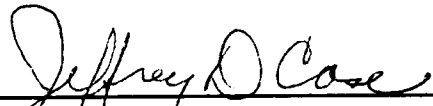
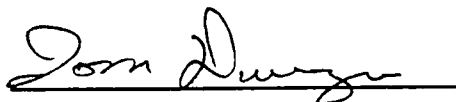


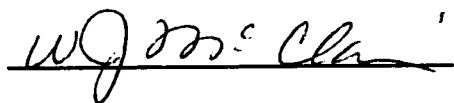
To the Graduate Council:

I am submitting herewith a thesis written by Gary W. Haney entitled "The Development of a Network Management Query Language." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.


Jeffrey D. Case, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


Vice Provost
and Dean of the Graduate School

Statement of Permission to Use

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Law D. Jones

Date 12/31/90

**The Development of a Network Management
Query Language**

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Gary W. Haney

May 1991

For Denise

The best thing that ever happened to me was meeting you.

ACKNOWLEDGEMENTS

Dr. Jeffrey Case, my thesis adviser, is renowned for his knowledge in the field of network management. I owe special thanks to him for his assistance in the development, growth, and finishing of this work. He always expressed his opinions in a positive manner and his constructive criticism was invaluable as this work progressed.

I am especially grateful to those who read the entire thesis and helped with corrections, criticism and encouragement: Dr. Bill McClain and Dr. Tom Dunigan. Many thanks to all readers for taking time out of their schedules and serving on the thesis committee.

I also express gratitude to Dr. Bill McClain, Dr. David Straight, and Dr. Jeff Case for the excellent classroom instruction I have received as a graduate student.

I also express gratitude to Ms. Veronica Maya and Ms. Jean Smith of the Oak Ridge Resident Graduate Program for their devotion to their jobs and concern for the students. Thanks also to Kedem Kaminsky for Figure 6.

My most personal and deepest thanks I owe to my wife, Denise Haney, not only for her encouragement, patience, and understanding, but for her willingness to perform added responsibilities that allowed me the time needed to complete this work. I am also grateful to all those who encouraged me to continue: parents, brothers, in-laws, and friends.

ABSTRACT

Network management is receiving increased attention from both vendors and users. International standards for network management have not considered the user interface within the initial scope of standardization. Without a standard user interface for obtaining management information, users operating in a multi-vendor environment may be required to learn multiple syntaxes and semantics to communicate with each vendor's version of the same standardized protocols. A common network management query language would require that users learn only one form of interface to accomplish their tasks. This research presents a proposal for the utilization of SQL as a common network management query language.

Table of Contents

Chapter	Page
1. Introduction to the Problem	1
1.1. Statement of the Problem	4
1.2. Benefits of the Proposed System.	4
1.2.1. Single Point of Reference	5
1.2.2. Migration and Preservation.	5
1.2.3. Reduced Training	6
1.2.4. Support of Multiple Protocols.	6
1.3. Drawbacks of the Proposed System	7
1.3.1. Tailored Interfaces.	7
1.3.2. Single Point of Failure.	8
1.4. Significance of the Study.	8
1.4.1. Impact on the Network Community	8
1.5. General Research Questions.	9
1.6. Assumptions	9
1.7. Limitations of the Study	10
2. Background Research	11
2.1. The Need for a Common User Interface	11
2.1.1. Desirability of a Standard User Interface	13
2.2. ISO/OSI and TCP/IP Network Management Standards.	14
2.2.1. Efforts to Define a Common CMOT/SNMP Interface.	17
2.3. What Should the Interface Be?	18
2.4. How Should the User Interface Be Presented?.	22
2.5. Other Non-traditional Uses of SQL.	25
3. Implementation Methods and Procedures	27
3.1. Mapping SQL to Network Management Applications.	27
3.1.1. Mapping of Accepted SQL Datatypes to MIB-based Variables ...	27
3.1.2. Security of the Historical Information.	29
3.1.2.1. Grant and Revoke Commands.	29
3.1.2.2. Views as a Security Mechanism.	31
3.2. Design of the NMQL	33

3.2.1. Management Domains for an Enterprise	35
3.3. Environment of the NMQL Prototype and Data Collection	
Methods.	39
3.3.1. Creation of the NMQL Historical Database.	40
3.3.2. The NMQL Manager.	40
3.3.3. The NMQL Agent.	45
3.4. Implementation of the NMQL	46
3.5. Analysis Procedures.	47
3.6. Question Testing.	47
3.6.1. Question 1: Operands of SQL	47
3.6.2. Question 2: Operators of SQL.	48
3.6.3. Question 3: Mapping of CMIP to SQL	48
3.6.4. Question 4: Common Interface Characteristics	50
4. Results and Findings	52
4.1. Question 1: Operands of SQL.	52
4.2. Question 2: Operators of SQL	56
4.3. Question 3: Mapping of CMIP to SQL	59
4.3.1. Multiple Syntaxes.	59
4.3.2. Semantics	61
4.4. Common User Interface Characteristics	63
4.4.1. API Support	64
4.4.2. Windowing Support	66
4.4.3. Command Line Support	67
5. Conclusions and Summary.	69
5.1. Summary	69
5.1.1. SQL as an NMQL.	69
5.1.1.1. Operand Limitations.	69
5.1.1.2. Operator Limitations.	70
5.1.1.3. Suggested Solutions to Recognized Limitations.	70
5.1.2. Common User Interface	72
5.1.3. Future Deployment of an NMQL.	72
5.2. Conclusions	74
5.3. Recommendations for Further Study	74
Bibliography	76

Appendices.	83
Appendix A - Source Code for Bridge_Schema_Create.sc	84
Appendix B - Source Code for the NMQL_Manager.sc	99
Appendix C - Source Code for the NMQL_Agent.sc	120
Appendix D - Necessary Command Files.	148
Appendix E - User's Guide to the NMQL.	151
Vita.	153

List of Figures

Figure 1 - Network Management in the ISO/OSI Environment	16
Figure 2 - Comparison of SNMP with CMIP.	18
Figure 3 - Detailed Description of Network Management in the ISO Environment.	34
Figure 4 - Diagram of the COMPNET Enterprise Network	37
Figure 5 - Domain Partitioning of Blakely Enterprises Network.	38
Figure 6 - An SQL API for CMIP	62

List of Tables

Table I - Mapping of SQL Verbs and Commands to NMQL	28
Table II - Specification for Managed Object LANBridge.	41
Table III - Specification for Managed Object LANBridge (cont.)	42
Table IV - List of Dynamic SQL Statements.	44
Table V - Abstraction of Network Management Commands	49
Table VI - Abstraction of CMIP Network Management Commands	50
Table VII - Mapping between SQL, NM Abstractions, and Pass-thru Functions	53

Development of a Network Management Query Language

**Gary W. Haney
Computer Science Department
University of Tennessee, Knoxville**

Chapter 1

Introduction to the Problem

During the early years of computer networking, when computer networks consisted of direct point-to-point links between a few computer systems within close proximity of each other, network management was not thought of as a major issue. Today, however, computer networks are much more diverse in nature. As a result, the ability to manage networks has become a major budget line-item in many companies.

Most computer networks are very diverse in architecture, speed, and network type. These networks can consist of anything from a few microcomputers sharing disk drives to world-wide networks consisting of multi-vendor equipment with diverse processing capacities. Many computer vendors characteristically implement networking of their products by using proprietary protocols, via a product-specific user interface, which may or may not communicate with another vendor's equipment.

In large organizations, network communication in multi-vendor environments is more likely to be required for those organizations to conduct their business properly. Communications in a multi-vendor, multi-protocol environment may require the use of an internet. An internet is simply a network of networks; it is designed to allow multiple networks utilizing multiple protocols to communicate over a common internet protocol using gateway devices to translate among the individual protocols.¹

Management of such large networks has become increasingly complicated, and sophisticated tools are needed to monitor and control the information flow without imposing severe restrictions on normal network operation. Most computer vendors provide system software that will permit management of the network from the individual computer system's point of view; however, few vendors provide network management tools from the network's point of view, and even fewer provide network management tools for an internet.

Several international standards committees (*e.g.*, Organization for International Standards/Open Systems Interconnection (ISO/OSI), Internet Activities Board (IAB), Consultative Committee for International Telephone and Telegraph (CCITT), Institute of Electrical and Electronics Engineers (IEEE), American National Standards Institute (ANSI), *etc.*) are proposing methods of implementing

¹Tanenbaum, Andrew S., *Computer Networks*, (Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981), pp. 353-367.

global, multi-vendor network management. Migration from multiple, proprietary protocols to the international standard protocols receives substantial attention from both vendors and users. However, many of these same vendors and users have developed network management applications using proprietary protocols and would like to preserve the software and hardware investment already made, and this may impede the standards development.

As stated above, the international standards committees are currently focusing on the protocols needed to implement systems which are consistent with the seven-layer model as defined by the ISO/OSI and how these protocols should co-exist. Little research exists concerning the standardization of a user-interface to be used by network management personnel. Yoshikazu Kobayashi, in a paper entitled "Standardization Issues in Integrated Network Management,"² states that standardization of the high-level management information category is currently considered a long-term issue. Development of a network management query language would greatly contribute to vendor interoperability and would also aid the end-user community by providing a single set of syntax and semantics that could be used on any network management package.

²Kobayashi, Yoshikazu, "Standardization Issues in Integrated Network Management," Integrated Network Management I, B. Meandzija and J. Westcott, Ed., Elsevier Science Publishers, Amsterdam, The Netherlands, International Federation for Information Processing, 1989, pp. 79-90.

1.1. Statement of the Problem

The purpose of this thesis research is to design, prototype, implement, and test a Network Management Query Language (NMQL). Current computer networks are multi-lingual, meaning multiple protocols co-existing simultaneously on the same network. Bridges and gateways provide the capability for dissimilar computer systems to communicate in multi-lingual environments. However, management of multi-lingual networks requires multiple network management software packages and possibly multiple management stations. To function in multi-lingual environments, network management applications developers are required to develop numerous interfaces to perform a single application. A network management query language can provide the capability of utilizing multiple, disparate network management protocols from a single workstation through a single, applications interface.

