Table of Contents

A Report on Report Writers

Stewart Hill
Hewlett-Packard Co.
Computer Language Lab
19447 Pruneridge Ave.
Cupertino, CA, 95014

INTRODUCTION

Reports have long been an integral part of business computing. For many years, report programs could only be written by trained programmers; such programs can be difficult to write and tedious to verify. But in recent years a growing number of languages and products have made report writing much easier. Many more people can now create and produce reports, even without training in standard programming languages.

Report Writers come in many different forms and with vastly different user interfaces. They range from statements imbedded in a programming language to forms-driven stand-alone report products. With the advent of friendly report writers, several questions must also be raised: 1) What kind of performance is achievable with a report writer? 2) What features should be expected from a report writer, and how might these vary from product to product? 3) Which users should reasonably have access to a report writer, and what users do not need such access?

This paper addresses the last two questions: what report writers do, and who should use them. Performance is addressed only in a general manner, as it is not within the scope of this paper to compare actual performance data. Such data has been obtained previously[1], and it is subject to change.

Several different report writers were examined in preparation for this paper, though these are by no means all that are available. The intent of this examination is not a product comparison, but an attempt to present the various implementation styles. This type of investigation brings out both the similarities and the differences which can be found in report writing systems. Additionally, these products are intended for different types of users, and looking at them side by side helps delineate the intended audience.

Each of these products stands on its own merits, and no judgements on them will be made. Figure 1 is a list of the products examined. These were chosen because reference documentation was readily available and because they represent the current range of user interfaces to report writers. The "style" listed is very general. The fourth generation languages do not all use the same style of interface.

---

[1] A Performance Comparison of HP3000 Report Writers, Roger W. Lawson, September 1987 INTEREX (Business Users) Proceedings

PRODUCT	COMPANY	Style
HP Business BASIC	HP	Imbedded 3GL
Powerhouse(r) QUIZ	COGNOS	4GL
HP Visor	HP	Forms-driven/SQL (HP-UX)
The WRITE STUFF	PROTOS	4GL
Business Report Writer	HP	4GL Report Specific
ASKPLUS	COGELOG	QUERY-based
COBOL Report Module	ANSI	Imbedded 3GL

Figure 1: Sample Report Writer Products

REPORT WRITER FUNCTIONALITY

What is a Report Writer?

To answer this question, one must first determine what is meant by a "report." In general, all output can be considered a report, no matter how unstructured it may be. However, such output is uncontrolled and far too general a definition to use. Instead, let us define a report to be the structured, formatted output of repetitive data, plus the ability to summarize that data. This definition certainly does not cover all reports, but it does describe the concepts behind most reports.

A report writer is a powerful controller. Its primary purpose is to control the structure of the output and the formatting of data within that structure. In addition, a report writer provides summary data for use within the report itself. The actual workings of a report writer are complicated and can be difficult to understand. Nevertheless, these controls relieve the programmer of an enormous amount of bookkeeping, which leads to better programs in less time.

To accomplish its goals, a report writer requires three distinct parts: a page layout controller, a data formatter, and a data retriever. In some products, these pieces are closely linked and are intended to be used only with one another. Other report writers provide these parts as independent controls which may be used with or without one another. Whatever the implementation, these controls are able to produce reports ranging from the very simple to the very complex.

Page Layout and Summary Controls

Report writers have two sets of controls for structuring report output. One set defines the layout of a page, which usually corresponds to a printed piece of paper. The other set controls the overall report, including the grouping and summarizing of data. All of the examined report writers provide these controls.

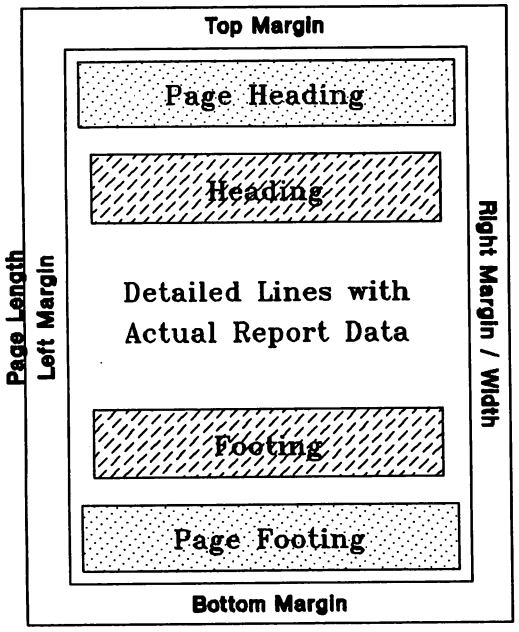

Figure 2: Page Layout

Figure 2 shows a typical report page, including some blocks associated with groups of data. All of the examined report writers define these parts of a page in some way. They also provide controls for customizing each area of the page. The terms used to describe the page layout are:

