# Development and Implementation of a Flexible Reporting Software Application

by

Enrique R. Siaca

B.S. Electrical Engineering, Massachusetts Institute of Technology, 1988

Submitted to the Department of Electrical Engineering and Computer Science and the Sloan School of Management in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

and

MASTER OF SCIENCE IN MANAGEMENT

In conjunction with the Leaders for Manufacturing Program at the Massachusetts Institute of Technology
May, 1994



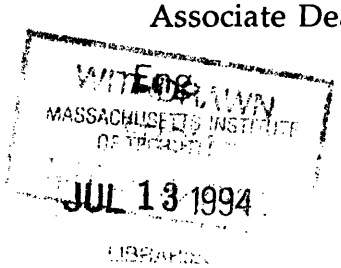
Signature of Author _____
Department of Electrical Engineering & Computer Science
Sloan School of Management
May, 1994

Certified by _____
Steven B. Leeb, Assistant Professor of Electrical Engineering
Department of Electrical Engineering, Thesis Supervisor

Certified by _____
James Utterback, Professor of Management and Engineering
Sloan School of Management, Thesis Supervisor

Accepted by _____
Frederic R. Morgenthaler
Chairman, Committee on Graduate Students

Accepted by _____
Jeffrey A. Barks
Associate Dean, Sloan Master's and Bachelor's Programs



# Development and Implementation of a Flexible Reporting Software Application

by
Enrique R. Siaca
B.S. Electrical Engineering, Massachusetts Institute of Technology, 1988

Submitted to the Department of Electrical Engineering and Computer Science and the Sloan School of Management in partial fulfillment of the requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
MASTER OF SCIENCE IN MANAGEMENT

## Abstract

The high-capacity 3 1/2" disk drive market provides suppliers with the opportunity to achieve differentiation through better service. Periodic product performance reports may help to improve service in three distinct ways. They can provide customers with commonly requested information, trigger feedback on the various reported metrics, and guide efforts to identify and solve specific customer problems.

This document discusses the development and implementation of a flexible software application capable of providing periodic, product performance reports to Digital Equipment Corporation's disk drive customers. The software application was eventually able to customize the reports, generate new reports, perform data updates, and handle new data sources. It was also flexible enough to accommodate organizational changes that could affect the report generation activities.

Thesis Supervisors:

Steven Leeb, Assistant Professor of Electrical Engineering
James Utterback, Professor of Management & Engineering

## Acknowledgements

I wish to acknowledge the Leaders for Manufacturing Program for its support of this work. Special thanks go to the people at Digital's Small Form Factor organization who put up with me for the six and a half months of the internship and gracefully answered all of my questions.

Thanks also go to my advisors, Dr. Jim Utterback and Dr. Steven Leeb for their guidance both during the internship and during the preparation of this document.

I also wish to acknowledge my manufacturing philosophy classmates for discussing their own experiences and helping me to better understand the total picture of the manufacturing enterprise. Their advise throughout the year has allowed me to distill from my internship experience a much better document than it would have been otherwise.

Finally, I would like to thank my mother Dina and my father Ramón for their unwavering support and encouragement. Without them I would not have made it this far.

# Table of Contents

# List of Figures

# List of Tables

# CHAPTER 1
# Industry and Project Overview

This document describes the implementation of a flexible software application developed to generate a series of product performance reports addressing internal and external information needs within a manufacturing organization. The work was performed during a 6 1/2 month internship at Digital Equipment Corporation's disk drive manufacturing facility in Colorado Springs, Colorado. In particular, it involved work within the Small Form Factor organization which assembled and tested 3 1/2" hard disk drives for internal and external customers.

This chapter briefly introduces the relevant competitive aspects of the high-capacity disk drive industry that served to motivate the internship project. An overview of the research project then follows. The various stages and results of the project are then discussed in more detail in the subsequent chapters.

## 1.1 Industry environment and supplier/customer communications

In the 1970's, IBM built the first of what are known today as Winchester disk drives. These original drives were 14" in diameter and their primary characteristic was an assembly of read/write heads and disks enclosed in a clean cavity. Over the years, Winchester disk drives have gradually decreased in diameter from 14" to 8" to 5 1/4" to the current industry leader, the 3 1/2"

drive. At the same time, their market has spread from mainframe storage applications to include personal computers and laptops.

Today, the high-capacity disk drive OEM[1] business generates over two billion dollars in yearly revenues[2] by supplying drives to high-end systems including microcomputers, workstations, and RAID[3] systems. It is characterized by constant product innovations focused on increasing areal density[4] and improving product performance characteristics such as how quickly a drive can find, read, and write data. These improvements must come without sacrificing the lower prices, better product quality, and higher reliability demanded by the market.

> *"[Areal density has] been rising at about 60% per year."*[5]
>
> - Peter Franklin, business manager of OEM storage at Digital Equipment Corp.

> *"Traditionally, hard-drive prices drop between 5% and 6% per quarter."*[6]
>
> - Peter Knight, senior vice president of systems development at Conner Peripherals Inc.

Within this fast paced environment, drive manufacturers fiercely compete to supply a limited number of OEM customers. The competition has turned the industry into one where both technological innovation and commodity pricing are necessary for a manufacturer to succeed.

---

[1]Original Equipment Manufacturer; customer that integrates the product into a larger system.
[2]Computer Industry Forecasts, January 15, 1993.
[3]Redundant Arrays of Independent Disks.
[4]Areal density refers to the amount of data that can be stored on recording media per unit area.
[5]Computer Reseller News, December 20, 1993.
[6]IBID

The suppliers that are first to reach the market with new products have a distinct competitive advantage. Products must initially go through an expensive qualification process with OEM customers that might last several weeks and involve hundreds of evaluation drives. Therefore, many customers will only qualify two or three suppliers and ignore those that don't have a product available on time. Unless qualified suppliers perform poorly, latecomers then face a very uphill battle. They must convince customers that they can offer enough of a better product and service to warrant spending more money on additional qualifications.

Nevertheless, winners of a qualification cycle are faced with several other cycles during the year, forcing established suppliers to remain alert to changes in customer demands. These demands might include hardware as well as software changes to the drives, new bus interfaces for the drives, and more stringent labeling and packaging requirements. Manufacturers who are not able to quickly adapt to their customers' demands risk losing their customer base.

> *"The technology continues to move at a frantic pace. There's no question life cycles for products are shorter."*[7]
>
> - Greg Brashier, director of Storage Dimensions' LANStor Unit.

In particular, the level of required communication between supplier and customer is becoming particularly high in this business. Customers who put their brand names on the drives or install them in their systems want to be

---

[7]Computer Reseller News, December 20, 1993.

notified about how the drives are performing. For example, information on manufacturing yields and the percentage of drives that fail during and after installation is increasingly being requested by customers. The causes of drive failures and corresponding corrective action plans are also important to customers.

The internship project was undertaken with the belief that a supplier that could establish a broad communication channel with its customers, and had a strong commitment to customer satisfaction, would develop a definite edge over its competitors. Such a supplier would be able to maintain strong links with its existing customers, react faster to their changing needs, and convince potential customers to consider it over other suppliers.

## 1.2 Improving communications between suppliers and customers

In an attempt to broaden the communications between Digital and its OEM customers, the internship project introduced a series of monthly product performance reports for the different hard drive models purchased by Digital's customers. These reports were eventually made available to the people working directly with customers. The new information was expected to generate questions that would improve communication between the Small Form Factor organization, the customer contacts, and the customers, as well as within different functional areas of the Small Form Factor organization.

The general progress of the project consisted of three distinct definition activities starting with the creation of a sample product performance report. This activity was followed by the development of the software tools necessary

for the periodic generation of the reports, and the definition of the steps required for using the software and generating the reports.

Figure 1.1 shows the overlap of these three definition activities over time. The triangular shapes are intended to represent a move from the initial broad definitions to the more concrete ones that were later implemented.

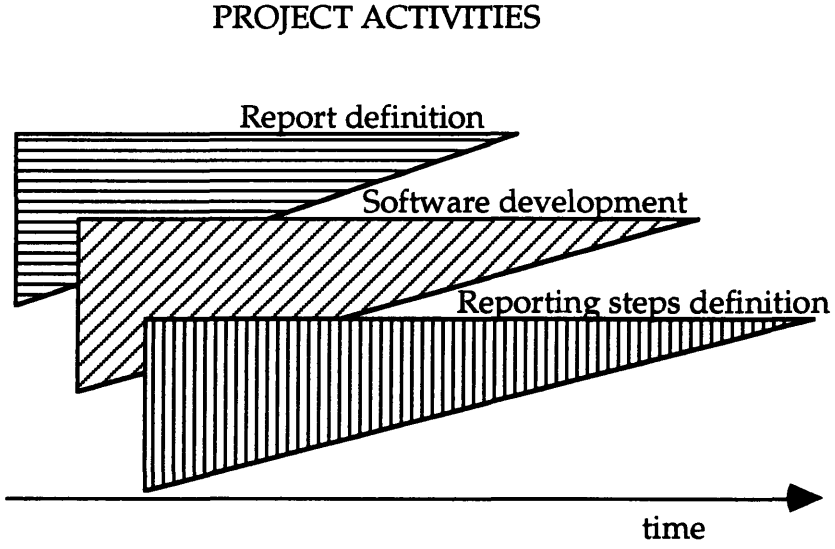



Figure 1.1: Internship Project Activities

The next three chapters will discuss these activities in order and the effect of their interactions on the various development efforts. First, Chapter 2 includes the development of the product performance reports from initial concept to final implementation. Chapter 3 describes the construction of the software tools that enabled the generation of the reports, while Chapter 4 follows the development of the activities related to the periodic generation of the reports.

Afterwards, Chapter 5 gives a more detailed description of the main features of the software applications that were developed, while Chapter 6 describes the final sequence of activities required to generate the reports. Finally, Chapter 7 provides the opportunity to step back and analyze the effects and impact of the product performance reports. Some comments will be provided on the events that followed the end of the internship and suggestions will be given for improving the development and implementation methodology of similar future projects.

# CHAPTER 2

## Report Definition and Development

This chapter will review the process by which a given information need and a series of reports to satisfy this need were identified during the internship. In particular, the chapter will show how a development approach using quick iterations followed by requests for feedback can integrate new information processing innovations quickly and effectively into an organization.

Before describing the information need that was addressed by the reports, some additional background on Digital's Small Form Factor organization is required. This manufacturing group belonged to a larger storage organization operating out of Massachusetts and having a business office that dealt directly with customers. Under this arrangement, Small Form Factor operated such that exposure to customers was mostly limited to receiving drive orders, returned drives, and occasional customer visits (usually by new customers that were thinking of qualifying a new product). As the organization moved to serve more and more external customers, the need for better communication became apparent. For example, many customers changed order quantities and requested delivery dates and a lot of rescheduling effort was needed to accommodate these changes. Demands for modifications to the disk drives and for more and/or different drive tests were also being voiced by multiple customers.

To attend to customer needs, a customer satisfaction group operated within the Small Form Factor organization. This group served as a link to a series of

account teams working directly with the customers. These account teams operated out of the separate storage business office and usually consisted of a sales representative, a manager for the account, and an applications engineer. The customer satisfaction group, in its role as a communication link, would bring feedback from the customers into the manufacturing decision making process, and provide information on the manufacturing operations to the account teams. Figure 2.1 illustrates the position of the customer satisfaction group along the communication path between Small Form Factor manufacturing and the customers.

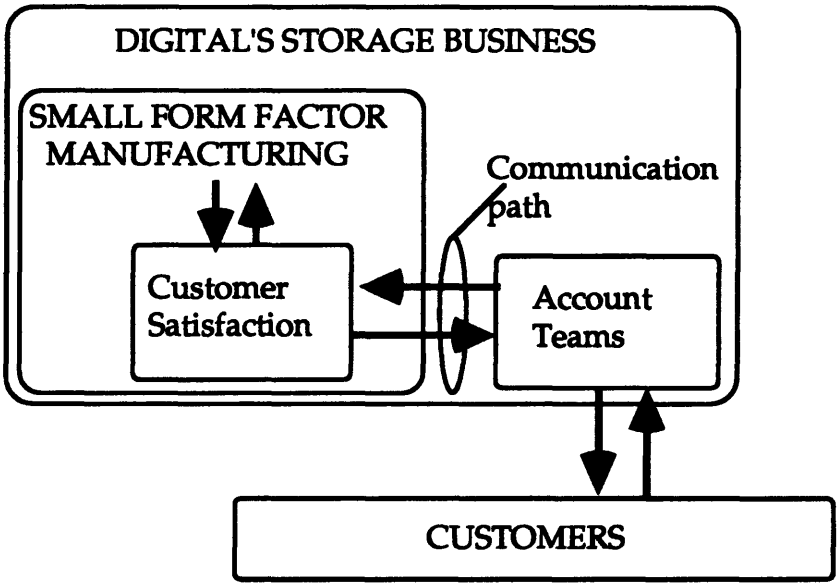



Figure 2.1: Communication Path to Customers

Note again how the account teams belonged to an organization separate from Small Form Factor. This surfaced as a problem later on in that the expectations on how the product performance reports would be used turned out to be inconsistent between customer satisfaction and the account teams.

With this in mind, an explanation will now be given of how poor access to information was a major hurdle to improved communications with customers. Afterwards, discussion will center around the evolution of the product performance reports that were eventually created to address this communication need, from their initial conception to the actual final product.

## 2.1 The information needs of the customer satisfaction group

As a starting point, I would like to present an initial problem statement as it was posed by the customer satisfaction group. This initial problem definition will be explored in further detail, and the information obstacles that were found will then be discussed.

In brief, the problem faced by the customer satisfaction group can be stated as follows:

> *Increasingly, customers demand more information on product and process performance metrics and there is no way to easily gather and communicate this information in a periodic, cohesive fashion.*

In addition, the perception by the customer satisfaction group was that more customers would be demanding this information in the future, and that more detailed information would be demanded. Therefore, the mechanisms to gather data and generate reports including databases, software applications, and the activities related to their use needed to be in place to handle these new requests.

During the early part of the internship, the customer satisfaction group provided a description of a variety customer information needs. In addition, memorandums, contracts, and other documents were provided to support the anecdotal information. The following signals pointed to a growing customer need for product information:

- *Several customers were asking for product performance information:* There were information requests as part of the product qualification process, negotiated access to product and process information as part of new contracts, and direct requests to account teams and customer satisfaction for product performance information.

- *Account teams were asking for periodic information on product performance:* There were memorandums from account team members asking for this information in order to work out customer complaints more effectively. In particular, some customers were generating their own product performance information, and account team members did not have any internally provided information for comparison purposes.

- *Competitors were already providing some information to customers:* Some account team members had seen samples of product performance reports provided by competitors. From the interviews performed, it appears that the competitor's reports contained general information on quality and reliability of the disk drives.

At the root of the problem with reporting the requested information was the way the information systems were organized and the data were shared within the manufacturing plant. The information systems within the plant could be characterized as isolated pockets of functional information. Different data was controlled and used by different groups, and there was no overall data architecture or translation mechanisms to tie the various systems together. More specifically, in collecting the data for the product performance reports, the following problems were encountered.

- *Access:* Data contained in multiple computer systems required separate user accounts to become accessible. For an employee, this meant going to different system administrators to get the accounts, get approvals from a manager for each one, then get access to the data sources from the groups that managed the data, and finally, learn to use the different available interfaces to access the data.

- *Interpretation:* The meaning of the data was often unclear or misleading. There was some documentation available but it was frequently inadequate in providing a meaningful interpretation of the data. The best sources for gaining an understanding of what the data meant were the people who actually captured and maintained it through data-entry, execution of periodic file updates, etc. Even here, individual employees were often familiar with only portions of these activities and processes.

- *Integrity:* There were sources of errors from the data-entry process as well as timing problems in accessing the data. The timing problems, in particular, required knowledge of how often the data was entered, and how the

different files were updated. Changes in data collection procedures also introduced changes to the meaning of the data.

Figure 2.2 provides a graphical representation of how the occurrence of any of these three problems prevented the extraction of useful information from the available data.

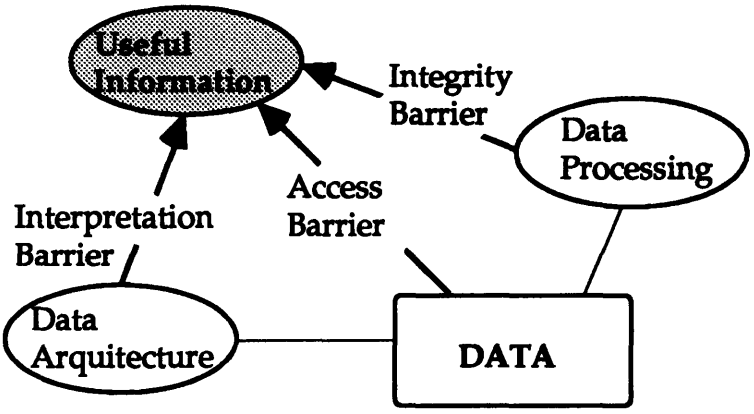



Figure 2.2: Barriers between Data and Useful Information

Examples of symptoms associated with these barriers included the following:

- *Data was shared through electronic reports:* Typically, users would receive periodic reports from other functional groups through electronic mail. Ad hoc querying of databases to answer specific questions was the exception rather than the norm.

- *Data that seemed the same was not the same in different systems:* For example, production and financial systems had fields to denote the shipment date of drives. In the case of the production database, shipment

date meant the date when the product left the end of the production line (not necessarily the shipment date). In the case of the financial database, shipment date meant the date of a particular stockroom transaction and an extensive understanding of all the transaction types was required to determine which ones were shipments.

- *Information systems suffered both major and minor changes that affected the integrity of the data:* Major changes occurred when new information systems were introduced and the data from the older system was transferred to them. Minor changes within an information system developed over time as the existing system was adapted to new business needs[8]. These changes jeopardized the integrity of the data in the eyes of those not familiar with them.

As evidenced by these symptoms, a division existed between the people who maintained and used the data often, and the people who didn't use the data as frequently and therefore did not have the necessary knowledge to correctly interpret it.

This situation proved problematic because the customers were asking for data that cut across functional boundaries. Typically, a member of the customer satisfaction group would collect internal memos and reports and either incorporated them directly into a customer report or included a retyped and revised version of these with the following results:

---

[8]For example, in one system a text field called REQUESTER was reassingned to capture customer request dates since no locations were originally set up to store this information.

- The presentation was not consistent across sections of a report.
- The reports used a lot of internal "lingo" that was hard for customers to understand.
- The reports had to be assembled individually.
- Data not available from internal memos or reports had to be obtained and interpreted by the customer satisfaction representative.

## 2.2 Definition of the customer product performance reports

From the original conversations with members of the customer satisfaction group, they expressed a desire for solving the above problems by providing a series of easy to generate customized reports to customers. These reports would contain a variety of information with different detail levels that could be included or excluded depending on the needs of each customer. These first conversations formed the basis for an initial broad description of the reports that would help address the varying information requirements of customers:

> *Periodic customer reports with product and process information that could be customized to incorporate specific customer requests.*

These new reports would help the customer satisfaction group to classify product information by customer since each customer had its own assigned part numbers. The reports would also provide factual rather than anecdotal information to manufacturing on specific customer problems, and would lead to better efforts to solve these problems.

As justification for the reports, the customer satisfaction group observed that:

- Customers were becoming more demanding over time in their information requirements and it was believed that this trend would continue.

- An expanding customer base would mean that more customers would eventually be asking for the same type of information.

- Providing reports now would allow for the reporting of standardized product metrics whereas if customers were left to ask for the information, they would each be asking for a different set of metrics to be reported.

- Providing the reports before customers asked for them would reflect a willingness on the part of Digital to anticipate customer demands that would otherwise be lost if customers had to ask for the information.

- There were communication breakdowns with customers where the perceived level of performance within Digital did not match the customer's perceptions and these breakdowns could be discovered through the use of the reports. Delivery performance was cited as an example here since Digital measured to a scheduled shipment date and assumed that the carrier would deliver as promised while the reality at the customer site could be different.

- There were customer specific issues that would be revealed by the information and could then be addressed on a customer by customer basis. For example, differences in the handling of the drives at the customer sites

could be checked when the defective drives returned from one site greatly outnumbered those from another.

The initial concept for the new customer product performance reports was a broad definition of an ideal way to provide information to account teams and customers. The definition provided a long term vision that served as a reference point, and gave a broader context to the more limited scope of the internship project.

The internship project only targeted a "minimum requirements" or baseline report while laying the groundwork for later improvements through an appropriate report structure and flexibility features. The work incorporated the core tasks that had to be accomplished in order for the baseline report introduction to be successfully implemented. Future report features beyond the minimum requirements would increase the attractiveness of the product performance reports, and allow for their evolution and adaptability to changing customer needs. Figure 2.3 describes the relationship between current practice (ad hoc reporting), a minimum requirements customer report, and an ideal report.

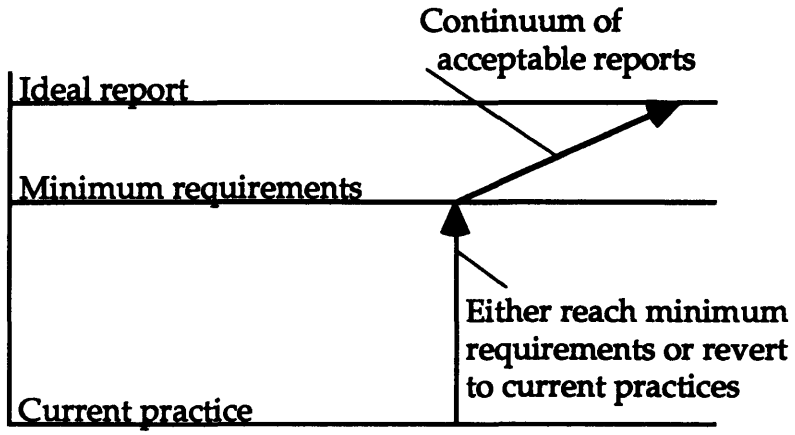



Figure 2.3: From Current Practice to an Ideal Report

The minimum requirements for introducing a new product performance report to customers were developed as follows. First, an examination was performed covering the various internal memos requesting and offering product information, ad hoc reports that had been prepared already, and customer grade reports[9]. After reviewing this information, I asked the customer satisfaction manager what were the requirements for offering a first cut report to the account teams in order to get their feedback.

- *Information must be complete:* The first cut report needed to include delivery performance, delivered quality, and reliability performance information for each drive part number.

- *Information must be valid:* Any report presented to account teams for feedback had to contain real data. That is, "Do not offer a sample report for feedback with information that you are not sure you can provide."

[9]Some customers provided Digital with evaluations on its performance as a supplier.

- *Information must be available:* Any report presented to customers needed to be maintainable (i.e. could be generated periodically). In other words, "Do not ask for feedback on a report that you can not provide on a consistent basis."

These were somewhat generic requirements, but it must be remembered that no benchmark product performance reports existed at the time, and that the final reports would be dependent on the available information and the capabilities of the software that would create them. In addition, customer contacts would have some say on how the reports would look and fulfilling unnecessary requirements would only waste development effort.

## 2.3 Development of the customer product performance reports

Given that no benchmark report existed, and that information requirements would eventually be affected by feedback from account teams and customers, a prototyping strategy was used for developing the new reports. The basic strategy consisted of iterating through a series of report contents and formats in order to converge towards a final report. These iteration cycles will be discussed next.

### 2.3.1 Initial throw-away prototypes

The initial iteration cycles occurred when a throw-away report prototype was created to discuss the formatting of the data. First, talks with customer satisfaction group members provided a sense of how the data would be presented and a preliminary sketch of what a report should look like. Then,

based on the information above, I created a sample report format. One iteration was performed on this sample, and very little feedback was obtained from customer satisfaction.

This initial prototype report is shown in Figure 2.4. It contained simple metric information as well as time charts to show trends. Individual shipments were also shown as it was felt that customers would also want to look at this information (particularly when orders were late). All was arranged on a single page for ease of distribution. Note that the algorithms for generating the metric information were not defined at this stage.

d i g i t a l

# Performance Report

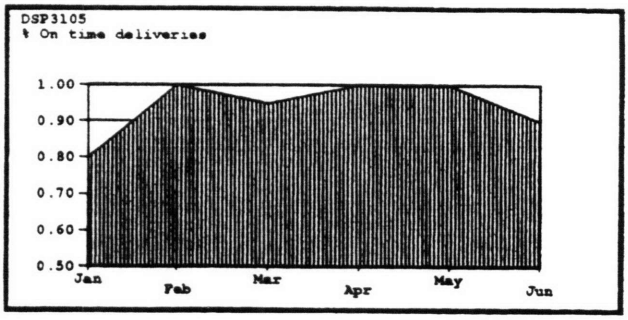
Customer: Oem Inc.

Date: June, 1993

## Delivery Performance

6 month average on-time delivery

98.7%



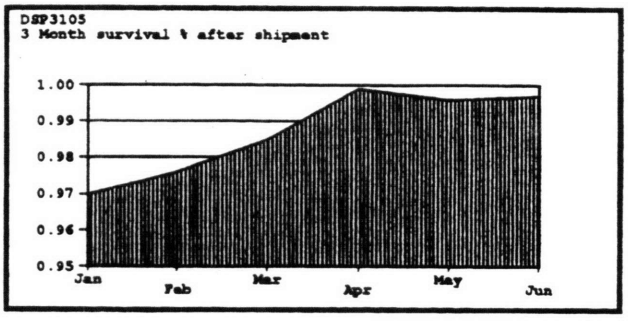

## Missed Order Information

| Customer P.O.#/Line # | | Ordered/Shipped | | Requested/Actual Receipt Date | | # Days Off |
|---|---|---|---|---|---|---|
| 133-0457 | 1 | 100 | 100 | 06-18-93 | 06-21-93 | +3 |
| | 2 | 100 | 50 | 06-18-93 | 06-21-93 | +3 |
| | | | 50 | 06-18-93 | 07-01-93est. | +13 |

## Delivered Quality

6 month average early life experience

99.4%




## Reliability Performance

6 month average MTBF

248,000hrs

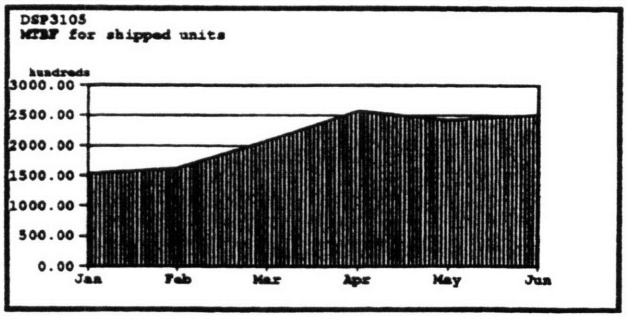



**Figure 2.4 (a): Initial Throw-away Report Prototype**

The value of the first prototype was not in the feedback it generated, instead it served two important purposes. First, it communicated to customer satisfaction the kind of data and the presentation capabilities that would be developed. Second, the template served as a basis for judging the adequacy of the reporting capabilities of the various software development tools that were later examined.

### 2.3.2 Report prototypes with real data

A second set of iterations was performed on throw-away prototype reports with actual data. These reports were generated using the chosen software development platform (see next chapter) so as to demonstrate their feasibility. The customer satisfaction group showed significant interest and provided considerable feedback on these prototypes. Selected account team members also saw the reports and gave feedback on them.

Figure 2.5 shows a sample report during this stage. The data requirements and algorithms necessary to generate the performance metrics were already defined for this report. Note also how the amount of information included grew from that of the initial prototype.

Digital Equipment Corp. Product DSP3105

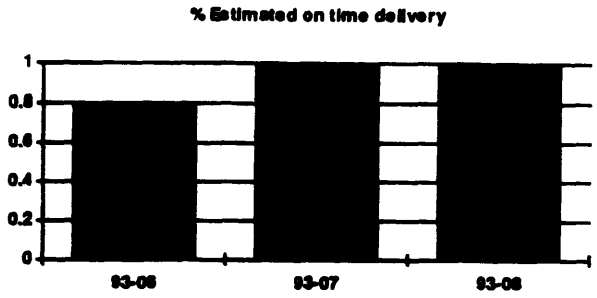
*06-Sep-93* Customer Part

| Customer Name | Part Number | Customer ID |
|---|---|---|
| | RH27A | -AA |

**Delivery Performance**

**3-month rolling average**

88.2%




| Customer Request Date | Est. days late (early) | P.O Number | Line No. | Order Quantity | Shipment variance |
|---|---|---|---|---|---|
| 09-Aug-93 | 0 | 7978CM3194 | 1 | 240 | 0 |
| 16-Aug-93 | 0 | 7978CW3194 | 1 | 240 | 0 |
| 23-Aug-93 | 0 | 7978CM3194 | 1 | 240 | 0 |
| 30-Aug-93 | 0 | 7978CW3194 | 1 | 240 | 0 |

Figure 2.5 (a): Throw-away Prototype Report with Real Data

Digital Equipment Corp. Product DSP3105

*06-Sep-93* Customer Part

| Customer Name | Part Number | Customer ID |
|---|---|---|
| | RH27A | -AA |

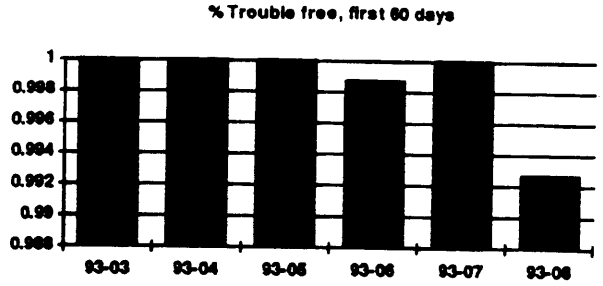
**Delivered Quality**

**3-month rolling average**

99.7%

**Defects per million - DPM:**

| | |
|---|---|
| 93-03 | 0 |
| 93-04 | 0 |
| 93-05 | 0 |
| 93-06 | 1,269 |
| 93-07 | 0 |
| 93-08 | 7,292 |




**Installed population**

12,263

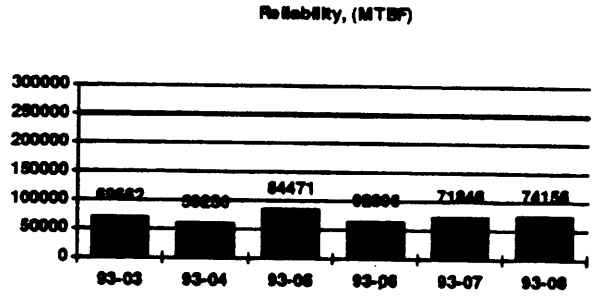



Figure 2.5 (b): Throw-away Prototype Report with Real Data

The customer satisfaction group used this prototyping stage to evaluate and decide whether to accept the reports. Several modifications were made to the product performance reports during this stage but a particular report structure was not cemented and enough flexibility was maintained to accommodate the later feedback from account teams. Also, since most feedback was obtained during meetings of the customer satisfaction group, this evaluation stage allowed the customer satisfaction group members to confront their different ideas of how the reports should look.

Additional feedback was then obtained from a selected account team members (as identified by the customer satisfaction group) which validated the general thrust of the reports. In addition, this feedback pointed to a need for more detailed information on product failures to direct improvement efforts based on the report data.

### 2.3.3 Towards the final reports

The final prototyping iteration cycles were performed on an evolutionary report prototype that became the final product performance report. Figures 2.6 (a) thru (g) show an example of the product performance reports that were finally selected for distribution to account teams.

|d| i|g| i| t|a| l|

# Product Performance Report

## Delmonico Inc.

**For the month of: 93-11**

**Products included:**

DSP3105

DSP3210

Figure 2.6 (a): Fiñal Customer Product Performance Report (Cover)

DIGITAL EQUIPMENT CORPORATION — PRODUCT PERFORMANCE REPORT FOR: 93-11

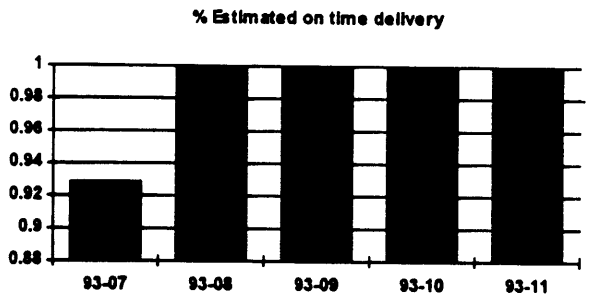
| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH27A -PW | DSP3105 | | 1.05 GB |

## Delivery Performance

**3-month rolling average**

100.0%

**Delivery Performance** is estimated by comparing the actual ship date and expected transit time to the customer request date. A detailed description of the algorithm used is available upon request.