1.2. Benefits of the Proposed System

This research defines a Network Management Query Language (NMQL) that permits network management application(s) to access multiple, heterogenous, multi-vendor, and multi-protocol network-manageable devices. In particular, the syntax and semantics of the interface between network management applications and network protocol entities are described.

1.2.1. Single Point of Reference

With the vast number of network management protocols available today, whether proprietary or open, a common NMQL is needed to isolate the applications programmer from the details of having to communicate with multiple, disparate syntaxes of the various network management applications. Network management application programmers are no longer required to implement the calling mechanisms of multiple protocols to retrieve information from devices. Instead, the application interfaces with the NMQL, and the NMQL retrieves the desired information from the requested devices. If historical information is requested, the NMQL retrieves the specified information from a stored database. Since information is retrieved from all devices through the NMQL, multiple network management workstations are no longer needed. A common network management query language allows application programmers to concentrate on their applications and not on the specifics of each network management package being utilized.

1.2.2. Migration and Preservation

As the international standards for network management evolve and become implementable, network managers and vendors are certain to migrate toward these standards. As this migration begins, network managers will demand and vendors will provide a common and internationally-accepted user interface for network management, precluding the proprietary interfaces. A single, standardized NMQL virtually eliminates the transition from one network management application syntax

to another by providing a uniform interface which remains unchanged for network management applications. In addition, the NMQL insulates the design and implementation of network management applications from the instability which typically characterizes network management today and thereby serves to preserve investments in network management applications.

1.2.3. Reduced Training

With the development of a common, standardized NMQL, network managers will be able to reduce the amount of training currently required for their support personnel to function in today's environment. The NMQL provides the single common interface across all vendor products and requires that network management support personnel be trained on one interface instead of several.

1.2.4. Support of Multiple Protocols

The NMQL described herein is intended to support a wide variety of network management protocols and strategies including: the standard Transmission Control Protocol/Internet Protocol (TCP/IP)³ network management protocol, *i.e.*, the Simple Network Management Protocol (SNMP)⁴; the standard OSI network management

³Postel, J., "Transmission Control Protocol," Internet Engineering Task Force, RFC 793, September 1981.

⁴Case, J., Fedor, M., Schoffstall, M., and Davin J., "A Simple Network Management Protocol", Request for Comments 1098, DDN Network Information Center, SRI International, Menlo Park, CA (March 1988).

protocol, *i.e.*, the Common Management Information Protocol (CMIP)^{5,6}; an experimental TCP/IP network management protocol, *i.e.*, Common Management Information Protocol over TCP/IP (CMOT)⁷; the Manufacturing Automation Protocol and Technical Office Protocol (MAP/TOP)⁸; as well as proprietary network management protocols.

1.3. Drawbacks of the Proposed System

Although there are many advantages realized by implementing a NMQL, there are also certain disadvantages which are described in the following paragraphs.

1.3.1. Tailored Interfaces

The NMQL developed herein requires carefully built and specifically tailored interfaces between the NMQL and the vendor-specific devices to support existing

⁵Information Processing Systems - Open Systems Interconnection, "Management Information Service Definition - Part 2: Common Management Information Service," International Organization for Standardization, Draft International Standard 9595-2, August 1988.

⁶Information Processing Systems - Open Systems Interconnection, "Management Information Protocol Specification - Part 2: Common Management Information Protocol," International Organization for Standardization, Draft International Standard 9596-2, August 1988.

⁷Warrier, U. and L. Besaw, "The Common Management Information Services and Protocol over TCP/IP", Request for Comments 1095, DDN Network Information Center, SRI International, Menlo Park, CA (April, 1989).

⁸Manufacturing Automation Protocol Version 3.0 - Implementation Release - Subject to Errata Changes, July 20, 1987.

vendor-specific network management packages. These interfaces must be updated when new releases of the vendor's software are installed.

1.3.2. Single Point of Failure

Since the NMQL serves as a single point of reference for the applications programmer, it also serves as a single point of failure for the network management applications. With existing systems, if one or more protocols are inoperative, only those devices using said protocols are affected. With this system, if the NMQL is not functioning, the entire spectrum of devices is affected.

1.4. Significance of the Study

Initial benefits from this project will be realized within the Department of Computer Science at the University of Tennessee. Should this project be successful, it is expected that the results will be useful as a prototype implementation of a future production-quality system for use by the University of Tennessee Computing Center and elsewhere on campus, and by the internetworking community in general.

1.4.1. Impact within the Network Community

The network community as a whole will benefit from the availability of a common query language for network management applications that incorporates the ISO/OSI network management protocol standards. This will enable both users and vendors to migrate their software from the current proprietary implementations to the open OSI standards.

1.5. General Research Questions

1. How can the operands of the Standard Query Language (SQL) be adapted to perform network management applications?
2. How can the operators of SQL be adapted to perform network management functions?
3. How can the syntax/parameters of the CMIP be mapped to the commands provided by SQL?
4. What are the characteristics of a generic/common user interface for network management applications?

1.6. Assumptions

The research will be performed under the following assumptions:

1. Sufficient Unix workstations and VAX equipment with appropriate network hardware and software will be available for use on an experimental basis.
2. The network environment at the University of Tennessee, Knoxville campus, will be used as a model to guide and evaluate the design of the proposed system. It is assumed that similar considerations in relation to the types of supported protocols and network architecture apply to the design of other multi-protocol network management systems.
3. The Digital Equipment Corporation's (DEC) LANBridge 150 and 200 utilizing RBMS version 2.0 will be the hardware and management protocol used to test the feasibility of a common network management query language for network management applications.
4. The feasibility of such a system can be extrapolated from experience with the use of a prototype system in a limited area with a limited variety of network management agents.

1.7. Limitations of the Study

The prototype is operated subject to the following limitations:

1. Only the hardware mentioned above (LANBridge 150 and 200) is to be implemented in this prototype.
2. Information relating to the nodes being managed will be contained in separate SQL relations (*i.e.*, tables). Incorporation of a global network management database is left for further study.
3. The network/system manager is charged with the task of maintaining the integrity of all SQL relations. Standardized SQL tools will be used to create, change, or delete relations in the "managed nodes" database.
4. Authentication/Authorization information for each network management agent is to be contained in the SQL database.
5. The use of an industry-standard Remote Procedure Call⁹ mechanism to communicate with remote devices is excluded from this study.
6. The prototype system will be implemented under 4.3 Berkley Standard Development (BSD) Unix systems and DEC VAX systems. Portions of the prototype will include use or adaptations of existing network management systems. The objective is threefold: (1) to demonstrate the feasibility of the proposed design rather than to implement an entirely new system from scratch; (2) to provide a means of consolidating multiple network management systems into one logically functioning unit; (3) to provide a solid, consistent, and standardized user interface to network management personnel.

⁹Sun Microsystems, Inc. "Remote Procedure Call Protocol." Request for Comments 1057. Menlo Park, CA: DDN Network Information Center, SRI International, June 1988.

Chapter 2

Background Research

2.1. The Need for a Common User Interface

Typical network management centers in most large corporations contain a truly heterogeneous environment (*i.e.*, multiple management workstations executing multiple network management software packages). These software packages usually consist of proprietary software utilizing proprietary network management protocols with highly-customized user interfaces. When the ISO/OSI protocol suite is completed, it is expected that most of the network management vendors will migrate to those "open" protocols, at which time integrated network management can become a reality. Within this paper, integrated network management is defined as the capability to manage and control multi-vendor, multi-protocol networks from a single network management workstation.

Little research has been produced relating to standardizing the user interface to these "open" protocols. The ISO/OSI reference model does not currently include the standardization of the network management user interface in its suite of internationally standardized protocols.¹⁰ As stated earlier, Yoshikazu Kobayashi

¹⁰Aronoff, R., Chernick, M., Hsing, K., Mills, K., and Stokesberry, D., "Management of Networks Based on Open Systems Interconnection (OSI) Standards: Functional Requirements and Analysis", NIST Special Publication 55-175, U.S. Dept. of Commerce, November 1989, p. 85.

refers to standardization of network management information at the higher levels as a long-term goal.¹¹ The Corporation for Open Systems (COS) is a non-profit organization comprised of vendors, users, and researchers chartered to accelerate the introduction of OSI-conformant services and protocols. COS has formed a Network Management Subcommittee that is currently working heavily with the MAP/TOP users group and other vendors and users to develop a COS platform for network management.¹² The COS platform is focused on identifying which entities need management and how these entities are to be managed. COS does not currently attempt to standardize the user interface to the management platform.

Even without a standard user interface for network management, integrated network management will provide the network manager with the capability to migrate across vendor boundaries from a single workstation. However, the network manager or operations personnel will still have to remember how to manipulate each vendor's customized user interface. Lack of a consistent and uniform presentation of network information to the designated personnel leads to chaotic management of the network.¹³ Network operations personnel and/or network

¹¹Kobayashi, Yoshikazu, Ibid, p. 87.

¹²Faurer, L. D., "Network Management in an OSI Environment", Computer Standards and Interfaces, 7, North Holland, (1988) pp 155-59.

¹³Dupuy, A., et al., "Network Fault Management: A User's View," Integrated Network Management, I, B. Meandzija and J. Westcott, Ed., Elsevier Science Publishers, Amsterdam, The Netherlands, IFIP, 1989, p. 105.

managers should not be required to memorize the syntax and semantics of multiple network management languages. Instead, a common network management query language that will provide a single network management interface needs to be developed.