- **PAGE LENGTH**: The total number of lines on a page.
- **LEFT MARGIN**: The number of spaces printed before each line of output. Most report writers include this in the format of each line, rather than providing a specific control statement.
- **RIGHT MARGIN (or WIDTH)**: Indicates the last column in which data can be written. This is usually based on the output device.
- **TOP and BOTTOM MARGIN**: The number of blank lines which occur at the top and bottom of every page. Many report writers combine these with the PAGE HEADING and PAGE FOOTING controls, rather than having separate controls.
- **PAGE HEADING**: Defines data to be printed at the top of each page.
- **PAGE FOOTING**: Defines data to be printed at the bottom of each page.

- **HEADING**: Defines data to be printed before each summary grouping. (Described below)
- **FOOTING**: Defines data to be printed after each summary grouping. This area is often used to display totals or counts of the preceding detailed data.

Some additional controls for the layout are:

- **PAGENUM**: The current page number.
- **SET PAGENUM**: A means of modifying the current page number.
- **NEED**: A means of specifying that a certain number of lines must be on the page before the next output takes place. A new page is started if fewer lines are left on the current page.

NOTE

The terms above do not represent syntax from any one report writer. These terms are commonly used and tend to convey their intent directly.

Among the report structure controls, the most important feature is the ability to define groups of data, called summary groups. These groups are usually based on a change in the value of a data base field or a variable. Such a value change causes a break, at which time the report writer may take some action. These breaks constitute the heart of report writer control, allowing great volumes of data to be printed and summarized in meaningful (and readable) reports.

Reports may use several breaks at the same time; these breaks are nested to provide different levels of detail with the report. The key to understanding report writers is understanding how these nested breaks work. This is relatively simple, but not necessarily straightforward. A careful reading of a manual, along with some hands-on experience, should be enough for most people to learn about a report writer.

The number of break levels (also called summary levels) available in the report writers varies, but nine levels appears to be the minimum number. Report writer controls allow a HEADING and a FOOTING to be defined at each break level, and summary data and automatic totalling can be kept for each level. These controls provide the programmer with great flexibility for designing the look of a report, and the ability to put out meaningful data.

The typical report structure controls are:

- **REPORT HEADING**: Defines data to be displayed only at the beginning of the report. Cover pages and introductions may be defined here. Not all report writers provide this facility.
- **REPORT FOOTING**: Defines data to be displayed only at the very end of the report. Often used to display grand totals for a report. All examined report writers provide this.
- **BREAK**. Defines the data base field, variable, or expression to use for grouping data. Multiple BREAK levels are allowed in order to define different levels of detail.

- **TOTAL**: Defines fields, variables, or expressions to be totalled automatically. This can be done at each break level. Usually a function is defined for printing this total, although some report writers do this automatically.

- **COUNT**: Provides an automatic count of the number of detailed items printed in a group. This value can be accessed at each break level.

- **AVERAGE**: Just returns TOTAL/COUNT. Almost all report writers provide this function.

- **REPORT**: Tells the report writer to process detailed data. This activates checks for BREAK controls, page boundary checks, and totalling. Some report writers provide this control automatically.