**Units shipped during the month of : 93-11**

| Customer Request Date | Est. days early (late) | P.O Number | Shipped Quantity |
|---|---|---|---|
| 02-Nov-93 | 0 | 29012574 | 43 |
| | 0 | 29012603 | 120 |
| 04-Nov-93 | 0 | 28008330 | 28 |
| 04-Nov-93 | 0 | 28008462 | 12 |
| | 0 | 28090345 | 160 |
| 09-Nov-93 | 0 | 20071166 | 120 |
| | 0 | 20071180 | 210 |
| 11-Nov-93 | 0 | 20071166 | 120 |
| 16-Nov-93 | 0 | 20071166 | 80 |
| 16-Nov-93 | 0 | 20071166 | 20 |
| | 0 | | 0 |
| | 0 | | 200 |
| 16-Nov-93 | 0 | 20071166 | 20 |
| 17-Nov-93 | 0 | 20071166 | 120 |
| 19-Nov-93 | 0 | 29012644 | 6 |
| 22-Nov-93 | 0 | 20071166 | 120 |
| 23-Nov-93 | 0 | 20071166 | 120 |
| | 0 | 25989811 | 200 |
| 29-Nov-93 | 0 | 20071166 | 120 |



Figure 2.6 (b): Final Customer Product Performance Report (Delivery)

DIGITAL EQUIPMENT CORPORATION PRODUCT PERFORMANCE REPORT FOR: 93-11

| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH27A -PW | DSP3105 | | 1.05 GB |

## Delivered Quality

**3-month rolling average**

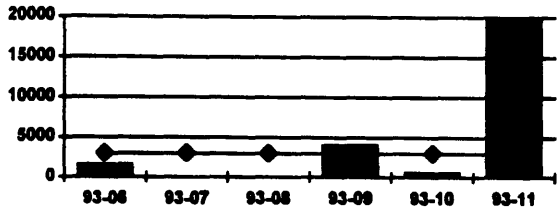
6,364 DPM

**Delivered Quality** is based on field returns and is an assessment of the performance of the product for its first 60 days, primarily in our customer's facilities.

$$DPM = \left( \frac{1{,}000{,}000 \times [\text{quality returns for month}]}{[\text{shipments for month}]} \right)$$

Where quality returns are units returned within 60 days of shipment. A description of the detailed algorithm used is available upon request.

Delivered Quality in Defects Per Million




Delivered Quality for Entire Customer Base

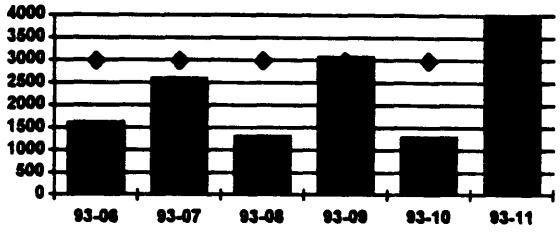





Figure 2.6 (c): Final Customer Product Performance Report (Quality)

**DIGITAL EQUIPMENT CORPORATION** **PRODUCT PERFORMANCE REPORT FOR:** 93-11

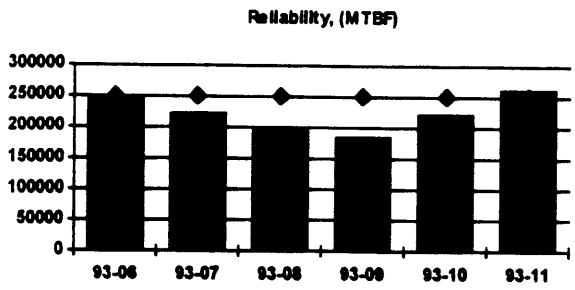
| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH27A -PW | DSP3105 | | 1.05 GB |

## Reliability Performance

| Customer's Installed population |
|---|
| 14,122 |

| Customer's Total units shipped |
|---|
| 19,394 |

**Reliability Performance** is an estimate of field performance at end-user applications, based on quantity of units shipped and field returns. A detailed description of the algorithm used is available upon request.



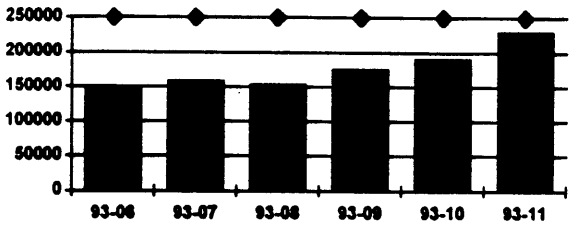






Figure 2.6 (d): Final Customer Product Performance Report (Reliability)

**DIGITAL EQUIPMENT CORPORATION** **PRODUCT PERFORMANCE REPORT FOR: 93-11**

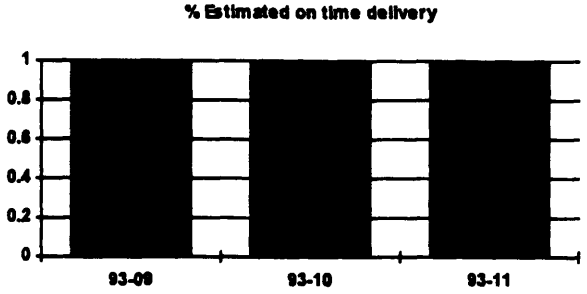
| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH20E -PW | DSP3210 | NONE | 2.10 GB |

**Delivery Performance**

**3-month rolling average**

100.0%

**Delivery Performance** is estimated by comparing the actual ship date and expected transit time to the customer request date. A detailed description of the algorithm used is available upon request.




**Units shipped during the month of : 93-11**

| Customer Request Date | Est. days early (late) | P.O Number | Shipped Quantity |
|---|---|---|---|
| 02-Nov-93 | 0 | 29012602 | 2 |
| | 0 | 29012603 | 10 |
| | 0 | 20945693 | 10 |
| | 0 | 29807676 | 30 |



Figure 2.6 (e): Final Customer Product Performance Report (Delivery)

DIGITAL EQUIPMENT CORPORATION PRODUCT PERFORMANCE REPORT FOR: 93-11

| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH20E -PW | DSP3210 | NONE | 2.10 GB |

## Delivered Quality

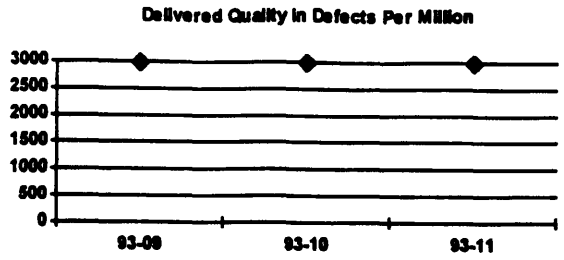
**3-month rolling average**

0 DPM

**Delivered Quality** is based on field returns and is an assessment of the performance of the product for its first 60 days, primarily in our customer's facilities.

$$DPM = \left( \frac{1{,}000{,}000 \times [\text{quality returns for month}]}{[\text{shipments for month}]} \right)$$

Where quality returns are units returned within 60 days of shipment. A description of the detailed algorithm used is available upon request.



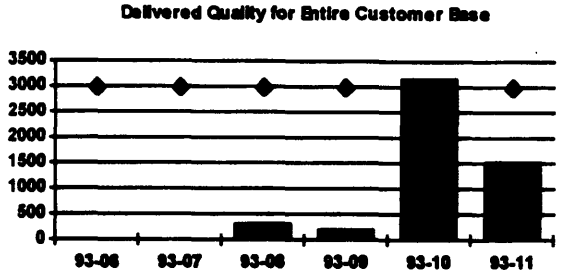






Figure 2.6 (f): Final Customer Product Performance Report (Quality)

DIGITAL EQUIPMENT CORPORATION PRODUCT PERFORMANCE REPORT FOR: 93-11

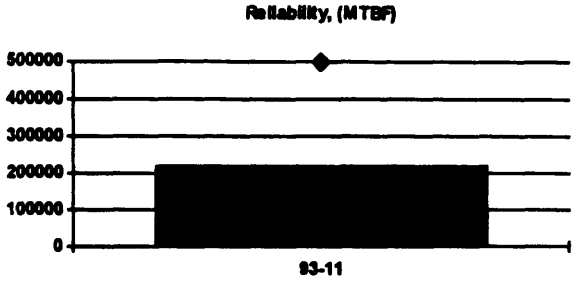
| Customer Name | Internal Part No. | Sales Part No. | Customer Part No. | Capacity |
|---|---|---|---|---|
| Delmonico Inc. | RH20E -PW | DSP3210 | NONE | 2.10 GB |

**Reliability Performance**

| Customer's Installed population |
|---|
| 30 |

| Customer's Total units shipped |
|---|
| 182 |

**Reliability Performance** is an estimate of field performance at end-user applications, based on quantity of units shipped and field returns. A detailed description of the algorithm used is available upon request.



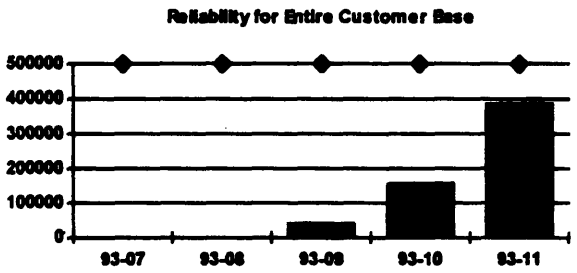






Figure 2.6 (g): Final-Customer Product Performance Report (Reliability)

From the customer satisfaction group's perspective, this stage did not represent significant changes to the content of the reports. On the other hand, this stage signaled that a report architecture and a report generation process were now in place and that availability of the product performance reports would not be a problem.

Additional feedback was obtained at this stage from other account team members. This new round of feedback uncovered the fact that certain account team members (particularly those involved in sales) were looking at the reports as a marketing tool and wished to report only "benign" information to customers. Taking different cuts of the data (by application, customer type, etc.) was suggested as a way to better achieve the marketing potential of the reports, but customer satisfaction did not view the reports as a product marketing tool.

Finally, during this stage some account team members suggested that other storage products (i.e. tapes) should be included with the reports since customers would want to have information for them too. This issue reached beyond the Small Form Factor organization and was not resolved by the end of the internship.

## 2.4 Contents of the customer product performance reports

A product performance report consists of a cover page and a page each for the delivery performance, delivered quality, and reliability performance of every disk drive part number included in the report.

### 2.4.1 Cover page and header information

The cover page (Figure 2.6 (a)) includes the name of the customer that purchased the drives, the month covered by the report information, and a list of the drive sales part numbers that will be covered in the report. Inside the report, the page headers also contain the customer name, and month information. In addition, the headers provide the capacity of the corresponding drive for that page plus a box with the part numbers that designate the drive at the customer's organization, at Digital's sales organization, and at the Small Form Factor organization. This box therefore serves as a translation mechanism when people from different organizations refer to the same disk drive model.

### 2.4.2 Delivery performance section

The delivery performance section (Figure 2.6 (b) & (e)) contains a column graph with up to six months worth of delivery performance information, a three-month aggregate delivery metric, as well as information on individual shipping transactions. Table 2.1 shows the equations used for generating the delivery performance metrics.

| METRIC | EQUATION |
|---|---|
| Delivery performance expressed as % (for the graph) | (#on time shipments during the month)* / (#shipments during the month) |
| 3-month aggregate delivery performance expressed as % | (#on time shipments during last 3 months) / (#shipments during last 3 months) |

* i.e. #on time shipments = Number of on time shipments based on scheduled ship date.

Table 2.1: Delivery Performance Metrics

The graph serves to show delivery trends for a given customer while the shipment information provides the volume of drives requested by the customer as well as of the how large and how late or early were the missed orders.

Note also that on time shipments are based on meeting a scheduled ship date so that variations in transit time are not taken into consideration. Information on when the order actually was received by the customer is not available for inclusion in the reports. Also, delayed orders sometimes shipped via a faster carrier to make up the time and the original scheduled ship dates were left unchanged so that these orders would be counted late when in fact they were on time.

### 2.4.3 Delivered quality section

The delivered quality section (Figure 2.6 (c) & (f)) contains two column graphs with up to six months worth of delivered quality information for a specific customer's part number and for the total population of drives of that capacity

and size. It also contains a three month aggregate quality metric expressed as defective units per million (DPM). Table 2.2 shows the equations used to generate the delivered quality metrics.

| METRIC | EQUATION |
|---|---|
| Delivered quality expressed as DPM (for the graph) | $\frac{(\#\text{shipped} - \#\text{quality returns during month})(1M)}{(\#\text{shipped during month})}$* |
| 3-month aggregate delivered quality expressed as DPM | $\frac{(\#\text{shipped} - \#\text{quality returns in last 3 months})(1M)}{(\#\text{shipped during last 3 months})}$ |

*Quality returns are defined as those that occur within 60 days of shipment.

Table 2.2: Delivered Quality Metrics

The graphs serve to show quality trends for a given customer and to compare the performance of the customer's drive with the total population of similar drives (i.e. same capacity and size). The constant line through each graph represents the quality goal for the product.

The delivered quality metric is designed to catch drive problems that occur at the OEM customer's manufacturing facility. As such, this is the metric that would reflect problems with the handling of the drives.

Note also that the delivered quality metric is susceptible to wide swings due to changes in shipment volumes. For example, a customer that receives 100 drives in a given month and returns 10 drives from a much larger shipment the previous month will show 100,000 DPM for that month which is

misleading (i.e. the customer is not necessarily returning 10% of all drives). This effect is less pronounced when viewing the 3-month aggregate metric.

### 2.4.4 Reliability performance section

Finally, the reliability performance section (Figure 2.6 (d) & (g)) contains two column graphs with up to six months worth of reliability information for a specific customer's disk drives and the total population of drives of that capacity and size. It also contains an estimate of the installed population of the customer's drives and the total drives shipped up to date. Table 2.3 shows the equations used to generate the reliability performance metrics.

| METRIC | EQUATION |
|---|---|
| Reliability performance expressed as MTBF** (for the graph) | $\frac{\text{(5-months of run hours for installed population)*}}{\text{(\#reliability returns in last 5 months)***}}$ |
| Installed population expressed as number of drives | (#drives shipped up to two months earlier) - (#returned drives up to two months earlier) |
| Total units shipped expressed as number of drives | (#drives shipped up to current report month) |

*Run hours are calculated as 730/month for each drive starting after 60 days of shipment.
**MTBF or mean time between failures is a standard measure of product reliability.
***Reliability returns are defined as those that occur after 60 days of shipment.

Table 2.3: Reliability Performance Metrics

As in the delivered quality case, the graphs here serve to show trends for a given customer and to compare the performance of the customer's drive with the total population of similar drives. The constant line through each graph represents the reliability specification for the product.

The installed population number tries to capture the number of drives that have left the OEM manufacturing site and are installed in working systems. It serves as a guideline to how stable the reliability metric is because reliability performance tends to jump significantly while the installed population is small. Total units shipped can be used to determine whether additional drives will soon be installed and how much of an effect they will have on near term reliability performance.

# CHAPTER 3

## Definition and Development of the Software Applications

This chapter contains a detailed description of how the project's two complementary software applications were conceptually defined, as well as how they were developed. The discussion will show how organizational constraints influenced the many design decisions required to create the software applications that generated the product performance reports.

This development work was started immediately after the definition of the minimum requirements report and began with an analysis of the various tasks that would be required to provide periodic product performance reports. These tasks can be divided into four steps.

- Data Collection - Getting the required data from the various sources.
- Data Processing - Operating on the collected data.
- Report Generation - Creating multiple reports from the processed data.
- Report Distribution - Making the reports available to their audience.

Figure 3.1 illustrates the sequential relationship between these four steps.

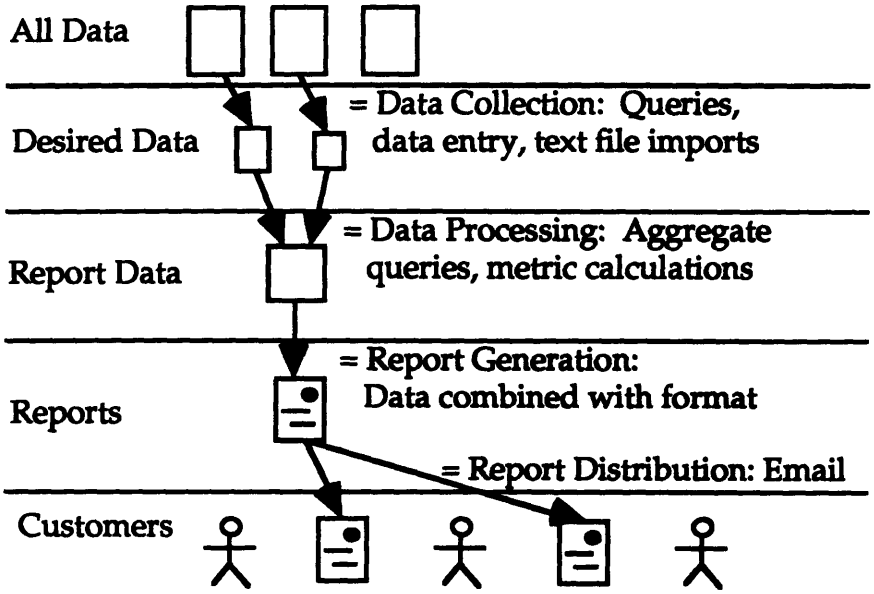



Figure 3.1: Flow of Data and Associated Tasks

As the most complex steps, data collection and processing are discussed first along with the software application that was developed to perform them. Then, the discussion will shift to the report generation and distribution steps and to their corresponding software application.

## 3.1 Data collection and processing

Data collection can be defined as the tasks required to extract the desired data from the various sources within the manufacturing plant's computer network. Data processing can be defined as the additional tasks required to transform the collected data into a form that could be used to generate reports. Before the development of the software to execute these tasks could begin, the data sources for the reports had to first be identified. After this was done,

several mechanisms to extract the desired data were then explored and the best ones selected.

### 3.1.1 Searching for the data sources

The obvious starting places for identifying the required data sources were Small Form Factor's internal reports and memos that were already being generated and shared with the account teams and customers. A list of the required data and its possible sources was then identified after informal interviews with the originators of the internal reports and memos. This data included:

- Shipped and returned drives information kept on VMS/Rdb[10] relational databases.
- Shipping transaction information on flat files[11] belonging to the financial/inventory system.
- Customer and part number information being transferred into a Microsoft ACCESS[12] database.
- Other data on part numbers and customers that was not kept in an organized fashion anywhere within the plant.

Selecting the final data sources from the original list required answers to additional questions. In some instances, the data were not available while in others the data were available from multiple sources. In these cases, several considerations were involved in determining which data would be used to generate the product performance reports. Questions included:

---

[10]Relational database management system for Digital's VAX/VMS architecture.
[11]Files containing unformatted data and used by older record management systems.
[12]Relational database management system for Microsoft Windows.

- Which source was original and which was a copy?
- Which source was most readily available?
- Which data approximated most closely the desired data?
- When did the data become available at each source?
- How closely did the data match that currently used for internal metrics?

Selection of the right data sources was not a static decision making process. The changing business practices and systems brought up the possibility that a given data source might need to be dropped in favor of another. In particular, changes to the order administration system and the database for returned drives were being considered but had not been implemented at the end of the internship. Nevertheless, the data sources selected to generate the product performance reports were:

- *Warranty Returns Database:* This Rdb relational database was picked as the source for information on disk drives returned from customers.

- *Inventory/Financial Database:* This transaction system was selected as the source of aggregate as well as individual order shipping information[13].

- *Drive Configuration Database:* This Microsoft ACCESS database was picked as the source of information on new disk drive families[14], as well as customer specific part numbers.

---

[13]It is interesting to note here that the aggregate information did not match the individual order information.

[14]Disk drive families were grouped according to capacity and the external dimensions of the drives.

- *Customer satisfaction group:* The group was selected to provide and maintain customer names & sites, drive storage capacity for each part number, and quality/reliability specifications for each disk drive family since this data was not available from existing data sources.

Table 3.1 shows a description of the various data needed to generate the reports. Each description is followed by a sample value and the name of the corresponding data source.

SAMPLE DATA AND SOURCE

| Data Description | Sample Value | Data Source |
|---|---|---|
| Customer Name | Crossbow Intl. | Customer Sat. |
| Part Number (Digital) | RH27A-AA | Drive Config. |
| Customer Specific Part No. | Crossbow-1 | Drive Config. |
| Capacity | 1.05 GB | Customer Sat. |
| Quality/Reliability Specs | 3K DPM/500K MTBF | Customer Sat. |
| Requested Shipment Date | 02-JAN-1993 | Inventory Sys. |
| Actual Shipment Date | 02-JAN-1993 | Inventory Sys. |
| Returned Date | 10-FEB-1994 | Warr. Returns |

Table 3.1: Examples of Required Data for Reports

### 3.1.2 Evaluating the data collection and processing mechanisms

The work of creating the mechanisms for data collection and reporting was started while the definition of the data sources was still in progress. Several software tools were evaluated and the best ones were then selected.

The evaluation stage was an effort to establish the capabilities of the various software tools that could perform data collection and processing. It was

conducted through communication with users of the software, observation of how these software tools were already being used, and through the actual use of the tools. Evaluations were performed on:

- Structured query language (SQL[15]) embedded in command files
- Network server data querying/reporting applications
- PC data querying/reporting applications
- Network server special query files to generate reports (inventory system)
- Network server spreadsheet applications

As a self-imposed constraint, the evaluations were limited to software tools known within the Small Form Factor organization and available on the manufacturing plant's computer network. These tools were abundant, and locally available know-how would be able to support them if necessary. At the end of the evaluation stage, Microsoft ACCESS SQL queries, along with some command files, were the mechanisms selected to execute the data collection and processing tasks. This final selection was a result of the maintenance application concept that will be discussed in the next section.

## 3.2 The data maintenance application

Two main alternatives were considered for performing the data collection and processing tasks; decentralized and centralized. A decentralized approach would allow different individuals within the customer satisfaction group to get the portion of the required data they were most familiar with, and would

[15]SQL (Structured Query Language) is a standard data definition and data manipulation language for relational databases.

require less effort of each individual. A centralized approach would simplify the process and the coordination necessary for these activities, but would place all the data collection burden on a single person. This person might be unfamiliar with some of the data and unable to check for inconsistencies.

A centralized PC based data maintenance application was only pursued after an individual with the necessary PC hardware was assigned to collect the data and generate the reports. As envisioned, the software application would serve as the sole interface to all the data maintenance (collection & processing) tasks. This single interface would hide from the user the details of the many and diverse tasks necessary for the generation of the report data. Figure 3.2 provides a simple schematic of this data maintenance application concept.

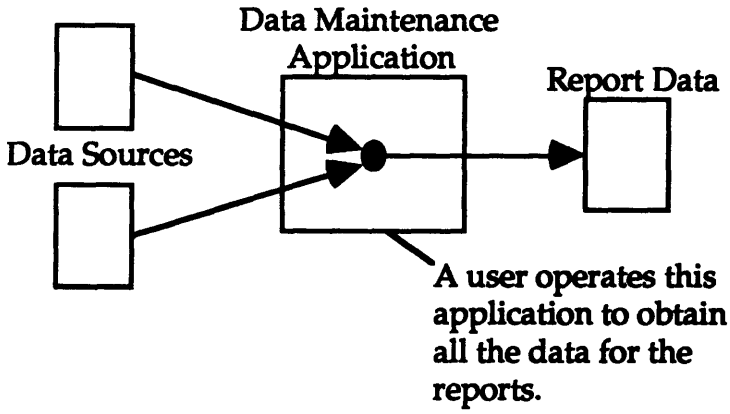



Figure 3.2: Initial Concept for Maintenance Application

The maintenance application would have a main menu listing all the data updates to be performed, as well as customized windows for each update requesting specific information. Ideally, the person performing the data

collection and processing would only need to step through the menu of updates once a month in order to obtain all the necessary data for the reports. In addition, help features allowing modification and addition of updates would be in place to help the user to adapt to future data requirements.

Microsoft ACCESS was chosen as the software development platform for this application because of its report generating capabilities, as well as its ability to access Rdb databases on the network. The software development took the form of two distinct steps:

- Integration of data updates into a single user interface.
- Development of help/flexibility enhancing features.

The first step was implemented through the use of a main menu screen with a sequential list of the various update steps necessary to obtain the report data. The complexity behind the updates was captured within the data maintenance application so that the update logic and network connections were hidden from the user. The user needed only to click on the menu buttons in sequence in order to perform the updates. Figure 3.3 shows the main menu window that the user saw when using the data maintenance application.

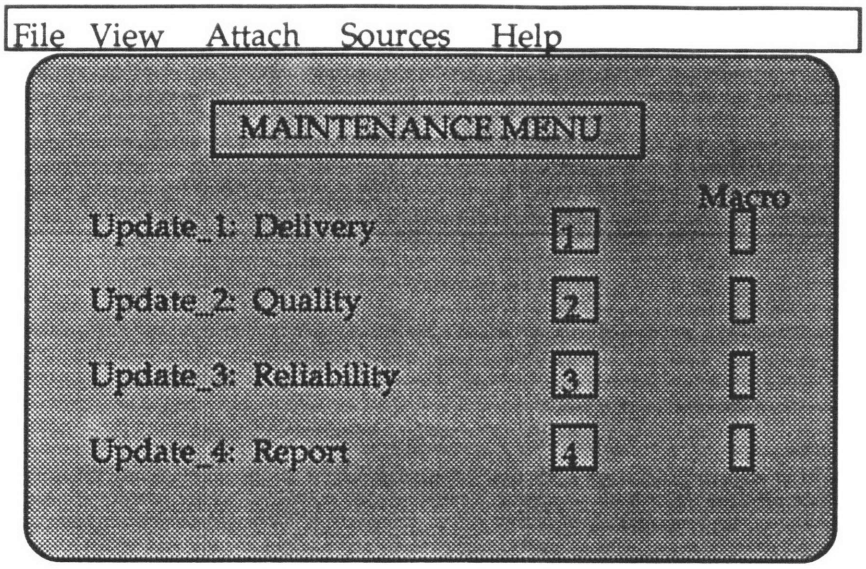



Figure 3.3: Main Menu of Data Maintenance Application

Because different updates could take different forms, a second customized update window was introduced to carry the update parameters and other information necessary to perform a specific update. For example, if the update needed to generate three month aggregate data, the user would open the update window from the main menu and enter the desired month for the update (user defined parameter). A second month would then be derived during the execution of the update (derived parameter) to frame the three-month window. If an update required access to a text file, the path and file name would also appear on the customized update window. Figure 3.4 shows how a user, by clicking on a particular update button could open the corresponding update window.

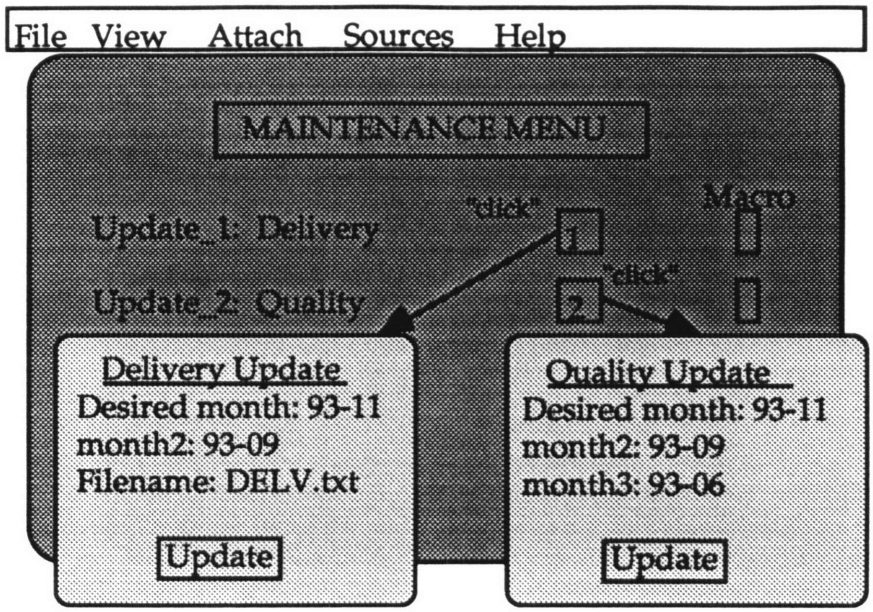



Figure 3.4: Opening the Update Windows

The updates themselves were executed after the user entered the required parameters and clicked on the Update button. In general, they consisted of a series of SQL queries performed as part of the Microsoft ACCESS code, or as part of an external command file. An example of what an update query looked like is shown on Figure 3.5.

SAMPLE UPDATE QUERY (SQL)

```
Select "93-11", Part_Number, Count(*) from RETURN_DATA
Into TEMP_1
Where Return_Date Between 01-NOV-1993 And 30-NOV-1993
Group by Part_Number;
```

This query counts the number of returned drives for each part number for the month of November and creates a new table TEMP_1 with three columns containing the string "93-11", the part number, and the total returns for the month respectively.

Figure 3.5: Sample Update Query

The data collection and processing mechanisms were integrated into the data maintenance application as follows:

- The Warranty Returns Database was accessed by the maintenance application through Microsoft ACCESS SQL queries and a database connectivity protocol to Rdb.

- The Financial/Inventory system was accessed by a command file to produce output text files with the desired data. The maintenance application could execute the command file, and then import the resulting output files for further processing. Some user editing was required for the macro files that generated the output data, and for the output files themselves to insure that the imports would not generate an error.

- The Drive Configuration Database data was updated by an automatic batch process that copied the file across the network from a public directory to a local one. Through the use of pointers, the maintenance application automatically knew where to search for the required information within this database.

- The information required from the customer satisfaction group was set up as two text files in a special group area. These two files could then be accessed and edited by the group using a simple text editor. The files where then copied periodically to a PC accessible directory to allow importing into the maintenance application.

In addition, a Microsoft ACCESS local PC database was created to keep the aggregate monthly data for the performance metrics. There were two reasons for this approach. First, the computation of the metrics was performed using the latest month's data from the original sources and the previous month's data from the local PC database. Getting all the data every month from the original sources was not practical because the data was not always available, there would be longer processing times, and higher use of the network could lead to more update failures. Second, a local PC database allowed speedy ad hoc querying of the aggregate information that would otherwise be unavailable or very slow.

Figure 3.6 shows the revised concept for the maintenance application. Note that it now allowed external data collection mechanisms through the execution of command files, and it also included the local PC database.

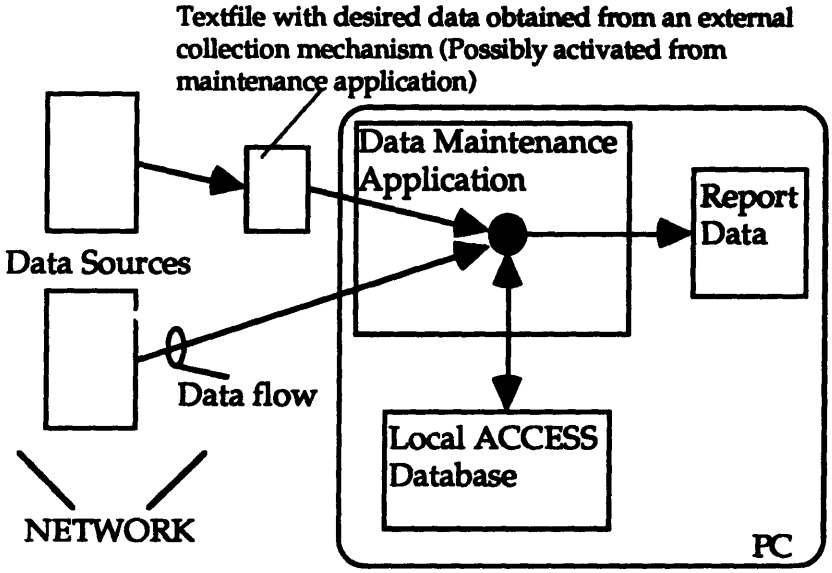



Figure 3.6: Revised Data Maintenance Application Concept

If needed, the user could change a particular update to point to a new data source and still run the update. On the other hand, changing a data source required some knowledge of the network environment and software.

