POWERHOUSE STANDARDS

Christopher D. Brayman
Brant Computer Services Limited
2605 Skymark Avenue, Suite 400
Mississauga, Ontario CANADA L4W 4L5
Telephone: (416) 9790

INTRODUCTION

During the years that COBOL became the standard of third generation languages (3rd GL's), the industry adopted many standards in structured programming and system design methodologies as MIS departments scrambled to protect themselves from the whims of creative, but inconsistent, computer professionals. The "techies" of this language generation were forced to standardize to protect companies' often huge investments in computer systems.

Standards were developed, such as "Structured Warner-Orr", for writing COBOL applications. In addition, much attention was placed on design methodologies resulting in a multitude of textbooks describing the best approach to systems analysis and design. Certain leaders emerged, such as "Yourdon", who managed to capture all components of the system development lifecycle (SDLC) in a fashion we could understand.

Today, we are still struggling to understand and deal with the impact of fourth generation languages (4th GL's) on traditional programming standards. Certainly, 4th GL's have changed and have eliminated many of the problems found in 3rd GL environments. The inherent organization and structure of the POWERHOUSE products, the leading 4th GL on mini-computers world-wide, have reduced significantly the risk that a creative "techie" will build an application that no one else could possibly maintain. However, even with this structure, I have seen some radically different looking POWERHOUSE code over the past five years.

In our company, we have a variety of projects ongoing in different computer languages. As a professional service organization offering a variety of solutions to businesses in the HP environment, we employ the "old" and "new" schools of computer professionals. Applications are being developed or maintained in low-level languages such as ASSEMBLER, in 3rd GL's (COBOL, RPG, FORTRAN, PASCAL), and 4th GL's (POWERHOUSE, SPEEDWARE). Regardless of the level of programming language, one constant remains:

> Standards in program development are necessary to ensure consistancy between programmers. No matter how well structured the language, "techies" will always find a way to impose their creativity and do it differently.

The intent of this technical paper and subsequent presentation at the IUG in Orlando is to discuss some of the internal standards we have adopted in our POWERHOUSE applications on HP3000 MPE boxes. Some of these standards you may agree with, others you may not. We make no claim to know the "best way" since, by their very nature, standards are simply that -- "standards".

I. POWERHOUSE PRODUCTS

POWERHOUSE is a family of fourth generation language products authored by Cognos of Ottawa, Canada. The software is comprised of a dictionary, a report writer (QUIZ), a menu and screen generator (QUICK) and a transaction processor (QTP). In addition to these development products running on a variety of hardware platforms, Cognos offers the following applications: POWERHOUSE Graphics, an end-user Report Writer (The Expert); a financial accounting package (Multiview); a spreadsheet program (Powerplan); and, a development and documentation tool (The Architect).

General Standards

1. *A minimum level of source program documentation is required* for all QUICK, QUIZ, and QTP programs. This includes:

   - o name of program (MPE filename, group account)
   - o version number
   - o name of programmer
   - o title of program
   - o expanded description
   - o dates created and modified
   - o explanation of major changes

   NOTE: Use the DESCRIPTION verb for expanded descriptions in QUICK screens.

2. If KSAM records are not very static, the binary trees must be frequently rebuilt which is CPU-intensive. As well, file-locking is handled differently for KSAM files (in comparision to IMAGE)

since locks occur for the file around a write, update, and *read. As such, KSAM files are only used for generic keyword searches.*

3. Security specified at the file and element levels inside IMAGE is restrictive and costly to change as data must be unloaded and reloaded to a rebuilt database. Therefore, file and element level security should be kept as simple as possible. *Apply security using POWERHOUSE application security.*

4. Any incoming data from external systems should go through a set of validation checks prior to entry of the production database. The following steps are typically performed:
   - o run QTP batch edits against data
   - o add good records to production base
   - o report rejected records
   - o add rejected records to a correction database and use QUICK validation screens to modify and edit rejected records
   - o rerun QTP batch edit against data and add corrected records to production base.

   NOTE: We try to encourage corrections at the source from external systems.