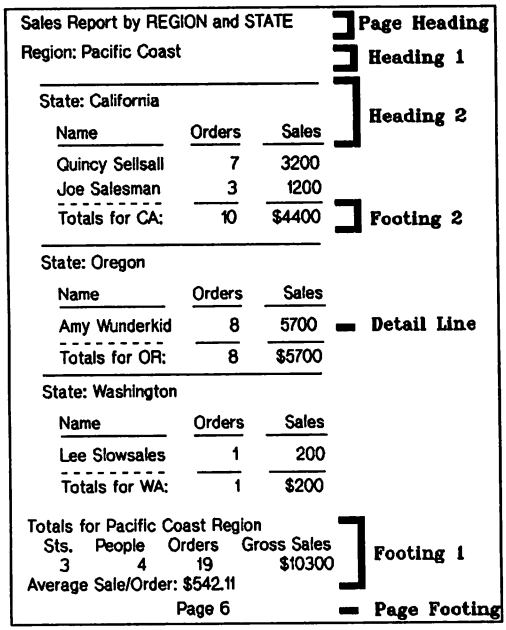

Figure 3: Report showing nested BREAK levels.

The example page shown in figure 3 is from a simple report which uses most of the report writer control features. In particular, two different break levels are shown, one for regions and one for states. The most detailed level is that for states; the employees and sales figures are shown at this level. The region level provides less detail: the only real information at this level is in **Heading 1** and **Footing 1**. There may be other break levels defined in this report as well. An example of this would be an even less detailed break, such as by country.

The program in figure 4 shows the control features that would be used to produce the example report page in Figure 3. Most of the actual printing has been left out, except to show the use of some control features. (The syntax used has been made up; however, the clarity of the syntax is typical of report writers which do not use visual interfaces.)

```
PAGE HEADING USES 2 LINES
      PRINT "Sales Report by REGION and STATE"
      BLANK                                          ! prints a blank line
PAGE FOOTING USES 1 LINE
      PRINT "Page " PAGENUM, CENTERED
HEADING 1 USES 2 LINES: BREAK IF Region CHANGES
      TOTAL FOR: Sales
      AVERAGE FOR: Sales
      COUNT OF: State, Orders, Employee
      ...
FOOTING 1 USES 5 LINES
      ...
      PRINT COUNT(State) COUNT(Employee) COUNT(Orders) Total(Sales)
      ...
      PRINT AVERAGE(Sales)
HEADING 2 USES 5 LINES: BREAK IF State CHANGES
      TOTAL FOR: Sales
      COUNT OF: Orders
      ...
FOOTING 2 USES 2 LINES
      ...
      PRINT "Totals for" State ":" COUNT(Orders) TOTAL(Sales)
...
! Now select the data from datasets and print the report
CHOOSE Region
      CHOOSE Employee, State, Orders, Sales
         SORT BY Region, State
         REPORT
```

Figure 4: Report Program

Additional Features

The controls listed above only define the minimal set needed for a good report writer. Most report writers provide some additional capabilities. Each new function only enhances the ability of a report writer to produce the desired output. One should always consider these features when attempting to choose a report writer. Fortunately, most capabilities beyond those listed can be found in several report writer products.

Typical examples of new capabilities are the addition of HIGH and LOW functions. These functions trace the highest and lowest value of break expressions. Such a function is not provided in all report writers directly, but the functions can be very useful. Another example is the addition of statements to suppress output. Such statements can be used to produce summary reports without defining a new report. Again, not all report writers have or need such statements.

Format Controls

Format controls provide the mechanisms for defining both line formats and item formats. Line formats control what each unique line of a report should look like, including the positions of data items in the line. Item formats control the display of each data item in a line, such as the format for numeric output. Since report requirements vary considerably, the formatting portion of report writers must be flexible and powerful.

Line Formatting

Line formats usually fall into one of two categories: "guide line" specifications and "columnar description" specifications. The type of specification depends upon the user interface for a particular report writer product. One could argue all day about which interface is better, but in reality both provide sufficient power for defining very complex reports. In any case, every unique line must be defined and then referenced by the report.

Guide Lines use a visual definition for each report. The user enters the line exactly the way it should be printed, using special markers for printing data in the line. Depending upon the product, this may occur directly in a file or report, or this may be done on a terminal. The report writer is then told when each line should be printed.

Columnar Descriptions combine data item formats with special functions to control the line format. The data item formats specify the field length for each item. Typical special functions are TAB, which skips to a particular column, and SPACE, which prints a specified number of blanks. The line format is built by indicating in what order data should be printed, with the output functions specifying the columns in which output should begin.

Some report writers provide a default output format. In these cases, the size and type of the data control the line format. In addition, the headings and footings may be given default formats as well. This is a convenient feature for accessing data quickly, but most reports will override these defaults to produce a better looking report.