The features that were included to add flexibility to the software application included:

- Ability to add new updates and update windows to the menu. This feature addressed the evolving information requirements of customers.

- Ability to "run" updates in descriptive mode where the logic of the steps could be followed without actually running the update. This capability allowed modification of updates by non-programmers.

- Ability to create a new update using the Microsoft ACCESS graphic querying interface and then document its logic. This feature also addressed the limited programming support available within the manufacturing plant.

- Ability to execute external command files in batch mode. This feature enabled SQL experts to create data update routines separate from the maintenance application. The user of the maintenance application could then integrate and execute these routines from the main menu.

- Ability to read external text files from the update menu. This capability particularly helped when automatic updates were executed via a periodic batch job. As one of the update steps, the user could be reminded to scan the batch job's log file to verify that the job's last execution was error free.

- Ability to document updates and data sources from relational databases. This feature kept the documentation current on changes to update logic and data sources. Unfortunately, the documentation of the source text files to create a complete picture of all data sources was not implemented by the end of the internship.

These features will be dealt with in more detail in Chapter 5. Meanwhile, focus will now shift to the report generation and distribution tasks.

### 3.3 Report generation and distribution

The report generation activities dealt with the creation of the product performance reports for multiple customers and disk drive types. The report distribution activity involved making the reports available to the intended audience.

The software application that would execute these tasks would be used monthly and had to be able to:

- Generate many reports (40+) across all customers and disk drive models.
- Generate multiple report formats (i.e. different presentations of the data).
- Include graphs and tables on the reports to show trends.
- Expand on the reports (Add new sections).
- Distribute the reports to a large audience (40+).
- Distribute the reports to different sites (10+).

The need to provide flexibility in the report formats was evident from the initial round of feedback interviews. There was a strong indication from account team members that detailed information would be needed on drive failures. There was also a possibility that different report formats might be required for different customers. One account team member described the problem of providing a complete report as "The spiral of chasing data". Data would be presented to a customer who would then ask for more data. Because of this situation, presenting a complete picture that included the detailed data was seen as important.

### 3.4 The reporting application

Because of the requirement to produce scores of sophisticated reports on a monthly basis, the selection of an appropriate software platform was critical to shortening the development time of the reporting application. The reporting application concept, in this case, followed the software development platform's selection. As a result, the concept definition and development work was focused on exploiting the underlying capabilities of the chosen software platform in an organized way that would hide some of the programming complexity from the user.

The reporting application was defined under the assumption that the report data was available. It would provide a single user interface that allowed for multiple ways of formatting and presenting the report data. Figure 3.7 shows a simple diagram of the concept.

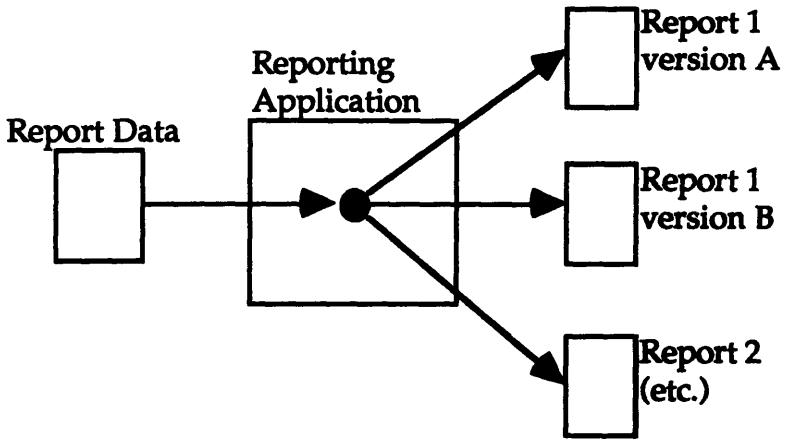



Figure 3.7: Initial Concept for Reporting Application

An early evaluation of the various available tools attempted to find a software development platform with the best report generation capabilities. Using this guideline, Microsoft ACCESS was selected as the development platform. This PC software tool, complete with its own database management system, had the ability to generate multiple reports quickly in a wide range of formats, and also enabled customization of the reports. Since none of these tasks required procedural programming, the use of this tool saved a lot of development efforts and allowed experienced ACCESS users to modify the maintenance and reporting applications without the help of programmers.

The reporting application would be independent of the data maintenance application to the extent that once the data updates were performed, copies of the reporting application's executable file could be distributed to allow for generation of the customer performance reports as needed.

To maintain a consistent interface across applications, the reporting application would have a main menu listing all the available reports, as well as customized windows for each report that would allow viewing, printing, and other report-specific commands. To avoid confusion between the applications, a color coding scheme was implemented with tones of green for the maintenance application windows, and tones of red for the reporting application windows.

For the reporting application, the menu incorporated two existing internal reports that used the same data as the customer product performance reports. The generation of these reports, previously prepared by a person within the customer satisfaction group, was now included as part of the tasks performed by the reporting application. In this way, the data that was looked at internally was consistent with the data reported to customers. Figure 3.8 shows how similar the reporting menu was to the maintenance menu.

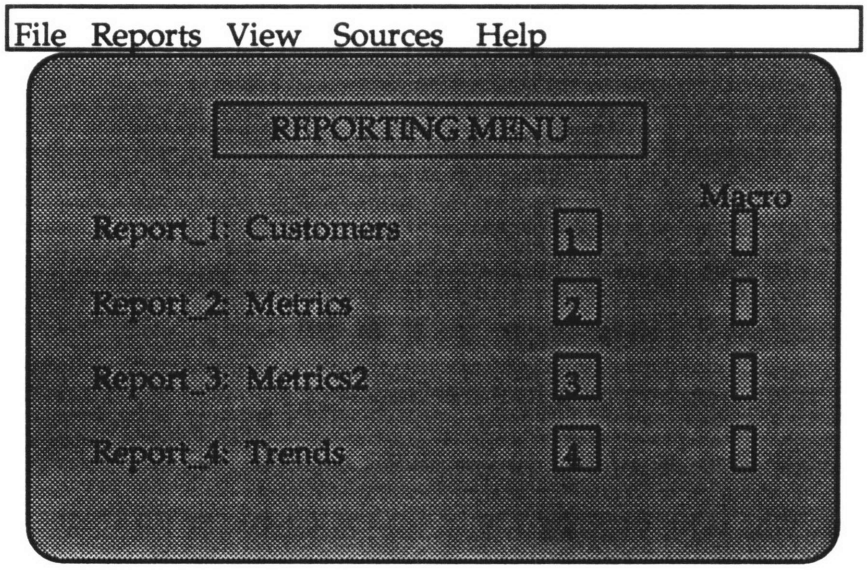



Figure 3.8: Main Menu for Reporting Application

Again, as in the case of the maintenance application, a second report specific window was introduced that could be customized to have options that were particular to a given report. Standard options for each report window would be "View" and "Print". Figure 3.9 shows two examples of these report windows.

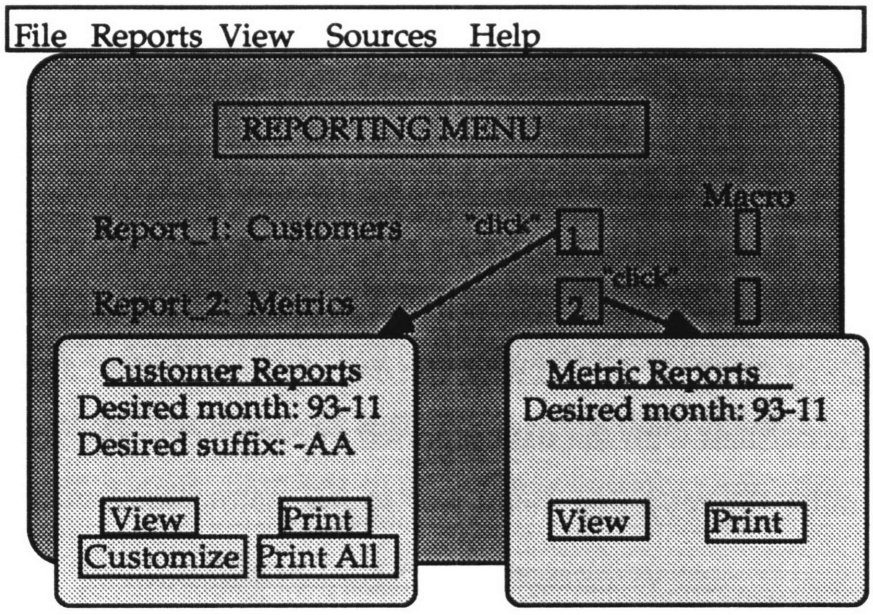



Figure 3.9: Examples of Report Windows

The organization of the reports themselves followed a hierarchical model. A main report contained the basic organization of the data, as well as information on page headers and footers. Sub reports containing different information would then be linked to the main report. This structure, due to time constraints, was fully exploited only for the customer product performance reports.

For these reports, a customization window was created that allowed up to six sub reports to be linked to the main report. This window was easily accessible from the Customer Reports window as seen in Figure 3.10.

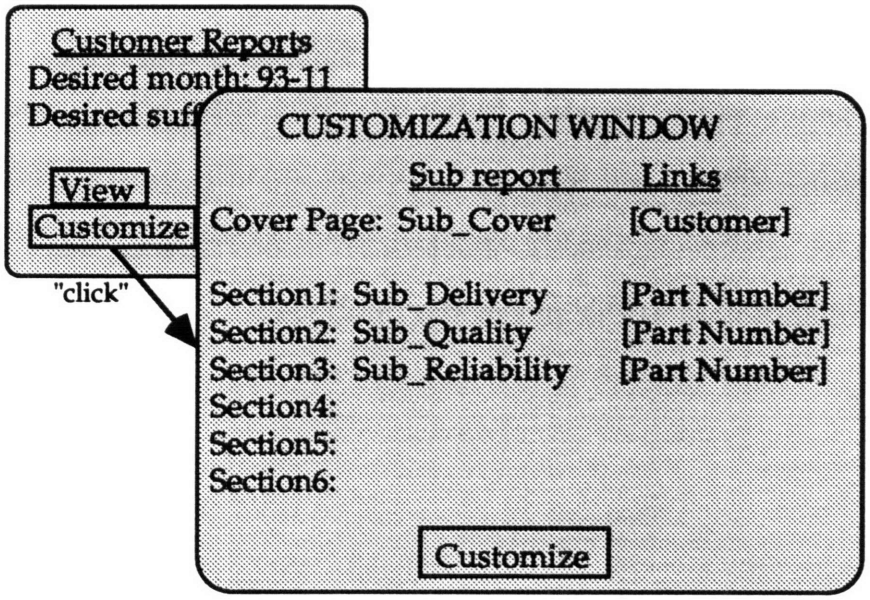



Figure 3.10: Customization Window

The user of the reporting application could exploit this customization feature in several ways. First, (s)he could easily delete or rearrange sections within the main report by deleting and reordering the names of the linked sub reports. Second, a given sub report could be modified (using Microsoft ACCESS) and saved with another name. The user could then rename the linked sub report to use the new version. Finally, if a new sub report was created, it could be easily be added to the list of linked sub reports. The customization feature will be discussed in more detail in Chapter 5.

For distribution of the reports, electronic mail was chosen as the fastest and easiest way to perform this task. Postscript files were created for all the

customer reports by the reporting application through a Print All button. File names were created automatically based on the part numbering scheme. For example, if a customer's part numbers ended with the suffix "-AA", its product performance report file would be named AA.EPS. The files could then be electronically mailed to account team members through the execution of a command file (external to the reporting application). The question of whether the account teams would take the time to print the postscript files, or whether it was too much of an inconvenience was raised but remained unanswered at the end of the internship.

# CHAPTER 4

# Definition and Implementation of Process Activities

This chapter describes the development and implementation of the data gathering and report generating activities associated with the product performance reports and related software applications. Of concern here are the actual steps that individuals had to execute to produce the monthly reports, from data collection to report distribution.

## 4.1 Definition of the activities associated with the new reports

During the course of the project, the definition of a sequence of activities that would allow periodic generation and distribution of the product performance reports was very much linked to the development of the related software applications. An obvious goal in establishing such a process was to minimize the incremental burden the reporting activities would place on the customer satisfaction group.

An additional goal, which surfaced during the project, was that these activities should be able to withstand changes to the Small Form Factor organization with minimum modifications. There were several signs of organizational fluidity that emphasized the need for this additional goal.

- Three people joined, and one person left the customer satisfaction group during the course of the internship.

- There was a reorganization of the Small Form Factor groups during the internship, and further changes after I left the site.

- The role of the customer satisfaction group was not rigidly defined in that there was no group charter with a specific description of the group's responsibilities.

These signs cautioned against data gathering and reporting activities that relied on too much coordination within the customer satisfaction group and the Small Form Factor organization. Instead, they suggested a more centralized approach with one person having most of the responsibility for the reporting activities. An individual outside the customer satisfaction group was eventually assigned with these tasks, and the initial goal of minimum additional work became dominant once again.

The new activities necessary to generate the product performance reports were classified as follows.

In terms of the additional burden they generated vs. existing and related activities:

- *New activities generating additional work:* The collection of monthly delivery data was an example as this was not previously done.

- *New activities reducing the overall work:* The collection of monthly quality/reliability data by part number allowed for the automatic generation

of metrics by customer type (i.e. OEMs vs. distributors) that were previously performed by an individual in customer satisfaction.

- *New activities modifying the existing work:* The generation of internal product quality and reliability reports based on field data was incorporated into the new reporting tasks because these reports used the same data as the customer product performance reports.

In terms of the additional burden they generated as a function of time:

- *Initial one time activities:* These include the installation of the software, placing the files in the appropriate directories, and establishing the pointers to the databases and files.

- *Additional training and learning activities:* These refer to the time spent in learning to perform an update, as well as fixing minor data, software, or network problems.

- *Steady state process activities:* These refer to the time an experienced operator requires to execute all the data collection and report generation tasks assuming no problems occur.

Given that no dedicated individuals were originally available to generate the product performance reports, I considered the following approach. The data collection and processing would be performed by individual members of the customer satisfaction group into a group database. The product performance reports could then be generated with the reporting application as required by

having a PC connected to the network and available to the group. Figure 4.1 provides a graphical representation of this approach.

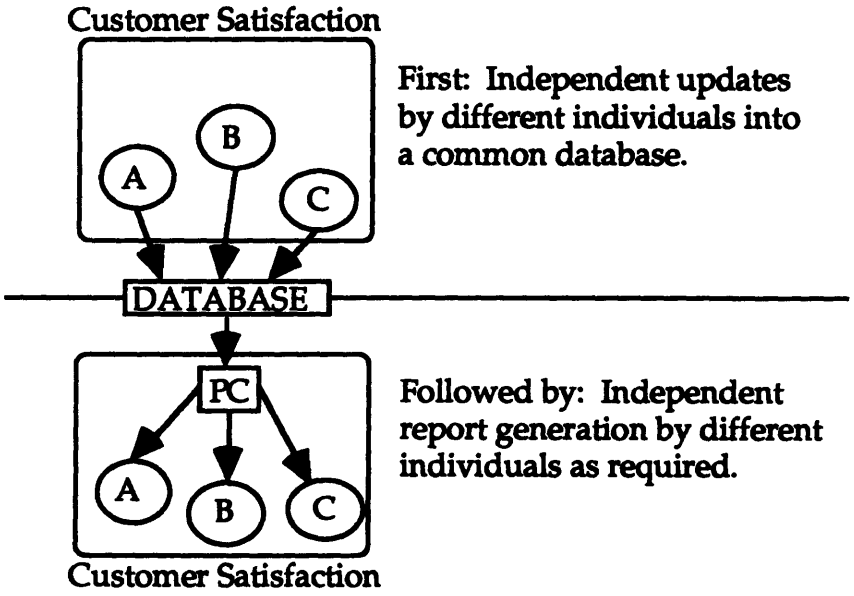



Figure 4.1: Initial Concept for Reporting Activities

Note that a PC based, centralized data maintenance application had not been considered at this point. The following logic supported the decentralized approach instead.

- Members of the customer satisfaction group had access to different network nodes and had access to different applications. A decentralized data collection approach would allow customized mechanisms for collecting data suited to each individual's knowledge and environment.

- A common database could be set up for the group so that the collected data would be shared among the various members. An underlying assumption

was that the data collected to generate the reports was useful for other purposes also.

- Dividing the update task among the customer satisfaction group members would minimize the impact of the incremental work necessary to generate the product performance reports.

The idea here was that if the information used for the reports was relevant to the work of the customer satisfaction group, this approach would prove feasible.

To test this idea an Rdb database with the relevant information was put in place and made accessible to the customer satisfaction group. The observed behavior was that use of the collected data was minimal after the data became available. There were various factors that could have contributed to the observed behavior.

- The customer satisfaction group preferred to obtain information from the people closest to the data for accountability. That is, the people who provided the data would be accountable for interpretation and integrity of the data. On the other hand, data maintained by the customer satisfaction group might not have this accountability.

- The available database server applications were not user friendly, accessible, or fast enough to provide easy access to the data. A survey of the customer satisfaction group indicated that these applications were either unknown or infrequently used by the group.

In essence, the underlying assumption about the usefulness of the data was wrong. Either making the information available didn't provide additional value, or the availability was not high enough for the data to be useful. During this time, other problems associated with a decentralized approach also became apparent.

- The coordination needs would require that one person have an understanding of all the activities necessary to update the reports. This would place a considerable burden on that individual.

- It would be harder to adapt/document the reporting activities to accommodate changes to the Small Form Factor organization (since a different set of coordination requirements might surface).

- Also, the use of customized mechanisms for each data collection task would be too inflexible to adapt to changes in the data sources (i.e. reprogramming would be needed to adapt to the new sources).

Because of these factors, introducing updates to the group to show maintainability turned out to be an impossible task for the available time frame. The transfer of these responsibilities to a single individual outside the group led to the centralized approach and the creation of the data maintenance application that helped to solve these problems.

Under the new approach, activities would be totally performed by one person who would operate the data maintenance and reporting applications once a

month and step through menus of update and report generation tasks. This person would have an overall perspective of the process, would be able to pinpoint specific activities that failed, and could then contact the people responsible for fixing them.

Support would still be needed from the customer satisfaction group in providing additional product information required for the product performance reports, as well as the information necessary for the electronic distribution of the reports. Separate shared text files were created so that the customer satisfaction group could update this required data. Nevertheless, these update tasks would now be peripheral and independent of the main data gathering and reporting activities. Figure 4.2 provides a graphical representation of the revised approach.

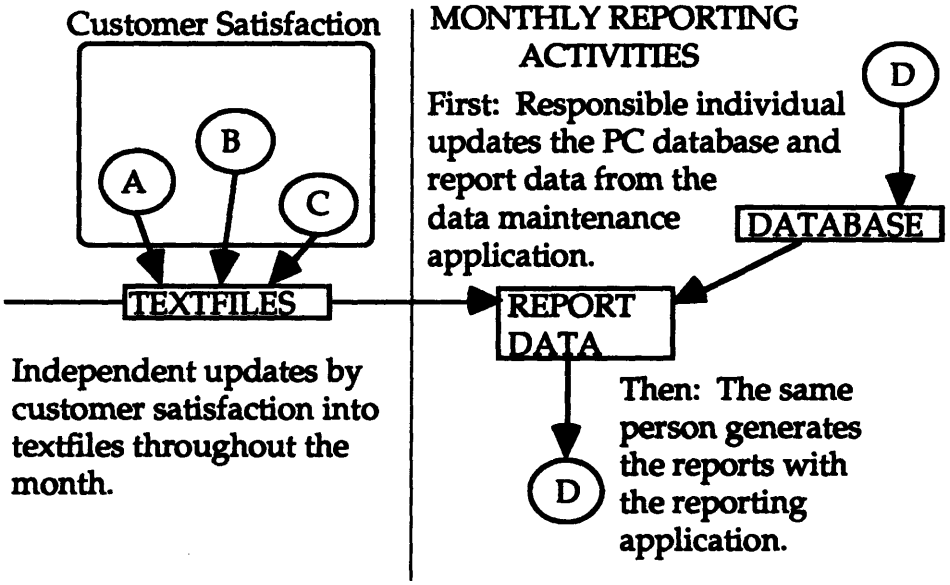



Figure 4.2: Revised Approach to Reporting Activities

## 4.2 Development of the data gathering and reporting activities

The development of the data gathering and reporting activities attempted to bridge the existing obstacles to a centralized updating process. As will be seen, some of these obstacles were better addressed than others. Additional problems were found during the implementation stage but feedback was obtained and improvements were made to correct them and simplify the various activities.

### 4.2.1 Initial implementation problems

There were strong initial factors that impeded centralization of the data updating process and these had to be addressed once the centralized approach was chosen. Some of these factors were as follows:

- It was hard for one person to gain access to the data across various systems. This was evidenced by the multiplicity of accounts, passwords, and clearances that had to be obtained to access all the pertinent data sources.

- Considerable learning was required to understand how to obtain data from different sources. It was my experience that an understanding of data interpretation and integrity was necessary in order to obtain the right data.

- The data gathering and reporting process had too many steps for one person to execute it reliably. Too many disparate tasks were required, and considerable improvements were needed to allow one person to execute all the steps and minimize the opportunities for errors.

The sequential menu interface that was provided with the maintenance and reporting applications enabled an organized and simple update and report generation process that could be ideally performed by a single individual in less than two hours. The complexity of the access to the data was enclosed within the maintenance application and was hidden behind the simple menu interface. The application knew where to look for the data whether a database or textfile, based on a series of pointers and descriptions of the sources. The setting up of the various pointers was an initial activity that was performed once during the installation of the software tools. The single interface also made the sequence of activities considerably easier to document, but some problems persisted.

### 4.2.2 Remaining problems

If the responsibility for the activities were transferred to a different person, the change would require reestablishing all the data connections for the new individual. This would involve getting accounts in all the relevant network nodes, gaining access to the appropriate directories, as well as establishing network drives on the PC environment. In other words, while the complexity of data access was hidden from the chosen individual, it would resurface if the process had to be transferred to another person.

Also remaining was the problem of capturing the knowledge that was required to interpret the data and analyze its integrity. Text files with information on the various sources of the report data, as well as the algorithms used to compute the metrics, were left in a common directory accessible to the customer satisfaction group to communicate some of the data

integrity and interpretation findings. It was unclear at the end of the project whether these files provided an acceptable way to capture this knowledge.

### 4.2.3 Implementation of the activities

As the data gathering and reporting activities were taking shape, the data maintenance and reporting software applications were being developed and refined on a test PC. Later, these files were transferred to the PC of the individual responsible for the data gathering and reporting activities. The transfer occurred in three distinct process runs spanning three consecutive months.

1st run, "on line" help: For the first run, the data maintenance and reporting applications were installed along with the local database on the user's PC. The responsible person was then able to perform the monthly updates with some help and instructions. The transfer of report generation activities was not attempted during this run. On the other hand, the run allowed significant feedback to be obtained on the user friendliness of the maintenance application's interface and improvements were then made to the software tool's menus and options.

2nd run, execution with on site support: The revised software applications were installed and the responsible person executed the updates with minimal assistance (given by phone). The report generation steps were included as part of the activities for the first time. Additional fine tuning of the maintenance and reporting applications was performed after this run.

3rd run, unaided execution: The final revisions to the software applications were left installed before the end of the internship and documentation was created for the software tools and the update process. The responsible person was able to execute the updates and generate and distribute the reports with aid from the documentation only.

### 4.3 Final location of the collected data

Some space should now be dedicated to the decision of where to store the monthly data collected for reporting purposes as this was one of the last decisions made before the end of the internship. Centralization of the process could still allow the collected data to reside in an Rdb relational database that would be accessible to the customer satisfaction group if needed. The other option was to store this data on the PC accessible only to the person responsible for the updates. There were tradeoffs involved in each approach.

A PC database was more attractive because:

- Queries of Rdb databases using the data maintenance application did not allow the deletion of records.

- Queries of Rdb databases from the data maintenance application ran considerably slower than comparable queries of PC based Microsoft ACCESS databases.

- Updates of Rdb databases from the data maintenance application were not very robust (connections would hang frequently), as they depended on the settings and traffic on the network servers.

- Setting up the security for an Rdb database was a complex task (particularly for people who wanted access from other nodes) and required additional maintenance of access lists.

A network database was more attractive because:

- The data would be accessible for ad hoc queries by the customer satisfaction group.

- The data on Rdb databases could be easily edited by members of the customer satisfaction group through the use of Rdb user applications whereas editing textfiles required more care.

As discussed before, the final decision was to put the delivery, quality, and reliability data in a PC based Microsoft ACCESS database, and keep the data that needed updating by customer satisfaction on textfiles in a common directory. The decision was taken because:

- In particular, delivery, quality, and reliability data would not be accessed on a regular basis by the customer satisfaction group based on the experience with the test Rdb group database.

- Everyone in the customer satisfaction group knew how to edit textfiles while not everyone knew how to operate the Rdb applications. This was evidenced from the responses of the customer satisfaction group to an administered survey.

## 4.4 Comments on the implementation of the reporting activities

In retrospect, it seems obvious that the centralized approach should have been pursued from the start of the project. The level of coordination, particularly in an environment of constant organizational changes, proved to be too high for a decentralized approach to succeed. On the other hand, it did not become clear until later that all the data gathering steps could be brought together to be executed by a single person.

The selection of a key individual to coordinate and perform the data collection was an important milestone of the project. I believe that this selection could have been done earlier and it would have speeded process development and implementation. At the very least, more emphasis should have been placed on exploring possible staffing arrangements and hardware/software availability for the data maintenance and reporting activities earlier in the project.

Customer satisfaction's rejection of the group database also caused delays in the development and implementation of the data gathering activities. As mentioned before, it is my belief that the lack of acceptance was due to both the requirement for proper accountability of the data, as well as the inadequacy of the available user applications to access the data. Maintenance

of the group database also would have required additional work (i.e. database security) that the PC database approach avoided. Nevertheless, this exploration proved useful in discovering how the newly collected data fit into a set of broader information needs within the customer satisfaction group.

# CHAPTER 5

# Description of the Developed Software

This chapter contains a more detailed description of the developed software's structure and interaction with the manufacturing plant's computer network. Emphasis will be placed on the flexible features that were incorporated into the software to allow it to adapt to evolving business needs.

## 5.1 Software overview

The software package that was developed consists of two Microsoft ACCESS executable files that perform distinct functions. These two applications execute most of the data maintenance and reporting activities mentioned earlier. They will be discussed in some detail as this was where most of the software development effort was directed.

Both of these files reside on a 486 PC[16] with 8MB of RAM[17] and a 1.05GB hard drive[18]. In addition to DOS[19] and WINDOWS[20], the associated installed software necessary to run these files includes:

Microsoft ACCESS V1.1

Microsoft ODBC Administrator V1.0

---

[16]Personal computer with having an Intel 80486 microprocessor.
[17]Eight thousand bytes of random access memory (RAM).
[18]1.05 million bytes of storage capacity.
[19]Standard PC operating system.
[20]Microsoft's PC graphic user interface (GUI).

Digital ODBC Rdb Driver V1.0

Digital PATHWORKS V4.1

Microsoft ACCESS is required to execute the applications while the other software is needed in order to obtain Rdb relational data and text data across the network. More will be said about this other software in the section on network integration. Figure 5.1 illustrates the data flow between the two applications, the local PC database, and the network.

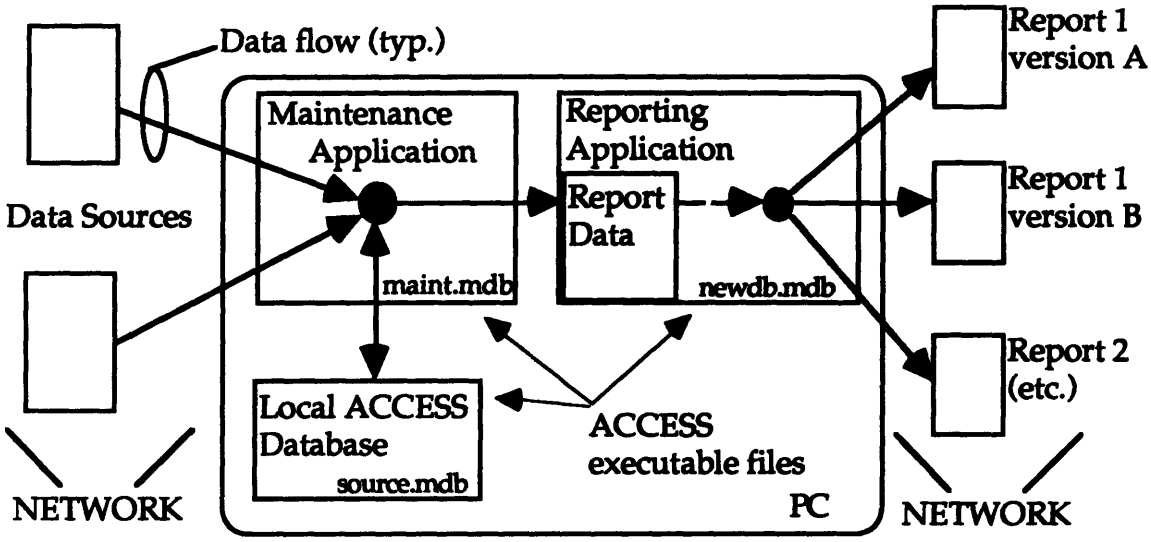



Figure 5.1: Data Maintenance and Reporting Applications

## 5.2 Data maintenance application description

The data maintenance application is a Microsoft ACCESS executable file called MAINT.MDB. This is a menu driven software application that collects every month, the data for the product performance reports from the various data

sources across the network. Additionally, it manipulates this data into the format required by the reporting application (report data). The data that need to be stored for future reference is sent to a local Microsoft ACCESS database file.

The maintenance application file does not store any data. Instead, it uses pointers to relational database tables[21] and text files to access and manipulate data from several network locations. The ODBC[22] protocol is specifically used to obtain data from Rdb tables. There are two main parts to the way the maintenance application is organized, the user interface and the update architecture.

### 5.2.1 User interface

The user interface includes those components of the software that allow a user to execute the update code, as well as add or delete existing updates to the menu[23]. Its main components are a main menu form in conjunction with up to eight update specific forms (Update_1 to 8) and macros (Macro_1 to 8). A form in this context can be described as a window with displayed data and possibly buttons that execute certain actions. These actions are contained in macros which are nothing but saved sequences of user commands. Finally, a macro usually executes an update procedure written in ACCESS BASIC (see below).

---

[21]Relational data depository organized as data fields (columns) and records (rows).
[22]Microsoft's open database connectivity protocol to access relational databases.
[23]Each item on the menu is considered a separate update.

The main menu form is the one initially displayed by the application and serves as a common interface to the various updates. The update specific forms and macros are opened from this main form.

By clicking on a button and following a series of prompts, the user can immediately create a new update form and macro. If the user has created a procedure (i.e. UPDT) that executes a given update, (s)he would only need to open the newly created macro for editing and insert a command to run the procedure in order to incorporate the new update into the menu. Figure 5.2 shows the relationship between the form (Update_1) and the macro (Macro_1) for the first update.