5. *The use of USE files inside source programs for handling standard system-wide calculations or global hilite options is encouraged.* The most common example is for screen hilite options maintained in one source file:

   example:

   ```
   HILITE DATA INVERSE UNDERLINE
   HILITE ID INVERSE
   HILITE MESSAGE INVERSE UNDERLINE AUDIBLE
   ```

   In this way, the hilite options for a system of screens can be changed by modifying the one USE file source statements and recompiling all the screens in the system.

   This feature is not a new one. The COPYLIB in COBOL environments provides a similar capability.

6. When constructing DEFINE statements in QUIZ and QTP where the value of the defined item is calculated based on the values of multiple items or expressions, use the "IF...ELSE" structure. If it is based on the value of only one item, use the "CASE...WHEN" form.

7. Benchmark testing has shown that it is typically faster to extract data into subfiles by QUIZ rather than QTP. Therefore, use QUIZ for creating subfiles where possible.

8. In large systems, it can often become difficult to differentiate among database files and elements and other temporary variables, defined expressions, and alias files. Therefore, we use standard prefixes as follows:

   T -- for temporary variables
   D -- for defined variables
   A -- for alias files

QDD Standards

1. Increase the blocking factor on QSCHEMAC's.

2. Include global options for things like date formats at the beginning of QDD source files.

3. Use RELEASE and VERSION verbs to control enhancement releases. This is especially important in large, complex and dynamic applications.

4. Include descriptions for all files.

5. Include descriptions and help messages for all elements.

6. Any logical edits and display functions for elements should be included in the dictionary.

   example:

```
ELEMENT CASH-AMOUNT 9(008)V9(002) &

   HEADING "Cash^Amount"      &
   LABEL "Cash Amount"        &
   SIGN LEADING "-"           &
   PICTURE "^^^,^^^,^^"       &
   FLOAT "$"                  &
   VALUES - 100 TO 100000     &
   HELP "this field represents the cash"
        "amount of order detail transactions
```

7. Use common element names for like items.

   example:

   DESCRIPTION

   rather than always creating unique items with names such as INVOICE-DESC, PRODUCT-DESC, etc.

QUICK Standards

1. Major verbs in QDESIGN should be placed in the following order in source programs:

   SCREEN
   TEMPORARY
   FILE
   TEMPORARY
   DEFINES
   ITEMS
   GLOBAL HILITES (use file)
   TITLES (as they occur)
   FIELDS (as they occur)
   PROCEDURES (recommended order from QUICK manual)

2. The QKGO file establishes a wide range of run time parameters for the QUICK screens. Default values are set for all parameters with the ability to increase or decrease values according to the application requirements. System programmers must analyze the machine enivronment for available memory, input/output limitations and CPU power. *In larger complex POWERHOUSE applications, parameters should be tuned to ensure optimal use of stack, internal buffers and extra data segments.*

   As a general rule, the programmer should analyze each user group in an application for resource requirements and establish separate QKGO files. Different users will access only a few screens and others may access many. Different screens may have very different requirements for work areas in stack.

   Special attention should be given to the following parameters:

   A. Application Lines

      This parameter involves the stacking options of the SCREEN statement, the application lines QKGO parameter and available

terminal memory. The parameter controls the number of lines of simulated terminal memory used for stacking QUICK screens.

If a user moves between a small number of screens, stack all the screens on different 24-line blocks of terminal memory.

In large application systems where a user moves unpredictably through screens, map screens according to levels. The master menu is mapped onto application lines 1 to 24. All menus and screens at level two are mapped to application lines 25 through 48. This is repeated to the deepest level in the system.

B. Procedure Code

This parameter sets the number of 256-byte pages reserved for procedure code in the user stack. Procedure code records from the compiled screen file on disc are read when an associated function is requested (i.e. the user types "E" in the ACTION FIELD and QUICK moves the Entry Procedure into stack). Records are moved into this allocated space according to a paging system maintained by QUICK.

If the parameter setting is less than the number of required procedure code records for the activity on the screen (the threshold level), subsequent activities will require at least the threshold level of reads to the screen file on disc. *Therefore, it is essential that user's busiest screen be analyzed for its threshold level and the parameter for procedures code be set to this value.*

C. Rollback Buffer, Secondary Blocks, Segment Size, and Screen Table

These four parameters work together to control a secondary "paging system" maintained by QUICK in extra data segments. The paging system is used to store spillovers from stack when rolling back updates and screen table information. It is much faster to load screens from extra data segments rather than the original compiled screen file on disc. As such, performance can be improved in on-line applications where large "paging systems" can be maintained.