Data Item Formatting

Virtually all report writers provide advanced formatting capabilities for individual fields. This allows the programmer to shape the report so that numeric and alphabetic data are placed properly in the report output. There are many ways to provide the formatting power needed, and each report writer provides its own specification technique.

Many report writer products use the COBOL **PIC** descriptor, or something very similar, to format data. This lends some familiarity to the report writer for many programmers and still provides a very flexible formatting system. Other report writers may mix COBOL formatting with their own formatting, or even provide a new type of format. Not much time is required to learn data format controls, so this should be of little concern to new users.

Data Retrieval

Reports are generally used to display large volumes of data, or to display small pieces of data from a large set. This implies that most reports will use data from a data base, rather than from files or from keyboard entry. Given this, it is understandable that most report writers are geared toward making data base access fast and simple.

All but one of the report writers examined run on MPE, and therefore use IMAGE or TurboIMAGE for data base retrieval. HP Visor runs on HP-UX, and it uses a relational data base. Syntactically, access to the data base differs greatly among these products. For example, COBOL and HP Business BASIC use intrinsics and built-in statements; HP Visor uses SQL or a form-driven access; the fourth generation languages all define their own statements for selecting data.

What became clear during this investigation is that the language type of the report writer is very important. That is, fourth generation languages (including HP Visor) provide easier access to the data base than third generation languages. Report programs in third generation languages are required to know much more about the data base definition than programs using the fourth generation languages. This generally means that 4GL report programs can be written faster and (arguably) more clearly than equivalent 3GL programs.

One fact is crystal clear. No one should have to write reports *without the aid of a report writer*. The bookkeeping combined with accessing the correct data makes for a very complicated task. Such programs are much harder to maintain and change than *any* program that uses a report writer.

Third Generation Language Access

There are two parts to data base structure: the definitions of and relationships between datasets (tables in a relational system), and the format of the data in each set. In a database like TurboIMAGE, the "format" includes the location of the data in a record as well as its size and type. This information must be made available to any programs which access the data base.

Standard languages usually access data bases through intrinsic or library calls. Some languages contain an interface to hide the details of these calls, but the net result is still a system call. The task of making these calls correct is left to the programmer. This implies that report programs assume or are told a great deal about the data base structure.

A report program may determine data base structure in many ways. Normally, the program source code tells which datasets to use, and how to link them together. Few programs verify the linking information at run-time. The data format may be obtained from a dictionary interface, or the programmer may code this directly. In any case, the programming language itself knows nothing about the data base, and the report program knows no more than the programmer gives it.

Third generation report programs are implicitly dependent upon the data base structure. The data base and the programs may change independently, and these changes are not automatically reflected in both. This may lead to errors when reports are finally produced. On the other hand, data base structures do not undergo radical changes often; such changes usually include time to update associated programs.

Fourth Generation Language Access

Fourth generation languages access data bases in a variety of ways. While most of the features turn out to be the same, syntax and performance vary greatly. Yet even more striking are the differences from third generation language programs.

As one should expect, fourth generation programs must specify what datasets (tables) to use, and which fields are needed by the report program. Unlike their third generation counterparts, however, these programs often attempt to define an automatic link between the datasets. In addition, these languages determine where the fields occur in the dataset, along with the size and type of each field. This field information is obtained without user intervention or specification.

The programmer's job becomes much easier once the report writer itself has access to the data base structure. Data fields may be used without regard to *exactly* how the data is retrieved. The process of

linking multiple datasets becomes much simpler, even to the point of not being needed. The report program no longer needs explicit calls to retrieve data, as this is done automatically by the language statements. This allows the programmer to concentrate on the data required and the report format itself.

Most fourth generation languages provide easy data base access through the use of a dictionary or a schema definition. This interface may use a standard dictionary product or may require a separate definition file. Regardless, the languages can use this definition at run-time (or compile time) to provide the dataset linkage and the data formats. If minor (or even major) changes in the data base structure take place, the report programs take this into account automatically (or require a simple recompilation). Program source changes are required far less often.

Non-Data-Base Data Access

Data bases are certainly not the only source for report data, especially when the data is transmitted from an outside source. Data may reside on tape, in files, or be sent to the report program directly. In such cases, the data might be structured, but may also occur as "free-form" input.