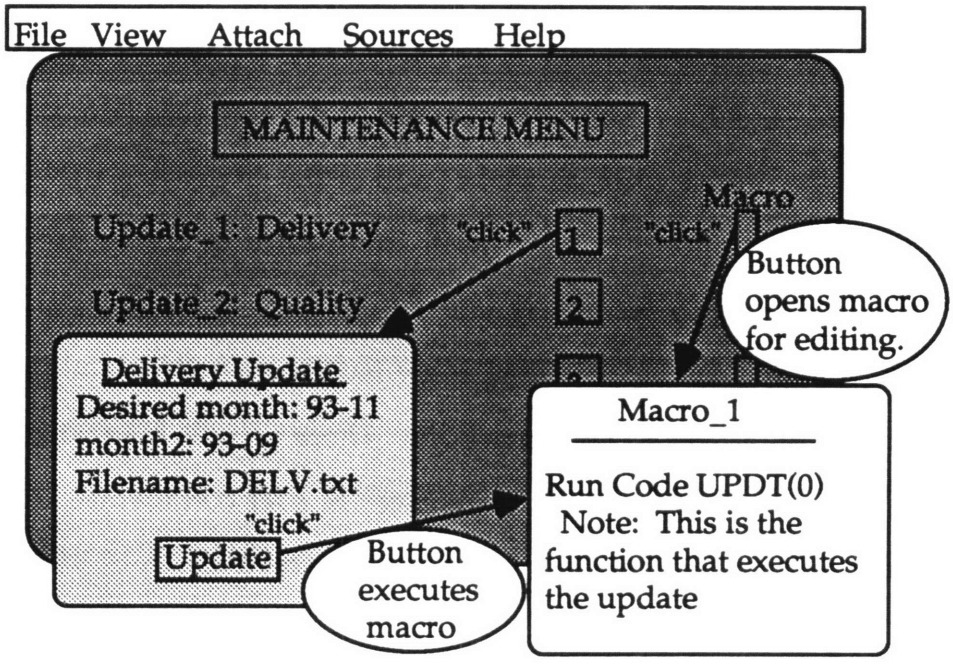



Figure 5.2: Relationship between Update Form and Update Macro

Update routines developed outside of Microsoft ACCESS can still be called or monitored from within the maintenance application. These routines can be

used to execute automatic updates by making use of a server's batch facility, or to provide greater speed over the ACCESS based updates by eliminating some of the network overhead. In these cases the update macro would contain a command to run a batch file.

For the update procedures developed within the maintenance application, a special debugging feature is included allowing the user to follow the logic behind a given update. Each update procedure is structured so as to receive a numeric argument that corresponds to different ways of execution. The argument can be equal to zero, one, or two and the execution occurs as described on Table 5.1 (Note: Here we continue to use UPDT as the name of our sample update procedure).

| Function Call: | Description: |
|---|---|
| UPDT(0) | Runs the actual update. |
| UPDT(1) | Shows the SQL statements for the various queries in appropriate sequence. |
| UPDT(2) | Shows the description associated with the various queries and thus conveys the logic behind the queries in narrative form. |

Table 5.1: Calls to an Update Procedure

To make use of either of the debug modes (i.e. one or two), the user need only to change the argument of the update procedure in the appropriate macro and then run the update as usual. Instead of the regular update, a series of text windows will appear describing the progression of the update.

### 5.2.2 Update architecture

Data updates can be viewed as a sequence of queries or data manipulation commands in SQL that are a function of a number of parameters or variables that are subject to change (i.e. monthly). The updates access source database tables that provide the needed data, and destination tables that receive the collected and processed data. Source text files are treated in a similar fashion to relational table data. They are first imported into a temporary ACCESS database table, then are manipulated through the use of SQL, and are deleted afterwards.

As mentioned above, the sequence of data manipulation steps is contained in an ACCESS BASIC procedure. This procedure contains the source code with the necessary commands to perform an update. The sample structure of an update procedure is shown on Figure 5.3 below.

```
Define Function UPDT(n)                    'Name of the update function
      Update form = Update_1               'Name of the form used for the update
      month2 = month1 - 6 months           'Parameter month2 is derived
            .
            :
      SuperQueryMaster ("Updt_Make_1",n)       'The rest of the module is
      SuperQueryMaster ("Updt_Make_2",n)       'a series of queries that
      SuperQueryMaster ("Updt_Alter_1",n)      'are executed in order.
      EmptyTable ("DELIVERIES",n)
      etc.
End Function
```

Note: See the previous section for an explanation of the argument n.

Figure 5.3: ACCESS BASIC Update Procedure

The first part of the update procedure defines the name of the update form, file pointers, and parameters corresponding to the desired update. The update form, in particular, contains the values for the pointers to any required text files and the user specified and derived parameters. For example, if data must be obtained within a six-month window, the user might specify the last month of the search window (month1) by typing it on the update form. The procedure would then derive the other required month (month2) by subtracting six from the user supplied month.

The second part of the update procedure is nothing but a sequential execution of the queries that make up the update. There are two types of these query commands, standard generic queries, and queries that are particular to a given update.

A standard query is one that executes essentially the same way except for a few parameters that can be changed. An example of a standard query is EmptyTable("tablename",n). This standard query receives the name of a database table as an argument and proceeds to delete all the records of that table.

Queries that need to refer to a number of fields of specific tables must be custom created for a given update and executed through the use of the SuperQueryMaster("queryname",n) function. This function accesses a table of stored query definitions that contains three fields; name of the query, query description, and SQL code. It references the table by query name in order to execute the appropriate SQL code (or display the description of the query depending on the value of the argument n). Figure 5.4 shows how

queries called from a procedure can be viewed on a special form that displays the information of each query entry.

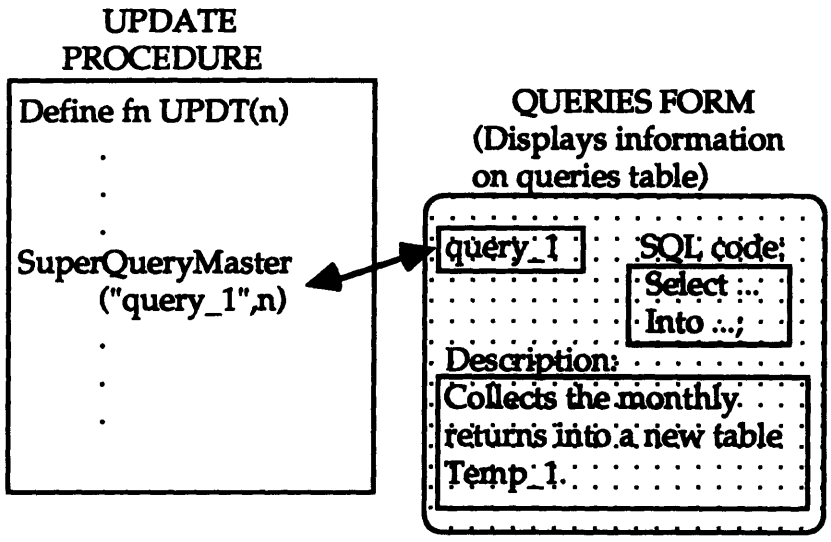



Figure 5.4: Viewing Information on Update Queries

Both the update procedure, as well as the query information can be printed to generate the documentation for a given update. If a particular query is modified, it can be printed and substituted for the older version to keep the documentation up to date.

## 5.3 Reporting application description

The reporting application is a Microsoft ACCESS executable file called NEWDB.MDB. This is a menu driven software application that generates, every month, the customer product performance reports along with other internal reports.

This reporting application stores its own data to generate the reports. Therefore, once the report data is updated by the maintenance application, the reporting file can be copied or mailed electronically to multiple people who could then generate the reports that they needed[24]. There are two main parts to the way this software tool is organized, the report architecture and the user interface.

### 5.3.1 Report architecture

To create an appropriate level of flexibility for the customer product performance reports and simplify some of the complexity surrounding their creation the following architecture was used. A given report would be composed of a main report, and a series of sub reports following a hierarchical arrangement. The main report would be created based on an audience and topic, and would consist of a particular data organization, and page header/footer information. The sections of this main report would be added as sub reports and linked to the main report through appropriate data fields as determined by the data organization of the main report. As an example, the final organization of the product performance reports is shown on Figure 5.5 below.

[24]Assuming they had a PC with Microsoft ACCESS installed.

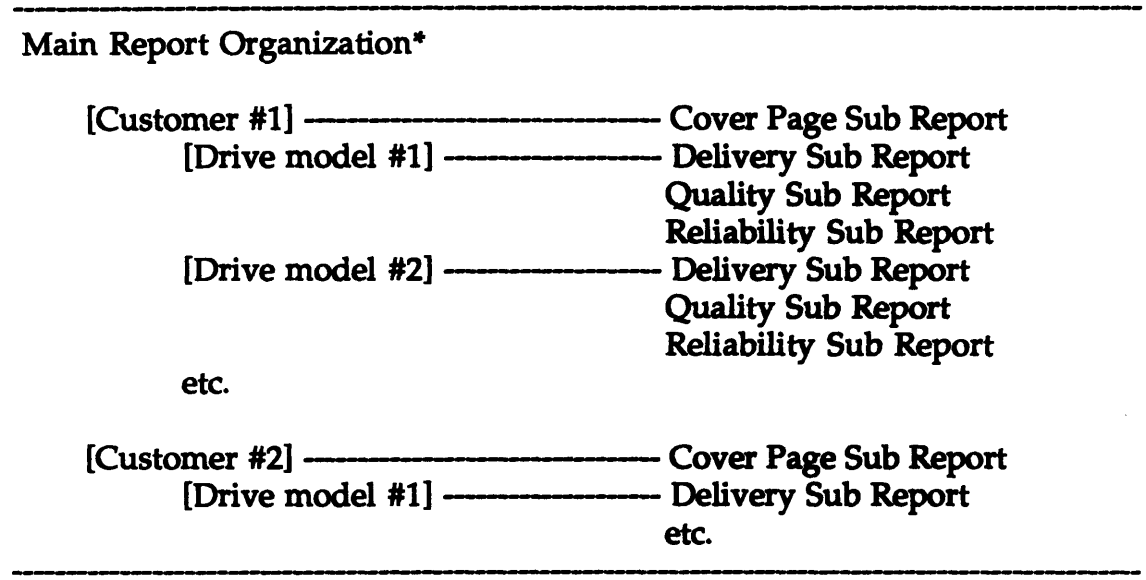



Figure 5.5: Structure of Product Performance Reports

A link between the main report and a sub report refers to both the location of the sub report within the main report organization, and to the sub report data that is to be displayed. The link between the Delivery Sub Report above and the main report consists of the sub report location; the first performance metric to appear for each drive model, and the displayed data; the delivery metric only for a given drive model and customer.

The advantages that this arrangement provides over having a single complex report are many. First of all, if the user wants to modify the presentation of the delivery data, (s)he only needs to modify the Delivery Sub Report. If the user wants to create a new sub report variant, then (s)he can edit the sub report and then save the modifications as Delivery Sub Report #2 and change the main report link to point to the new sub report (i.e. #2). Also, the order of

the sub reports can be easily rearranged, and up to 3 additional sub reports can be added to the main report. Again, all this can be done by changing the links to the main report using the customization window described in Chapter 3. With a single complex report, determining where to make the changes and how to make them would be much more difficult.

### 5.3.2 User interface

The user interface includes those components of the software that allow the user to view, print, and print to file the various reports, as well as add or delete existing reports from the menu. Its main components are a main menu form along with up to eight report specific forms (Report_1 to 8) and macros (Macro_1 to 8).

The main menu form is displayed initially by the application and serves as a common interface to the various reports. It looks very similar to the main menu form of the maintenance application (except for the colors and some of the wording) in order to maintain a uniform interface across the two applications. The report specific forms and macros are opened from this main form and are related in the same way that the maintenance application's update forms and macros are related.

The report specific forms are similar to the update forms except there are buttons for viewing and printing instead of updating. The macro for the reports also operates like the update macros except it contains commands for viewing and printing that must be specified when a new report is added to the menu. Similarly to the maintenance application, new reports can be

added to the main menu by selecting a New Report menu option and following a sequence of dialog boxes.

## 5.4 Integration into the network environment

This section describes how the PC application files interact with the computer network at the manufacturing plant. First, a brief description will be given on the plant's computer network and how the PC containing the two applications is connected and interacts with the network. Then, the various data interactions will be described in more detail.

### 5.4.1 The PC and the network

Figure 5.6 is a schematic representation of Digital's computer network showing the existing arrangement where different network nodes contain different kinds of information.

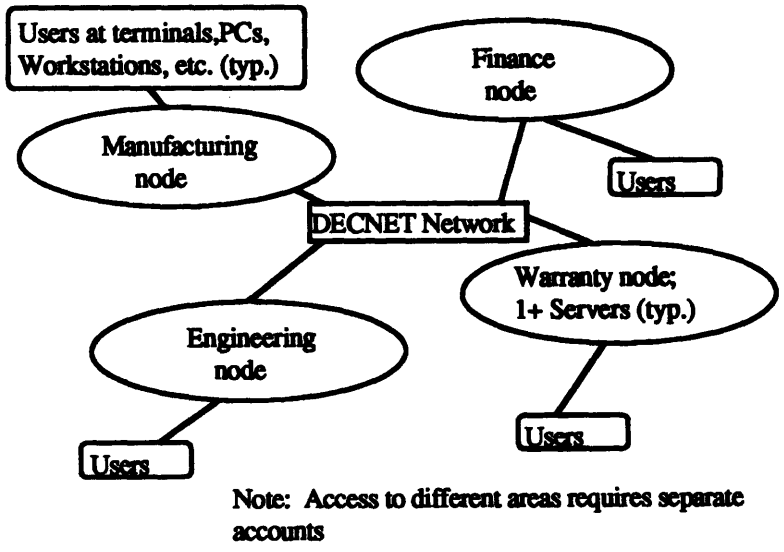



Figure 5.6: Digital's Small Form Factor Computer Network

Each network node has a different system administrator in charge of installing and updating software as well as setting up user accounts and granting file storage space. Users typically operate out of a main account but will have accounts on other nodes if necessary.

The PC running the Microsoft ACCESS applications is connected to the plant's DECNET[25] network to gather the data from the various network nodes and to distribute the reports via electronic mail[26]. Digital PATHWORKS client server software connects the PC based software to the network. To establish the connections, the user of the PC software must have accounts on the various nodes and access to the directories and data sources.

Once the data connections are set up, the client server and ODBC software become mostly transparent to the user of the maintenance and reporting applications. Directories on network nodes look like PC directories, and network database tables look like local ACCESS tables. In the maintenance application, changing a data source only requires minimal modifications to the relevant update procedures (although changing the source is not trivial). This is a very important flexibility requirement for the maintenance application since the sources of the required data can change over time as mentioned earlier.

---

[25]Digital's networking software that runs on server and client nodes in both local area and wide area networks.

[26]Remote digital sites were connected to Colorado's network and could receive email.

### 5.4.2 Using network drives to transfer data

PATHWORKS allows a user's network directory to become a network drive on the PC. DOS commands can then be performed on the files contained in this directory. These network directories are exploited in two distinct ways.

From a data gathering perspective, text files located on such a directory can be imported into the maintenance application as if they were regular DOS files. From a report distribution perspective, output postscript files can be sent to the network directory and then a command file executed to distribute the postscript files via electronic mail.

### 5.4.3 Obtaining Rdb data from the network

Data requests sent across the network operate in the following manner. First, ACCESS generates a request for Rdb data. This request is passed to the ODBC Administrator residing on the PC. The administrator makes use of Digital's ODBC Rdb Driver to reformulate the request in a format that can be sent over to a network server with an SQL/Services facility. This facility manages the Rdb data requests that travel across the network. SQL/Services gathers the requested information from the node where the data resides and returns it to the PC. The process is illustrated in Figure 5.7 below.

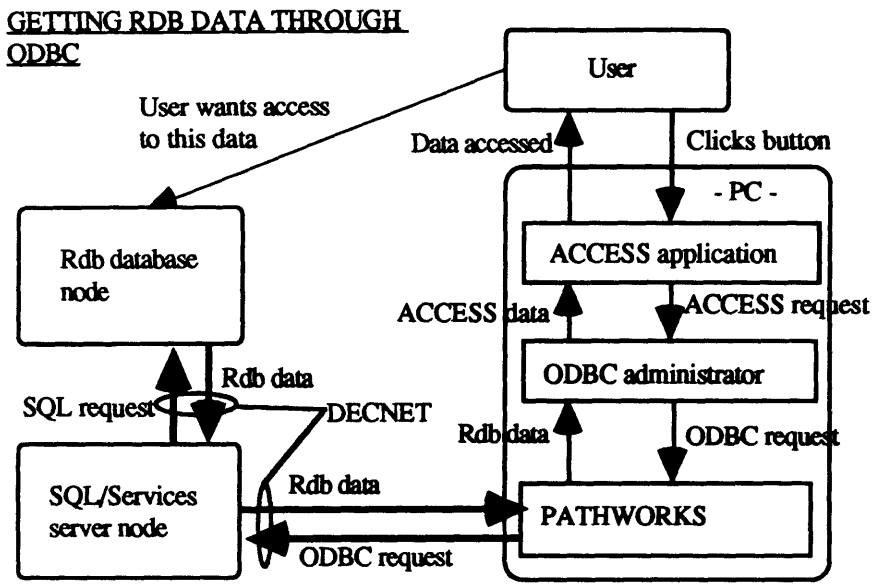



Figure 5.7: Getting Rdb Data through ODBC

The ODBC approach allows access to a variety of data sources provided that an ODBC driver exists for them. On the other hand, there are various drawbacks to ODBC used with Rdb. First, the driver does not have full capabilities for data manipulations. In particular, the ODBC driver cannot delete rows from remote Rdb tables at present. Also, attempted connections will fail when the SQL/Services facility is busy, and the connection is sensitive to the settings of both SQL/Services, and of Rdb at the remote data source. On the other hand, Digital supports this approach and it is expected that most of these problems will be eliminated over time.

# CHAPTER 6

## Description of the Data Gathering & Report Generating Process

This chapter will provide a description of how the implemented data gathering and report generating process worked at the end of the internship. Some problems and issues that arose both during and after implementation will also be discussed to show why automation was not taken any further.

### 6.1 Execution of the data gathering and reporting activities

The process of data gathering and report generation worked as two sets of distinct activities after implementation. There were activities performed by the customer satisfaction group, as well as those performed by the person responsible for generating the reports.

During the course of the month, the customer satisfaction group updated two text files with information on customers and part numbers. Judging from the rate of change of this data, these updates would be infrequent (1 or 2 lines of text might change every month). Also, customer satisfaction maintained the electronic mail distribution list for each customer's account team.

At the end of the month, the person running the updates generated two output text files from the financial/inventory database to be used for the data updates. The monthly updates were then executed sequentially by stepping through the menu selections within the maintenance application. The end

result of this work was to ready the report data before generating the reports. Note that the time required for these tasks would ideally be less than an hour and that constant attention to the maintenance application would not be required during the whole time.

The individual performing the updates then switched to the reporting application's main menu by clicking a button that automatically invoked this application from the data maintenance application. Following the reporting main menu, the user then generated and electronically mailed two internal reports (postscript files) with aggregate product metrics that were to be reviewed at the next staff meeting of the Small Form Factor organization. The user then generated and printed one report containing the performance metrics of all customer part numbers, and gave it to the customer satisfaction manager. Finally, individual postscript files of the customer reports were generated and mailed to the account teams through the execution of a command file that referenced the distribution lists maintained by the customer satisfaction group. Copies of the postscript files were also left in a directory accessible to the customer satisfaction group.

## 6.2 Problems with the execution of the activities

The real data gathering and report generating process was somewhat more problematic than the simple description above. As a result, more coordination was needed to generate the reports. The following issues required additional coordination and flexibility in the execution of the updates:

- *Data on returned drives for a given month was not fully entered until a few days after the end of the month:* The number of days would vary depending on how many drives were returned near the end of the month. Therefore, the person running the updates had to confirm with the group entering the returns data that all returned drives had been entered into the warranty/returns database before running the updates.

- *The files containing the shipment information were updated at the end of the fiscal month:* If the fiscal month ended before the calendar month, then the last few days of shipments would be missed[27]. This data collection activity was altered to get the shipments information and the returns for the fiscal month instead of the calendar month even though the reports didn't indicate this.

- *New part numbers could appear in the configuration database before being updated by customer satisfaction:* In this case, an error occurred during the report generation activity and the updates and reports had to be rerun after customer satisfaction had made the necessary changes.

Because of the different times at which data became available, the person responsible for data maintenance and report generation had to first verify that the data was good before proceeding with an update. It is primarily because of this limitation that the data maintenance activities were not fully automated.

Other issues that surfaced during and after implementation included:

---

[27]Unless they were looked-up in a different file with the current month information.

- *The scheduled shipment date for an order was sometimes not updated to reflect the change to a faster shipping method:* Many times, this change allowed an order to get to the customer on time but this would not be recognized during the delivery performance calculations. The result was that some orders that shipped on time based on the estimated transit time, appeared as late orders on the reports. In rare instances, the scheduled ship date was not entered for a given shipment but an inspection and editing of the output file containing the shipping transactions would permit the correction of this error.

- *Problems with the SQL/Services facility sometimes caused an update to crash:* This situation led to multiple update attempts. It was apparently a result of increased use of the facility (i.e. the SQL/Services were shared across multiple users). A dedicated SQL/Services facility on a different network node was being considered to avoid this problem.

- *Changes to the PC configuration sometimes caused errors:* The individual responsible for the reports frequently upgraded both PC software and hardware and these activities sometimes resulted in unexpected errors during the operation of the maintenance and reporting applications.

These examples illustrate some of the difficulties in automating the set of data gathering and report generating activities to produce reliable reports. On the other hand, the need for periodic data gathering and reporting activities served to sustain a higher awareness of the existing data integrity and accessibility problems.

# CHAPTER 7

# Impact and Reactions to the Project

This chapter will first provide a summary of the main decision points affecting the direction of the internship project. It will then discuss subsequent events and feedback obtained after the introduction of the product performance reports. From the discussion, a better approach to similar projects will be proposed. Finally, some areas will be pointed out that should receive further research emphasis in the future.

## 7.1 Review of the main project decisions

Among the many internship activities, certain decisions were of particular importance during the course of the internship project as they significantly affected the direction of the development work. In chronological order, these were:

- *The initial definition of the report concepts, in particular the "minimum requirements" report and its corresponding data and format:* These definitions provided a clear set of requirements for the internship project, bounded the scope of the project, and guided the selection of the software platform that would be used to develop the maintenance and reporting applications.

- *The selection of Microsoft ACCESS as the platform for creating the product performance reports:* This selection launched the software development efforts to create the maintenance and reporting applications.

- *The selection of a centralized approach to the various data maintenance and reporting tasks:* The selection of a single individual to perform these tasks guided the development of both the data maintenance application and the data collection and processing activities. It also narrowed down the available approaches to dealing with data accessibility, integrity, and interpretation problems.

- *The selection of the locations for data residence:* The selection of a PC based ACCESS database to hold the aggregate monthly data for the product performance reports greatly limited the accessibility to the data at the same time that it simplified the security necessary to protect this data. This was the last significant decision that was taken. Afterwards, efforts concentrated on improving and debugging of the software applications, and simplifying and documenting the data gathering and reporting activities.

Even though these decisions led to the successful implementation of the customer product performance reports, questions remained. First, it was unclear at the end of the internship how successful the reports would be in improving relations with customers. Second, it was hard to tell how successful was the sequence and timing of the decisions compared to other possible approaches.

## 7.2 Feedback and reactions to the product performance reports

Beginning two months after the end of the internship and the introduction of the reports, feedback was sought from both the manufacturing site as well as from account teams on the impact and reactions to the reports. This feedback was only obtained from a limited number of individuals through loosely structured phone interviews.

Because the interviews were limited and some of the statements were contradicting, a comprehensive assessment of the reactions to the product performance reports could not be made. However, it seemed that the following had occurred:

- The customer satisfaction group was beginning to ask more questions as its members became more familiar with the data contained in the reports.

- An internally generated pareto of drive failures was being matched to the overall quality and reliability metrics reported to customers.

- Changes were made to the counted returned drives to eliminate those that were only returned for upgrades and therefore were not field failures.

- At least some account team members were using part of the information provided on the reports but it was not clear if they were using just the aggregate information for a given drive capacity and size or the customer specific information provided in the product performance reports.

- At least some account team members wanted additional information that was not included with the reports.

- The automatic distribution of the product performance reports was stopped and customers were not being shown the reports.

Though sketchy at best, these observations called into question both the development approach chosen to create the product performance reports, as well as the effectiveness of the reports themselves, at least in the short term.

### 7.3 Suggestions for a better development approach

Based on the internship experience, and on the perceptions of what transpired afterwards, the following recommendations are made for improving the development approach to a similar project:

- *From the beginning, work with the direct customer contacts to find out which information is needed by them and by customers, and why it is needed.*

Sufficient feedback from account teams was not obtained early enough during the project. This earlier feedback would have helped the customer satisfaction group to better anticipate the account team and customer responses to the reports later on.

Instead, the approach taken was to involve first the key account team members that were already pushing for the information on product

performance. The logic behind this approach was that these people would be more receptive to the reports and more willing to provide constructive feedback. Others, who could be going through a period of strained relations with manufacturing, might have a negative or indifferent reaction towards the reports if these were not sufficiently accurate or helpful.

Therefore, the account team members were exposed to the reports following a loose order of who should see them first. Feedback was implemented along the way so that the account team members who got to see the reports later on were exposed to more polished and rigid versions. In the end, differences between those who viewed the reports as a marketing tool, and those who viewed them as a quality improvement tool could not be properly addressed when they surfaced.

- *Establish early on what goals will be set and what improvement actions will be taken within manufacturing and by customer contacts based on the reported information.*

There wasn't enough searching early on for answers to questions about how the data would be used and acted upon. In other words, people tended to agree that it was good to have the data but the mechanisms needed for improving it were not made clear.

Also, only two members of the satisfaction team took a leadership role in pushing for the development of these reports. The involvement of the other (newer) members was far less and would have helped in identifying possible mechanisms for improving the performance metrics.

- *Establish what additional information would be needed internally to guide the improvement efforts.*

Without concrete improvement targets, the ability to correctly interpret the data did not become a priority. Therefore, the missing information that would be needed to lead improvement efforts was not identified.

- *Establish responsibilities early on for data gathering, report generation, and corrective actions.*

Not discussing earlier what human, hardware, and software resources would be available to implement the data gathering and reporting activities was an oversight that ultimately delayed the implementation of the product performance reports. An earlier assessment of the available resources would have allowed a faster development and hand-off of the data gathering and reporting activities to the responsible individual. This would have allowed the reports to become available to account teams earlier and would have allowed for feedback on the product performance reports to be requested earlier.

- *Develop a set of reports for internal use in parallel with those developed for the customers.*

The customer product performance reports were too bulky to use internally. Customer satisfaction group members and account team members would have to refer to data across many customers and needed a report that would

provide the information for all their customers in a more condensed manner. This way, the difference in product performance metrics across customers would become explicit.

- *Leave a responsible person, and an action plan in place for the further development and improvement of the product performance reports.*

At the end of the internship, it was clear that more sections would have to be added to the product performance reports to provide additional information that was being requested by account team members. Also, the customization capabilities of the reporting applications needed further development and a strategy for their implementation. No person was identified as being in charge of any of these tasks.

## 7.4 Suggestions for further research

In addition to the project specific recommendations, additional topics were identified that could provide challenging research opportunities. These topics include:

- *Achieving a better understanding of the requirements associated with the successful introduction of new bottom-up information system innovations in multiple stages across different groups.*

- *Developing guidelines for the analysis and selection of software development platforms to satisfy evolving information needs in a distributed computing environment.*

- *Developing analysis tools for achieving a better understanding of the relationship between a given manufacturing organization and its information systems infrastructure.*

These of course, are only a few examples of additional research topics suggested by this project.

# References


Anagnostopoulos, Paul C., VAX/VMS Writing Real Programs in DCL, Digital 1989.

Avison, D., Kendall, J.E., DeGross, J., Human, Organizational, and Social Dimensions of Information Systems Development, North Holland . 1993.

Date, C.J. , An Introduction to Database Systems Vol.1 5th edt., Addison-Wesley 1990.

Digital manual, VAX Rdb/VMS Guide to Using SQL/Services, Digital (Int) 1990.

Digital manual, VAX Rdb/VMS Guide to Using SQL, Digital (Int) 1990.

Heeg, G., Magnusson, B., Meyer, B., Technology of Object-Oriented Languages and Systems, Tools 7, Prentice Hall 1992.

Massiglia, Paul , Digital Large System Mass Storage Handbook, Digital 1986.

Meyer, Marc H., Innovation in Large Organizations: A Study of the Diffusion of Decision Support Technology, MIT Masters Thesis, 1980.

Microsoft manual, Microsoft ACCESS User's Guide, Microsoft 1992.

Perkinson, Richard C., Data Analysis, The Key to Data Base Design, QED 1984.

Rogers, Everett M., Diffusion of Innovations, 3rd edt., New York Free Press 1986.

St. John Bate, Joseph, Vadhia, Dinesh B., Fourth-Generation Languages under DOS and UNIX, BSP Professional Books, 1987.

Trimble, J.H. , Chappell, D., A Visual Inroduction to SQL, Wiley 1989.

Tsichritzis, Dionysios C., Lochovsky, Frederick H., Data Models, Prentice-Hall 1982.

## Appendix A

# DATA MAINTENANCE AND REPORTING APPLICATION USER'S GUIDE

## INTRODUCTION

This software package is designed to let you easily access data from various sources and incorporated into a variety of customized reports. It is composed of two distinct modules:

Maintenance module - This module takes care of all the data update and maintenance routines. It leaves the data in an appropriate format for generating the reports.

Reporting module - This module generates various reports from the available data and includes capabilities for customizing, altering, or generating new reports.

## REQUIREMENTS

**Hardware:**
Personal Computer:
Intel 386, 486 microprocessor recommended.
8 MB of RAM, 16 MB recommended.
Network card

**Software:**
Microsoft Windows (Application developed using 3.1)
PATHWORKS/DECNET Software for attaching Rdb tables, connecting network drives, and running remote updates.
Microsoft ACCESS (Application developed using version 1.1)
ODBC Administrator for attaching Rdb tables
Digital's Rdb Driver for communicating with Rdb tables (Application developed using version T1.1).

**Network Installation:**
VMS/Rdb 4.1 or greater for remote databases.
An accessible network node running SQL/Services 4.1 or greater for communication with Rdb tables.

**References:**

| | |
|---|---|
| For a user: | Introduction to Microsoft ACCESS. |
| For development work: | A Visual Introduction to SQL. |
| | Microsoft ACCESS User's Guide |
| | Microsoft ACCESS Introduction to Programming |

## INSTALLATION

Copy the following files to your C:\ACCESS directory:

| | |
|---|---|
| SOURCE.MDB | Data Module: Holds local updated data. |
| MAINT.MDB | Maintenance Module: Performs data updates. |
| NEWDB.MDB | Reporting Module: Generates reports. |

Notes:

- If you have attached tables from an SQL database, you will have to reattach them. Make sure you delete each table before reattaching the new one.

- PATHWORKS has to know the nodes where the SQL server and the database reside. You can type "C:\decnet\list known nodes" from the DOS prompt to verify this and "C:\decnet\define node *address* name *nodename*" to add a new node.

- You must open each update window in the Maintenance Module (MAINT.MDB) and change the file paths where applicable (by selecting **[Edit][Design View]** while viewing the update window).

# INSTRUCTIONS FOR RUNNING A MONTHLY UPDATE

**I- Get information from MAXCIM.**
Shipments and delivery information is obtained from the MAXCIM inventory transaction system. Queries to this system are performed using FIS reports that generate output text files with the requested data. The FIS reports specify the data to be obtained, the output format, and the name of the output file for the requested data. The common file extension for these reports is .FCF.

For the monthly product performance updates, two of these FIS reports are used to obtain data on shipping transactions and total shipments for the month. They have corresponding output files that store the requested data once it is obtained from MAXCIM.

| FIS Report | Output File |
|---|---|
| MTBF.FCF | SHIPS.TXT |
| OEM_LOAD.FCF | LOAD_OUT.DAT |

To use these FIS reports, you first need to edit them so that the output data will be for the correct month. The lines that need to be edited are marked with an (*). Example:

```
A:TRANSACTION DATE >= "100193"V        !*Selection Expression 1
      (change the date above)_/          \_(asterix indicates lines to
edit)
```

1- To get the total shipments for the month, edit MTBF.FCF as follows:
- Transaction Date must be equal or greater than the first day of the update month (fiscal).
- Transaction Date must be equal or less than the last day of the update month (fiscal).
- Edit the Break Logic Section so that the update month appears as YY-MM.