Therefore, if the machine environment has sufficient memory, increase the size of the "paging system", as required by the application. Typically some form of stack monitoring tool will be necessary to analyze this environment.

3. *Use default screen processing where possible.* Avoid unnecessary procedure code that must be paged in and out of stack as part of the procedural code "paging system". Apply editing to fields with verbs such as VALUES and PATTERN directly.

   example:

   When doing a conditional lookup to a reference file, use an associated DISPLAY or SILENT field to perform the conditonal processing with a VALUES verb, this technique could be used rather than writing procedural code with GETS, etc.

```
FIELD CUSTOMER-NO OF PROJECTS &
        LOOKUP ON CUSTOMER-MASTER

FIELD CUSTOMER-STATUS ID SAME DISPLAY &
        VALUE "A"
```

   NOTE: This approach would be appropriate where a custom error messages was deemed unnecessary by the designers, and excessive volumes of procedural code was otherwise required in more essential processing on the screen.

4. *Where the logic of the QUICK screen requires that procedural control over reading and updating of a file must occur, use DESIGNER files.* Avoid declaring files as SECONDARY when none of the fields appear on the screen and accessing the files only need occur under certain circumstances.

5. *Use REFERENCE files only for LOOKUP ON.* If a procedural GET is required, use a DESIGNER file.

6. *Use the DETAIL file for one-to-many relationships on a single screen format,* rather than two screens with the second screen declaring the primary file of first screen as a MASTER, etc.

   DETAIL files avoid the unnecessary loading of a separate screen into the user stack and extra data segments.

7. *Use the ALIAS file in the following circumstances:*

   - o accessing a file on a different path or mode
   - o changing multiple key vlaues in IMAGE chains

8. Use the DELETE files with caution because of the application implications of automatically deleting detail transactions. As well, these deleted records must be rolledback into the Rollback

Buffer in primary stack, then to extra data segments or temporary disc files as necessary. *Negative performance may occur with large DELETE files.*

9. It is important to understand the field processing cycle within QUICK as well as the use of FIELDTEXT and FIELDVALUE during this cycle. Four QUICK procedures are used to control the cycle in the following order:

INPUT ----> EDIT ----> PROCESS ----> OUTPUT

i) Input. This procedure is used to manipulate data before any editing is performed by the EDIT procedure. *Any changes to the user-entered value should be made to FIELDTEXT.*

ii) EDIT. This procedure is used to perform any additional editing after those performed by field verbs such as VALUES and PATTERN as part of the ACCEPT verb. There are two values associated with the field. The new value entered by the user should be referenced by FIELDTEXT (for character items) or FIELDVALUE (for numeric and date items). *The old value should be referenced by OLDVALUE (fieldname) to address the value in the record buffer.*

iii) PROCESS. This procedure is used to perform calculations after the newly entered value has been placed in the record buffer. It immediately follows the EDIT procedure. *The value of the field should be referenced by the actual name of the field.*

iv) OUTPUT. This procedure is used to modify data between storage in the record buffer and output back to the terminal screen. *The screen display is altered by modifying the value of FIELDTEXT.*

10. *Proper qualification with file references using "OF filename" for all fields in FIELD verbs and procedural code items should be done.*

example:

```
FIELD AMOUNT OF CASH-RECEIPTS

PROCEDURE PROCESS AMOUNT OF CASH-RECEIPTS
BEGIN
     IF AMOUNT OF CASH-RECEIPTS > 1000
          THEN BEGIN
          .
          .
          .
          END
END
```

Proper qualifications as described will avoid errors for ambiguous file references when adding new files to the processing of existing screens.

11. *Include PUT verbs for DESIGNER files in the UPDATE procedure* to ensure that the DESIGNER file is rolledback automatically by QUICK, if an error occurs during the PUT to the database. When performing these PUTS outside the UPDATE procedure or when modifying the UPDATE procedure directly, use the STARTLOG and STOPLOG verbs (as QUICK normally does in the default UPDATE procedure) to control IMAGE logging if it has been enabled.