Report writers provide a wide variety of access to alternate data sources. This ranges from accepting input from any source to restricting input to a data base. Third generation languages tend to be more adept at accepting any type of input; their I/O is normally disjoint from the report writer itself. Some fourth generation languages also do well, while others require a fixed format for each record processed. Still other products are based solely on access to a data base; in these cases, the data must be merged into a data base before the report can be generated.

Access to file and other source data is of little concern to most operations. The vast majority of reports are generated from data bases. This explains why most report writer products provide generous and easy access to data bases instead of concentrating on general input.

Performance Considerations

As with any program or language, performance is an important issue. Unfortunately, report writers are not always well understood, and consequently take the blame for poor performance. Report performance can be measured and in many cases improved. To be realistic, one must examine the three major pieces of report writers to determine where performance may suffer.

Report Writer Controls

The cost of the automatic report writer controls can be measured, albeit with some difficulty. To truly define their cost, however, a report program must be compared to an equivalent program which does *not* use a report writer. As stated earlier, writing such a program can be quite difficult.

Report writers pay a small penalty for providing generalized control statements. These statements can produce virtually any report, but this fact precludes many optimizations for specific reports. When a report writer is not used, a program may take advantage of knowledge about the incoming data or about the report layout. This allows a specific report program to minimize the data value checks and page checks which must take place during report output. On the other hand, very complicated reports may require just as many checks, and optimization may not be possible.

In general, report writer controls do not significantly decrease performance. Most of the work performed by these statements must be duplicated in hand-written report programs. The increased development and maintenance time for such programs outweigh the overhead costs for report writer controls.

Formatting

Performance improvements are indeed possible during data formatting, although the amount of improvement is very language dependent. Many report writer products give the user great latitude in controlling the output. More importantly, the improvements which might be made in a hand-written program are generally available in the report writers as well. In other words, the non-report-writer programs will not fare much better than a typical report writer program when it comes to data output.

The most likely way to improve report output speed is to change the report format itself. This allows faster data output specifications, thereby significantly reducing report output time. A typical situation occurs with currency signs in reports. The currency symbol can usually be "floated" so that it prints adjacent to a number; printing the symbol in a fixed location is much easier for the report writer. If currency symbols are printed on every line of a report, changing the format to use a fixed column (or even remove the currency symbol) will allow the report to be produced faster.

Data Retrieval

Data base access is by far the most critical performance aspect of any report writer. A report program may spend anywhere from *fifty to ninety percent* of its time in the data base! Obviously, any changes which can reduce this time will impact performance significantly.

Report writers do not generally prevent better data base access. Indeed, they should give the programmer more time to think about data base performance. Unfortunately, report writers sometimes make data retrieval too easy. The very powerful data base statements allow such easy retrieval that performance is overlooked. This fact becomes obvious when a report writer program replaces an old hand-written report program; the old program probably considers every aspect of performance, while the new program simply produces the same output without regard to performance.

There are some typical situations which occur during data base retrieval. One of the most common occurs when a program is based upon a request for a report, but the programmer does not closely consider the amount of data being retrieved. For example, suppose the following request is made:

```
Produce a report listing all orders over $10,000
which are 90 days or more past due.
```

This report is quite simple to produce, especially if the report writer has date arithmetic (many do). For example, the selection might be:

CHOOSE Order_amount >= 10000 AND DATE - Payment_due_date >= 90

There is nothing wrong with the selection criteria above. But suppose that the average order for this company is more than $10,000, and most of their customers pay on time. What performance can be expected in this case? The report writer will first find every order of more than $10,000, which will be most orders; then the check will be made for the number of days the bill is overdue. If this information resides in separate datasets, performance will be poor.

NOTE

Report writers often implement *partial evaluation* for selecting data. This means that the evaluation of a second selection criteria depends upon the results of the first criteria. In the case above, the payment due date will not be checked unless the order amount is $10,000 or more.

In order to improve on this, the programmer must consider the likelihood of satisfying each condition; the report writer itself cannot do this. Performance can be increased greatly by changing the selection criteria:

CHOOSE DATE - Payment_due_date >= 90 AND Order_amount >= 10000

Now the order amount will only be retrieved if the payment is overdue. Since this does not apply to most orders, fewer data base records will be read.