Note: This file gets its data from the INHIS data file in MAXCIM. INHIS gets new data on the first Monday after the end of a fiscal month. You might have to run MTBF.FCF an additional time to get the last few days from the calendar month by having it call INTRS instead of INHIS. You can then edit the output file SHIPS.TXT to add the missing shipments.

2- Shipping transactions for the month. Edit OEM_LOAD.FCF as follows:
- Date Last Shipment must be equal or greater than the first day of the update month (calendar).

- Date Last Shipment must be equal or less than the last day of the update month (calendar).

**II- Enter the Maintenance Module**

You can enter the Maintenance Module by double-clicking on the file MAINT.MDB or opening this file from MSACCESS. When you enter this module, you will see a window (General Menu) with a list of update options. The options are arranged to be run in sequence since later updates depend on data gathered by previous ones.

When you click on the update number, a second screen will appear with information for that specific update. Verify that the correct month for the update and correct input file names are displayed, then click the **[Run Update]** or **[Execute Batch]** button to execute the update routine.

1- Select and run update #1 from the General Menu.

- This update will ask you to edit the FIS files if you haven't done so yet. Once it runs, it will also ask you to open the output files to verify and edit the data. Expect to wait a few minutes before you are able to look at the output files.

- You should eliminate the last line of the file indicating the number of records selected, as well as any empty lines. Also, you will see 2 consecutive date fields in each row of LOAD_OUT.DAT. If the first one reads "00-XXX-0000", you should edit it to match the second field, and replace the characters that follow the second field with a zero. Example:

| | |
|---|---|
| Before editing: | "00-XXX-0000", "23-OCT-1993", %3429, |
| After editing: | "23-OCT-1993", "23-OCT-1993", 0, |

Note: The strange string "00-XXX-0000" means that there was no scheduled ship date entered to ship the units. The number after the two dates is supposed to be the variance between scheduled and actual ship date in days and therefore is meaningless when there is no scheduled ship date. The above example presumes that the units shipped on time.

2- Select and run update #2 from the General Menu.

- This update gets the monthly ships and returns by part number for the month. It puts the information into a temporary table, and then will prompt you to continue if you want to update (Do you want to update? append? delete?). You should select yes for these prompts to continue, or select no if you want to cancel the update.

Note: This update takes the longest to run because it gets its return data from the RATS warranty database across the network. If you get an ODBC call failed message during this update, it might mean:

1)Either the SQL server, or database node (DEATER) is not working. Contact the system manager to verify.

2)No processes are available at the SQL server. Try again later. Also, if the PC hangs while executing an ODBC call, you might want to reconnect to the network to insure that old processes that were not killed are making the SQL server unavailable.

3- Select and run update #3 from the General Menu.

- This update gets the individual shipping transactions for each part number for the month and uses this data to generate the delivery performance metrics. Again, you will see messages that will ask you if you want to update/append/delete.

- In addition, this update will have messages indicating that errors have been encountered and you should proceed in the following manner:

1)If the error indicates that fields were deleted, it should be ignored. This update tries to convert information in a text field to a date format and it is just informing you that the text field did not contain a valid date.

2)If the error indicates that records were deleted, the update will create a table called "Import Errors - *yourname*". You should check this table after the update is finished (Select [View][Database], click on the tables icon, and double-click on the table name). The table will indicate the text file line numbers that were not imported. If the line number contained valid data, you must edit it and rerun the update (See instructions for editing the MAXCIM output files in section 1 above).

4- Select and run update #4 from the General Menu:

- This update generates the 5-month window MTBF by part number for the month. It puts the information into a temporary table, and then will prompt you to continue if you want to update (Do you want to update? append? delete?). You should select yes for these prompts to continue, or select no if you want to cancel the update.

5- Select and run update #5 from the General Menu:

- This update copies the necessary information to generate reports for a given month to the Reporting Module (NEWDB.MDB) tables. It should run without prompting you.

- If you get an import error, you should verify that the CUST_ACC.TXT and PART_INF.TXT are formatted correctly, and then rerun the update.

Note: If you want to generate reports for a month that you already updated, you can rerun this update for that month (don't run the other updates), and you can then generate the reports.

6- Click on the **[To Reports]** button on General Menu to go to the Reporting Module.

# INSTRUCTIONS FOR GENERATING THE MONTHLY REPORTS

**I. Enter the Reporting Module**

You can enter the Reporting Module by double-clicking on the file NEWDB.MDB or opening this file from MSACCESS. When you enter this module, you will see a window (General Menu) with a list of report options.

When you click on the report number, a second screen will appear with information for that specific report. Verify that the correct month for the report is displayed.

1- The following two options are standard for all reports:

- **[View]** will open a window displaying the report in Print Preview mode. You can proceed to print the report by selecting **[File][Print]** from the top menu, or you can return to the report window by closing this window.
- **[Print]** will send the report directly to the printer. In addition, there is an option of printing to a file by selecting the Print To File box. If you choose this option, you will then be prompted for an output file name.

2- Options for the customer reports:

- With the customer reports you have an additional option of specifying the customer id for viewing or printing. This option is initially set up to "-*" meaning that all the customer ids (suffixes) will be included (The asterix acts as a wild card). You can edit this box to get information for certain customers only. Examples:

  Entering "-AB" will generate a report only for the customer with the "-AB" customer id.

  Entering "-A*" will generate a report for all customers with a customer id starting with "-A".

- **[Print Individual Files]** will print all the customer reports to individual files. You will be prompted for the file path that will contain the files. The individual files will be created with the customer id as the name and .EPS as the extension. Example:

  AB.EPS will be the file created for the customer with the "-AB" id.

- **[Customize]** will open a window showing the links between the main report and its subreports. From this window, you can rearrange the order in which sections are reported, use different section variants, or

add new sections to the main report. See the **MODIFYING/CREATING NEW REPORTS** section for more information on this feature.

## MODIFYING/CREATING NEW UPDATES

**I - Setting up Data Sources/Recipients**

1- Determine the sources of the data to be updated.

- The sources of the update data can be Rdb tables, MAXCIM output text, other text, MSACCESS tables, etc.
- You can use **[View][Tables]** to check if the data you need already exists in a table. Each table will have a description of the data and will tell you whether it is original, aggregate, or formatted for reports. You should use an original or aggregate table as a data source.

2- Determine where you will store the updated data.

- You can update the data directly to a report formatted table, or you can set up a new SOURCE.MDB table to accumulate the historical information, and pass along only the relevant information to the report table.

- If you want to have a historical table, you must create it first in SOURCE.MDB (double-click the file or open from within MSACCESS). Select **[File][New][Table]** to create a new table, and then save it after you enter its field names and data types. Then select **[File][Exit]** and reopen MAINT.MDB.

- To add a new report formatted table, you must first go to the Reporting Module (**[File][To Reports]**), and then select **[Sources][New Table]**. Save this table after you enter the field names and data types and return to the Maintenance Module (**[File][To Maintenance]**).

3- Attach all the tables to the maintenance module MAINT.MDB

- Any new table must then be attached to the Maintenance Module. Within MAINT.MDB, select **[Sources][Attach Table]** to do this. Finally, select **[View][Tables]** (update table info? YES) and enter the source, type, and description for your new table.

4- Create a new import format if necessary to import an input table.

- If you will be importing text as part of an update, you will need to specify the format of the imported data. Within the Maintenance Module, select **[Sources][Imp/Exp Specs]** to view the existing import export formats.
- You can create new delimited or fixed width formats if the existing ones are not appropriate. If you want to import text with the first row containing field names, make sure you include the suffix "WF" at the end of your new specification name.

**II - Creating and debugging an update**

1- Create a new update window.

- To create a new update, you must first create a window for it. Push the **[New Update]** button in General Menu to create a new window. You will be asked for an update title that will appear on the top of the update window, for an update description that will appear on the General Menu (after the "Update_#:" characters), and for the template window to use.

- You can chose from one of the following template windows for your update:
    1)No text file. This window comes with the predefined parameters *month1, month2, month3, month4, month5, firstday, lastday, newday,* and *oldday.*
    2)Text file. This window comes with predefined parameters *month1, month2, month3, firstday, lastday,* and *import_file.* The parameter *import_file* should contain the name of the text file you want to import.
    3)Monitor log. This window allows you to view the current log file of an externally running update. It uses the parameter *log_file* to store the DOS directory path and file name of the file to be viewed.
    4)Run batch. This window allows you to execute an external VMS batch job. It uses the parameter *batch_file* to store the full VMS name of the file to be viewed. You do not need to include username and password as part of the file description as this update will ask you for them when it executes.

2- Assign values to the parameters in the new update window.
- The parameter *month1* will automatically default to the previous month (YY-MM). You can use this value to get other months that you might need for your update. For example, if you need the product returns for the last five months, you can define *month2* as follows:

    *month2* = **GetNewMonth**([*month1*],-4)
    Note: In this example, *month2* is 4 months before *month1*. You can use the macro command **SetValue** to assign the value to *month2*.

  Then, you can use these parameters in a query that will look for returns between *month2* and *month1* inclusive (a five month window).

- If you selected the textfile, monitor log, or run batch option, you can edit the filename by opening the update window and selecting **[Edit][Design View]**. Double-click the box that should contain the file name and enter the desired name using the following format:

*="D:\PATH\FILENAME.TXT"*

Where *D* is the drive letter, *PATH* is the directory path, and *FILENAME.TXT* is the filename. For a VMS file specification, the name should look like this:

*="NODE::DISK:[DIRECTORY]FILENAME.COM"*

Where *NODE::DISK:[DIRECTORY]* is the VMS path to the command file, and *FILENAME.COM* is the name of the command file to run as a batch.

- You can also create new parameters within design view. Select **[View][Toolbox]** from the top menu and drag a text box (top left of toolbox) into the form. You should reference the Microsoft ACCESS User's Guide for information on unbound text boxes for additional help.

3- Create the queries and data imports/exports necessary to create an update:

- Data updates consist of a series of queries and data import/export operations executing in sequence. To create a new query, select the **[Updates][New Query]** option from General Menu. You can have queries that create new tables, and queries that append records, delete records, or update record fields in existing tables.

- In creating new queries, I use the convention XXXXTYPE# for the query name where XXXX is a four character code for the update, TYPE is the type of action query (Delete/Insert/Make/Update), and # is a digit or character to order the queries.

- You can use the following macro commands to execute queries and text import/exports from the macro associated with the update:

  **OpenQuery** - Executes the specified query.
  **TransferText** - Performs the text transfer action.

- In addition, the following functions can be called with the **RunCode** macro command:

  **EmptyTable(***tablename***,0)** - Deletes all records from a table.
  **DeleteTable(***tablename***,0)** - Deletes a table.
  **NullsToZeroes(***tablename,fieldname***,0)** - Changes field null values to 0.

Note: *tablename* is a string with a database table name. **DeleteTable** only detaches attached tables, it does not actually delete them.

**III - Final debugging and implementation.**

1- Organize and document your queries.

- Once you have tested the queries, you can select **[View][SQL code]** from General Menu to organize and document them. A window will appear with query records containing the query name, sql code, and description.

- To add your new queries, scroll past the last record until you get one with blank fields. Then, proceed as follows:
  1)Under query name, type the name of one of your update queries.
  2)Use the **[Get SQL]** button to fill the sql code field with the information contained in your query.
  3)Type a description of what your query does in the description field. The record will be saved automatically when you scroll in either direction, or if you close the window.
  4)After you enter all your queries, close this window. Select **[View][Database]** from the General Menu, click on the queries icon, and then delete all your queries by clicking on the query name and selecting **[Edit][Delete]** from the menu.

3- Create a new module and invoke the queries from a master function in this module.

- To create a new module, select [Updates][New Module] from General Menu. You can then define a master function that will be called to execute the update. I name the master function using the four character code for the update. Following is the sample syntax for a function created to execute the TEST update:

```
Function TEST (way)
Dim a, upform as Form
'---- Set Values -----
upform = [Forms][Update_3]
upform("month2") = GetNewMonth(upform("month1",-4)

'---- Run the Update ----
a = EmptyTable("TestTable",way)
a = SuperQueryMaster("TestInsert1",way)
      :
End Function
```

- The **SuperQueryMaster** function is used to execute the queries records that you entered using the **[View][SQL code]** option. It is called using the following syntax:

*a* = **SuperQueryMaster**(*queryname,way*)

Where *a* is a dummy variable, *queryname* is the name of the query as it appears in the View SQL code form, and *way* is one of the following integer values.

*way* = 0 - Execute the SQL code associated with queryname.
= 1 - Show a message displaying the SQL code associated with queryname.
= 2 - Show a message displaying a description of what queryname does.
= 3 - Create a new query called queryname (it does not get executed).

- The **GetImportDelim** and **GetImportFixed** functions are used for importing text into an MSACCESS table. You can invoke them with the following syntax:

*a* = **GetImportDelim**(*filename, tablename, formatspec, way*)
*a* = **GetImportFixed**(*filename, tablename, formatspec, way*)

Where *a* is a dummy variable, *filename* is the name of the text file to be imported, *tablename* is the name of the table to be created, *formatspec* is the Import/Export specification, and *way* is one of the following:

*way* = 0 - Execute the text import operation.
= other - Show a message displaying a description of what the function does.

4- Change the update macro so it calls the master function.
- From the General Menu, select the view macro button for the new update. Edit the macro to leave only the command **RunCode** to run your master function. To follow the logic of your update before running it, give the master function an argument value of 2.(i.e. TEST (*way*) and set *way*=2 to test the logic of your update.

5- After you test the update, call the function with *way*=0 to run the actual update.

**IV- Documentation**
You can print the module containing the main function, as well as the queries associated with the update to document your update.

To print the module, select **[View][Database]**, click on the modules icon, open the desired module, and select **[File][Print]**. To print the queries, select **[View][SQL code]** and then click on the **[Print]** button to print the desired pages.

**V- Menu Commands**
Following is a list of available menu options to help you create, maintain and document data updates and sources/recipients within Maintenance Module:

**General_Menu:**

| | |
|---|---|
| **[File]** **[To Reports]** | Exit from the Maintenance Module and start the Reporting Module. |
| **[File]** **[Close]** | Close the General_Menu window and show the tool bar. Use this option to exit to the regular MSACCESS mode. |
| **[File]** **[Exit]** | Exit the Maintenance Module and return to the operating system. |

[View]
[SQL Code]

This option opens a window that allows you to see all the query information used for the different updates. In each query record you will see the following fields:
- Query Name: The query name is used when invoking query execution from ACCESS BASIC with the SuperQueryMaster function.
- Sql Code: This is the actual SQL code that gets executed by SuperQueryMaster. You may edit this field to change the characteristics of the query and use <CNTL>ENTER to insert lines and make the code more readable.
- Description: This is the description that appears when updates are called with 2 as their argument. By giving descriptions to your queries and calling the update with the argument 2, you can follow the update logic without executing it.

Available buttons

[<<] & [>>] These buttons will let you scroll 5 records at a time in either direction.

[Print] This button will let you print some or all of the query records.

[Repl] This button will let you replace a given string sequence with a new one. For example, if you move an update from position 4 to 5, you can change all instances of Update_4 to be Update_5 for the corresponding query records.

[Make Query] This option will let you create a query out of the information in the query record. The query will be given the same name as the query name of the record. You can then edit this query by opening it to design view, and avoid making explicit changes to the SQL.
NOTE: You cannot make a query that calls a table that doesn't exist. You will need to create the table before executing this command.

[Get SQL] Once you are done creating or editing a query, you can import its SQL code by first typing the query name on the corresponding field and then hitting this button. Once the SQL appears on the sql code field, you can edit it for readability, add a description of what the query does, and print the record.

[View]
[Tables]

This option will let you view all the currently attached tables. It offers to run a query on the tables to refresh the table information. There are three ways in which this information will be displayed:

- The table name, table source, and description appear on the screen, but the table field names do not show up. On an SQL server table, this might mean that the ODBC connection failed when it tried to get the table fields information. Also, tables that were previously attached but have been deleted will also show up in this manner and you will need to delete them.

- The table name and field names appear on the screen, but the table source and description do not show up. This signals a newly attached table. You should enter the table source and description on the screen to document this table.

- The table name, source, description, and fields, all appear on the screen. All tables should be displayed this way unless recent deletions/additions have occurred.

[View]
[Database]

Using this command, you can select the appropriate icon and view all the various system objects:

Icon

Tables - Will list local and attached tables.
SYS_ denotes tables with system information.
TMP_ denotes temporary tables.
=> (Black Arrow) denotes attached tables.

Queries - Will list any queries that you have created for editing or
for a new update.

Forms - Will list the various forms used by the application.
Update_# - Denotes the form associated with a particular
update.
TEMPLATE#_ - Denotes a template form for an update.
Other forms are used by the application in various ways.

Reports - There are no reports associated with this module.

Macros - Will list system and update macros.
General_Update_Macro - General Menu commands.
General_Tables_Queries_Macro - View SQL code/Tables
commands.
General_Help_Macro - Help messages for the interactive help
screen.
Menu_ - Macros containing the customized menu commands.
Macro_# - Macro associated with a given update.
TEMPLATE#_ - Macro associated with a given template form.

Modules - Will have ACCESS BASIC functions and update
routines.
Analyzer_ - Functions used to get attached table information.
SYS_ - Modules containing functions used by the system.
Update_ - Modules containing specific update routines.

**[Updates]**
**[New Query]**

This option will let you create a new query for your update.

**[Updates]**
**[New Module]**

This option will create a new module to allow you to code your update.

**[Sources]**
**[Attach Table]**

This option will guide you through the steps of attaching a new table to the Maintenance Module. You can attach other MSACCESS tables, SQL Server tables, and others.

Make sure you first delete a table that you are trying to reattach, or else it will be attached with a "1" appended at the end of its name and all your queries will still refer to the old table. For example, if you have an attached table "CUSTOMERS" and you reattach without deleting it first, MSACCESS will keep the old table and give the new attachment the name "CUSTOMERS1".

**[Sources]**
**[Imp/Exp Specs]**

This option will let you view the list of defined import/export formats for both, fixed width, and delimited text files. You can add/delete or modify formats with this selection.

If the name of the import/export specification ends with the characters "WF" (with fields), these functions will import the first row of the text file as the field names for the table.

**[Sources]**
**[ODBC Specs]**

When you try to attach a table from an SQL Database, MSACCESS will ask you for an ODBC Data Source. This Data Source is a pointer to the type and location of the SQL database.

When you select this option, you will see a window with the names of the defined data sources and the installed ODBC drivers. You can create a data source to an Rdb Database by clicking on the RdbDriver and then clicking on the Add Name button.

You can modify an existing data source by first clicking on its name and then clicking the configure button.

## MODIFYING/CREATING NEW REPORTS

**I- Determine/create a data source**

You can generate a report using one of the following data sources:

1)Use an existing table or query. Select **[View][Database]** and click on the tables or queries icon to view the available sources. Double-click on the query or table name to view its contents.

2)Create a new query. You can reformat the data from existing tables into a new query and use the query as the data source for your report. To do this, select **[Sources][New Query]** from the General Menu.

3)Create a new table. If you need information that is not currently available, you can select **[Sources][New Table]** to create a new source for the data you need. If you need to update the new table's information from outside sources, you should attach it to the Maintenance Module (see instructions above).

**II- Create the reports**

1- Creating a main report.

- You can use the **[Reports][New Main]** option to create a new report once you determine the source of the report data. Select the fields you are going to use to organize your main report, and then push the **[Create]** button to create the main report. This option automatically adds the prefix "Rep_" to the report name you specify.

Note: This option uses the TEMPLATE1_Rep report as a template for the main report. You can modify this template if you want to change the default format for the main report.

2- Creating a subreport.

You can use the **[Reports][New Sub]** option to create a new subreport. Select the fields and headings you are going to display on your subreport, and then push the **[Create]** button to create the subreport. This option automatically adds the prefix "Sub_" to the report name you specify.

Note: This option uses the TEMPLATE1_Sub report as a template for the subreport. You can modify this template if you want to change the default format for the subreport.

3- Using the Report Wizard.

- Microsoft ACCESS also offers a Report Wizard utility that will step you through the creation of a new report.

**III- Link the main report with its sub reports.**

1- Add the subreport to the main report.

- To link a subreport to a main report, you must first open the main report to design view. To open the main report, select [View][Database] from General Menu, click on the reports icon, select the desired main report, and click the [Design] button. You can then select the database window ([Window][Database]), and drag and drop the subreport into the desired main report section.

2- Link the subreport to the main report.

- Consult the Microsoft ACCESS User's Guide to link the subreport to the main report.

- The [Customize] button in the Customer Product Performance Reports lets you view the different subreports that are linked to the main report. You can select a different subreport from the drop down list to customize this report. Up to six report sections are available for each part number. If you need to change a report section, save the currently linked subreport under a different name. Then make the modifications to the copy. Finally, change the name of the linked subreport to the new name and view the report.

- Existing linkages are as follows:

| Menu Selection | Main Report | Sub Reports |
|---|---|---|
| (1) | Rep_METR_by_capacity | Sub_QUA2_1<br>Sub_REL2_1 |
| (2) | Rep_METR_by_type | Sub_QUA3_1<br>Sub_REL3_1 |
| (3) | Rep_CUST_by_part_no | Sub_COV1_1<br>Sub_DEL1_1<br>Tny_ORD1_1<br>Sub_QUA1_1<br>Sub_REL1_1 |
| (4) | Rep_Trends_by_capacity | NONE |

**IV- Add a new report option within General Menu**

1- Add a new window for the report.

While in the General Menu window, push the [New Report] button to add a new report option to the menu. The application will ask you for a title and a description of the report information. Once the new menu option appears, you will need to modify the corresponding macro to View/Print the new report.

2- Link the macro to the main report.

In General Menu, push the view macro button for the macro corresponding to your new report option. The macro will have instructions on how you

should modify its commands to point to your new report. Modify the commands accordingly and then save the macro.

3- Test the new report option.
After you are done modifying the macro, you should test the new report option by opening the report window from General Menu and attempting to View/Print the report.

**V- Menu Commands**
Following is a list of available menu options to help you create, maintain and modify the reports within the Reporting Module:

**General_Menu:**

| | |
|---|---|
| **[File]** | Exit from the Maintenance Module and start the |
| **[To Reports]** | Reporting Module. |
| **[File]** | Close the General_Menu window and show the tool bar. |
| **[Close]** | Use this option to exit to the regular MSACCESS mode. |
| **[File]** | Exit the Maintenance Module and return to the operating |
| **[Exit]** | system. |
| **[Reports]** | This option will let you generate a new report with page |
| **[New Main]** | header and footer information. |
| **[Reports]** | This option will let you generate a sub report. |
| **[New Sub]** | |
| **[Reports]** | This option will call the Microsoft ACCESS Report |
| **[Wizards]** | Wizard utility that will take you through the necessary steps for generating a sample report. |

**[View]**
**[Database]**

Using this command, you can select the appropriate icon and view all the various system objects:

Icon

Tables - Will list local and attached tables.
SYS_ denotes tables with system information.
TMP_ denotes temporary tables.
=> (Black Arrow) denotes attached tables.

Queries - Will list any queries that you have created for editing or
for a new update.

Forms - Will list the various forms used by the application.
Update_# denotes the form associated with a particular update.
TEMPLATE#_update denotes a template form for an update.
Other forms are used by the application in various ways.

Reports - There are no reports associated with this module.

Macros - List system and report macros
Macro_# - Macro associated with a report option.
General_Menu_macro - General menu commands.
General_Help_macro - Help messages for the interactive help screen.
Make_New_Report_macro - New Main/Sub commands.

Modules - Lists system modules.
SYS_ - Prefix for system modules.
CUST_ - Special module for the CUST_by_part_no report.

**[Sources]**
**[New Table]**

Create a new table to serve as a data source for a new report.

**[Sources]**
**[New Query]**

Create a new query to serve as a data source for a new report.

**[Help]**
**[General Menu]**

On line help information for the General Menu options.

# Appendix B

## Documentation for Updates

Shipping Transactions
Aggregate Shipments and Returns
MTBF Calculations
Report Data Updates

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'=============================================================
======
'        MODULE FOR UPDATING THE MONTHLY DELIVERY
INFORMATION
'           USING THE SHIPPING TRANSACTIONS DATA FROM MAXCIM
'
'     SOURCE TABLES:   LOAD_OUT.DAT (Text File)
'
'   TEMPORARY TABLES:   ShipDets1    (Local)
'                       ShipDets2    (Local)
'
'     UPDATED TABLES:   SHIPMENTS    (SOURCE.MDB)
'                       MTBF_5MTH    (SOURCE.MDB)
'=============================================================
======

Function SHPS (way)
Dim a, infile As String, upform As Form
Dim temp1 As String, temp2 As String

'-------------------------------------------------
Set upform = [Forms]![Update_3]
infile = upform("import_file")
temp1 = "ShipDets1": temp2 = "ShipDets2"
'-------------------------------------------------

'1== [Set the first and last days for the desired update month ] ==
'=============================================================
=====
    upform("firstday") = GetFirstDay(upform("month1"))
    upform("lastday") = GetLastDay(upform("month1"))
    upform("message") = "Starting update": DoCmd RepaintObject

'2== [Empty ShipDets1 and ShipDets2 tables to receive new information] ==
'=============================================================
==========
   a = EmptyTable(temp1, way)      'Delete Entries from ShipDets1 Table.
   a = EmptyTable(temp2, way)      'Delete Entries from ShipDets2 Table.

'3== [Put the new information into the ShipDets2 Table]  ==
'=============================================================
    a = GetImportDelim(infile, temp1, "loader_format", way)
      'Add latest month Entries to ShipDets1 Table.
   a = SuperQueryMaster("ShpsInsert1", way)
```

```
        'Convert request date from ShipDets1 to ShipDets2.

'4== [Update the SHIPMENTS and DELV_MTH tables] ==
'================================================
    upform("message") = "Updating SHIPMENTS": DoCmd RepaintObject
    a = SuperQueryMaster("ShpsUpdate1", way)
        'Update latest month Entries to SHIPMENTS Table.
    a = SuperQueryMaster("ShpsDelete1", way)
        'Delete records from ShipDets2 that are already in SHIPMENTS.
    a = SuperQueryMaster("ShpsInsert2", way)
        'Add remaining records from ShipDets2 to SHIPMENTS.
    upform("message") = "Updating DELV_MTH": DoCmd RepaintObject
    a = SuperQueryMaster("ShpsDelete2", way)
        'Delete month Aggregates in DELV_MTH Table.
    a = SuperQueryMaster("ShpsInsert3", way)
        'Add new info for the month to DELV_MTH Table.
    upform("message") = "Update completed": DoCmd RepaintObject


End Function
```

Query Name: ShpsDelete1

SQL Code: <CNTL>ENTER=Newline

```
DELETE  DISTINCTROW ShipDets2.*
FROM ShipDets2, SHIPMENTS,
  ShipDets2 INNER JOIN SHIPMENTS ON
  ShipDets2.so_number =
  SHIPMENTS.SO_NUMBER,
  ShipDets2 INNER JOIN SHIPMENTS ON
  ShipDets2.line_no = SHIPMENTS.SO_LINE
WITH OWNERACCESS OPTION;
```

Query Description:

Deletes the records from the ShipDets2 table that were already updated in SHIPMENTS.

Query Name: ShpsDelete2

SQL Code: <CNTL>ENTER=Newline

```
DELETE  DISTINCTROW DELV_MTH.*
FROM DELV_MTH
WHERE ((DELV_MTH.month =
  [Forms]![Update_3]![month1]))
WITH OWNERACCESS OPTION;
```

Query Description:

Deletes the records from DELV_MTH for the update month.

Query Name: ShpsInsert1

Query Description:

Transforms the imported text field [customer_request_date] into a date field. It moves data from ShipDets1 to ShipDets2 to accomplish this.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO ShipDets2 ( so_number, line_no,
part_no, cust_id, order_qty, ship_qty, var_qty,
sched_date, ship_date, var_days, po_no, late_flag,
req_date )
SELECT  DISTINCTROW ShipDets1.so_number,
ShipDets1.line_no, ShipDets1.part_no,
ShipDets1.cust_id, ShipDets1.order_qty,
ShipDets1.ship_qty, ShipDets1.var_qty,
ShipDets1.sched_date, ShipDets1.ship_date,
ShipDets1.var_days, ShipDets1.po_no,
ShipDets1.late_flag, ShipDets1.req_date
FROM ShipDets1
WHERE ((ShipDets1.so_number Is Not Null) AND
(ShipDets1.line_no Is Not Null) AND
(ShipDets1.part_no Is Not Null))
WITH OWNERACCESS OPTION;
```

Query Name: ShpsInsert2

Query Description:

Inserts new records from ShipDets2 into SHIPMENTS that were not already updated.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO SHIPMENTS ( SO_NUMBER,
SO_LINE, PART_NO, CUST_ID, ORDER_QTY,
SHIP_QTY, VAR_QTY, SCHED_DATE, SHIP_DATE,
VAR_DAYS, PO_NO, LATE_FLAG, REQ_DATE )
SELECT  DISTINCTROW ShipDets2.so_number,
ShipDets2.line_no, ShipDets2.part_no,
ShipDets2.cust_id, ShipDets2.order_qty,
ShipDets2.ship_qty, ShipDets2.var_qty,
ShipDets2.sched_date, ShipDets2.ship_date,
ShipDets2.var_days, ShipDets2.po_no,
ShipDets2.late_flag, ShipDets2.req_date
FROM ShipDets2
WITH OWNERACCESS OPTION;
```

**Query Name:** ShpsUpdate1

**Query Description:**

Updates existing records in SHIPMENTS with the updated shipping information on ShipDets2.

**SQL Code:** <CNTL>ENTER=Newline

```
UPDATE DISTINCTROW ShipDets2, SHIPMENTS,
ShipDets2 INNER JOIN SHIPMENTS ON
ShipDets2.so_number = SHIPMENTS.SO_NUMBER,
ShipDets2 INNER JOIN SHIPMENTS ON
ShipDets2.line_no = SHIPMENTS.SO_LINE
 SET SHIPMENTS.SO_NUMBER =
[ShipDets2].[so_number], SHIPMENTS.SO_LINE =
[ShipDets2].[line_no], SHIPMENTS.PART_NO =
[ShipDets2].[part_no], SHIPMENTS.CUST_ID =
[ShipDets2].[cust_id], SHIPMENTS.ORDER_QTY =
[ShipDets2].[order_qty], SHIPMENTS.SHIP_QTY =
[ShipDets2].[ship_qty], SHIPMENTS.VAR_QTY =
[ShipDets2].[var_qty], SHIPMENTS.SCHED_DATE =
[ShipDets2].[sched_date], SHIPMENTS.SHIP_DATE =
[ShipDets2].[ship_date], SHIPMENTS.VAR_DAYS =
[ShipDets2].[var_days], SHIPMENTS.PO_NO =
[ShipDets2].[po_no], SHIPMENTS.LATE_FLAG =
[ShipDets2].[late_flag], SHIPMENTS.REQ_DATE =
[ShipDets2].[req_date]
WITH OWNERACCESS OPTION;
```

**Query Name:** ShpsInsert3

**Query Description:**

Inserts updated records for the month into the DELV_MTH table.

**SQL Code:** <CNTL>ENTER=Newline

```
INSERT INTO DELV_MTH ( part_number,
customer_id, month, total_orders, late_orders,
On_time_percentage )
SELECT DISTINCTROW SHIPMENTS.PART_NO,
SHIPMENTS.CUST_ID, [Forms]![Update_3]![month1]
AS month, Count(SHIPMENTS.SO_NUMBER) AS
CountOfSO_NUMBER,
Sum(SHIPMENTS.LATE_FLAG) AS
SumOfLATE_FLAG, 1-
Sum([LATE_FLAG])/Count([SO_NUMBER]) AS
On_time
FROM SHIPMENTS
WHERE ((SHIPMENTS.SHIP_DATE Between
[Forms]![Update_3]![firstday] And
[Forms]![Update_3]![lastday]))
GROUP BY SHIPMENTS.PART_NO,
SHIPMENTS.CUST_ID
WITH OWNERACCESS OPTION;
```

```
Option Compare Database   'Use database order for string comparison
Option Explicit
'=============================================================
======
'    MODULE FOR UPDATING THE MONTHLY SHIPS/RETURNS
INFORMATION
'              USING DATA FROM MAXCIM AND RATS
'
'    SOURCE TABLES:  SHIPS.TXT       (Text File)
'                RETURN_DATA     (RATS.RDB)
'                 PART_NUMBER_DATA (RATS.RDB)
'
'  TEMPORARY TABLES:   TMP_Shp
'                TMP_Ret
'                TMP_Ret2
'
'    UPDATED TABLES:   SHIPS_RETS_DATA  (SOURCE.MDB)
'                QUAL_MTH       (SOURCE.MDB)
'=============================================================
======

Function SHRT (way)
Dim a, upform As Form
Dim temp1 As String, temp2 As String, temp3 As String
Dim infile As String

'-----------------------------------------------
Set upform = [Forms]![Update_2]
infile = upform("import_file")
temp1 = "TMP_Shp": temp2 = "TMP_Ret": temp3 = "TMP_Ret2"
'-----------------------------------------------

'1== [Set the first and last days for the desired update month ] ==
'=============================================================
=====
    upform("firstday") = GetFirstDay(upform("month1"))
    upform("lastday") = GetLastDay(upform("month1"))
    upform("message") = "Starting update": DoCmd RepaintObject
    DoCmd SetWarnings False

'2== [Set up temporary TMP_Shp Table to contain the updated information]
==
'=============================================================
==============
    a = SuperQueryMaster("ShrtMake1", way)
        'Create the TMP_Shp Table from SHIPS_RETS_DATA.
```

```
    a = GetImportFixed(infile, temp1, "ships_loader_format", way)
       'Transfer text from maxcim fis into TMP_Shp
    upform("message") = "Got shipments": DoCmd RepaintObject

'3== [The temporary TMP_Ret and TMP_Ret2 Tables will contain returns
info] ==
'=============================================================
=================
    a = SuperQueryMaster("ShrtMake2", way)
       'Put all the returns from the desired month into the TMP_Ret Table.
    upform("message") = "Got returns": DoCmd RepaintObject
    a = SuperQueryMaster("ShrtMake3", way)
       'Put the reliability returns from the desired month into the TMP_Ret2
Table.
    upform("message") = "Got reliability returns": DoCmd RepaintObject

'4== [First, add the return data to existing records on TMP_Shp] ==
'=============================================================
======
    a = SuperQueryMaster("ShrtUpdate1", way)
       'Update the return information of existing records in TMP_Shp.
    a = SuperQueryMaster("ShrtDelete1", way)
       'Delete records from TMP_Ret that are already in TMP_Shp.
    a = SuperQueryMaster("ShrtDelete2", way)
       'Delete records from TMP_Ret2 that are already in TMP_Shp.