2.1.1. Desirability of a Standard User Interface

A single, standardized user interface offers several fairly obvious advantages:

- * *Reduced training costs:* Network managers, programmers, and operations personnel can move from environment to environment without the added cost of retraining.
- * *Applications portability:* Applications developed in one environment can be ported to another environment with little or no change.
- * *Applications longevity:* Applications developed implementing a standard are assured a longer lifetime.
- * *Customer preference:* If software applications are purchased and all vendors support the same user interface, customers are able to concentrate on the problem of selecting the system that best fits their own technical requirements without having to factor the user interface into their decision.
- * "Any robust interface with wide acceptance provides a platform for the development of software packages, making it economically feasible for many vendors to provide the interface on their machines. In turn, this stimulates the sale of existing software for that platform and provides the capital to develop more software, leading to even wider acceptance of the interface, leading to more development, and so on."¹⁴

However, a standard user interface also carries with it some disadvantages:

- * *A standard may stifle creativity:* Implementers or in-house developers may be preempted from providing a good solution to some problem, because the standard already specifies an alternative, maybe less satisfactory solution.
- * *No standard is perfect:* Despite the tremendous amount of work required for a standard to be designed, critiqued, re-designed, implemented, and

¹⁴Fong, K. and Reinstedler, J., "The Development of an OSI Application Layer Protocol Interface", Computer Communication Review, (1986), p. 26.

finally approved, no single standard can solve the world's problems. Any user is probably able to cite deficiencies in any standard that they are currently implementing. Also, what one user states as being a disadvantage that should be corrected, another will claim it as an advantage and demand that it not be changed. (Note: This is more commonly referred to as "the politics of standards.")

It is the purpose of this research to show that the advantages of implementing a standard far outweigh the accepted disadvantages.

2.2. ISO/OSI and TCP/IP Network Management Standards

Both the ISO/OSI and the TCP/IP standards committees are actively involved in the development of integrated network management standards. The TCP/IP committee, referred to as the Internet Activities Board, has selected two network management protocols. In RFC 1052¹⁵, Vinton Cerf stated that the Simple Network Management Protocol¹⁶ (SNMP) has been recognized as the network management protocol for the Internet for the short term and that Common Management information protocol over TCP/IP¹⁷ (CMOT) has been recognized as the Internet management protocol for the long term. CMOT is also to be designed to ease the migration of the TCP/IP world to the ISO/OSI world. The ISO/OSI committee has stated in ISO-9595 and ISO-9596 that the CMIP and the

¹⁵Cerf, V., "IAB recommendations for the development of Internet network management standards", Request for Comments 1052, DDN Network Information Center, Menlo Park, CA (April 1988).

¹⁶Case, J., Fedor, M., Schoffstall, M., and Davin, J., Ibid.

¹⁷Warrier, U. and Beesaw, L., Ibid.

CMIS will be the network management protocol for open systems. All three recognized network management protocols utilize the Management Information Base¹⁸ (MIB) as the virtual data store and the Structure of Management Information¹⁹ (SMI) as the dictionary that denotes the structure of management information. Recently, the OSI community has adopted the Guidelines for the Definition of Managed Objects²⁰ (GDMO) as its tool for creating object definitions. Figure 1²¹ depicts a summary description of the CMIP/CMIS architecture for network management. Note that distributed instances of MIB data are physically located at each of the ISO/OSI's seven layers. System Management Applications Entities (SMAE) are the means for conducting systems management on an ISO/OSI system. The SMAE can be viewed as a "tool box" containing various management services and protocols for exchanging management

¹⁸McCloghrie K., and Rose, M. T., "Management Information Base for TCP/IP-based Internets," Request For Comments 1066, Internet Engineering Task Force, (1988).

¹⁹Rose, M. and McCloghrie, K., "Structure and identification of Management Information for TCP/IP-based internets", Request for Comments 1065, DDN Network Information Center, SRI International, Menlo Park, CA, (Aug. 1988).

²⁰Information Processing Systems - Open Systems Interconnection, "Guidelines for the Definition of Managed Objects", Draft International Standard 10165-4, International Organization for Standardization, Omnicom International, June 1990.

²¹Wakid, S., Brusil, P., and LeBarre, L., "Coming to OSI: Network Resource Management and Global Reachability", Data Communications, (December 1987), pp. 137-150.

information between open systems. This exchange of information is implemented using the CMIP/CMIS protocol.

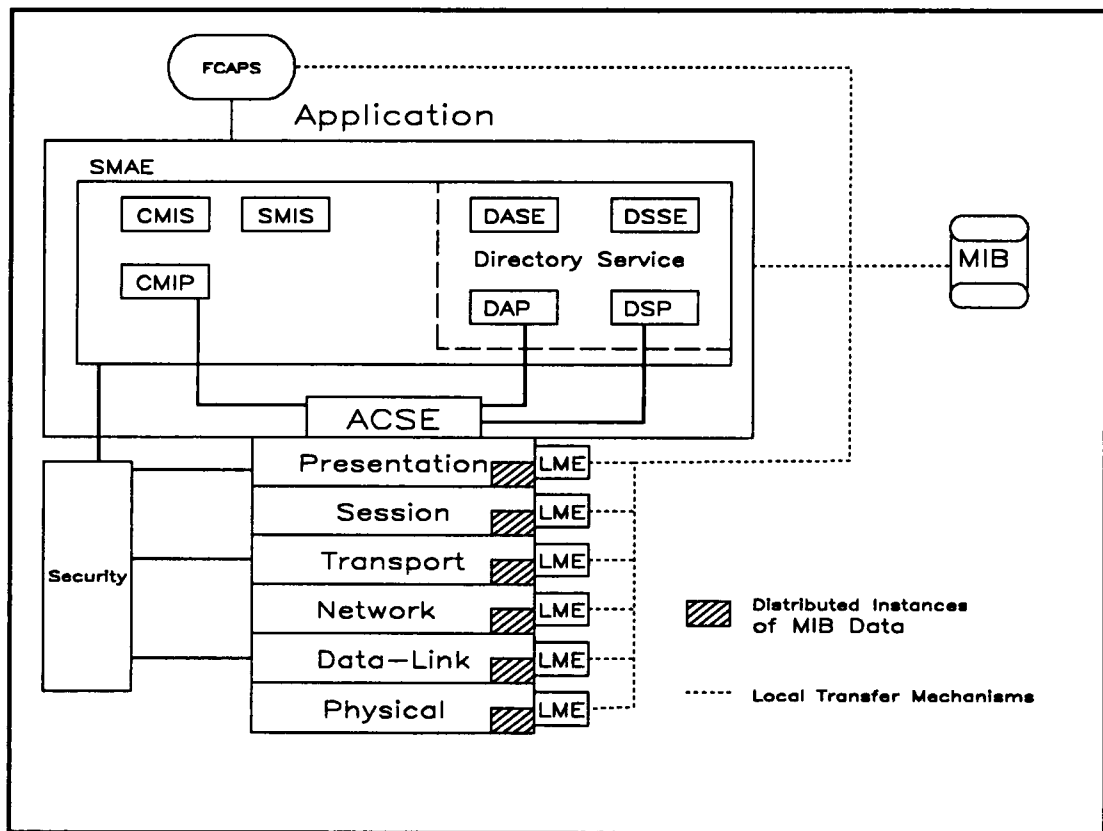


Figure 1 - Network Management in the ISO/OSI Environment
(Source: Data Communications, December 1987)

The five functional areas of systems/network management defined by ISO/OSI are: Fault, Configuration, Accounting, Performance, and Security. Each functional area is being standardized independently and each also has a different time schedule for review and acceptance by the standards committees.

Figure 2 is a diagram comparing the short-term network management standard for TCP/IP (*i.e.*, SNMP) with the ISO/OSI network management standard CMIP. Note that SNMP utilizes the connectionless-oriented User Datagram Protocol

(UDP)²² as its transport mechanism, while CMIP utilizes a Connection-Oriented Transport Protocol Specification described in ISO-8073. The utilization of Ethernet (ISO-8802.3²³ or IEEE-802.3²⁴) in the lower layers of Figure 2 is purely for comparison purposes.

Current work within the IETF and ISO/OSI is heavily involved in either protocol implementation, in the cases of the CMIP and the CMOT, or in the enhancement of the definition of managed objects, as in the case of the SNMP. Neither standards organization has described a standard method of how "management information" should be interfaced to the user.

2.2.1. Efforts to Define a Common CMOT/SNMP Interface

During the early segments of this research, an attempt by the author was made to establish a common Applications Programmable Interface (API) between the two prominent network management protocols for the Internet -- SNMP and CMOT. The author drafted a document describing the need for such an API and a Common API Working Group was established within the Internet Engineering Task Force

²²Postel, J., "User Datagram Protocol", Request for Comments 768, Internet Engineering Task Force, DDN Network Information Center, SRI International, Menlo Park, CA, (1988).

²³Information Processing Systems - Open Systems Interconnection, "Carrier Sense Multiple Access with Collision Detect (CSMA/CD)," International Organization for Standardization, International Standard 8802.3, 1988.