Not all report writers allow absolutely optimal access to a data base. The more control a report writer gives to the programmer, the higher the chances of peak performance. But again, this must be traded off against the increased development and maintenance time for such a program. And the most important point is that programmers should not overlook performance just to get the report produced.

Third Generation vs. Fourth Generation Language

Historically, third generation languages yield report programs that run faster than fourth generation programs. While constant improvements are being made to fourth generation languages, one must consider this factor when deciding which report writer to use for a particular report. (Report generators which produce third generation programs should not be considered fourth generation products in this case.) In many cases, the performance gains here are not significant enough to consider changing from one language for another. These issues should be raised before the report program is started. But when performance does become critical, one must look at all the alternatives.

The friendliness and power of fourth generation languages comes from their ability to hide many implementation details from the user. That is, they let the user specify what to do without worrying too much about how things will be done. Unfortunately, this very aspect of the fourth generation languages means that they must typically do more work than an equivalent third generation language. Optimizations are more likely in third generation programs as well, since the programmer has direct access to the executing code. The net result is that third generation report programs run faster.

Many fourth generation languages interpret their programs, and they do this very quickly. Most third generation languages are compiled (some fourth generation products supply a compiler also). A compiled program can significantly outperform an interpreted program. For report programs, the performance difference depends upon how much time is spent in the data base; the difference is greater if less time is spent retrieving data. Again, these factors must be examined carefully before embarking upon a reimplementation of a report; the performance improvement might not be worth the time spent to rewrite the program.

MATCHING REPORTS, USERS, AND REPORT WRITERS

Clearly, many report writers provide the common functions needed to produce a report. But this does not mean that all report writers are equally easy to use, or that all types of users can make efficient use of

any report writer. Different products are intended to be used by different users, and performance considerations may also influence the decision of what kind of report writer to use.

NOTE

The discussion that follows concerns the use of a report writer to create reports, not just to run a report. Only performance issues are relevant to the actual execution of a report.

There are several factors which influence the decision to use a particular type of report writer. Some of the important aspects of this decision are: the type of report, user background, user interface, and performance. The influence of each of these factors must be weighed when deciding what report writer should be used.

Users are not easily categorized, as their experience ranges from systems programmers to completely non-technical users. Proper report writer usage does depend upon the user's background; therefore, some groups must be defined. For this discussion, four types of users are defined: technical, semi-technical, occasional, and end users.

Types of Reports

Reports can be classified into a few fuzzy categories. These groups help determine the amount of work and the amount of experience needed to produce a report. A report may move from one group to another as modifications are made to the report, but the categories will still reflect the total amount of effort required. For descriptive purposes, these categories will be used: simple query-based reports, simple repetitive reports, complicated infrequent reports, and complicated repetitive reports.

Simple Reports

The query-based report represents the "on-the-fly" report, in which any data may be needed at any time. These reports are often run one time, or at irregular intervals. In many cases, a permanent report program cannot be written for such a report. Even when this is possible, this type of report may not be worth saving as a permanent part of a system. The query-based report is ideally suited to default report formatting, if the report writer provides this.

The simple repetitive type represents reports with straightforward page layouts and easy data access. These reports are run frequently. This type of report usually gets saved as a permanent feature to provide instant access to the report. Repetitive reports consist of known useful reports and query-based reports identified as being used frequently. Once again, default formats (if available) may be used with this type of report.

Complicated Reports

A "complicated" report refers to either a sophisticated report format, difficult data access, or, more often, to both. This report type implies that greater expertise, as well as a clear understanding of the desired output, are required. Complicated reports rarely use a default format when it is provided.

The frequency of producing a complicated report affects different factors in the report program. Infrequent reports allow the programmer to concentrate effort on the report format, with less emphasis

on the performance of the report writer. **Repetitive** reports are run frequently or regularly. These reports tend to emphasize performance much more, especially when data base access is inefficient. These two types of reports require different tuning efforts to make them work well.

Types of User Interfaces

Report writers use three distinct user interfaces, which will be referred to as imbedded, programmatic, and visual. These interfaces are not mutually exclusive, and in fact several report writers provide facets of each, yet each interface seems to define its own set of users.

