
   c) "**Window Event: Open Window, before setting up for reading**" – `OMIT('**EndOmit**')`
   d) "**Before first record retrieval**" - `**EndOmit**` - on the next line you will want to place your `FileLabel{Prop:Sql} = 'select statement here'` – *this causes the compiler to ignore the View opening and setting statements for the report.*
   e) "**Top of GetNextRecord ROUTINE**" – `NEXT(FileLabel)` then on the following line another `OMIT('**EndOmit**')`
   f) "**GetNextRecord ROUTINE, after NEXT**" - `**EndOmit**` - *this causes the compiler to ignore the next(view) of the report.*
3. **Get Record Count for Accurate Progress bar.**
   a) "**Before Opening Progress Window**" – `Tablename{prop:sql} = 'select count(*) from tablename where Your conditions here';NEXT(Tablename);RecordsToProcess = FirstColumnInTable` *(must be an integer)*.
   *b)* "**GetNextRecord ROUTINE, after NEXT**" – (if you are using Embed #2 this follows the code listed there) – *Unless you are using a single table report the CW template increments the RecordsProcessed and RecordsThisCycle by BYTES(PrimaryTable) which will negate the positive effect of having an accurate count of the records you are going to display. Thus you can just copy the generated display information and paste it, replacing the BYTES(PrimaryTable) with 1.*

```
IF ERRORCODE()
  LocalResponse = RequestCancelled
  EXIT
ELSE
  LocalResponse = RequestCompleted
END
RecordsProcessed += 1    !BYTES(PrimaryTable) replaced by 1
RecordsThisCycle += 1    !BYTES(PrimaryTable) replaced by 1
IF PercentProgress < 100
  PercentProgress = (RecordsProcessed / RecordsToProcess)*100
  IF PercentProgress > 100
    PercentProgress = 100
  END
  IF PercentProgress <> Progress:Thermometer THEN
    Progress:Thermometer = PercentProgress
    ?Progress:PctText{Prop:Text} = FORMAT(PercentProgress,@N3) &|
        '% Completed'
    DISPLAY()
  END
END
OMIT('**EndOmit**')
```

   c) "**Procedure Routines**" - `**EndOmit**` - *this causes the compiler to ignore the display progress portion of the report, any routines you might write would come below this.*

# StupidTempTable Theory

Developed by Troy Sorzano, Team TopSpeed, Information Packaging Unlimited.

1) Why do we need this theory.
   a) So you don't need to have file relationship established in the CW dictionary for every join you wish to use.
   b) So you don't need to have a key or index established for every ordering statement you wish to use in joined tables.
   c) So you can easily return and manipulate calculated fields from the server.
2) How the theory works.
   a) A validation table (StupidTempTable) exists on the File server with a single field of each data type that is used by your application.
   b) This table does not appear in the CW Dictionary, but is used for validating Local TempTables defined in the code section of procedures. When the local table is defined it uses the NAME attribute for the table itself and for each of the columns you wish to return. CW doesn't care about the order, or size, just that the column name and type exist in the table on the back end.
   c) You can open as many locally defined temporary tables referencing the StupidTempTable as you wish, as long as they have different Labels, but the same NAME. (Be sure to close them when you are done)
   d) You can have as many columns of the same NAME in a TempTable as you wish, as long as the labels are different.
3) Implementing the theory.
   a) Build your validation table on the server. There need be only one.

```
StupidTempTable
X_INT        INT
X_VARCHAR    VARCHAR(255)
X_CHAR       CHAR(255)
X_MONEY      MONEY
X_TEXT       TEXT
Etc.
```

   b) Create a local table in the data section of your procedure.

```
TempTable  FILE,DRIVER('ODBC'),NAME('StupidTempTable'),OWNER(Glo:ConnectString),PRE(Stt)
Record      RECORD
Provider_ID   LIKE(Pro:Provider_ID),NAME('X_INT')
Pro_Name      LIKE(Pro:Name),NAME('X_VARCHAR')
Account_ID    LIKE(Acc:Account_ID),NAME('X_INT')
Acc_Name      CSTRING(40),NAME('X_VARCHAR')
Directions    LIKE(Pro:Directions),NAME('X_TEXT')
Distance      LONG,LIKE('X_INT')                    !for Server Calculated field
             END
            END
           END
```

   c) Retrieve the data into the fields

```
TempTable{Prop:Sql} = 'select p.provider_id,'&|
                      'p.name,a.account_id,a.name,p.directions,'&|
                      'round(sqrt(power(p.Zip_XCoord - a.Zip_XCoord,2) +  '&|
                      'power(p.Zip_YCoord - a.Zip_YCoord ,2)),0) DISTANCE from '&|
                      'providers p, accounts a, acctproviders ap where '&|
                      'a.state = <39>PA<39> and ap.account_id = a.account_id '&|
                      'and p.provider_id = ap.provider_id order by a.name, '&|
                      'distance' .
```

   d) Process the data retrieved as you would any other table