²⁴Institute of Electrical and Electronics Engineers, "Carrier Sense Multiple Access with Collision Detect,"

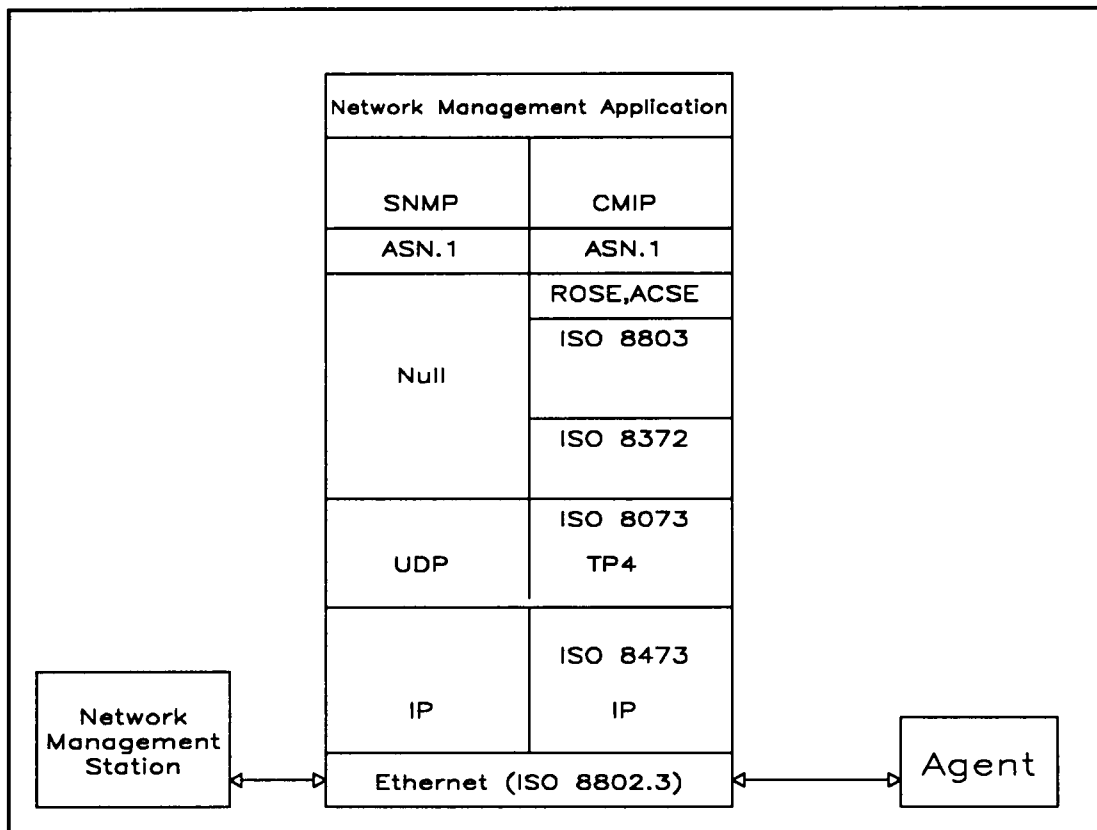


Figure 2 - Comparison of SNMP with CMIP.

(IETF) of the IAB. The API would function at a layer above both SNMP and CMOT and beneath the specific network management application. The goal of this API research was to allow network management applications to function independently of the management protocol recognized by any given device. The IETF working group was disbanded after no working implementation of CMOT was available.

2.3. What Should the Interface Be?

Management of a network can be generally represented in three phases: the collection of management data, the processing and/or monitoring of management

data, and the modification of management data. This data can either be realtime or historical in nature and can be physically stored on the respective devices or in some remote location away from the managed devices. With that generalization in mind, management of a network can be equivalenced to the management of data; the network can then be perceived as a distributed database, the respective network management protocol(s) can be equated to file access mechanism(s), the SMI taking the role of the data dictionary, and the MIB becoming the data views. Therefore, network management can be thought of as a highly-specialized form of database management remotely similar to that of On-Line Transaction Processing (OLTP). OLTP implements realtime transactions on remote, distributed databases.

Standard Query Language²⁵ (SQL) was adopted by ANSI in 1986 as the query language for relational database systems. SQL was originally developed by IBM in the mid-1970's under the project name SEQUEL to provide an easy-to-use and consistent user interface to both technical and non-technical personnel for use in database management applications.²⁶ SQL has gained wide acceptance from both vendor and user communities that primarily utilize data management software.

²⁵ANSI X3.135-1986, American National Standards for Information Systems - Database Language - SQL, American National Standards Institute, New York, N.Y., (October 1986).

²⁶Kroenke, David M., Database Processing: Fundamentals, Design, Implementation, 2nd Edition, Science Research Associates, Chicago, Ill., (1983), pp. 265-6.

Given the generalization above, it is to be shown that SQL (or a variant similar to SQL) can be adopted as a network management query language (NMQL). The NMQL and its supporting relational database environment are used to provide Management Information Base (MIB)-oriented network management, temporal control of network management operations (*i.e.*, the ability to specify execution time for network management commands), grouping of network management resources by defining new symbolic objects in the data dictionary (the SMI), and the grouping of network management operations via a procedure or macro facility. Additionally, the NMQL can be used to provide a common, easy-to-use, and consistent user interface to network managers and management operations personnel.

Unni Warriar and Carl Sunshine²⁷ have presented an idea for integrated network management through a platform of heterogeneous devices utilizing multiple software systems in cooperation with each other. In one segment of the paper, the authors propose the adoption of a Network Management Language (NML) based on a variant of SQL that would serve as the user interface for network management applications. The authors state that one possible generic format of an SQL-NML command might resemble:

²⁷Warriar, U. and Sunshine, C., "A Platform for Heterogeneous Interconnection Network Management," Integrated Network Management, I, B. Meandzija and J. Westcott (Ed.), Elsevier Science Publishers, Amsterdam, The Netherlands, IFIP, (1989), pp. 13-24.

*<NML command> ::= <SQL statement> [AT <time-value>] FOR <duration>
[EVERY <period>]*

where the *AT*, *FOR*, and *EVERY* statement clauses have obvious translations as time parameters for the given NML command. The authors state that they are currently prototyping certain stages of the system described by the paper, but give no details. It is this idea that will be expanded herein and developed into a network management query language system.

An example NMQL query, based on the above research, which requires realtime querying of the physical device resembles the following:

*SELECT LINE1_RECEIVED_FRAMES, LINE2_RECEIVED_FRAMES FROM
LANBRIDGE AT 17:00 FOR 24:00 EVERY 00:10;*

Example A

where *LINE1_RECEIVED_FRAMES* and *LINE2_RECEIVED_FRAMES* are columns of a SQL database called *LANBRIDGE*. The *AT*, *FOR*, and *EVERY* clauses are used to designate time beginning, length, and interval, respectively. These time clauses are non-standard SQL and are the additions to the existing SQL standard which form the basis of the NMQL.

Another example NMQL query which does not require physical querying of the device might resemble:

*SELECT LINE1_RECEIVED_FRAMES, LINE2_RECEIVED_FRAMES FROM
LANBRIDGE WITH BRIDGE_ID = "LAN SEGMENT A";*

Example B

where *LINE1_RECEIVED_FRAMES*, *LINE2_RECEIVED_FRAMES*, and *BRIDGE_ID* are columns of a SQL database called *LANBRIDGE*. The above statement is a standard SQL statement and will return all *LINE1_RECEIVED_FRAMES* and *LINE2_RECEIVED_FRAMES* rows in the table *LANBRIDGE* with a *BRIDGE_ID* equal to "LAN SEGMENT A".

Why choose SQL? SQL has been widely accepted in the Management Information Systems (MIS) world as the query language for relational databases. All major vendors have developed or are developing SQL interfaces to their proprietary DBMSs. Some vendors support only SQL as their user interface. SQL has proven itself as an acceptable interface to a wide variety of managers, users, and programmers. SQL or an augmented form of SQL should be able to perform the same type of functions within the context of network management.

2.4. How Should the User Interface be Presented?

Presentation of a particular user interface can generally be accomplished in one of three ways: either graphical, textual, or a combination of both. The advantages of an object-oriented, graphical interface are relatively obvious.²⁸ These advantages include: quicker-paced learning, better retention, ease of use, etc. However, certain persons may prefer a textual-based user interface.

²⁸Foley, James D., Wallace, Victor L., and Chan, Peggy. "The Human Factors of Computer Graphics Interaction Techniques," IEEE CG&A, (Nov 1984), 4(11):13-48.

If a graphical interface is to be used, this interface should conform to the X.11 standard. The X.11 standard, originally conceived at the Massachusetts Institute of Technology in 1984 under the name Project Athena, has been accepted by several international standards organizations including the Open Software Foundation (OSF), the X/Open Consortium, the U.S. National Institute of Standards and Technology (NIST), and the IEEE 1003.0 POSIX working group.²⁹ The advantage to using the X.11 Window System³⁰ standard in graphical applications is that user interfaces can be identical in all applications independent of the computer hardware. With identical user interfaces, corporations and users can worry less about having to learn a new vendor-customized user interface.

If a textual interface is chosen, two sub-options exist: verbose and condensed. Verbose user interfaces are of the forms presented in Examples A and B. A condensed user interface that is analogous to Example A might resemble:

```
-S L1_RCV_FRMS, L2_RCV_FRMS -F LANBRIDGE -A 17:00 -FO 24:00 -E
00:10;
```

Example C

²⁹Digital Equipment Corporation, Windowing - A Guide to the DECWindows Desktop Environment, Digital Press, (1989), pp. 9-14.

³⁰Scheifler, Robert W. and Gettys, Jim S., The X Window System, *ACM Transactions on Graphics*, 5(2):79-109, April 1986, Special Issue on User Interface Software.

While this type of user interface is somewhat more cryptic, it does provide a quicker means of accomplishing the same task as Example A by requiring less typing, and some well-trained professionals may prefer this type of user interface.

In a paper describing an architecture for integrated network management, Amatzia Ben-Artzi described an idea of presenting different styles of user interfaces depending on the type of user.³¹ The users of Netview[™], CMOT, and SNMP are presented with their respective style of interfaces. This idea has certain merits in that users are only presented with the information they need to perform their job functions adequately. The problem of how to handle the massive volume of network management data is thereby reduced by network workgroups having different views of network management data. However, the demerits of this idea are: (1) personnel may need retraining when job functions are changed and (2) multiple user interfaces to new products must also be developed.

The ideal network management user interface is one that is simple, easy-to-use, understandable, implementable, and extensible. The casual or untrained user must be able to use this interface without referencing numerous manuals to determine the syntax for a particular query. However, the professional or experienced user should be able to customize the interface to suit their personal preferences.

³¹Ben-Artzi, Amatzia, "Architecture for a Multi-Vendor Network Management System," Integrated Network Management, I, B. Meandzija and J. Westcott, Ed., Elsevier Science Publishers, North Holland, Amsterdam, (1989), pp. 445-453.

Another problem associated with integrated network management addressed by Mr. Ben-Artzi is the lack of a solid definition for management information (*e.g.*, required and optional parameters for data-link bridges) stored in the MIB³². Without a solid definition of the network management information contained in the MIB, users will be faced with resolving object definition ambiguities and object non-existence as they attempt to incorporate multi-vendor equipment into their environment. This problem is addressed further in subsequent sections of this paper.