Imbedded report writers occur as statements in a standard, third generation language. These statements provide page controls and automatic totalling; the standard language provides the control over data base access and general output. The imbedded statements may extend output control or data base access as needed. The COBOL report module is an excellent example of an imbedded report writer.

The term **programmatic** refers to products which define their own language for the report writer. This includes fourth generation languages as well as some products specifically designed to produce reports. These report writers provide all controls necessary for writing a report, including data base access. Most programmatic products are reasonably small and friendly to use when compared to imbedded report writers.

Visual report writers use forms and other screen formats to guide the user. The forms allow a user to indicate what data to use, how to print the data, and what a page should look like. This type of interface is extremely easy to use, as no real "programming" is involved. Some report writers provide a visual mode as an alternative to the programmatic or imbedded interface.

Of course, reports may be generated by programs which do not use a report writer. This alternative is always open and must be considered when determining how a report program should be written.

The Occasional User

As the title implies, the occasional user does not need a report writer very often. This type of user relies on others to write report programs and possibly even run them. But every now and then, a report must be produced quickly, or privately, and the user cannot wait for someone else to write the program. The occasional user is *not* an end user; some technical knowledge is required. Managers are probably the best example of occasional users.

The infrequent and impromptu use of a report writer implies that query-based reports will be produced. Such reports are run in order to get immediate information. As a result, formatting and performance are not key areas of concern for most of these reports. If a report needs to be run repeatedly, even if only a few times a year, someone else can be assigned to write the report program.

The best interface for an occasional user is a visual one. This type of interface allows the user to see what will be produced before the report data is printed. In some cases, the report writer will guide the user until a minimal report can be printed. For simple query reports, this is ideal.

A programmatic interface provides a good environment for infrequent users. Data base access remains easy, and default formats are common in these products. The page layout and automatic summaries are still simple to use. However, programmatic interfaces tend to require more work on the user's part, as both statements and syntax must be remembered (or looked up). In addition, some knowledge of the data base is essential with this interface.

An imbedded language report writer is not designed for the occasional user. Too much knowledge of the language, the data base, and the report writer is required. Even a former technical user will have to look up a substantial amount of information. Very quickly the user will realize that the report takes too long to produce.

Most companies have relatively few occasional users. When choosing a report writer product, these users should not be a major factor in the decision process. The needs of the technical and semi-technical users far outweigh the needs of a few people to produce uncommon reports.

The Semi-technical User

Semi-technical users are a growing segment of the programmer population. This has been spurred by the tremendous growth of fourth generation languages, program generators, and user-friendly, specialized products. All of these products make programming much easier to understand, which has allowed many people to write custom programs.

The semi-technical user need not have any formal training or programming experience. Indeed, a user may not be able to write anything in a third generation language. This may prevent the writing of a general purpose program, but it does not prevent a great number of useful programs from being written. The semi-technical user is not a full-time programmer, but may have some programming duties. On the other hand, this user may use a report writer during spare moments just to improve the work environment.

This user group will create many more reports than the occasional user. Any simple reports can be produced, in addition to some complicated reports. The user's own expertise must be the guide to creating new reports; advanced users should be able to produce very fancy reports. Rarely used or custom complicated reports probably require too much time for the semi-technical user to create; repetitive reports also require time, but often are worth the person's effort.

Both visual and programmatic report writers are available and used by these users. One could debate for days about which style is better. The important point is that the report writer provide the capabilities needed by the semi-technical user. As a person writes more reports, the sophistication of the reports can increase. The report writer should not impede this progress.

Imbedded report writers may be used by some semi-technical users, but in general this is not the case. General purpose languages are more difficult to learn and to use than fourth generation products. Even the size of a standard language can be daunting for some users. In addition, the relatively unfriendly environment for writing programs will drive away most semi-technical users.

Unlike reports produced by the occasional user, the performance of a semi-technical user's report program can be important. As indicated earlier, a small change in data base access can significantly impact system performance. Many users are not trained to recognize this situation, and even fewer know how to make effective changes. One must rely on the more advanced users or a "technical" user to help streamline these report programs. The choice of a report writer which allows this streamlining is important if many semi-technical users are writing programs.