'5== [Then, add new records with return data for part numbers that didn't
ship] ==
'=============================================================
==================
    a = SuperQueryMaster("ShrtInsert1", way)
       'Insert return information into TMP_Shp for product that didn't ship
during the month.

'6== [Now TMP_Shp is updated and we can delete the temporary tables] ==
'=============================================================
==========
    a = DeleteTable(temp2, way)
    a = DeleteTable(temp3, way)

'7== [Finally, Set null values within TMP_Shp to zeroes] ==
'=============================================================
    a = NullsToZeroes(temp1, "SHIPPED", way)
    a = NullsToZeroes(temp1, "RETURNED", way)
    a = NullsToZeroes(temp1, "REL_2MTH_RETURNS", way)
    upform("message") = "Ready to update": DoCmd RepaintObject
```

```
    DoCmd SetWarnings True

'8== [Now put the TMP_Shp info into SHIPS_RETS_DATA and delete
TMP_Shp] ==
'=============================================================
==============
    a = SuperQueryMaster("ShrtUpdate2", way)
        'Update SHIPS_RETS_DATA with values from TMP_Shp.
    a = SuperQueryMaster("ShrtDelete3", way)
       'Delete records from TMP_Shp that already were updated in
SHIPS_RETS_DATA.
    a = SuperQueryMaster("ShrtInsert2", way)
       'Instert records from TMP_Shp that were not already present in
SHIPS_RETS_DATA.
   a = DeleteTable(temp1, way)

'9== [Update QUAL_MTH from the SHIPS_RETS_DATA information]  ==
'=============================================================
=
    a = SuperQueryMaster("ShrtDelete4", way)
       'Delete update month records from QUAL_MTH.
    a = SuperQueryMaster("ShrtInsert3", way)
       'Append updated information to QUAL_MTH from
SHIPS_RETS_DATA.
    a = SuperQueryMaster("ShrtUpdate3", way)
      'Set Quality = 0% where quality returns >= shipped.
    upform("message") = "Update completed": DoCmd RepaintObject

End Function
```

Query Name: ShrtDelete1

Query Description:

Delete records in TMP_Ret that were already updated in TMP_Shp.

SQL Code:

<CNTL>ENTER=Newline

```
DELETE  DISTINCTROW TMP_Ret.*
FROM TMP_Ret, TMP_Shp,
  TMP_Shp INNER JOIN TMP_Ret ON
  TMP_Shp.PART_NO = TMP_Ret.part_no,
  TMP_Shp INNER JOIN TMP_Ret ON
  TMP_Shp.CUST_ID = TMP_Ret.customer_id,
  TMP_Shp INNER JOIN TMP_Ret ON
  TMP_Shp.SHIP_MTH = TMP_Ret.month
WITH OWNERACCESS OPTION;
```

Query Name: ShrtDelete2

Query Description:

Delete records in TMP_Ret2 that were already updated in TMP_Shp.

SQL Code:

<CNTL>ENTER=Newline

```
DELETE  DISTINCTROW TMP_Ret2.*
FROM TMP_Shp, TMP_Ret2,
  TMP_Shp INNER JOIN TMP_Ret2 ON
  TMP_Shp.PART_NO = TMP_Ret2.part_no,
  TMP_Shp INNER JOIN TMP_Ret2 ON
  TMP_Shp.CUST_ID = TMP_Ret2.customer_id,
  TMP_Shp INNER JOIN TMP_Ret2 ON
  TMP_Shp.SHIP_MTH = TMP_Ret2.month
WITH OWNERACCESS OPTION;
```

Query Name: ShrtDelete3

SQL Code:

<CNTL>ENTER=Newline

```
DELETE  DISTINCTROW TMP_Shp.*
FROM TMP_Shp, SHIPS_RETS_DATA,
 TMP_Shp INNER JOIN SHIPS_RETS_DATA ON
 TMP_Shp.PART_NO =
 SHIPS_RETS_DATA.PART_NO,
 TMP_Shp INNER JOIN SHIPS_RETS_DATA ON
 TMP_Shp.CUST_ID =
 SHIPS_RETS_DATA.CUST_ID,
 TMP_Shp INNER JOIN SHIPS_RETS_DATA ON
 TMP_Shp.SHIP_MTH =
 SHIPS_RETS_DATA.SHIP_MTH
WITH OWNERACCESS OPTION;
```

Query Description:

Delete records in TMP_Shp that were already updated in SHIPS_RETS_DATA.

Query Name: ShrtDelete4

SQL Code:

<CNTL>ENTER=Newline

```
DELETE  DISTINCTROW QUAL_MTH.*
FROM QUAL_MTH
WHERE ((QUAL_MTH.month =
 [Forms]![Update_2]![month1]))
WITH OWNERACCESS OPTION;
```

Query Description:

Delete records for the update month in the QUAL_MTH table.

**Query Name:** ShrtInsert1

**Query Description:**

Insert return information into TMP_Shp for parts that did not ship for the month.

**SQL Code:** <CNTL>ENTER=Newline

```
INSERT INTO TMP_Shp
 ( PART_NO, CUST_ID, SHIP_MTH, RETURNED,
 REL_2MTH_RETURNS )
SELECT  DISTINCTROW TMP_Ret.part_no,
 TMP_Ret.customer_id, TMP_Ret.month,
 TMP_Ret.returned, TMP_Ret2.rel_2mth_returns
FROM  TMP_Ret, TMP_Ret2,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.part_no = TMP_Ret2.part_no,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.customer_id = TMP_Ret2.customer_id,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.month = TMP_Ret2.month
WITH OWNERACCESS OPTION;
```

**Query Name:** ShrtInsert2

**Query Description:**

Insert update information from TMP_Shp into the SHIPS_RETS_DATA table.

**SQL Code:** <CNTL>ENTER=Newline

```
INSERT INTO SHIPS_RETS_DATA ( PART_NO,
 CUST_ID, SHIP_MTH, SHIPPED, RETURNED,
 REL_2MTH_RETURNS )
SELECT  DISTINCTROW TMP_Shp.PART_NO,
 TMP_Shp.CUST_ID, TMP_Shp.SHIP_MTH,
 TMP_Shp.SHIPPED, TMP_Shp.RETURNED,
 TMP_Shp.REL_2MTH_RETURNS
FROM TMP_Shp
WHERE ((TMP_Shp.SHIP_MTH =
 [Forms]![Update_2]![month1]))
WITH OWNERACCESS OPTION;
```

Query Name: ShrtInsert3

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO QUAL_MTH
 ( part_number, customer_id, month, shipped,
 qual_returns, quality )
SELECT  DISTINCTROW
 SHIPS_RETS_DATA.PART_NO,
 SHIPS_RETS_DATA.CUST_ID,
 SHIPS_RETS_DATA.SHIP_MTH,
 SHIPS_RETS_DATA.SHIPPED,
 [RETURNED]-[REL_2MTH_RETURNS] AS
 QUAL_RETURNS,
 1-[QUAL_RETURNS]/[SHIPPED] AS QUALITY
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.SHIP_MTH =
 [Forms]![Update_2]![month1]))
WITH OWNERACCESS OPTION;
```

Query Description:

Insert aggregate quality information for the month from SHIPS_RETS_DATA into QUAL_MTH.

Query Name: ShrtMake1

SQL Code:

<CNTL>ENTER=Newline

```
SELECT  DISTINCTROW SHIPS_RETS_DATA.*
INTO TMP_Shp
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.PART_NO Is Null)
 AND (SHIPS_RETS_DATA.CUST_ID Is Null) AND
 (SHIPS_RETS_DATA.SHIP_MTH Is Null))
WITH OWNERACCESS OPTION;
```

Query Description:

Make a temporary table TMP_Shp with the same structure as the SHIPS_RETS_DATA table it will later update.

Query Name: ShrtMake2

SQL Code: <CNTL>ENTER=Newline

```
SELECT DISTINCTROW
 PART_NUMBER_DATA.PN_MAIN AS part_no,
 "-" & [PN_SUFFIX] AS customer_id,
 [Forms]![Update_2]![month1] AS month,
 Count(RETURN_DATA.RECORD_KEY) AS returned
INTO TMP_Ret
FROM RETURN_DATA, PART_NUMBER_DATA,
 RETURN_DATA INNER JOIN
 PART_NUMBER_DATA ON
 RETURN_DATA.CUST_PART_NUMBER =
 PART_NUMBER_DATA.CUST_PART_NUMBER
WHERE ((PART_NUMBER_DATA.PN_PREFIX Not
 Like "B*") AND
 (RETURN_DATA.RETURNED_DATE Between
 [Forms]![Update_2]![firstday] And
 [Forms]![Update_2]![lastday]) AND
 (PART_NUMBER_DATA.PN_SUFFIX Not Like "Z*"))
GROUP BY PART_NUMBER_DATA.PN_MAIN, "-" &
 [PN_SUFFIX]
WITH OWNERACCESS OPTION;
```

Query Description:

Insert the total returns for the month by part number into a new table called TMP_Ret.

Query Name: ShrtMake3

SQL Code: <CNTL>ENTER=Newline

```
SELECT DISTINCTROW
 PART_NUMBER_DATA.PN_MAIN AS part_no,
 "-" & [PN_SUFFIX] AS customer_id,
 [Forms]![Update_2]![month1] AS month,
 Count(RETURN_DATA.MTTF_HOURS) AS
 rel_2mth_returns INTO TMP_Ret2
FROM RETURN_DATA, PART_NUMBER_DATA,
 RETURN_DATA INNER JOIN
 PART_NUMBER_DATA ON
 RETURN_DATA.CUST_PART_NUMBER =
 PART_NUMBER_DATA.CUST_PART_NUMBER
WHERE ((RETURN_DATA.RETURNED_DATE
 Between [Forms]![Update_2]![firstday] And
 [Forms]![Update_2]![lastday]) AND
 (RETURN_DATA.MTTF_HOURS>1) AND
 (PART_NUMBER_DATA.PN_PREFIX Not Like "B*")
 AND (PART_NUMBER_DATA.PN_SUFFIX Not Like
 "Z*"))
GROUP BY PART_NUMBER_DATA.PN_MAIN,
 "-" & [PN_SUFFIX]
WITH OWNERACCESS OPTION;
```

Query Description:

Insert the total reliability returns for the month by part number into a new table called TMP_Ret2. Reliability returns are those where
 [MTTF_HOURS] > 1
(That is, returned over two months after shipment).

Query Name: ShrtUpdate1

Query Description:

Update the records in TMP_Shp with the information from TMP_Ret and TMP_Ret2 for part numbers that shipped during the month.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Shp, TMP_Ret,
 TMP_Ret2,
 TMP_Shp INNER JOIN TMP_Ret ON
 TMP_Shp.PART_NO = TMP_Ret.part_no,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.part_no = TMP_Ret2.part_no,
 TMP_Shp INNER JOIN TMP_Ret ON
 TMP_Shp.CUST_ID = TMP_Ret.customer_id,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.customer_id = TMP_Ret2.customer_id,
 TMP_Shp INNER JOIN TMP_Ret ON
 TMP_Shp.SHIP_MTH = TMP_Ret.month,
 TMP_Ret LEFT JOIN TMP_Ret2 ON
 TMP_Ret.month = TMP_Ret2.month
SET TMP_Shp.RETURNED = [TMP_Ret].[returned],
 TMP_Shp.REL_2MTH_RETURNS =
 [TMP_Ret2].[rel_2mth_returns]
WITH OWNERACCESS OPTION;
```

Query Name: ShrtUpdate2

Query Description:

Update existing records in SHIPS_RETS_DATA with the information from TMP_Shp.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW SHIPS_RETS_DATA,
 TMP_Shp,
 SHIPS_RETS_DATA INNER JOIN TMP_Shp ON
 SHIPS_RETS_DATA.PART_NO =
 TMP_Shp.PART_NO,
 SHIPS_RETS_DATA INNER JOIN TMP_Shp ON
 SHIPS_RETS_DATA.CUST_ID =
 TMP_Shp.CUST_ID,
 SHIPS_RETS_DATA INNER JOIN TMP_Shp ON
 SHIPS_RETS_DATA.SHIP_MTH =
 TMP_Shp.SHIP_MTH
SET SHIPS_RETS_DATA.SHIPPED =
 [TMP_Shp].[SHIPPED],
 SHIPS_RETS_DATA.RETURNED =
 [TMP_Shp].[RETURNED],
 SHIPS_RETS_DATA.REL_2MTH_RETURNS =
 [TMP_Shp].[REL_2MTH_RETURNS]
WHERE ((TMP_Shp.SHIP_MTH =
 [Forms]![Update_2]![month1]))
WITH OWNERACCESS OPTION;
```

**Query Name:** ShrtUpdate3

**Query Description:**

Set quality = 0% in QUAL_MTH for those part numbers where
[quality returns] >= [shipped]
during the month.

**SQL Code:**

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW QUAL_MTH
SET QUAL_MTH.quality = 0
WHERE ((QUAL_MTH.qual_returns >=
  QUAL_MTH.shipped) AND (QUAL_MTH.month =
  [Forms]![Update_2]![month1]))
WITH OWNERACCESS OPTION;
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'=============================================================
======
'        MODULE FOR UPDATING THE MONTHLY MTBF INFORMATION
'        USING THE 5-MONTH WINDOW OF RUN HOURS ALGORITHM
'
'    SOURCE TABLES:   MTBF_5TMH
'                     SHIPS_RETS_DATA
'
'  TEMPORARY TABLES:   TMP_Mtbf
'                      TMP_t1
'                      TMP_t2
'                      TMP_t3
'                      TMP_t4
'
'    UPDATED TABLES:   MTBF_5MTH
'=============================================================
======

Function MTBF (way)
Dim a, upform As Form
Dim temp1 As String, temp2 As String, temp3 As String
Dim temp4 As String, temp5 As String

'-------------------------------------------------
Set upform = [Forms]![Update_4]
temp1 = "TMP_t1": temp2 = "TMP_t2": temp3 = "TMP_t3"
temp4 = "TMP_t4": temp5 = "TMP_Mtbf"
'-------------------------------------------------

'1== [Set values of relevant months for calculations] ==
'=====================================================
    upform("month2") = GetNewMonth(upform("month1"), -4)
      'Set earliest month for returns.
    upform("month3") = GetNewMonth(upform("month1"), -2)
      'Set latest month for run hours.
    upform("month4") = GetNewMonth(upform("month1"), -6)
      'Set earliest month for run hours.
    upform("month5") = GetNewMonth(upform("month1"), -1)
      'Set previous MTBF month.
   upform("message") = "Starting update": DoCmd RepaintObject
   DoCmd SetWarnings False    'Turn off warnings.

'2== [Set up TMP_Mtbf Table to receive update totals]  ==
'=====================================================
```

```
    a = SuperQueryMaster("MtbfMake1", way)
        'Create table TMP_Mtbf with previous month data from  MTBF_5MTH.

'3== [Get run hour totals for this month into TMP_Mtbf] ==
'=============================================================
    a = SuperQueryMaster("MtbfInsert1", way)
        'Get 1-month run hrs for last 5-months of installed base into TMP_Mtbf.
    a = SuperQueryMaster("MtbfMake2", way)
        'Add run hours from previous MTBF and send total to TMP_t1 Table.
    a = SuperQueryMaster("MtbfUpdate1", way)
        'Update TMP_Mtbf run hours from TMP_t1 values for this month.
    a = DeleteTable(temp1, way)      'Select and delete the TMP_t1 Table.
    upform("message") = "Got run hours": DoCmd RepaintObject

'4== [Get reliability returns for this month into TMP_Mtbf] ==
'=============================================================
==
    a = SuperQueryMaster("MtbfMake3", way)
        'Get 5-month returns from SHIPS_RETS_DATA into TMP_t2 Table.
    a = SuperQueryMaster("MtbfUpdate2", way)
        'Update TMP_Mtbf returns from TMP_t2 values for this month.
    a = DeleteTable(temp2, way)
        'Select and delete the TMP_t2 Table.
    upform("message") = "Got returns": DoCmd RepaintObject

'5== [Calculate MTBF values in the TMP_Mtbf Table]  ==
'=====================================================
    a = SuperQueryMaster("MtbfUpdate3", way)
        'Set MTBF = (run_hours)/(returns) where returns are greater than zero.
    a = SuperQueryMaster("MtbfUpdate4", way)
        'Set MTBF = 1,000,000 where returns=0 and run_hours>0.
    upform("message") = "Calculated MTBF": DoCmd RepaintObject

'6== [Get installed population values into TMP_Mtbf Table]  ==
'=============================================================
==
    a = SuperQueryMaster("MtbfMake4", way)
        'Get installed population from SHIPS_RETS_DATA into TMP_t3 Table.
    a = SuperQueryMaster("MtbfUpdate5", way)
        'Update TMP_Mtbf installed population from TMP_t3 values.
    a = DeleteTable(temp3, way)
        'Select and delete the TMP_t3 Table.

'7== [Get total shipment values into TMP_Mtbf Table]  ==
'=====================================================
    a = SuperQueryMaster("MtbfMake5", way)
```

```
        'Get total shipments from SHIPS_RETS_DATA into TMP_t4 Table.
    a = SuperQueryMaster("MtbfUpdate6", way)
        'Update TMP_Mtbf total shipments form TMP_t4 values.
    a = DeleteTable(temp4, way)       'Select and delete the TMP_t4 Table.
    DoCmd SetWarnings True        'Turn on Warnings.

'8== [Put the TMP_Mtbf info into MTBF_5MTH and delete TMP_Mtbf] ==
'=============================================================
========
    upform("message") = "Updating MTBF_5MTH": DoCmd RepaintObject
    a = SuperQueryMaster("MtbfUpdate7", way)
        'Update MTBF_5MTH with values from TMP_Mtbf.
    a = SuperQueryMaster("MtbfDelete1", way)
       'Delete records from TMP_Mtbf that already were updated in
MTBF_5MTH.
    a = SuperQueryMaster("MtbfInsert2", way)
       'Instert records from TMP_Mtbf that were not already present in
MTBF_5MTH.
    a = DeleteTable(temp5, way)       'Select and delete the TMP_Mtbf Table.
    upform("message") = "Update completed": DoCmd RepaintObject

End Function
```

Query Name: MtbfDelete1

SQL Code:

<CNTL>ENTER=Newline

```
DELETE  DISTINCTROW TMP_Mtbf.*
FROM TMP_Mtbf, MTBF_5MTH,
 TMP_Mtbf INNER JOIN MTBF_5MTH ON
 TMP_Mtbf.PART_NO = MTBF_5MTH.PART_NO,
 TMP_Mtbf INNER JOIN MTBF_5MTH ON
 TMP_Mtbf.CUST_ID = MTBF_5MTH.CUST_ID,
 TMP_Mtbf INNER JOIN MTBF_5MTH ON
 TMP_Mtbf.SHIP_MTH = MTBF_5MTH.SHIP_MTH
WITH OWNERACCESS OPTION;
```

Query Description:

Deletes records in TMP_Mtbf that are already updated in MTBF_5MTH.

Query Name: MtbfInsert1

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO TMP_Mtbf
 ( PART_NO, CUST_ID, RUN_HRS, SHIP_MTH )
SELECT  DISTINCTROW
 SHIPS_RETS_DATA.PART_NO,
 SHIPS_RETS_DATA.CUST_ID,
 730*Sum([SHIPPED]) AS run_hrs,
 [Forms]![Update_4]![month1] AS month
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.SHIP_MTH Between
 [Forms]![Update_4]![month4] And
 [Forms]![Update_4]![month3]))
GROUP BY SHIPS_RETS_DATA.PART_NO,
 SHIPS_RETS_DATA.CUST_ID
WITH OWNERACCESS OPTION;
```

Query Description:

Gets total drives for most recent 5 months of installed population from SHIPS_RETS_DATA. It then multiplies this quantity by 730 run_hours/month and inserts this information into new TMP_Mtbf records for the update month.

Query Name: MtbfInsert2

Query Description:

Insert records in TMP_Mtbf for update month that don't already exist in MTBF_5MTH.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO MTBF_5MTH
 ( PART_NO, CUST_ID, SHIP_MTH, RUN_HRS,
 RETURNS, MTBF, INST_POP, TOT_SHIPS )
SELECT  DISTINCTROW TMP_Mtbf.PART_NO,
 TMP_Mtbf.CUST_ID, TMP_Mtbf.SHIP_MTH,
 TMP_Mtbf.RUN_HRS, TMP_Mtbf.RETURNS,
 TMP_Mtbf.MTBF, TMP_Mtbf.INST_POP,
 TMP_Mtbf.TOT_SHIPS
FROM TMP_Mtbf
WHERE ((TMP_Mtbf.SHIP_MTH =
 [Forms]![Update_4]![month1]))
WITH OWNERACCESS OPTION;
```

Query Name: MtbfMake1

Query Description:

Creates the temporary table TMP_Mtbf with previous month MTBF information. Note: previous month run hours are used as part of the update algorithm.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT  DISTINCTROW MTBF_5MTH.PART_NO,
 MTBF_5MTH.CUST_ID, MTBF_5MTH.SHIP_MTH,
 MTBF_5MTH.RUN_HRS, MTBF_5MTH.RETURNS,
 MTBF_5MTH.MTBF, MTBF_5MTH.INST_POP,
 MTBF_5MTH.TOT_SHIPS INTO TMP_Mtbf
FROM MTBF_5MTH
WHERE ((MTBF_5MTH.SHIP_MTH =
 [Forms]![Update_4]![month5]))
WITH OWNERACCESS OPTION;
```

Query Name: MtbfMake2

Query Description:

Creates a temporary table with [update_month_runhrs] = [previous_month_runhrs] + [one_month_runhrs_for_last_5_months]. This is the desired run hours total per the 5 month window algorithm.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT DISTINCTROW TMP_Mtbf.PART_NO,
 TMP_Mtbf.CUST_ID, Sum(TMP_Mtbf.RUN_HRS)
 AS SumOfRUN_HRS, [Forms]![Update_4]![month1]
 AS month INTO TMP_t1
FROM TMP_Mtbf
WHERE ((TMP_Mtbf.SHIP_MTH Between
 [Forms]![Update_4]![month5] And
 [Forms]![Update_4]![month1]))
GROUP BY TMP_Mtbf.PART_NO,
 TMP_Mtbf.CUST_ID
WITH OWNERACCESS OPTION;
```

Query Name: MtbfMake3

Query Description:

Creates a temporary table with the reliability returns from the most recent 5 months by part number.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT DISTINCTROW
 SHIPS_RETS_DATA.PART_NO,
 SHIPS_RETS_DATA.CUST_ID,
Sum(SHIPS_RETS_DATA.REL_2MTH_RETURNS)
 AS SumOfREL_2MTH_RETURNS,
 [Forms]![Update_4]![month1] AS month
INTO TMP_t2
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.SHIP_MTH Between
 [Forms]![Update_4]![month2] And
 [Forms]![Update_4]![month1]))
GROUP BY SHIPS_RETS_DATA.PART_NO,
 SHIPS_RETS_DATA.CUST_ID
WITH OWNERACCESS OPTION;
```

Query Name: MtbfMake4

Query Description:

Creates a temporary table with [installed_population] = Sum([shipped] - [returns]) up to 2-months ago.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT DISTINCTROW
  SHIPS_RETS_DATA.PART_NO,
  SHIPS_RETS_DATA.CUST_ID,
  Sum([SHIPPED]-[RETURNED]) AS inst_pop,
  [Forms]![Update_4]![month1] AS month
INTO TMP_t3
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.SHIP_MTH <=
  [Forms]![Update_4]![month3]))
GROUP BY SHIPS_RETS_DATA.PART_NO,
  SHIPS_RETS_DATA.CUST_ID
WITH OWNERACCESS OPTION;
```

Query Name: MtbfMake5

Query Description:

Creates a temporary table with [total shipped] = Sum([shipped]) up to the update month.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT DISTINCTROW
  SHIPS_RETS_DATA.PART_NO,
  SHIPS_RETS_DATA.CUST_ID,
  Sum(SHIPS_RETS_DATA.SHIPPED) AS tot_ships,
  [Forms]![Update_4]![month1] AS month
INTO TMP_t4
FROM SHIPS_RETS_DATA
WHERE ((SHIPS_RETS_DATA.SHIP_MTH <=
  [Forms]![Update_4]![month1]))
GROUP BY SHIPS_RETS_DATA.PART_NO,
  SHIPS_RETS_DATA.CUST_ID
WITH OWNERACCESS OPTION;
```

**Query Name:** MtbfUpdate1

**Query Description:**

Updates the update month run hours for all part numbers in TMP_Mtbf, with the correct values contained in TMP_t1.

**SQL Code:**

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf, TMP_t1,
 TMP_Mtbf INNER JOIN TMP_t1 ON
 TMP_Mtbf.PART_NO = TMP_t1.PART_NO,
 TMP_Mtbf INNER JOIN TMP_t1 ON
 TMP_Mtbf.CUST_ID = TMP_t1.CUST_ID,
 TMP_Mtbf INNER JOIN TMP_t1 ON
 TMP_Mtbf.SHIP_MTH = TMP_t1.month
SET TMP_Mtbf.RUN_HRS = [SumOfRUN_HRS]
WITH OWNERACCESS OPTION;
```

**Query Name:** MtbfUpdate2

**Query Description:**

Update the reliability returns for the update month in TMP_Mtbf, from the values in TMP_t2.

**SQL Code:**

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf, TMP_t2,
 TMP_Mtbf INNER JOIN TMP_t2 ON
 TMP_Mtbf.PART_NO = TMP_t2.PART_NO,
 TMP_Mtbf INNER JOIN TMP_t2 ON
 TMP_Mtbf.CUST_ID = TMP_t2.CUST_ID,
 TMP_Mtbf INNER JOIN TMP_t2 ON
 TMP_Mtbf.SHIP_MTH = TMP_t2.month
SET TMP_Mtbf.RETURNS =
 [SumOfREL_2MTH_RETURNS]
WITH OWNERACCESS OPTION;
```

Query Name: MtbfUpdate3

Query Description:

Update MTBF in TMP_Mtbf to be
(run_hrs)/(returns)
for the current update month, where (returns)>0.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf
SET TMP_Mtbf.MTBF = [RUN_HRS]/[RETURNS]
WHERE ((TMP_Mtbf.SHIP_MTH =
 [Forms]![Update_4]![month1]) AND
 (TMP_Mtbf.RETURNS>0))
WITH OWNERACCESS OPTION;
```

Query Name: MtbfUpdate4

Query Description:

Update MTBF in TMP_Mtbf to be 1,000,000 where
(returns)=0 and (run_hrs)>0.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf
SET TMP_Mtbf.MTBF = 1000000
WHERE ((TMP_Mtbf.SHIP_MTH =
 [Forms]![Update_4]![month1]) AND
 (TMP_Mtbf.RETURNS=0) AND
 (TMP_Mtbf.RUN_HRS>0))
WITH OWNERACCESS OPTION;
```

Query Name: MtbfUpdate5

Query Description:

Update the installed population for the update month in TMP_Mtbf from the values in TMP_t3.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf, TMP_t3,
 TMP_Mtbf INNER JOIN TMP_t3 ON
 TMP_Mtbf.PART_NO = TMP_t3.PART_NO,
 TMP_Mtbf INNER JOIN TMP_t3 ON
 TMP_Mtbf.CUST_ID = TMP_t3.CUST_ID,
 TMP_Mtbf INNER JOIN TMP_t3 ON
 TMP_Mtbf.SHIP_MTH = TMP_t3.month
SET TMP_Mtbf.INST_POP =
 [TMP_t3].[inst_pop]
WITH OWNERACCESS OPTION;
```

Query Name: MtbfUpdate6

Query Description:

Update the total shipped for the update month in TMP_Mtbf from the values in TMP_t4.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW TMP_Mtbf, TMP_t4,
 TMP_Mtbf INNER JOIN TMP_t4 ON
 TMP_Mtbf.PART_NO = TMP_t4.PART_NO,
 TMP_Mtbf INNER JOIN TMP_t4 ON
 TMP_Mtbf.CUST_ID = TMP_t4.CUST_ID,
 TMP_Mtbf INNER JOIN TMP_t4 ON
 TMP_Mtbf.SHIP_MTH = TMP_t4.month
SET TMP_Mtbf.TOT_SHIPS =
 [TMP_t4].[tot_ships]
WITH OWNERACCESS OPTION;
```

Query Name: MtbfUpdate7

Query Description:

If records for the update month exist in MTBF_5MTH, then update them to the values in TMP_Mtbf.