2.5. Other Non-traditional Uses of SQL

SQL has been utilized in other non-traditional applications environments. For example, Akio Kato *et al* have developed an Organized Bit-map Assist Query (OBAQ) which addresses object-oriented relational database retrieval using visual information.³³ OBAQ is an application level program utilizing a high-level database manipulation function through interactive processing. In another similar work, Bogdan Czejdo *et al* developed a graphical query interface for databases

³²McCloghrie, K. and Rose, M., "Management Information Base for network management of TCP/IP-based internets", Request for Comments 1066, DDN Network Information Center, SRI International, Menlo Park, CA, (August, 1988).

³³Kato, A., Togawa M., Suzuki, K., and Oda K., "A Visual Interface for an Object-Oriented Query in Relational Databases," Systems and Computers in Japan, Vol. 19 No. 11, (1988), pp. 62-64.

which utilizes SQL as the underlying query language.³⁴ A hybrid form of SQL called QUEL³⁵ was used in the development of an Engineering Query Language.³⁶ The Engineering Query Language was developed to provide a SQL-type of user interface to technical engineers. In accomplishing this task, the developers of the Engineering Query Language translated SQL verbs (*e.g.*, SELECT, INSERT, *etc.*) into verbs that had more intrinsic meaning to their users.

³⁴Czejdo, B., Reddy, V., and Rusinkiewicz, M., "Design and Implementation of an Interactive Graphical Query Interface for a Relational Database Management System," IEEE Computer Society - Workshop on Visual Languages, (1988), pp. 14-19.

³⁵Stonebraker, M., Wong, E., and Kreps, P., "The Design and Implementation of INGRES", ACM Transactions on Database Systems (3), (1976), pp. 189-122.

³⁶Howard, H. C. and Howard, C. S., "User Interfaces for Structural Engineering Relational Data Base", Engineering with Computers (4), (1988), pp. 239-249.

Chapter 3

Implementation Methods and Procedures

3.1. Mapping SQL Functions to Network Management Applications

For the NMQL to be implementable, some form of mapping between SQL and network management operations must exist. This mapping should allow the managing process to retrieve/insert values from/to the specified agent.

One method of achieving this mapping is to redefine some of the standard SQL functions so that they perform dual role functions (retrieval of historical information from the database and retrieval of realtime information from the agent). The actual function performed is dependent on the format of the entered command. Also in order to perform certain network management functions, the SQL command language might have to be extended to allow the addition of a select set of network management-related verbs.

3.1.1. Mapping of Accepted SQL Datatypes to MIB-based Variables

Table I shows an example of how certain SQL verbs can be utilized in both realtime and historical environments to accomplish network management functions (*i.e.*, domain management, retrieval of management information, restricting access to management information, *etc.*). The elements of the table are the SQL verb, the function of that verb in the SQL environment without NMQL extensions, and the function of the verbs when NMQL-specific extensions are added. The network

Table I - Mapping of SQL Verbs & Commands to NMQL

SQL Verb	w/o NMQL Extensions	with NMQL Extensions
Select	retrieves historical information from the currently selected table or view as defined in SQL.	retrieves realtime information from physical device and updates the historical table.
Insert	places a new row in the historical table for the device as defined in SQL.	creates new variables for the designated physical device and updates the historical table.
Update	replaces an existing row in the historical table for the device.	modifies the existing values of the specified parameters and updates the historical database.
Delete	removes an existing row(s) from the historical table for the current device.	removes a variable(s) from the designated physical device.
Declare Schema	declares the currently active network management domain.	N/A
Declare Table	declares the currently active device for which historical management information can be obtained and realtime updates of the MIB variables will be allowed.	N/A
Rollback	Same as defined in standard SQL.	N/A
Close	Same as defined in standard SQL.	N/A
Join	Same as defined in standard SQL.	N/A

management functions of GET and SET can be accomplished by using the NMQL statements Select and Insert, respectively.

3.1.2. Security of the Historical Information

Standard SQL implements database security by using a combination of two methodologies.³⁷ The first methodology is to restrict, explicitly and selectively, access to the database by using the GRANT and REVOKE commands to control user access. The second methodology is to segment the table into many views and allow users to access only those views necessary for them to complete their job.

3.1.2.1. Grant and Revoke Commands

In standard SQL, the owner of the particular table or view and the database administrator are the only users that can issue GRANT and/or REVOKE commands. The ISO/ANSI standard SQL syntax for the GRANT and the REVOKE commands is:

```
GRANT {privilege_list | ALL}
ON {table_name [(column_list)] | view_name [(column_list)]}
TO {user_list | PUBLIC}
```

```
REVOKE {privilege_list | ALL}
ON {table_name [(column_list)] | view_name [(column_list)]}
FROM {user_list | PUBLIC}
```

³⁷Emerson, Sally, et al, The Practical SQL Handbook, Addison-Wesley Publishing Co., Reading, Mass., (April 1989) p. 297.

When the ALL qualifier is used in place of a list of privileges, all SQL privileges are either granted or revoked for that user, group of users, or all users (*i.e.*, PUBLIC).

An implementation of the GRANT privilege command is shown below in Example D. This command gives the users Smith and Jones permission to update (change) already existing values and to insert new values for any field (column) in the *LANBridge100* table.

GRANT insert, update ON LANBRIDGE100 TO Smith, Jones

Example D

An implementation of the REVOKE command shown below in Example E would remove Smith's privilege to update the *bridge_address* field of the *LANBRIDGE100* table only. The user Smith would still be allowed to update any remaining columns in the *LANBRIDGE100* table.

REVOKE update ON LANBRIDGE100(bridge_address) FROM Smith

Example E

One potential problem that may occur when using GRANT and REVOKE is that sometimes the ordering of commands may cause conflicting results. For example, the sequence of GRANT and REVOKE commands listed below in Example F leave the user Smith (who is a part of PUBLIC) without any select permission on the table *LANBRIDGE100*:

*GRANT select,insert,update ON LANBRIDGE100(bridge_address,trans_frms)
TO smith*

REVOKE select ON LANBRIDGE100 FROM PUBLIC

Example F

In contrast, the same statements in Example F issued in reverse order would accomplish the desired result of granting select, insert and update operations on two columns of the *LANBRIDGE100* table and revoking the select operation from everyone else. The occurrence of conflicting results when utilizing GRANT and REVOKE is mentioned here due to its relevancy if SQL is adopted as an NMQL.

3.1.2.2. Views as a Security Mechanism

In standard SQL and general relational database technology, views are used to simplify rather complex queries.³⁸ Views are virtual tables that do not exist as "real and independent" entities in the database. The elegance of views is their transparency to the end-user. The database performs all necessary transactions required to construct the view from the physical table. When utilizing views in database applications, the only possible evidence noticed by end-users is a possible degradation in the speed of the data retrieval.

Views are used in business applications (1) to focus the casual, possibly naive, end-user on the small subset of data that they really need; (2) to simplify the number of steps an end-user must take to accomplish their everyday tasks; and (3)

³⁸Emerson, Sally, Ibid, pp 270-1.

to customize the method of presenting the data contained in the database to the end user.³⁹ Although the methods of utilizing views mentioned above can also be realized in the network management applications area, views can be used to provide an increase in the security of the data contained in the database.

Through a view, users can query and modify only the data the created views present to them. For example, certain network users/operators may need to see only the columns referring to the packets transmitted and received on a single line in the LANBRIDGE100 table. Such a view is created by issuing the following SQL (NMQL) statements:

```
CREATE VIEW LANBRIDGE_LINE1 (Bridge, Trans_frames, Rcvd_frames)
  AS SELECT LANBRIDGE100.Bridge, LANBRIDGE100.Trans_frames,
  LANBRIDGE100.Rcvd_frames FROM LANBRIDGE100
```

A created view containing this information only meets the needs of its end-users and does not expose the contents of the entire LANBRIDGE table to them.

Views, when used in cooperation with the GRANT and REVOKE commands, provide a layered means of controlling user access to the database from two distinct levels. Created views allow the database to be divided into sub-tables containing information that is relevant to particular groups of personnel. The GRANT and REVOKE commands can then be used to limit the types of privileges given to the individuals within that respective group.

³⁹Ibid, p. 290.

3.2. Design of the NMQL

The ISO/OSI model for networks exemplifies a multi-vendor, distributed computing environment that will provide a certain hardware transparency to the end-user. The CMIS/CMIP specification for network management is one example among many (*e.g.*, File Transfer, Access, and Management⁴⁰ (FTAM), Message Handling Service⁴¹ (MHS), *etc.*) of peer-to-peer processing in a distributed computing environment.

With CMIS/CMIP, network functions are performed by one or more manager processes communicating with one or more agent processes (See Figure 3). The managing process receives the request for information from the user, formats the request into a CMIP compliant packet, sends the request to the specific agent(s) and waits for a reply. Each specified agent process receives the request from the manager, decodes the request, collects the necessary information from the managed objects (Note: Managed objects are an abstraction of the actual physical entities), and returns the information to the requesting manager. This communication between manager and agent processes can occur in a variety of forms (*e.g.*, on the same physical machine, within the same local area network (LAN), within the

⁴⁰Information Processing Systems - Open Systems Interconnection, "File Transfer, Access, and Management - Part 1: General Introduction", International Standard 8571-1.

⁴¹CCITT Recommendation X.400, (Red Book, 1984), "Message Handling Systems: System Model-Service Elements.