12. Very complex screens using numerous file structures and lots of procedural code for processing and edits will perform more poorly than simple screens. *Therefore, adopt the KIS principle (Keep It Simple) in the design of individual screen programs.* Spread the processing of multiple files over multiple screens where possible.

13. The design of a QUICK screen hierarchy should balance the logical requirements of the application with the size of the stack required for deep structures. Some logical structures may encourage a very deep hierarchy and conflict with the stack limitations of the machine environment.

    If the logical structure does not demand a deep hierarchy, consider building shallow structures. The savings realized by this design approach can be used to increase work areas in stack and improve system performance.

14. Consider using an external call to a 3rd GL subroutine in the following circumstances:
    i) when on-line edits and/or procedural processing involves very large volumes of procedural code;

ii) where a standard routine is accessed by many users concurrently and requires immediate response.

QUIZ Standards

1. Major verbs in QUIZ should be placed in the following order in source programs:

```
ACCESS
DEFINE
CHOOSE
SELECT FILE
SELECT IF
SORT
REPORT
NOREPORT
FOOTING (in order of control breaks)
FINAL FOOTING
INITIAL HEADING
HEADING AT
PAGE HEADING
(MPE FILE STATEMENTS)
SET
BUILD
```

2. Production reports should be kept in compiled form.

3. Fully qualify all linkages in ACCESS statements and all elements used in the report.

   example:

```
ACCESS EMPLOYEE-MASTER LINK EMPLOYEE-NUMBER TO &
     EMPLOYEE-NUMBER OF TIME-RECORDS
SORT ON EMPLOYEE-NUMBER OF EMPLOYEE-MASTER &
     ON FUNCTION-CODE OF TIME-RECORDS
REPORT EMPLOYEE-NUMBER OF EMPLOYEE-MASTER &
     FUNCTION-CODE OF TIME-RECORDS &
     HOURS-WORKED OF TIME-RECORDS
BUILD
```

4. Wherever possible, use "SELECT file IF" rather than "SELECT IF", since fewer evaluations must be performed to decide if the record complex should be kept or the individual file read.

5. Divide complicated reports into two passes. Extract and select a minimum set of records in the first pass. In the second pass, report information as appropriate.

6. Logical element characteristics such as headings, picture clauses, and floating dollar signs should be included in the dictionary.

7. When using "SELECT IF", specify selection based on items higher up in the ACCESS statement first (primary file first). This allows QUIZ to eliminate records based on partially satisfied selection conditions.

   example:

```
ACCESS EMPLOYEES LINK EMPLOYEE-NUMBER TO &
     EMPLOYEE-NUMBER OF TIME-RECORDS

     SELECT IF EMPLOYEE-STATUS OF EMPLOYEES="A" AND
     FUNCTION-CODE OF TIME-RECORDS = "77"
```

   In this way, records from TIME-RECORDS will only be read if the first condition is met; that is, if the Employee Status of Employees is equal to "A". As well, when multiple conditions are on the same file level, they should be placed according to the most likely condition.

8. Always include a NOREPORT verb in QUIZ reports to specify the report contents if no record complexes are reported.

9. Conditional expressions in DEFINE statements should be organized so that the most likely conditions are evaluated first.

QTP Standards

1. Major verbs in QTP should be placed in the following order in source programs:

```
RUN
GLOBAL TEMPORARY
REQUEST ONE
ACCESS
TEMPORARY
DEFINE
CHOOSE
SELECT FILE
SELECT IF
```

```
SORT
OUTPUT
ITEMS
SUBFILE
MPE FILE STATMENTS
SET
REQUEST TWO
.
.
.
BUILD
```

2. All QTP programs should be tested using a QUIZ program equivalent to ensure the proper understanding of record complexes that are constructed for the OUTPUT phase.

3. Production QTP runs should be kept in compiled form.

4. Fully qualify all linkages in ACCESS statements and all elements used in the run.

5. Conditional expressions in DEFINE or ITEM statements should be organized so that the most likely conditions are evaluated first.