SQL Code:

<CNTL>ENTER=Newline

```
UPDATE DISTINCTROW MTBF_5MTH, TMP_Mtbf,
 MTBF_5MTH INNER JOIN TMP_Mtbf ON
 MTBF_5MTH.PART_NO = TMP_Mtbf.PART_NO,
 MTBF_5MTH INNER JOIN TMP_Mtbf ON
 MTBF_5MTH.CUST_ID = TMP_Mtbf.CUST_ID,
 MTBF_5MTH INNER JOIN TMP_Mtbf ON
 MTBF_5MTH.SHIP_MTH = TMP_Mtbf.SHIP_MTH
SET MTBF_5MTH.RUN_HRS =
 [TMP_Mtbf].[RUN_HRS],
 MTBF_5MTH.RETURNS =
 [TMP_Mtbf].[RETURNS], MTBF_5MTH.MTBF =
 [TMP_Mtbf].[MTBF], MTBF_5MTH.INST_POP =
 [TMP_Mtbf].[INST_POP],
 MTBF_5MTH.TOT_SHIPS =
 [TMP_Mtbf].[TOT_SHIPS]
WHERE ((TMP_Mtbf.SHIP_MTH =
 [Forms]![Update_4]![month1]))
WITH OWNERACCESS OPTION;
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'=============================================================
======
'       MODULE FOR UPDATING THE REPORT MODULE INFORMATION
'             FROM THE VARIOUS SOURCE TABLES
'
'    SOURCE TABLES:  SHIPMENTS          (SOURCE.MDB)
'             DELV_MTH          (SOURCE.MDB)
'             QUAL_MTH          (SOURCE.MDB)
'             MTBF_5MTH         (SOURCE.MDB)
'             PART_INF.TXT      (Text File)
'             CUST_ACC.TXT      (Text File)
'             Revision History  (CONFIG.MDB)
'
'   TEMPORARY TABLES:  TMP_Cust
'             TMP_Part
'             TMP_Revs
'
'    UPDATED TABLES:  Delivery_Transactions (NEWDB.MDB)
'             Delivery          "
'             Delv_Aggregates       "
'             Quality           "
'             Qual_Aggregates       "
'             Reliability       "
'             Relb_Aggregates       "
'             Tot_Pop_Quality       "
'             Tot_Pop_Reliability   "
'             Cust_Info_Table       "
'=============================================================
======

Function REPT (way)
Dim a, infile1 As String, infile2 As String
Dim temp1 As String, temp2 As String, temp3 As String, upform As Form

'-------------------------------------------------------
Set upform = [Forms]![Update_5]
temp1 = "TMP_Cust": temp2 = "TMP_Part": temp3 = "TMP_Revs"
infile1 = upform("import_file"): infile2 = upform("import_file2")
'-------------------------------------------------------

'1== [Set additional values for the update] ==
'=============================================
    upform("month2") = GetNewMonth(upform("month1"), -5)
      'Set value for a month five months before current.
```

```
    upform("month3") = GetNewMonth(upform("month1"), -2)
      'Set value of earliest month for aggregates.
    upform("firstday") = GetFirstDay(upform("month1"))
      'Get the first day for the desired update month.
    upform("lastday") = GetLastDay(upform("month1"))
      'Get the last day for the desired update month.
   DoCmd SetWarnings False     'Turn warning messages off.
   upform("message") = "Starting update": DoCmd RepaintObject

    a = GetImportDelim(infile1, temp1, "cs_file_importer_WF", way)
      'Copy CUSTOMERS to TMP_Cust.
    a = GetImportDelim(infile2, temp2, "cs_file_importer_WF", way)
      'Copy PART_INFO to TMP_Part.
   a = SuperQueryMaster("ReptMake1", way)
      'Create new TMP_Revs Table.

'2== [Update the Delivery_Transactions table]  ==
'================================================
   a = EmptyTable("Delivery_Transactions", way)'Delete all records from
Delivery_Transactions.
   a = SuperQueryMaster("ReptInsert1", way)
       'Update Delivery_Transactions from SHIPMENTS.

'3== [Update the Delivery and Delv_Aggregates Tables]  ==
'=======================================================
   a = EmptyTable("Delivery", way) 'Delete all records from Delivery.
   a = SuperQueryMaster("ReptInsert2", way)
      'Append 6-month updated info to Delivery from
DELV_MTH/TMP_Part.
   a = EmptyTable("Delv_Aggregates", way) 'Delete all records from
Delv_Aggregates
   a = SuperQueryMaster("ReptInsert3", way)
      'Append new aggregate info for the month from Delivery.
   upform("message") = "Delivery updated": DoCmd RepaintObject

'4== [Update the Quality and Qual_Aggregates Tables] ==
'======================================================
   a = EmptyTable("Quality", way)  'Delete all records from Quality.
   a = SuperQueryMaster("ReptInsert4", way)
      'Append 6-month updated info to Quality from QUAL_MTH/TMP_Part.
   a = EmptyTable("Qual_Aggregates", way) 'Delete all records from
Qual_Aggregates.
   a = SuperQueryMaster("ReptInsert5", way)
      'Append new aggregate info for the month to Qual_Aggregates from
Quality.
   upform("message") = "Quality updated": DoCmd RepaintObject
```

```
'5== [Update the Reliability and Relb_Aggregates Tables] ==
'=============================================================
   a = EmptyTable("Reliability", way) 'Delete all records from Reliability.
   a = SuperQueryMaster("ReptInsert6", way)
      'Append 6-month updated info to Reliability from
MTBF_5MTH/TMP_Part.
   a = EmptyTable("Relb_Aggregates", way) 'Delete all records from
Relb_Aggregates.
   a = SuperQueryMaster("ReptInsert7", way)
      'Append new aggregate info for the month to Relb_Aggregates from
Reliability.
   upform("message") = "Reliability updated": DoCmd RepaintObject

'6== [Update the Total_Pop_Quality and Total_Pop_Reliability Tables] ==
'=============================================================
==========
   a = EmptyTable("Total_Pop_Quality", way) 'Delete all records from
Total_Pop_Quality.
   a = SuperQueryMaster("ReptInsert8", way)
      'Add new population information from Quality.
   a = EmptyTable("Total_Pop_Reliability", way) 'Delete all records from
Total_Pop_Reliability.
   a = SuperQueryMaster("ReptInsert9", way)
      'Add new population information from Reliability.
   upform("message") = "Populations updated": DoCmd RepaintObject

'7== [Update the By_Type_Quality and By_Type_Reliability Tables] ==
'=============================================================
=====
   a = EmptyTable("By_Type_Quality", way) 'Delete all records from
By_Type_Quality.
   a = SuperQueryMaster("ReptInsertA", way)
      'Append distribuitor info to By_Type_Quality from Quality/TMP_Cust.
   a = SuperQueryMaster("ReptInsertB", way)
      'Append non-distribuitor info to By_Type_Quality from
Quality/TMP_Cust.
   a = EmptyTable("By_Type_Reliability", way) 'Delete all records from
By_Type_Reliability.
   a = SuperQueryMaster("ReptInsertC", way)
      'Append distribuitor info to By_Type_Reliability from
Reliability/TMP_Cust.
   a = SuperQueryMaster("ReptInsertD", way)
      'Append non-distribuitor info to By_Type_Reliability from
Reliability/TMP_Cust.
   upform("message") = "Customer types updated": DoCmd RepaintObject
```

```
'8== [Update the Cust_Info_Table]  ==
'=====================================
   a = EmptyTable("Cust_Info_Table", way) 'Delete all records from
Cust_Info_Table.
   a = SuperQueryMaster("ReptInsertE", way)
      'Add latest info to Cust_Info_Table from
TMP_Cust/TMP_Revs/TMP_Part.
   a = DeleteTable(temp1, way)  'Delete the TMP_Cust table.
   a = DeleteTable(temp2, way)  'Delete the TMP_Part table.
   a = DeleteTable(temp3, way)  'Delete the TMP_Revs table.
   upform("message") = "Update completed": DoCmd RepaintObject
   DoCmd SetWarnings True

End Function
```

Query Name: ReptInsert1

Query Description:

Inserts the shipping details information for the update month into the Delivery_Transactions table.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Delivery_Transactions (part_number,
 customer_id, month, po_no, line_number, order_qty,
 ship_qty, sched_date, ship_date, req_date,
 SO_number, var_qty, var_days, late_flag)
SELECT DISTINCTROW SHIPMENTS.PART_NO,
 SHIPMENTS.CUST_ID, [Forms]![Update_5]![month1]
 AS month, SHIPMENTS.PO_NO,
 SHIPMENTS.SO_LINE,
 SHIPMENTS.ORDER_QTY,
 SHIPMENTS.SHIP_QTY,
 SHIPMENTS.SCHED_DATE,
 SHIPMENTS.SHIP_DATE,
 SHIPMENTS.REQ_DATE,
 SHIPMENTS.SO_NUMBER,
 SHIPMENTS.VAR_QTY, SHIPMENTS.VAR_DAYS,
 SHIPMENTS.LATE_FLAG
FROM SHIPMENTS
WHERE ((SHIPMENTS.SHIP_DATE Between
 [Forms]![Update_5]![firstday] And
 [Forms]![Update_5]![lastday]))
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert2

Query Description:

Inserts the delivery information by part number for the six months up to the update month into the Delivery table.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Delivery
 ( part_number, customer_id, month, total_orders,
 late_orders, On_time_percentage )
SELECT DISTINCTROW
 DELV_MTH.part_number,
 DELV_MTH.customer_id,
 DELV_MTH.month,
 DELV_MTH.total_orders,
 DELV_MTH.late_orders,
 DELV_MTH.On_time_percentage
FROM DELV_MTH, TMP_Part,
 DELV_MTH LEFT JOIN TMP_Part ON
 DELV_MTH.part_number = TMP_Part.PART_NO
WHERE ((DELV_MTH.month Between
 [Forms]![Update_5]![month1] And
 [Forms]![Update_5]![month2]))
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert3

Query Description:

Gets aggregate delivery information for part numbers and inserts it into Delv_Aggregates.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Delv_Aggregates
  (part_number, customer_id, month, capacity,
  total_orders, late_orders, delivery_aggregate)
SELECT  DISTINCTROW
  Delivery.part_number, Delivery.customer_id,
  [Forms]![Update_5]![month1] AS month,
  TMP_Part.CAPACITY,
  Sum(Delivery.total_orders) AS SumOftotal_orders,
  Sum(Delivery.late_orders) AS SumOflate_orders,
  (1-(Sum([late_orders])/Sum([total_orders]))) AS
  delivery_aggregate
FROM Delivery, TMP_Part,
  Delivery LEFT JOIN TMP_Part ON
  Delivery.part_number = TMP_Part.PART_NO
WHERE ((Delivery.month Between
  [Forms]![Update_5]![month1] And
  [Forms]![Update_5]![month3]))
GROUP BY Delivery.part_number,
  Delivery.customer_id, TMP_Part.CAPACITY
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert4

Query Description:

Inserts the quality information by part number into the Quality table for the six months up to the update month.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Quality ( part_number, customer_id,
  capacity, month, shipped, qual_returns, quality,
  specification )
SELECT  DISTINCTROW QUAL_MTH.part_number,
  QUAL_MTH.customer_id, TMP_Part.CAPACITY,
  QUAL_MTH.month, QUAL_MTH.shipped,
  QUAL_MTH.qual_returns, QUAL_MTH.quality,
  TMP_Part.QUAL_SPEC
FROM QUAL_MTH, TMP_Part,
  QUAL_MTH LEFT JOIN TMP_Part ON
  QUAL_MTH.part_number = TMP_Part.PART_NO
WHERE ((QUAL_MTH.customer_id Not Like "-Z*")
  AND (QUAL_MTH.month Between
  [Forms]![Update_5]![month2] And
  [Forms]![Update_5]![month1]) AND
  (([shipped]+[qual_returns])>0))
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert5

Query Description:

Inserts aggregate information from the Quality table into Qual_Aggregates.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Qual_Aggregates ( part_number,
 customer_id, month, capacity, total_shipped,
 quality_returns, quality_aggregate )
SELECT  DISTINCTROW Quality.part_number,
 Quality.customer_id, [Forms]![Update_5]![month1] AS
 month, Quality.capacity, Sum(Quality.shipped) AS
 SumOfshipped, Sum(Quality.qual_returns) AS
 SumOfqual_returns,
 (1-(Sum([qual_returns])/Sum([shipped]))) AS
 quality_aggregate
FROM Quality
WHERE ((Quality.month Between
 [Forms]![Update_5]![month1] And
 [Forms]![Update_5]![month3]))
GROUP BY Quality.part_number, Quality.customer_id,
 Quality.capacity
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert6

Query Description:

Inserts the reliability information by part number into the Reliability table for the six months up to the update month.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Reliability
 ( part_number, customer_id, month, run_hrs,
 rel_returns, mtbf, inst_pop, tot_ships, capacity,
 specification )
SELECT  DISTINCTROW MTBF_5MTH.PART_NO,
 MTBF_5MTH.CUST_ID, MTBF_5MTH.SHIP_MTH,
 MTBF_5MTH.RUN_HRS, MTBF_5MTH.RETURNS,
 MTBF_5MTH.MTBF, MTBF_5MTH.INST_POP,
 MTBF_5MTH.TOT_SHIPS, TMP_Part.CAPACITY,
 TMP_Part.MTBF_SPEC
FROM MTBF_5MTH, TMP_Part,
 MTBF_5MTH LEFT JOIN TMP_Part ON
 MTBF_5MTH.PART_NO = TMP_Part.PART_NO
WHERE ((MTBF_5MTH.CUST_ID Not Like "-Z*")
 AND (MTBF_5MTH.SHIP_MTH Between
 [Forms]![Update_5]![month2] And
 [Forms]![Update_5]![month1]))
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert7

Query Description:

Inserts the aggregate reliability information from the Reliability table into Relb_Aggregates.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Relb_Aggregates
 ( part_number, customer_id, month, capacity,
 total_run_hrs, reliability_returns, reliability_aggregate,
 tot_ships )
SELECT DISTINCTROW Reliability.part_number,
 Reliability.customer_id,
 [Forms]![Update_5]![month1] AS month,
 Reliability.capacity, Reliability.run_hrs,
 Reliability.rel_returns, Reliability.inst_pop,
 Reliability.tot_ships
FROM Reliability
WHERE ((Reliability.month =
 [Forms]![Update_5]![month1]))
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert8

Query Description:

Inserts 6 month quality information from Quality into Total_Pop_Quality. Information is aggregated by product capacity.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Total_Pop_Quality ( capacity,
 ship_mth, shipped, returned, quality, max_qual,
 min_qual, specification )
SELECT DISTINCTROW Quality.capacity,
 Quality.month, Sum(Quality.shipped) AS
 SumOfshipped, Sum(Quality.qual_returns) AS
 SumOfqual_returns,
 1-Sum([qual_returns])/Sum([shipped]) AS quality,
 Max(Quality.quality) AS MaxOfquality,
 Min(Quality.quality) AS MinOfquality,
 Avg(Quality.specification) AS AvgOfSpecification
FROM Quality
GROUP BY Quality.capacity, Quality.month
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsert9

Query Description:

Inserts 6 month reliability information from Reliability into Total_Pop_Reliability. Information is aggregated by product capacity.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO Total_Pop_Reliability
 ( capacity, ship_mth, run_hrs, returns, reliability,
 max_rel, min_rel, specification )
SELECT  DISTINCTROW Reliability.capacity,
 Reliability.month, Sum(Reliability.run_hrs) AS
 SumOfrun_hrs, Sum(Reliability.rel_returns) AS
 SumOfrel_returns, Sum([run_hrs])/Sum([rel_returns])
 AS mtbf, Max(Reliability.mtbf) AS MaxOfmtbf,
 Min(Reliability.mtbf) AS MinOfmtbf,
 Avg(Reliability.specification) AS AvgOfspecification
FROM Reliability
GROUP BY Reliability.capacity, Reliability.month
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsertA

Query Description:

Inserts quality information aggregated for distributors into the By_Type_Quality table.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO By_Type_Quality
 (capacity, ship_mth, shipped, returned, quality,
 max_qual, min_qual, cust_type )
SELECT  DISTINCTROW Quality.capacity,
 Quality.month, Sum(Quality.shipped) AS
 SumOfshipped, Sum(Quality.qual_returns) AS
 SumOfqual_returns,
 1-Sum([qual_returns])/Sum([shipped]) AS quality,
 Max(Quality.quality) AS MaxOfquality,
 Min(Quality.quality) AS MinOfquality, "Distributors"
 AS Expr1
FROM Quality, TMP_Cust,
 Quality INNER JOIN TMP_Cust ON
 Quality.customer_id = TMP_Cust.CUST_ID
WHERE ((TMP_Cust.FLAG Like "DIST*"))
GROUP BY Quality.capacity, Quality.month
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsertB

Query Description:

Inserts quality information aggregated for non-distributors into the By_Type_Quality table.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO By_Type_Quality
 ( capacity, ship_mth, shipped, returned, quality,
 max_qual, min_qual, cust_type )
SELECT  DISTINCTROW Quality.capacity,
 Quality.month, Sum(Quality.shipped) AS
 SumOfshipped, Sum(Quality.qual_returns) AS
 SumOfqual_returns,
 1-Sum([qual_returns])/Sum([shipped]) AS quality,
 Max(Quality.quality) AS MaxOfquality,
 Min(Quality.quality) AS MinOfquality, "Others"
 AS Expr1
FROM Quality, TMP_Cust,
 Quality LEFT JOIN TMP_Cust ON
 Quality.customer_id = TMP_Cust.CUST_ID
WHERE ((TMP_Cust.FLAG Not Like "DIST*"))
GROUP BY Quality.capacity, Quality.month
WITH OWNERACCESS OPTION;
```

Query Name: ReptInsertC

Query Description:

Inserts reliability information aggregated for distributors into the By_Type_Reliability table.

SQL Code:

<CNTL>ENTER=Newline

```
INSERT INTO By_Type_Reliability
 ( capacity, ship_mth, run_hrs, returns, reliability,
 max_rel, min_rel, cust_type )
SELECT  DISTINCTROW Reliability.capacity,
 Reliability.month, Sum(Reliability.run_hrs) AS
 SumOfrun_hrs, Sum(Reliability.rel_returns) AS
 SumOfrel_returns, Sum([run_hrs])/Sum([rel_returns])
 AS mtbf, Max(Reliability.mtbf) AS MaxOfmtbf,
 Min(Reliability.mtbf) AS MinOfmtbf, "Distributors" AS
 Expr1
FROM Reliability, TMP_Cust,
 Reliability INNER JOIN TMP_Cust  ON
 Reliability.customer_id = TMP_Cust.CUST_ID
WHERE ((TMP_Cust.FLAG Like "DIST*"))
GROUP BY Reliability.capacity, Reliability.month
WITH OWNERACCESS OPTION;
```

**Query Name:** ReptInsertD

**Query Description:**

Inserts reliability information aggregated for non-distributors into the By_Type_Reliability table.

**SQL Code:**

<CNTL>ENTER=Newline

```
INSERT INTO By_Type_Reliability
 ( capacity, ship_mth, run_hrs, returns, reliability,
 max_rel, min_rel, cust_type )
SELECT DISTINCTROW Reliability.capacity,
 Reliability.month, Sum(Reliability.run_hrs) AS
 SumOfrun_hrs, Sum(Reliability.rel_returns) AS
 SumOfrel_returns, Sum([run_hrs])/Sum([rel_returns])
 AS mtbf, Max(Reliability.mtbf) AS MaxOfmtbf,
 Min(Reliability.mtbf) AS MinOfmtbf, "Others" AS Expr1
FROM Reliability, TMP_Cust,
 Reliability LEFT JOIN TMP_Cust ON
 Reliability.customer_id = TMP_Cust.CUST_ID
WHERE ((TMP_Cust.FLAG Not Like "DIST""))
GROUP BY Reliability.capacity, Reliability.month
WITH OWNERACCESS OPTION;
```

**Query Name:** ReptInsertE

**Query Description:**

Inserts customer information into the Cust_Info_Table.

**SQL Code:**

<CNTL>ENTER=Newline

```
INSERT INTO Cust_Info_Table ( Customer_Name,
 part_number, customer_id, customer_part_no,
 sales_part_no, customer_AE, product_rev, capacity,
 gigabytes )
SELECT DISTINCTROW TMP_Cust.CUST_NAME,
 TMP_Revs.part_no, TMP_Revs.customer_id,
 TMP_Revs.cust_part, TMP_Part.SALES_PART,
 TMP_Cust.CUST_AE, TMP_Revs.opt_rev,
 TMP_Part.CAPACITY, TMP_Part.GIGABYTES
FROM TMP_Revs, TMP_Cust, TMP_Part,
 TMP_Cust RIGHT JOIN TMP_Revs ON
 TMP_Cust.CUST_ID = TMP_Revs.customer_id,
 TMP_Revs LEFT JOIN TMP_Part ON
 TMP_Revs.part_no = TMP_Part.PART_NO
WHERE ((TMP_Cust.CUST_NAME Is Not Null) AND
 (TMP_Revs.part_no Is Not Null) AND
 (TMP_Revs.customer_id Is Not Null) AND
 (TMP_Revs.opt_rev Is Not Null))
WITH OWNERACCESS OPTION;
```

Query Name: ReptMake1

Query Description:

Insert the latest revision information from Revision History into TMP_Revs.

SQL Code:

<CNTL>ENTER=Newline

```
SELECT  DISTINCTROW GetMain([Revision
History].[INTERNAL PART NUMBER]) AS part_no,
GetSufx([Revision History].[INTERNAL PART
NUMBER]) AS customer_id, First([Revision
History].[OPTION REVISION]) AS opt_rev,
Max([Revision History].[CCR ACCEPTANCE DATE])
AS acc_date, First([Revision History].[CUSTOMER
PART NUMBER]) AS cust_part
INTO TMP_Revs
FROM [Revision History]
WHERE (([Revision History].[EVALUATION UNIT?]
Like "N*") AND ([Revision History].[OEM MODEL
NUMBER] Not Like "DSP5*" And [Revision
History].[OEM MODEL NUMBER] Not Like "RZ7*"))
GROUP BY GetMain([Revision History].[INTERNAL
PART NUMBER]), GetSufx([Revision
History].[INTERNAL PART NUMBER])
HAVING (((GetMain([Revision History].[INTERNAL
PART NUMBER]))<>"NONE"))
WITH OWNERACCESS OPTION;
```

# Appendix C

## Summary of File Locations

PC Hard Drive (Drive C:)

Installed applications:
Microsoft Access 1.1 Installed
Microsoft ODBC Administrator Installed
DEC Rdb SQL Server Installed

Microsoft Access files:
MAINT.MDB Maintenance tool ACCESS file
NEWDB.MDB Reporting tool ACCESS file
RAWDB.MDB Local PC database file (ACCESS)

PC Network Drive (Drive D:)

CONFIG.MDB Configuration database file (copy)
CUST_DAT.TXT Customer information text file (copy)
PART_DAT.TXT Part number information text file (copy)
CONFIG.COM Command file
LOAD_OUT.DAT Monthly shipments text file (edited copy)
SHIPS.TXT Individual shipments for the month (edited copy)

MAXCIM Network Node:

SUB_FIS.COM File that generates Maxcim output files
MTBF.FCF Specification file for LOAD_OUT.DAT
OEM_LOAD.FCF Specifiaction file for SHIPS.TXT

Other Network Node:

CONFIG.MDB original in public directory.
RATS.RDB Warranty returns database

# Appendix D

## ACCESS BASIC Code

Maintenance Application
FUNCTIONS THAT EXIT TO THE DOS ENVIRONMENT
PROCEDURES CALLED BY THE VARIOUS FORMS
LIBRARY OF QUERY AND DATA TRANSFER FUNCTIONS
FUNCTIONS THAT PLAY SOUNDS
MISCELLANEOUS UTILITY FUNCTIONS

Reporting Application
GENERAL MENU FUNCTIONS
MISCELLANEOUS UTILITY FUNCTIONS
FUNCTIONS FOR CREATING A MAIN REPORT AND A SUB REPORT

```
    Option Compare Database   'Use database order for string comparisons
Option Explicit
'-----------------------------------------------------
'
'         FUNCTIONS THAT EXIT TO THE DOS ENVIRONMENT
'
'-----------------------------------------------------

'===== EXECUTE AN EXTERNAL BATCH USING DECNET COMMAND
=====
'============================================================
Function ExecuteBatch (dclfile As String)
Dim vTargetLoc As String, MyBatFile As Integer
Dim pos As Integer, username As String, password As String
Dim vAction1 As String, nodename As String, pathlength As Integer
ReDim filename(9) As String

'---- Send message for incorrect filename ----
If IsNull(dclfile) Or dclfile = "" Or Left(dclfile, 1) = "?" Then
   MsgBox "You need to specify a batch file to be executed.", 64, "ExecuteBatch"
   Exit Function
End If

'---- Get nodename and insert location for username/password ----
On Error GoTo clean_up
pos = InStr(dclfile, "::")
If pos = 0 Then Exit Function
pos = pos - 1
nodename = Left(dclfile, pos)

'---- Get the username and password information ----
username = InputBox("Username for " & nodename & ":", "Node where
batch executes")
If username = "" Then Exit Function
password = InputBox("Password for " & nodename & ":", "Node where batch
executes")
If password = "" Then Exit Function

'---- Insert username and password into the directory path ----
pathlength = Len(dclfile)
vTargetLoc = nodename & """" & username & " " & password
vTargetLoc = vTargetLoc & """" & Right(dclfile, pathlength - pos)
vAction1 = "C:\DECNET\NFT SUBMIT " & vTargetLoc & " >
C:\ACCESS\TESTDATA.LOG"

'---- Create and execute the batch file ----
```

```
MyBatFile = 1
Open "C:\ACCESS\TESTDATA.BAT" For Output As MyBatFile
Print #MyBatFile, "ECHO OFF"
Print #MyBatFile, vAction1
Print #MyBatFile, "ECHO Submitting batch process."
Print #MyBatFile, "EXIT"
Close MyBatFile
ExecuteBatch = Shell("C:\ACCESS\TESTDATA.BAT", 4)
MsgBox "Batch job submitted.", 64

clean_up:
'---- Erase the information on the batch file ----
Open "C:\ACCESS\TESTDATA.BAT" For Output As MyBatFile
Print #MyBatFile, " "
Close MyBatFile
Exit Function

End Function

'===== READ A LOG FILE ON A LOCAL/ATTACHED DRIVE =====
'=====================================================
Function MonitorLog (logfilename As String)
Dim recordname As String, completestr As String

'---- Get filename or exit if filename is not correct ----
On Error GoTo ErrorCheck
If logfilename = "?" Or logfilename = "" Then GoTo NeedFileName
Open logfilename For Input Access Read Shared As #1

'---- Get the contents of the log file ----
Do While Not EOF(1)
   Input #1, recordname
   completestr = completestr & recordname & Chr(13) & Chr(10)
Loop
Close #1
MonitorLog = completestr
Exit Function

NeedFileName:
MsgBox "You need a file name to get log information.", 64, "MSG_FORM
Message"
MonitorLog = "No file name given."
Exit Function

ErrorCheck:
MsgBox "Error Getting Log Information.", 52, "MSG_FORM Error"
```

```
MonitorLog = "Error looking for data."
Exit Function
End Function

'===== UNUSED SAMPLE FUNCTION =====
'==================================
Function Send ()
Dim MyTable As Table
Dim MyDB As Database
Dim vFileName As String, vMsgText As String
Dim vShell As String, vTargetLoc As String, MyBatFile As Integer
Dim vAction1 As String, vAction2 As String
Set MyDB = CurrentDB()
vFileName = "TESTDATA.DAT"
DoCmd TransferText A_EXPORTDELIM, "export_format", "CUSTOMERS",
vFileName, True
vTargetLoc = " COMET::$21$dua132:[siaca]*.*"
vAction1 = "C:\DECNET\NFT COPY " & vFileName & vTargetLoc & " >
TESTDATA.LOG"
vAction2 = "C:\DECNET\NFT SUBMIT
COMET::$21$dua132:[siaca]testmail.com"
MyBatFile = FreeFile
Open "TESTDATA.BAT" For Output As MyBatFile
Print #MyBatFile, vAction1
Print #MyBatFile, vAction2
Close MyBatFile

Send = Shell("TESTDATA.BAT", 1)

vMsgText = "Please check the TESTDATA.LOG file to confirm correct
processing"
MsgBox vMsgText, 48, "Warning!"