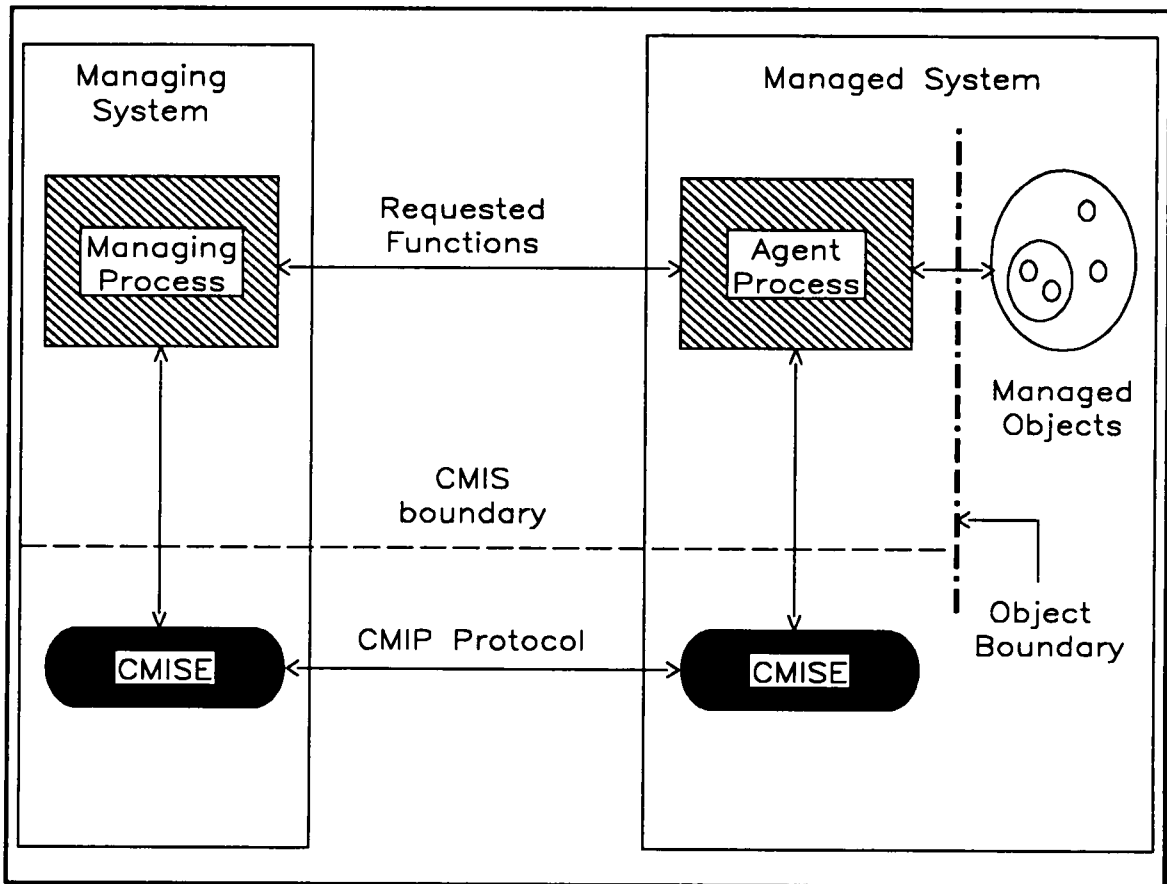


Figure 3 - Detailed Description of Network Management in the ISO Environment

same Wide Area Network (WAN), *etc.*) and is dependent on the managing process' jurisdiction within the network.

Therefore, the NMQL must also be designed to support a widely-distributed network management environment. On each *managing* node, a centralized database will be maintained to provide historical information about all agents and their objects that are under the reign of this manager. (Note: Depending on the management strategies deployed within a given company, the number of managing nodes may vary. For example, if only one managing node exists within a given

organization but that organization employs multiple network managers, the methods of securing the historical information mentioned in section 3.1.2 become more important.) On each *agent* within the network, a database will be established to keep only the current information relevant to that particular agent. The NMQL must perform the following functions: (1) accept NMQL commands from network managers; (2) interpret these NMQL commands into specific function invocations understood by the specific agent(s); (3) organize the results returned by the agent(s); (4) transpose the results into a relational form; (5) insert these results into the database; and (6) present the results to the manager. Anytime a manager changes the current values on any given agent, the historical database must be updated not only on the node from which the request was generated but also on every managing node that considers that particular agent as being "under its control."

3.2.1. Management Domains for an Enterprise

In large installations where the existence of multi-vendor equipment and multiple types of networks (*e.g.*, Ethernet, token ring, token bus, X.25, *etc.*) are more prevalent, there exists the potential for multiple network managers and the establishment of multiple network management policies. An enterprise can establish network management domains to partition its networks into logical groups

and then develop a common management policy for each group.⁴² For clarity, the following terminology⁴³ may be used when referring to domain management:

Domain - A set of objects (e.g., Hayes modems, Ungermann-Bass bridges, DEC terminal servers, etc.) to which a particular management policy applies. (Note: A network manager is also an object and set of network managers is a domain.),

Domain Type - Specifies the generic types(s) of object(s) in the set and the operations that can be performed on the instances of these objects,

Domain Instance - One occurrence of the defined object set, (Note: There may exist many objects of different or the same types in different domains, but referring to the same physical entity), and

Management Access Rule (MAR) - Defines a 5-tuple relation between a domain of network managers, a domain of managed objects, and a subset of management operations which a member of the management domain can perform on the objects.

To illustrate how domain management might be implemented, a small enterprise network named COMPNET will be created (See Figure 4). COMPNET is the Blakely Enterprises's COMPAny NETwork connecting all computing facilities of the company together. COMPNET consists of three LANs (LANA, LANB, and LANC) which network the computing facilities in building A, B, and C, respectively with a fiber-optic backbone to connect the building LANs to each other. Each building has a network manager for each of the respective LANs, and

⁴²Sloman, Morris and Moffett, Jonathan, "Domain Management for Distributed Systems", Integrated Network Management, I, B. Meandzija and J. Westcott, Ed., Elsevier Science Publishers, North Holland, Amsterdam, (1989), pp. 509-516.

⁴³Ibid.

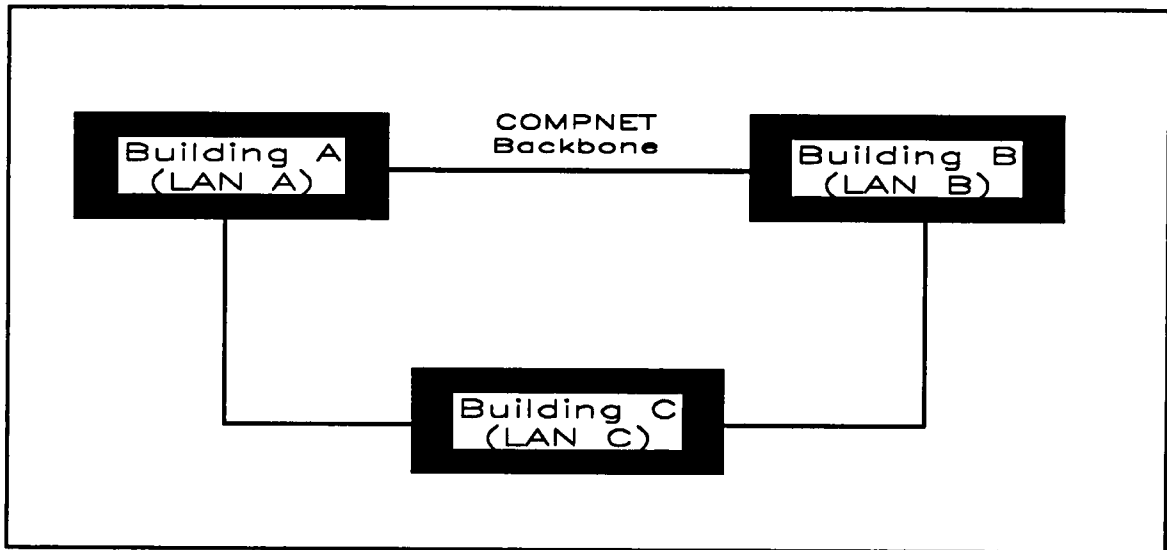


Figure 4 - Diagram of the COMPNET Enterprise Network

the COMPNET fiber-optic backbone also has its own network manager. The LANs of each building are connected to the backbone via a bridge. Figure 4 shows a diagram of COMPNET. Minimal corporate policies exist for network management of Blakely Enterprises networks and management is, for the most part, left up to the respective managers. Each manager has all privileges necessary to perform any functions on any device throughout COMPNET.

By establishing a domain management policy and grouping the objects of the network into common domains, network management of the Blakely Enterprise's networks can be made more organized by defining a management hierarchy, and the job functions of the network management personnel can become more defined through mapping their job function to a position in the management hierarchy. Figure 5 shows how such a reorganization utilizing the concept of domain management might look for Blakely Enterprises. Figure 5 shows that a domain for