The Technical User

The technical user category represents full-time programmers. These people often have formal training or much experience in the computing field. A background in third generation languages is very common, but not strictly necessary for report writers. Understanding how a program or system works is much more

important than language knowledge. The technical user employs this knowledge to write efficient report programs.

The technical user should be called upon to write the most complicated reports. Such reports can get bogged down in very exacting details which usually frustrate the more casual users. In addition, technical users should write or review any repetitive report programs, especially when they are run frequently. This ensures that the report programs attain the best performance.

All styles of report writers are usable by technical users, including languages which do not support report writer statements. Experience has shown that these users can produce fairly sophisticated reports very quickly when using either visual or programmatic report writers. Imbedded report writers require more time to use, but not nearly as much as a language with no report writer. The experienced user will weigh several factors before choosing what language to use for a report.

It is worth noting that not all report writers are equal for a technical user. There are reports which are extremely difficult to write in fourth generation languages. There are also reports which require great control over data base access in order to perform quickly; not all report writers provide the necessary control. On the other hand, *most* reports use very straightforward data base access and do not need such tight controls.

Performance is a key issue for technical users. The programmer must help the semi-technical user understand why a report runs slowly, and must help that user improve the report program. To do this, a thorough understanding of the report writer is needed, particularly with respect to data base access. The technical user's own report programs can take performance into account immediately.

Is one report writer enough?

Undoubtedly, many people would answer this question with a resounding "YES!" For occasional and semi-technical users, this answer is certainly correct. But a technical user has many factors to consider, and it may be that the correct answer to this question is actually "NO." Some of the key factors in this decision are performance, integration, and responsiveness.

Performance should be obvious by now. Every report writer creates programs with different performance. If performance is critical, an imbedded report writer may work best. On the other hand, many programmatic report writers provide sufficient controls to yield near maximal performance.

Integration is a tricky issue. This area refers to the need to merge the report program into an overall system. Some report writers are part of a much larger system, and so integration becomes an easy topic. But there are other report writers which run alone; the report programs from these products may have to be merged with other pieces to build a system. The technical user must determine the feasibility of this merger, as well as its affect on the user's system interface.

Responsiveness refers to the speed with which a new report can be created. If a request for a new report comes in, the technical user must determine how soon the report has to be available. Fourth generation products allow much faster response, in general, but this may be outweighed by other factors. This decision must be based on the business environment and customer needs.

Of course, the technical user may already have a report writer. In this case, the factors above, along with other possible motivations, may indicate that a second report writer is needed. The second report writer should not be the same style of program. For example, owning two programmatic report writers is probably not useful, but owning a programmatic and an imbedded report writer might be. This decision should not be made lightly, but it is worth considering.

THE END USER

Finally we come to the end user. At some point, everyone is an end user; all one has to do is run a program or an application. But the end user under discussion is the true end user. This person has little or no computing background and does not have the need to learn. Programming is not within the scope of this user's job. The user only wants to get information from the computer.

Given the user's background, a valid question to ask is "Should the end user be able to create new reports?" The answer should be no. Creating a report requires knowledge in two technical areas: 1) the report writer language or interface, and 2) the data base description. Some report writers are extremely simple to use, but one must still understand the terms used. Many users do not or cannot take the time to learn these details. In many cases, end users have neither the need nor the time to learn about a report writer.

Data bases can be even more confusing. End users with no computing background have no concept of how data bases are arranged, or why they are defined in a particular manner. Data base field names are often abbreviated, and there are fields which users know nothing about. In addition, security may dictate that end users remain unaware of some data base information. This precludes publishing the data base format for the end user, leaving no way to access the necessary information.

Some end users may have a genuine need to produce query-based reports. These users should be properly trained to use a report writer. The training itself moves the user into the semi-technical class. This will not change the user's job responsibilities; it will give the person the capability to do the job properly. Not all end users need to be trained; a few people can be trained and all query-based reports can be redirected to these users.

Assuming that the end user cannot be trained, what would it take to allow end users to produce query-based reports? Simply put, this requires a much more sophisticated interface than report writers provide. The interface must be customized to the job at hand, which indicates that several interfaces may be needed. The reports must be very easy to produce and the data base must be hidden completely. The capability to write an end user interface does exist today, but the programs can be quite complicated. A better alternative might be to write small report programs for the most common queries, and pass the less frequent queries on to someone with more training.