End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'-----------------------------------------------
'
'       PROCEDURES CALLED BY THE VARIOUS FORMS
'
'-----------------------------------------------

'===== UNUSED FUNCTION, UNDER DEVELOPMENT =====
'==============================================
Function AddParameter ()
Dim upform As Form
Dim a, textlabel As String, labelcontrol As Control, param As Control
Dim inputname As String
Set upform = Screen.ActiveForm

For a = 1 To 5
textlabel = "Text(" & a & ")"
If upform(textlabel).Visible = False Then
   Set labelcontrol = upform(textlabel)
    inputname = InputBox("Enter parameter name:", "New Parameter")
   If inputname = "" Then Exit Function
   inputname = Left(inputname, 10)
  DoCmd Echo False
   DoCmd DoMenuItem 3, 2, 0 'FormsDesignMenuBar, View, Design View
   labelcontrol.Visible = True
    upform(labelcontrol.caption).ControlName = inputname
    upform(inputname).Visible = True
   labelcontrol.caption = inputname
    DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
    DoCmd DoMenuItem 3, 2, 1 'FormsDesignMenuBar, View, Form
  DoCmd Echo True
   Exit Function
End If
Next
End Function

'===== GET INFORMATION FOR LOCAL AND ATTACHED TABLES =====
'=========================================================
Function GetAllTables ()
Dim db As Database, sn As Snapshot, a, attch As Integer
Dim tname As String, ttype As Long, ErrMsg As String

'---- Get a list of local and attached tables ----
On Error GoTo ErrorCheck
Set db = CurrentDB()
```

```
Set sn = db.ListTables()
a = EmptyTable("SYS_TableDetails", 0)

'---- Get the table fields ----
Do While Not sn.EOF
  attch = True
  ttype = sn.TableType
  tname = sn.Name
  If ttype = DB_TABLE Then attch = False
  If ttype = DB_QUERYDEF Then GoTo Do_a_loop
  If tname Like "MSys*" Then GoTo Do_a_loop 'Don't get system tables
  If tname Like "SYS_*" Then GoTo Do_a_loop 'Don't get program tables
  If tname Like "TMP_*" Then GoTo Do_a_loop 'Don't get temporary tables

  '---- Call the database analyzer to get table fields ----
   Call DumpTableInfo("MAINT.MDB", "SYS_TableDetails", tname, attch)
Do_a_loop:
sn.MoveNext
Loop
sn.Close
Exit Function

'---- On Error: Inform that table info is not available ----
ErrorCheck:
ErrMsg = "Could not get information for the " & tname & " table."
MsgBox ErrMsg, 64
Resume Next
End Function

'===== HIDE THE LAST UPDATE =====
'================================
Function HideUpdate ()
Dim a, vsb, Text, upd, mac, caption, msg, choice

'---- Search from last to first update position ----
On Error Resume Next
For a = 8 To 1 Step -1
  Text = "Text(" & a & ")"
   vsb = Forms!General_Menu(Text).Visible

  '---- Look for the last update in General Menu ----
  If vsb = True Then
    msg = "Are you sure you want to hide Update_" & a & "?"
    choice = MsgBox(msg, 36, "Hide Last Update")
    If choice = 7 Then GoTo HideUpdate_end
    upd = "Upd(" & a & ")"
```

```
    mac = "Mac(" & a & ")"

    '----- Hide Controls in General Menu -----
    DoCmd Echo False
     DoCmd OpenForm "General_Menu", A_DESIGN
     Forms!General_Menu(Text).caption = "Update_" & a & ": Not used."
     Forms!General_Menu(Text).Visible = False
     Forms!General_Menu(upd).Visible = False
     Forms!General_Menu(mac).Visible = False
     DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
     DoCmd OpenForm "General_Menu", A_NORMAL
    DoCmd Echo True
    GoTo HideUpdate_end
  End If
Next
HideUpdate_end:
End Function

'===== UPDATE THE TABLE INFORMATION ON THE SYSTEM TABLES
=====
'============================================================
Function MANT ()
Dim a, SQL As String

'----- Get updated table info? -----
On Error Resume Next
a = MsgBox("Update Table Info?", 36, "View Attached Tables")
If a = 7 Then Exit Function
a = GetAllTables() 'Put revised table info into SYS_TableDetails
DoCmd SetWarnings False

'----- Get the list of table names into a temporary table -----
SQL = "SELECT DISTINCTROW SYS_TableDetails.TableName INTO
TMP_maint  "
SQL = SQL & "FP.OM SYS_TableDetails GROUP BY
SYS_TableDetails.TableName  "
SQL = SQL & "WITH OWNERACCESS OPTION;"
a = QueryMaster("1", SQL, "", 0) 'Execute the Query

'----- Delete entries that already exist in SYS_TableSources -----
SQL = "DELETE DISTINCTROW TMP_maint.* FROM TMP_maint,
SYS_TableSources, "
SQL = SQL & "TMP_maint INNER JOIN SYS_TableSources "
SQL = SQL & "ON TMP_maint.TableName = SYS_TableSources.table_name
"
SQL = SQL & "WITH OWNERACCESS OPTION;"
```

```
a = QueryMaster("2", SQL, "", 0) 'Execute the Query

'---- Insert new entries into SYS_TableSources ----
SQL = "INSERT INTO SYS_TableSources (database_name, table_name) "
SQL = SQL & "SELECT DISTINCTROW '?', TMP_maint.TableName "
SQL = SQL & "FROM TMP_maint WITH OWNERACCESS OPTION;"
a = QueryMaster("3", SQL, "", 0) 'Execute the Query

'---- Delete temporary table when done ----
a = DeleteTable("TMP_maint", 0)
DoCmd SetWarnings True

End Function

'===== ADD A NEW UPDATE TO THE MENU =====
'=======================================
Function NewUpdate ()
Dim a, vsb, Text, upd, mac, caption
Dim title, upform, template, upmacro, tempmacro
Dim inmsg As String, newline As String, ErrMsg As String

On Error GoTo NU_ErrorCheck
newline = Chr(13) & Chr(10)
'---- Add a new line below if you have a new available template ----
'-------------------------------------------------------
   inmsg = "Enter: " & newline
   inmsg = inmsg & "(1) Update won't use input textfile" & newline
   inmsg = inmsg & "(2) Update will use input textfile" & newline
   inmsg = inmsg & "(3) Monitor external update only" & newline
   inmsg = inmsg & "(4) Execute external batch"

'-------------------------------------------------------

'---- Loop to find the first empty update slot ----
For a = 1 To 8
Text = "Text(" & a & ")"
vsb = Forms!General_Menu(Text).Visible

'---- When you find the empty slot do the following ----
If vsb = False Then

   '---- Set control and object names ----
   upd = "Upd(" & a & ")"    'Name of General_Menu update button
   mac = "Mac(" & a & ")"    'Name of General_Menu macro button
   upform = "Update_" & a    'Name of Update form
   upmacro = "Macro_" & a    'Name of Update macro
```

```
    '---- Set title for update ----
    title = InputBox("Enter the title for the desired update:", "New Update
Title")
    If title = "" Then Exit Function
    title = a & "-" & title

    '---- Set description of update ----
    caption = InputBox("Enter a short description for the desired update:",
"New Update Description")
    If caption = "" Then Exit Function
    caption = "Update_" & a & ": " & caption

    '---- Determine update template to use ----
    template = InputBox(inmsg, "Update Template")
    If template = "" Then Exit Function
    tempmacro = "TEMPLATE" & template & "_macro"
    template = "TEMPLATE" & template & "_Update"

    '---- Update General_Menu to make the new update visible ----
   DoCmd Echo False
    DoCmd OpenForm "General_Menu", A_DESIGN
    Forms!General_Menu(Text).caption = caption
    Forms!General_Menu(Text).Visible = True
    Forms!General_Menu(upd).Visible = True
    Forms!General_Menu(mac).Visible = True
    DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
     DoCmd OpenForm "General_Menu", A_NORMAL

    '---- Create a new template macro for the update ----
    DoCmd SetWarnings False
    DoCmd SelectObject A_MACRO, tempmacro, True
   DoCmd CopyObject , upmacro

    '---- Create a new template form for the update ----
    DoCmd SelectObject A_FORM, template, True
   DoCmd CopyObject , upform
    DoCmd SetWarnings True

    '---- Update information in the new form ----
    DoCmd OpenForm upform, A_DESIGN 'formname, DesignView
    Forms(upform)!Titletext.caption = title
    Forms(upform)!UpdButton.OnPush = upmacro
    DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
    DoCmd Close 2, upform 'Form, formname
```

```
    '----- Return to the General_Menu form -----
    DoCmd SelectObject A_FORM, "General_Menu", False
   DoCmd Echo True
   GoTo NewUpdate_end
End If
Next
NewUpdate_end:
Exit Function

'---- On Error:  Display error and exit -----
NU_ErrorCheck:
ErrMsg = "You Generated this error:" & Err & "."
If Err = 2544 Then ErrMsg = "Template macro " & tempmacro & " does not
exist."
MsgBox ErrMsg, 16, "New Update Error"
Exit Function

End Function

'===== REPLACE A GIVEN STRING WITH ANOTHER ONE =====
'===================================================
Function ReplaceString ()
Dim bigstring As String, instring As String, outstring As String
Dim pos As Integer, verify As Integer, outlen As Integer, biglen As Integer

'---- Enter string to be replaced and new string -----
outstring = InputBox("Enter name of string to replace:", "Replace String")
If outstring = "" Then Exit Function
instring = InputBox("New string:", "Replace String")
If instring = "" Then Exit Function

RepStr_loop:
'---- Set up to look for string to be replaced in the sql_code field -----
If IsNull([Forms]![Queries_form]![sql_code]) Then Exit Function
bigstring = [Forms]![Queries_form]![sql_code]
pos = InStr(bigstring, outstring)
outlen = Len(outstring)

'----- Replace all instances of the string appearing in the sql_code field -----
Do While (pos > 0)
   biglen = Len(bigstring)
    verify = MsgBox("Replace at " & pos & "?", 4, "Replace String")
   If verify = 6 Then
      bigstring = Left(bigstring, pos - 1) & instring & Right(bigstring, biglen -
(pos - 1) - outlen)
      [Forms]![Queries_form]![sql_code] = bigstring
```

```
   End If
   pos = InStr(pos + 1, bigstring, outstring)
Loop

'----- Ask to go to the next record to continue replacing -----
verify = MsgBox("Next Record?", 4, "Replace String")
If verify = 6 Then
   DoCmd GoToRecord , , A_NEXT
   GoTo RepStr_loop
End If
End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'-----------------------------------------------
'
'      LIBRARY OF QUERY AND DATA TRANSFER FUNCTIONS
'
'-----------------------------------------------

'===== DELETE A GIVEN TABLE =====
'================================
Function DeleteTable (tb_name As String, way)
On Error GoTo ErrorCheck
If way <> 0 Then
    MsgBox "Deletes the table " & tb_name & ".", 64, "Function DeleteTable"
    Exit Function
End If
DoCmd SelectObject A_TABLE, tb_name, True
DoCmd DoMenuItem 1, 1, 4   'DatabaseMenuBar , Edit, Delete
DeleteTable_end:
DeleteTable = 0
Exit Function

ErrorCheck:
Exit Function
End Function

'===== DELETE ALL THE RECORDS FROM A GIVEN TABLE =====
'=====================================================
Function EmptyTable (tablename As String, way)
Dim db As Database, qDelete As QueryDef, SQL As String

On Error Resume Next
If tablename = "?" Then Exit Function
If way <> 0 Then
    MsgBox "Deletes all records from " & tablename & ".", 64, "Function
EmptyTable"
    Exit Function
End If

Set db = CurrentDB()
SQL = "DELETE FROM " & tablename & ";"
Set qDelete = db.CreateQueryDef("EmptyAll", SQL)
qDelete.Execute
qDelete.Close
db.DeleteQueryDef ("EmptyAll")
EmptyTable = SQL
```

```
End Function

'===== DELETE TABLE RECORDS WHERE THE GIVEN CRITERIA IS TRUE
=====
'=============================================================
====
Function EmptyWhere (tablename As String, criteria As String)
Dim db As Database, qDelete As QueryDef, SQL As String

On Error GoTo ETW_ErrorCheck:
Set db = CurrentDB()
SQL = "DELETE FROM " & tablename & " WHERE " & criteria & ";"
Set qDelete = db.CreateQueryDef("EmptySome", SQL)
qDelete.Execute
qDelete.Close
db.DeleteQueryDef ("EmptySome")
EmptyWhere = SQL

ETW_ErrorCheck:
MsgBox "Error #" & Err, 16, "Error"
Resume Next

End Function

'===== IMPORT DELIMITED TEXT INTO A TABLE =====
'=============================================
Function GetImportDelim (infile As String, outtable As String, informat As
String, way)
Dim MSG As String, fields As Integer

On Error GoTo GID_ErrorCheck
fields = False
If Right(informat, 2) = "WF" Then fields = True
If IsNull(outtable) Then Exit Function
If way <> 0 Then
   MSG = "Imports delimited text to table " & outtable
   MSG = MSG & " from file " & infile & " using the "
   MSG = MSG & informat & " import format."
   MsgBox MSG, 64, "Function GetImportDelim"
   Exit Function
End If
DoCmd TransferText A_IMPORTDELIM, informat, outtable, infile, fields
Exit Function

GID_ErrorCheck:
MsgBox "Error #" & Err, 16, "Error"
```

```
Resume Next
End Function

'===== IMPORT FIXED WIDTH TEXT INTO A TABLE =====
'================================================
Function GetImportFixed (infile As String, outtable As String, informat As
String, way)
Dim MSG As String

On Error GoTo GIF_CheckError
If IsNull(outtable) Then Exit Function
If way <> 0 Then
   MSG = "Imports fixed length text to table " & outtable
   MSG = MSG & " from file " & infile & " using the "
   MSG = MSG & informat & " import format."
   MsgBox MSG, 64, "Function GetImportFixed"
   Exit Function
End If
DoCmd TransferText A_IMPORTFIXED, informat, outtable, infile
Exit Function

GIF_CheckError:
MSG = "Can't import file " & infile & ", error #" & Err
MsgBox MSG, 64, "Error on GetImportFixed"
Exit Function

End Function

'===== SET NULL VALUES TO ZERO FOR A PARTICULAR TABLE FIELD
=====
'============================================================
===
Function NullsToZeroes (tablename As String, fieldname As String, way)
Dim db As Database, qDelete As QueryDef, SQL As String
Dim SQL1 As String, SQL2 As String, MSG As String

'---- Send description of this function if way=0 ----
If way <> 0 Then
   MSG = "Sets all null instances of field " & fieldname
   MSG = MSG & " in table " & tablename & " to zero."
   MsgBox MSG, 64, "Function NullsToZeroes"
Exit Function
End If

'---- Execute query to turn nulls into zeroes ----
Set db = CurrentDB()
```

```
SQL1 = "UPDATE " & tablename & " SET " & fieldname & " = 0 "
SQL2 = "WHERE " & fieldname & " = Null;"
SQL = SQL1 & SQL2
Set qDelete = db.CreateQueryDef("EmptyAll", SQL)
qDelete.Execute
qDelete.Close
db.DeleteQueryDef ("EmptyAll")
NullsToZeroes = SQL
End Function

'===== EXECUTE/GIVE SQL/DESCRIBE/CREATE A GIVEN QUERY =====
'============================================================
Function QueryMaster (queryname As String, SQL As String, MSG As String,
way)
Dim db As Database, qUpdate As QueryDef
Dim ErrMsg As String, answer, queryparams As Snapshot

Set db = CurrentDB()
QueryMaster = 0
On Error GoTo CheckError

'---- Select execution: 0=Execute 1=SQL 2=Message 3=Create ----
If way <> 0 Then
   If way = 1 Then MsgBox SQL, 64, "SQL Code for " & queryname
   If way = 2 Then MsgBox MSG, 64, "Description of " & queryname
   If way = 3 Then Set qUpdate = db.CreateQueryDef(queryname, SQL)
   Exit Function
End If
Set qUpdate = db.CreateQueryDef(queryname, SQL)
Set queryparams = qUpdate.ListParameters()
DoCmd OpenQuery queryname
qUpdate.Close
db.DeleteQueryDef (queryname)
Exit Function

'---- Error handler ----
CheckError:
Const STOP_BOX = 16, OK_CANCEL_BOX = 49
Const QUERY_DEF_EXISTS = 3012, TABLE_DOESNT_EXIST = 3078
Const BAD1_SQL = 3129, BAD2_SQL = 3075, BAD3_SQL = 3070
Const BAD4_SQL = 3144, NO_PROC = 3146

If Err = QUERY_DEF_EXISTS Then
   ErrMsg = "If you continue, you will overwrite an existing "
   ErrMsg = ErrMsg & "query definition " & queryname & "."
   answer = MsgBox(ErrMsg, OK_CANCEL_BOX)
```

```
    If answer = 1 Then
        db.DeleteQueryDef (queryname)
        Resume
        Else Exit Function
    End If
End If

If Err = BAD1_SQL Or Err = BAD2_SQL Or Err = BAD3_SQL Or Err =
BAD4_SQL  Then
    ErrMsg = "The function " & queryname & " tried to execute an "
    ErrMsg = ErrMsg & "SQL statement with illegal syntax."
    MsgBox ErrMsg, STOP_BOX
    Exit Function
End If

If Err = TABLE_DOESNT_EXIST Then
    ErrMsg = "The function " & queryname & " looked for a table "
    ErrMsg = ErrMsg & "that doesn't exist."
    MsgBox ErrMsg, STOP_BOX
    Exit Function
End If

If Err = NO_PROC Then
    ErrMsg = "No processes available for ODBC call.  "
    ErrMsg = ErrMsg & "Try reinstalling your network connections, "
    ErrMsg = ErrMsg & "or reattaching the tables using another SQL server."
    MsgBox ErrMsg, STOP_BOX
    Exit Function
End If

ErrMsg = "You generated this error: " & Str(Err) & " in " & queryname & "."
MsgBox ErrMsg, STOP_BOX
Exit Function
End Function

'===== CALL QueryMaster WITH ARGUMENTS FROM
SYS_Update_Queries =====
'=============================================================
======
Function SuperQueryMaster (queryname As String, way)
Dim a, SQL As String, MSG As String, ErrMsg As String
Dim db As Database, tb_queries As Dynaset
Dim query_sql As String
Const STOP_BOX = 16

On Error GoTo SuperQueryMaster_ErrorCheck
```

```
Set db = CurrentDB()

'---- Get sql_code and descripton for the desired query name ----
query_sql = "SELECT DISTINCTROW "
query_sql = query_sql & "SYS_Update_Queries.sql_code, "
query_sql = query_sql & "SYS_Update_Queries.description "
query_sql = query_sql & "FROM SYS_Update_Queries "
query_sql = query_sql & "WHERE ((SYS_Update_Queries.query_name = '"
query_sql = query_sql & queryname & "')) "
query_sql = query_sql & "WITH OWNERACCESS OPTION;"
Set tb_queries = db.CreateDynaset(query_sql)

'---- Display message and exit if query name doen't exist ----
If tb_queries.EOF Then
   ErrMsg = queryname & " does not exist"
   ErrMsg = ErrMsg & ". Open the [SQL Code] window and "
   ErrMsg = ErrMsg & "verify that the query name is not misspelled."
   MsgBox ErrMsg, STOP_BOX
   Exit Function
End If
tb_queries.MoveFirst

'---- Display message and exit if SQL code doen't exist ----
If IsNull(tb_queries.sql_code) Then
   ErrMsg = "There is no sql code associated with " & queryname
   ErrMsg = ErrMsg & ". Open the [SQL Code] window to " & queryname
   ErrMsg = ErrMsg & " and add the SQL statement to the SQL code box."
   MsgBox ErrMsg, STOP_BOX
   Exit Function
End If

'---- Get SQL and Description ----
SQL = tb_queries.sql_code
If IsNull(tb_queries.description) Then
   MSG = ""
   Else MSG = tb_queries.description
End If
tb_queries.Close

'---- Call QueryMaster to execute according to the value of way ----
a = QueryMaster(queryname, SQL, MSG, way)
Exit Function

SuperQueryMaster_ErrorCheck:
MsgBox "Error #" & Err, 64
Exit Function
```

```
End Function
```

```
Option Compare Database   'Use database order for string comparisons
Declare Function SNDPLAYSOUND% Lib "mmsystem" (ByVal filename$,
ByVal Snd_Asynck%)
'-------------------------------------------------
'
'              FUNCTIONS THAT PLAY SOUNDS
'
'-------------------------------------------------

'===== PLAY THE GIVEN WAV FILE =====
'====================================
Function PlayThisWAV (w As String)
Dim X As Integer
X = SNDPLAYSOUND(Trim(w), 1)
End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'------------------------------------------------------
'
'        MISCELLANEOUS UTILITY FUNCTIONS
'
'------------------------------------------------------

'===== CHANGE A NULL VALUE TO THE STRING "None" =====
'=====================================================
Function ElimNull (fieldname)
Dim fieldvalue

fieldvalue = fieldname
If IsNull(fieldvalue) Then fieldvalue = "None"
ElimNull = fieldvalue
End Function

'===== GET THE FIRST DATE OF A GIVEN MONTH =====
'================================================
Function GetFirstDay (thismonth As Control) 'Argument as YY-MM

On Error Resume Next
If IsNull(thismonth) Then End
GetFirstDay = CVDate(Right(thismonth, 2) & "/01/" & Left(thismonth, 2))
End Function

'===== GET THE LAST DAY OF A GIVEN MONTH =====
'==============================================
Function GetLastDay (thismonth As Control) 'Argument as YY-MM
Dim thisdate, nextmonth

On Error Resume Next
If IsNull(thismonth) Then End
thisdate = CVDate(Right(thismonth, 2) & "/01/" & Left(thismonth, 2))
nextmonth = DateAdd("m", 1, thisdate)
GetLastDay = DateAdd("d", -1, nextmonth)
End Function

'===== GET PREVIOUS MONTH FROM TODAY AS YY-MM =====
'===================================================
Function GetLastMonth ()
Dim thisdate, newdate, newyear As String, newmonth As String

thisdate = Date
newdate = DateAdd("m", -1, thisdate)
```

```
newyear = Right(DatePart("yyyy", newdate), 2)
newmonth = DatePart("m", newdate)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
GetLastMonth = newyear & "-" & newmonth
End Function

'===== GET THE 5-CHARACTER MAIN PART OF A GIVEN PART
NUMBER =====
'============================================================
===
Function GetMain (Partno)
Dim testmain

testmain = Partno
If IsNull(testmain) Then
   testmain = ""
   GoTo setvalueMain
End If
testmain = Left(testmain, 5)
If Right(testmain, 1) = "-" Then testmain = Left(testmain, 4) & " "
If testmain = "NO PA" Then testmain = "NONE "
setvalueMain:
GetMain = testmain
End Function

'===== GET MONTH AS YY-MM FOR A GIVEN DATE =====
'============================================
Function GetMonth (datefield)
Dim newyear, newmonth

newyear = Right(DatePart("yyyy", datefield), 2)
newmonth = DatePart("m", datefield)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
GetMonth = newyear & "-" & newmonth
End Function

'===== GET A NEW DATE BY ADDING/SUBTRACTING DAYS TO THE
ARGUMENT =====
'============================================================
========
Function GetNewDay (thisday, delta)
Dim thisdate, newdate, newyear, newmonth, newday

thisdate = CVDate(Right(Left(thisday, 5), 2) & "/" & Right(thisday, 2) & "/" &
Left(thisday, 2))
newdate = DateAdd("d"; delta, thisdate)
```

```
newyear = Right(DatePart("yyyy", newdate), 2)
newmonth = DatePart("m", newdate)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
newday = DatePart("d", newdate)
If Len(newday) = 1 Then newday = "0" & newday
GetNewDay = newyear & "-" & newmonth & "-" & newday
End Function

'===== GET A NEW MONTH BY ADDING/SUBTRACTING MONTHS TO
THE ARGUMENT =====
'============================================================
==========
Function GetNewMonth (thismonth As Control, delta) As String
Dim thisdate, newdate, newyear As String, newmonth As String

thisdate = CVDate(Right(thismonth, 2) & "/01/" & Left(thismonth, 2))
newdate = DateAdd("m", delta, thisdate)
newyear = Right(DatePart("yyyy", newdate), 2)
newmonth = DatePart("m", newdate)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
GetNewMonth = newyear & "-" & newmonth
End Function

'===== GET THE SQL CODE FOR A GIVEN QUERY =====
'============================================
Function GetSql (qd_name As String)
Dim db As Database, Q As QueryDef
Dim SQL As String

On Error GoTo GetSql_ErrCheck
Set db = CurrentDB()
Set Q = db.OpenQueryDef(qd_name)
SQL = Q.SQL
GetSql = SQL
Exit Function

GetSql_ErrCheck:
MsgBox "Error #" & Err, 52
Exit Function

End Function

'===== GET THE DASH AND LAST TWO CHARACTERS OF A GIVEN
PART NO =====
'============================================================
======
```

```
Function GetSufx (Partno)
Dim testsufx

testsufx = Partno
If IsNull(testsufx) Then
   testsufx = ""
   GoTo setvalueSufx
End If
testsufx = Right(testsufx, 3)
If Left(testsufx, 1) <> "-" Then testsufx = Right(testsufx, 2) & " "
If Not (testsufx Like "*-*") Then testsufx = "-??"
setvalueSufx:
GetSufx = testsufx
End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'------------------------------------------------------
'
'              GENERAL MENU FUNCTIONS
'
'------------------------------------------------------

'===== HIDE THE LAST REPORT ON THE MENU =====
'============================================
Function HideReport ()
Dim a, vsb, text, upd, mac, caption, msg, choice

'----- Find the last report and hide it -----
On Error Resume Next
For a = 8 To 1 Step -1
text = "Text(" & a & ")"
vsb = Forms!General_Menu(text).visible

  If vsb = True Then
    '----- Verify if user wants to hide the report -----
    msg = "Are you sure you want to hide Report_" & a & "?"
    choice = MsgBox(msg, 36, "Hide Last Report")
    If choice = 7 Then GoTo HideReport_end
    upd = "Upd(" & a & ")"
    mac = "Mac(" & a & ")"

    '----- Hide the report on the General Menu form -----
    DoCmd Echo False
     DoCmd DoMenuItem 0, 2, 0 'FormMenuBar, View, DesignView
     Forms!General_Menu(text).caption = "Report_" & a & ": Not used."
     Forms!General_Menu(text).visible = False
     Forms!General_Menu(upd).visible = False
     Forms!General_Menu(mac).visible = False
     DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
     DoCmd DoMenuItem 3, 2, 1 'FormDesignMenuBar, View, Form
    DoCmd Echo True
    GoTo HideReport_end
End If
Next
HideReport_end:
DoCmd SelectObject A_FORM, "General_Menu", False
End Function

'===== CREATE A NEW REPORT WINDOW =====
'======================================
```

```
Function NewReport ()
Dim a, vsb, text, upd, mac, caption, title, upform, template, upmacro
Dim tempmacro, ErrMsg As String

'----- Find the first open location to insert the new report option -----
On Error GoTo NR_ErrorCheck
For a = 1 To 8
text = "Text(" & a & ")"
vsb = Forms!General_Menu(text).visible

 If vsb = False Then
   upd = "Upd(" & a & ")"    'Name of General_Menu update button
   mac = "Mac(" & a & ")"    'Name of General_Menu macro button
   upform = "Report_" & a    'Name of Update form
   upmacro = "Macro_" & a    'Name of Update macro

   '----- Enter the data to create the new report window -----
   title = InputBox("Enter the title for the desired report:", "New Update
Title")
   title = a & "-" & title

   caption = InputBox("Enter a short description for the desired report:", "New
Update Description")
   caption = "Report_" & a & ": " & caption

   template = 1
   If template <> 2 Or IsNull(template) Then template = 1
    tempmacro = "TEMPLATE" & template & "_macro"
    template = "TEMPLATE" & template & "_Report"

   '----- Add the new report option to the General Menu form -----
   DoCmd Echo False
    DoCmd DoMenuItem 0, 2, 0 'FormMenuBar, View, DesignView
    Forms!General_Menu(text).caption = caption
    Forms!General_Menu(text).visible = True
    Forms!General_Menu(upd).visible = True
    Forms!General_Menu(mac).visible = True
    DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
    DoCmd DoMenuItem 3, 2, 1 'FormDesignMenuBar, View, Form

   '----- Create a new template macro for the update -----
    DoCmd SetWarnings False
    DoCmd SelectObject A_MACRO, tempmacro, True
   DoCmd CopyObject , upmacro

   '----- Create a new template form for the update -----
```

```
    DoCmd SelectObject A_FORM, template, True
    DoCmd CopyObject , upform
    DoCmd SetWarnings True

    '----- Set up form buttons to point to the correct macro -----
    DoCmd OpenForm upform, 1 'formname, DesignView
    Forms(upform)!Titletext.caption = title
     Forms(upform)!ViewButton.OnPush = upmacro & ".open_rep"
     Forms(upform)!PrintButton.OnPush = upmacro & ".print_rep"
    DoCmd DoMenuItem 3, 0, 2 'FormDesignMenuBar, File, Save
    DoCmd Close 2, upform 'Form, formname

    DoCmd SelectObject A_FORM, "General_Menu", False
    DoCmd Echo True
    GoTo NewUpdate_end
End If
Next
NewUpdate_end:
Exit Function

NR_ErrorCheck:
ErrMsg = "You Generated this error:" & Err & "."
If Err = 2544 Then ErrMsg = "Template macro " & tempmacro & " does not
exist."
MsgBox ErrMsg, 16, "New Report Error"
Exit Function

End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'-----------------------------------------------
'
'       MISCELLANEOUS UTILITY FUNCTIONS
'
'-----------------------------------------------

'===== DISPLAY NEGATIVE NUMBERS BETWEEN PARENTHESIS =====
'=========================================================
Function accformat (anynumber)
If anynumber < 0 Then
   accformat = "(" & Abs(anynumber) & ")"
Else accformat = anynumber & " "
End If
End Function

'===== GIVE THE STRING EXPRESSION "NONE" TO A NULL VALUE
=====
'=============================================================
Function FillNull (Fieldvalue)
If IsNull(Fieldvalue) Then
   FillNull = "None"
Else FillNull = Fieldvalue
End If
End Function

'===== GET THE LAST MONTH FROM TODAY AS YY-MM =====
'================================================
Function GetLastMonth ()
Dim thisdate, newdate, newyear As String, newmonth As String
thisdate = Date
newdate = DateAdd("m", -1, thisdate)
newyear = Right(DatePart("yyyy", newdate), 2)
newmonth = DatePart("m", newdate)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
GetLastMonth = newyear & "-" & newmonth
End Function

'===== ADD A MONTH INTERVAL TO GET A NEW MONTH AS YY-MM
=====
'=============================================================
Function GetNewMonth (thismonth As Control, delta) As String
Dim thisdate, newdate, newyear As String, newmonth As String

thisdate = CVDate(Right(thismonth, 2) & "/01/" & Left(thismonth, 2))
```

```
newdate = DateAdd("m", delta, thisdate)
newyear = Right(DatePart("yyyy", newdate), 2)
newmonth = DatePart("m", newdate)
If Len(newmonth) = 1 Then newmonth = "0" & newmonth
GetNewMonth = newyear & "-" & newmonth
End Function

'===== RETURN THE ARGUMENT AS A STRING =====
'==========================================
Function GetString (ct_data As String) As String
GetString = ct_data
End Function
```

```
Option Compare Database   'Use database order for string comparisons
Option Explicit
'-------------------------------------------------
'
'      FUNCTIONS FOR CREATING A MAIN REPORT AND A SUB REPORT
'
'-------------------------------------------------

'===== CREATE A MAIN REPORT FROM A TEMPLATE =====
'================================================
Function CreateMain ()
Dim report_name As String, report_source As String
ReDim item(9) As String, group(9), display(9)
Dim a, i, upform As Form, cntrl As Control, ErrMsg As String

'---- Set the name and data source for the new report ----
On Error GoTo CM_ErrorCheck
Set upform = [Forms]![Make_Main]
If IsNull(upform("report_name")) Then Exit Function
report_name = upform("report_name")
report_name = "Rep_" & report_name
If IsNull(upform("report_source")) Then
   report_source = ""
Else report_source = upform("report_source")
End If

'---- Set values for groups and text boxes ----
For i = 1 To 3
   If upform("group(" & i & ")") = False Then upform("group(" & i & ")") = 0
   If upform("display(" & i & ")") = False Then upform("display(" & i & ")") =0
    If IsNull(upform("item(" & i & ")")) Then upform("item(" & i & ")") = ""
   item(i) = upform("item(" & i & ")")
   group(i) = upform("group(" & i & ")")
   display(i) = upform("display(" & i & ")")
Next

'---- Create the new report from the template report ----
DoCmd Echo False
DoCmd SelectObject A_REPORT, "TEMPLATE1_Rep", True
DoCmd CopyObject , report_name
DoCmd SetWarnings False
DoCmd OpenReport report_name, A_DESIGN
Reports(report_name).RecordSource = report_source

'---- Create the groups and text boxes in the new report ----
For i = 1 To 4
```

```
If item(i) <> "" Then
   a = CreateGroupLevel(report_name, item(i), group(i), group(i))
  If group(i) = True And display(i) = True Then
  Set cntrl = CreateReportControl(report_name, 109, 5 + a * 2, item(i), 0, 0)
  cntrl.ControlSource = item(i)
  cntrl.Left = (i - 1) * 1500
  ElseIf group(i) = False And display(i) = True Then
  Set cntrl = CreateReportControl(report_name, 109, 0, item(i), 0, 0)
  cntrl.ControlSource = item(i)
  cntrl.Left = (i - 1) * 1500
  End If
End If
Next
DoCmd Close A_REPORT, report_name
DoCmd SelectObject A_FORM, "Make_Main", False
DoCmd SetWarnings True
DoCmd Echo True
Exit Function

CM_ErrorCheck:
DoCmd SetWarnings True
DoCmd Echo True
If Err = 2501 Then
    DoCmd SelectObject A_FORM, "Make_Main", False
   Exit Function
End If
If Err = 2544 Then
   ErrMsg = "Need report TEMPLATE1_Rep to create main report"
   MsgBox ErrMsg, 16, "CreateMain Error"
   Exit Function
End If

MsgBox "Error #" & Err, 16, "CreateMain Error"
Exit Function

End Function

'===== CREATE A SUB-REPORT FROM A TEMPLATE =====
'===============================================
Function CreateSub ()
Dim report_name As String, report_source As String
ReDim item(9) As String, label(9)
Dim a, i, group As Integer
Dim upform As Form, cntrl As Control, ErrMsg As String

'----- Set values for form and report -----
```

```
On Error GoTo CS_ErrorCheck
Set upform = [Forms]![Make_Sub]
If IsNull(upform("report_name")) Then Exit Function
report_name = upform("report_name")
report_name = "Sub_" & report_name
If IsNull(upform("report_source")) Then
   report_source = ""
Else report_source = upform("report_source")
End If

'----Get values for fields and labels -----
For i = 1 To 4
    If IsNull(upform("label(" & i & ")")) Then upform("label(" & i & ")") = ""
    If IsNull(upform("item(" & i & ")")) Then upform("item(" & i & ")") = ""
   label(i) = upform("label(" & i & ")")
   item(i) = upform("item(" & i & ")")
Next

'----- Create a new subreport from the template subreport -----
DoCmd Echo False
DoCmd SelectObject A_REPORT, "TEMPLATE1_Sub", True
DoCmd CopyObject , report_name
DoCmd SetWarnings False
DoCmd OpenReport report_name, A_DESIGN
Reports(report_name).RecordSource = report_source

'---- Put fields and labels into new report ----
group = True
For i = 1 To 4
If item(i) <> "" Then
   a = CreateGroupLevel(report_name, item(i), group, group)
  group = False
  Set cntrl = CreateReportControl(report_name, 100, 5, "label" & i, 0, 0)
  cntrl.Caption = label(i)
  cntrl.Left = (i - 1) * 1500
  cntrl.Width = 1400
  Set cntrl = CreateReportControl(report_name, 109, 0, item(i), 0, 0)
  cntrl.ControlSource = item(i)
  cntrl.Top = 50
  cntrl.Left = (i - 1) * 1500
End If
Next
DoCmd Close A_REPORT, report_name
DoCmd SelectObject A_FORM, "Make_Sub", False
DoCmd SetWarnings True
DoCmd Echo True
```

```
Exit Function

CS_ErrorCheck:
DoCmd SetWarnings True
DoCmd Echo True
If Err = 2501 Then
    DoCmd SelectObject A_FORM, "Make_Sub", False
   Exit Function
End If
If Err = 2544 Then
   ErrMsg = "Need report TEMPLATE1_Sub to create subreport"
   MsgBox ErrMsg, 16, "CreateSub Error"
   Exit Function
End If

MsgBox "Error #" & Err, 16, "CreateSub Error"
Exit Function

End Function
```