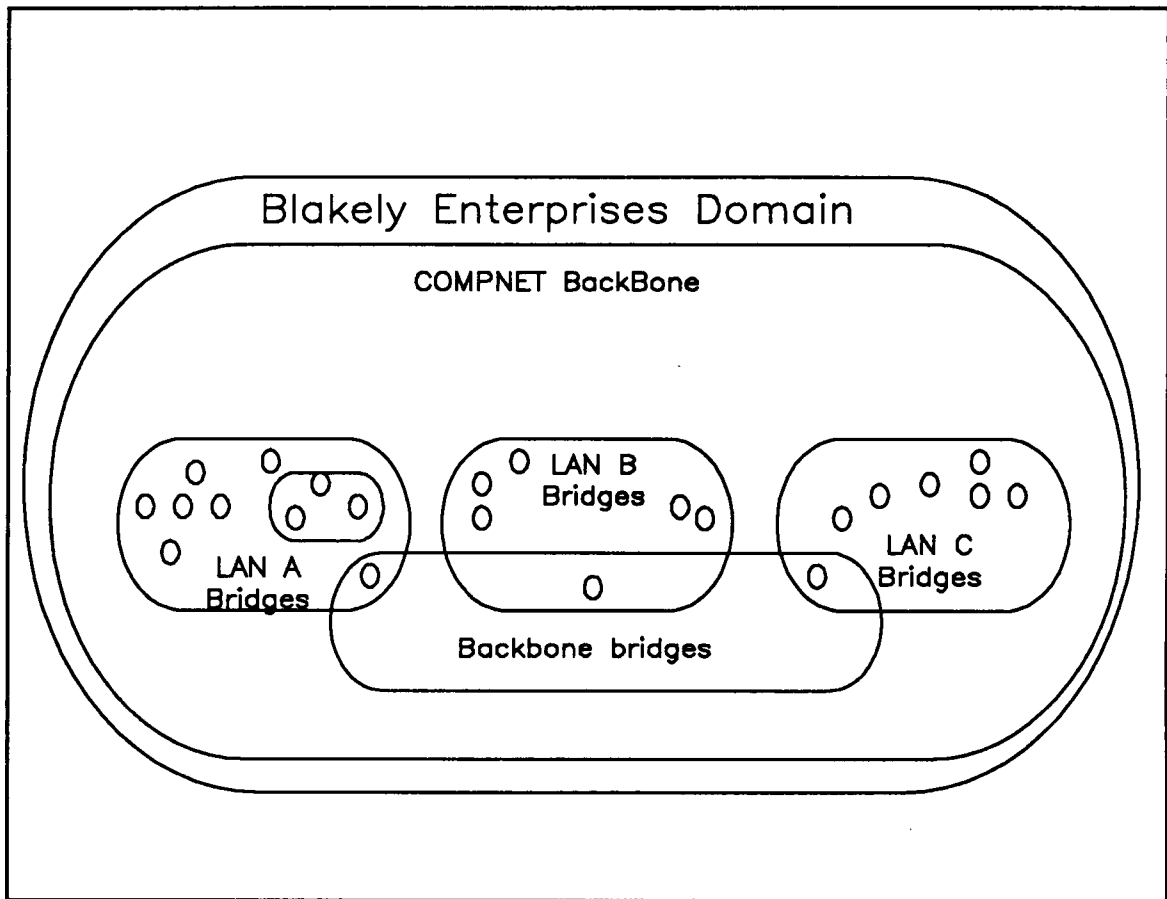


Figure 5 - Domain Partitioning of Blakely Enterprises Network
the entire enterprise is globally encompassing all other domains. This is done so that any other networks installed in the future will still remain within the enterprise, but distinct from the COMPNET domain.

Also note that the Backbone Bridges domain overlaps each of the LAN Bridges domains. This is to show that, in some cases, different domains may require the same or different types of MARs to the same type of object. This type of overlapping is termed "explicit overlapping" since one or more objects exist in the

object sets of multiple domains of the same type.⁴⁴ In this instance, the network manager sets of each of the LAN Bridges domains are given permission to read the statistics from the bridge that connects their respective LAN to the backbone, but only the manager set of the Backbone Bridges domain is allowed to modify any information contained in any of the bridges.

Domain management provides a means of segregating the networking components of an enterprise into logical, manageable sets and also provides a means of delegating the types of management authority given to each network manager and/or network management personnel. The historical information of the NMQL is designed to support domain-based network management by utilizing database tables and views as logical network management domains and by physically separating the domains with schemas. The MARs for each domain (view) are created by utilizing the GRANT and REVOKE commands described previously.

3.3. Environment of the NMQL Prototype and Data Collection Methods

The NMQL Prototype environment consists of a VAXStation 3100[™] utilizing the VAX/VMS[™] version 5.3 operating system. In this prototype, the VAXStation 3100[™] functions as both the NMQL Manager and the NMQL Agent.

⁴⁴Sloman, M. and Moffett, J., "Domain Management for Distributed Systems", Integrated Network Management, I, B. Meandzija and J. Westcott, Ed., Elsevier Science Publishing Co. Inc., Amsterdam, The Netherlands, (May 1989), pp. 510-11.

3.3.1. Creation of the NMQL Historical Database

The NMQL Prototype is built to allow archiving of historical management data by establishing a SQL/RDB database. The software that creates this historical database is named the "Bridge_Schema_Create.SC" program and is included as Appendix A.

The NMQL Historical Database is a relational database using VAX SQL as its query language. The historical database contains all managed object variables in a table. Tables II and III show the column name, length, and definition for each attribute of the managed object LANBridge. The NMQL Historical Database is implemented as a time-versioned relational database. The date and time are used as the primary keys for indexing the contents of the relation. The MIB address field is utilized as a secondary key. As more managed objects that support the NMQL are created, new relations containing the variables of the managed object are also created.

3.3.2. The NMQL Manager

The NMQL manager process consists of a top-level module written in the VAX C language and contains several SQL-based calls which utilize a hybrid version of SQL as the user interface. The database used in this research is the VAX Rdb™ version 3.0B. The query language used in this research is VAX SQL™ version 2.0.

Initially, dynamic SQL was chosen over the more traditional embedded SQL because it allows the user the freedom of entering SQL statements at run time on

Table II - Specification for Managed Object LANBridge

Field Name	Length	Definition
Bridge_Id	17 bytes	The character string containing the name of the bridge.
Bridge_Addr	16 bytes	The ethernet hardware address of this particular instance of a bridge.
Bin_Date	8 bytes	The 64-bit date of the query.
Bridge_Sec	4 bytes	The number of seconds since the line's counters were last reset.
Port_Restr	4 bytes	The number of times the port has been restarted.
Unkn_Dest_Addr	4 bytes	The number of frames received with an unknown destination address.
Frms_Filt	4 bytes	The total number of frames filtered.
Mult_Frms_Filt	4 bytes	The number of multicast frames filtered.
Frms_Addr_Filt	4 bytes	The number of frames filtered by address.
Frms_Proto_Filt	4 bytes	The number of frames filtered by protocol type.
Tot_Frms_Sent	4 bytes	The total frames sent by a given port.
Tot_Frms_Recv	4 bytes	The total frames received by a given port.
Tot_Bytes_Sent	4 bytes	The total bytes sent by a given port.
Tot_Bytes_Recv	4 bytes	The total bytes received by a given port.
Dev_Frms_Sent	4 bytes	The total frames sent by a given port.
Dev_Frms_Recv	4 bytes	The total frames received by a given port.
Dev_Bytes_Sent	4 bytes	The total bytes sent by a given port.
Dev_Bytes_Recv	4 bytes	The total bytes received by a given port.
Mult_Frms_Sent	4 bytes	The total multicast frames sent by a given port.
Mult_Frms_Recv	4 bytes	The total multicast frames received by a given port.
Mult_Bytes_Sent	4 bytes	The total multicast bytes sent by a given port.
Mult_Bytes_Recv	4 bytes	The total multicast bytes received by a given port.
Mult_Dev_Frms_Sent	4 bytes	The total multicast device frames sent by a given port.
Mult_Dev_Frms_Recv	4 bytes	The total multicast device frames received by a given port.
Mult_Dev_Bytes_Sent	4 bytes	The total multicast device bytes sent by a given port.

Table III - Specification for Managed Object LANBridge (cont.)

Field Name	Length	Definition
Mult_Dev_Bytes_Recv	4 bytes	The total multicast device bytes received by a given port.
Inv_Dev_Msg_Cnt	4 bytes	The count of invalid messages received.
Bad_Frms_Recv	4 bytes	The count of bad frames received.
Bad_Bytes_Recv	4 bytes	The total (in bytes) of the bad frames.
Xmt_Frms_Lost	4 bytes	A count of the lost transmitted frames.
Rcv_Frms_Lost	4 bytes	A count of the lost received frames.
Bad_Hello_Lim_Exc	4 bytes	The total of the number of times the bad hello limit was exceeded.
Collis_Tst_Fail	4 bytes	The number of times the collision test failed.
Xmt_Frms_Err	4 bytes	The number of times frame transmission resulted in an error.
Mult_Src_Frms_Rcv	4 bytes	The number of multicast source frames received.
Liftn_Exc_Frms	4 bytes	The number of Lifetime exceeded frames.
Liftn_Exc_Bytes	4 bytes	The number of bytes where the lifetime has been exceeded.
Rcv_overflow	4 bytes	The number of times the receive port has been flooded.
Oversize_Frm	4 bytes	The number of packets with width greater than threshold (1513 bytes).
Carrier_loss	4 bytes	The number of times the Ethernet carrier has been lost.
Lrnd_Addr	16 bytes	The address of a particular host or device that has recently transmitted packets.
On_Line	4 bytes	The line on which the bridge learned the address.

the command line of the process. dynamic SQL permits the application to be more flexible and more expandable. In contrast, embedded SQL requires that the user be queried for values that are plugged into already embedded SQL statements. The freedom of dynamic SQL, however, does not come inexpensively. Utilization of

Dynamic SQL in applications imposes substantial penalties on the application's performance and increases program complexity.⁴⁵ Unlike embedded SQL, dynamically-executed SQL statements are not a part of the program's source code. Dynamically-executed SQL statements are entered by the user on the command line and become part of the executing process only during run-time (*i.e.*, the command line is interpreted).

Only a subset of SQL statements can be executed dynamically.⁴⁶ Table IV lists the dynamically executable SQL statements and compares them to associated embedded SQL statements. Input parameters are either host variables or pointers to parameter markers in prepared SQL statements. Output parameters are either host variables or dynamic memory which are used to receive values of rows generated by SQL in response to a command or query. Parameter markers are substituted with either host variables or variables contained in dynamic memory when the statement is dynamically executed.

To utilize dynamic SQL, host languages must provide support for pointer variables that provide indirect access to memory by storing the address of data in addition to storing data directly in the variables. As of 1988, the only Digital

⁴⁵VAX SQL User's Guide, Digital Equipment Corporation, Order #: AA-JM33B-TE, Maynard, MA, (July 1988), p. 15-2.

⁴⁶VAX SQL An Overview, Digital Equipment Corporation, Maynard, MA, (1987), p. 11-5.

Table IV - List of Dynamic SQL Statements

Statement	Input Parameters	Output Parameters	Associated Embedded Statements
SELECT	Allowed	Required	PREPARE DECLARE CURSOR DESCRIBE (optional) OPEN FETCH
INSERT UPDATE DELETE	Allowed	N/A	PREPARE DESCRIBE (optional) EXECUTE EXECUTE IMMEDIATE (if no parameter markers)
CREATE ALTER TABLE DROP	Not Allowed	N/A	PREPARE EXECUTE EXECUTE IMMEDIATE
DECLARE DATABASE DECLARE TRANSACTION COMMIT ROLLBACK			
GRANT REVOKE			

supported VAX host languages that support dynamic SQL are C, PL/I, and Ada.⁴⁷

After subsequent research, the NMQL Manager was re-written to remove the dynamic SQL portions because the functions of the manager were re-defined to include: (1) installing the agent process, (2) parsing of the user's command, and

⁴⁷VAX SQL Reference Manual, Digital Equipment Corporation, Order No. AA-JM32B-TE, (July 1988), p. 4-158.

(3) determining if the command line is standard SQL or if the command line contained any NMQL extensions. If the command line is standard SQL, an SQL sub-process is spawned, and the application is placed into this sub-process. If the command line contains NMQL extensions, the NMQL Agent task is spawned. The NMQL Manager consists of one program entitled "MNQL_Manager.SC" and is included as Appendix B.

3.3.3. The NMQL Agent

The NMQL Agent process consists of a VAX C language process containing embedded VAX SQL statements. The NMQL Agent is created in a hibernated state by the NMQL Manager process. A default "wake up" interval of 30 minutes is used if no interval is specified by the user. Upon "waking up", the agent queries its managed object (in this case, a LANBridge 150 or LANBridge 200), and, after receiving the information, the Agent process updates the historical database and returns the requested information to the manager process. Upon completion, the NMQL Agent hibernates until the next scheduled "wake up" or until forced to "wake up" by an NMQL Manager process.

The NMQL Agent interfaces with the physical device by spawning a sub-process Digital Command Language (DCL™) task with a pre-specified command line (Note: Remote Bridge Management System (RBMS) version 2.0 does not support a callable interface, which would have been the preferred method). The command line given to the sub-process causes the direct execution of RBMS and

also directs RBMS to dump the bridge contents to a specified file. When RBMS completes, the DCL sub-process exits, control is returned to the NMQL Agent, and the NMQL Agent opens the data file, parses the data file, and enters the data into the historical database using embedded SQL.

While certainly not a practical method of communicating with a device, it did provide for a more rapid prototyping of the NMQL Agent and performed basically the same functions. The NMQL Agent consists of one file entitled "NMQL_Agent.SC" and is included as Appendix C. VAX DCL command files needed to build the source code for all NMQL modules is included as Appendix D.

3.4. Implementation of the NMQL

The implementation of the NMQL consists of two main components which are somewhat similar to the ISO/OSI CMIP architecture for performing network management operations, the NMQL Manager and the NMQL Agent. The NMQL Manager process is responsible for: (1) the user interface, (2) parsing of the command line to determine if the NMQL query is requesting historical or realtime information, (3) communication with the required agent(s), (4) security, and (5) displaying the requested information to the user. If the query requests historical information, the NMQL Manager process is also responsible for retrieving the information from the agent's database. The NMQL Agent process is responsible for: (1) receiving requests from NMQL Managers, (2) parsing the request for

errors, (3) retrieving the requested information from its MIB, and (4) returning the requested information to the NMQL Manager(s). While in this prototype, both the manager and the agent process reside on the same physical node; this would most likely not be the case in general. In actual operation, a given enterprise may consist of multiple manager processes responsible for many agents distributed throughout the enterprise.

3.5. Analysis Procedures

In this study, the analysis of the work presented herein was tested on LANBridge 150 and LANBridge 200 data link bridges from Digital Equipment Corporation utilizing the RBMS version 2.0 software. The RBMS software was incorporated into the NMQL agent software to provide a simple means of communicating with the LANBridge 150 and 200 devices.

3.6. Question Testing

A series of five research questions were used to direct the investigation of the SQL as a network management query language. Usage of SQL as an end-user interface for network management was researched and the advantages and disadvantages listed.

3.6.1. Question 1: Operands of SQL

Question 1 investigates mapping the SQL operands to operands commonly utilized in network management applications: How can the operands of SQL be adapted to perform network management operations?

ANSI SQL provides a rich set of operands and also provides a robust means of combining a series of operands and operators together to form complex database queries. The end-user interface presented by ANSI SQL is a somewhat verbose type of user interface. Options to this type of interface were researched and the findings are presented herein.

3.6.2. Question 2: Operators of SQL

Question 2 investigates mapping the operators available within ANSI SQL to those required within network management: To what extent is it feasible to utilize the operators available within SQL to store management information?

SQL supports only the following datatypes: character, character varying, integer, floating-point, and date. SQL utilizes 64-bit binary fields only to store date-type fields in an encrypted form. For SQL to be used in network management applications, some method of storing (the sometimes lengthy) ISO standard MIB addresses and routing tables needs to be designed.

3.6.3. Question 3: Mapping of CMIP to SQL

Questions 3 probes the problem of the development of a generic language that can be used to describe the syntax and semantics of multiple languages: How can the syntax/semantics of the different network management applications be described in a generic manner?

The syntax and semantics of various network management languages (SNMP, CMIP, CMOT, *etc.*) were reviewed and a matrix of the commands and the generic

functions were grouped together to determine if a generic language could be developed. A matrix (Table V) was also created that listed the generic functions of several network management packages. This matrix revealed that network

Table V - Abstraction of Network Management Commands

Information Retrieval	Information Alteration	Unsolicited Response	Connection Receipt	Connection Establish	Release
Generic Get	Unconfirmed Set Confirmed Set Unconfirmed Action Confirmed Action Unconfirmed Creation Unconfirmed Deletion	Generic Response Linked Response	Unconfirmed Receipt Confirmed Receipt	Initialize	Normal Abort

management commands can be categorized into six distinct areas: retrieval of information, altering of information, response to a request, unsolicited receipt of information, establishment of a connection, and releasing of a connection. Table VI lists the generic classes as columns in the table and shows a mapping of the CMIP protocol to these columns. Items in the table below containing an "A-" prefix are actually contained in the services provided by the Application Common Service Element^{48,49} (ACSE).

⁴⁸Information Processing Systems - Open Systems Interconnection, "Service Definition for the Application Control Service Element", International Standard 8649, August 1988.

Table VI - Abstraction of CMIP Network Management Commands

Information Retrieval	Information Alteration	Unsolicited Response	Connection Receipt	Connection Establish	Release
M-GET	M-SET M-ACTION M-CREATE M-DELETE	M-EVENT_REPORT		A-ASSOCIATE	A-RELEASE A-ABORT

3.6.4. Question 4: Common Interface Characteristics

Question 4 probes whether a common interface between network management applications and multiple network management protocols can actually exist, and, if it does, what are the characteristics of such an interface.

Early network management packages provided little, if any, support for interfacing a network management application with a network management protocol. Vendors typically provided their customers with one or more vendor-developed packages that could be used to monitor and management the customer's network. More recently, however, vendors have responded to requests from customers for more flexibility, ease of use, increased functionality, somewhat limited interoperability, *etc.* by developing applications programmable interfaces (APIs) into their network management protocol(s). APIs permit the end-user to customize the network management applications to suit their preference without

⁴⁹Information Processing Systems - Open Systems Interconnection, "Protocol Specification for the Application Control Service Element", International Standard 8650, August 1988.

having to rewrite the entire protocol. For example, IBM has developed an API for its NetView/PCtm product that allows MS-DOS assembly language programs to interface to the NetView/PC library to perform special and/or custom network management tasks.⁵⁰ These tasks could permit management of a previously-isolated network management system into the NetView arena, CMIP arena, *etc.*

⁵⁰NetView/PC Application Program Interface/Communications Services, Version 1.1, Second Edition, IBM Corporation, Order #SC30-3313-1, (October 1987)

Chapter 4

Results and Findings

This chapter presents the results of the research into each of the questions proposed in the previous chapter.

4.1. Question 1: Operands of SQL

How can the operands of SQL be utilized to perform network management functions?

Discussion of the various styles of user interfaces has been presented in previous sections. The purpose of this section is to review each of the various command functions available with traditional SQL and determine if it is suitable or if some additional capabilities are required for it to be accepted as a network management query language. Mapping of the functions provided by SQL to both the abstract network management functions defined in Table V and the ISO Systems Management Functions is used to prove that SQL does provide most, if not all, of the operands necessary for network management applications.

SQL provides the necessary functions to perform the five of the six generic types of network management commands as outlined in Table VII. However, SQL is somewhat inadequate in its ability to perform some of the more specific abstract network management commands. For example, SQL does not provide any one

Table VII - Mapping between SQL, NM Abstractions, and Pass-thru Functions

SQL Functions	Abstract Functions	Pass-thru Functions
SELECT	Generic Get	PT-GET
UPDATE	Unconfirmed Set	PT-SET
UPDATE,SELECT	Confirmed Set	PT-SET
UPDATE	Unconfirmed Action	PT-ACTION
UPDATE,SELECT	Confirmed Action	PT-ACTION
DELETE	Unconfirmed Delete	PT-SET
ALTER		PT-DELETE
DROP		
INSERT	Unconfirmed Create	PT-SET
CREATE		PT-CREATE
DECLARE	Initialize	N/A (Done thru ACSE)
COMMIT	Release	N/A (Done thru ACSE)
ROLLBACK	Abort	N/A (Done thru ACSE)
	Unconfirmed Receipt	PT-EVENT
	Confirmed Receipt	PT-EVENT
	Generic Response	
	Linked Response	

function to perform the abstract function Confirmed_Set. This would have to be mapped utilizing two SQL functions -- UPDATE and SELECT. The generic category in which the current SQL standard is insufficient is the Response category. Table VII shows the mapping between the SQL functions, the abstract network management functions (*i.e.*, Table V), and the Pass-thru services provided by the Object Management Function. The Object Management Function⁵¹, ISO-

⁵¹Information Processing Systems - Open Systems Interconnection, "Systems Management Part 1: Object Management Function", Draft International Standard 10164-1, (May 1990), pp. 6-21.

10164-1, defines what are called "Pass-thru" functions. Pass-thru functions provide a direct mapping onto the services provided by CMIS. SMFs that do not define mapping functions to CMIS unique to that SMF can utilize the pass-thru functions provided by Object Management.

Some of the abstract network management commands listed in Tables V and VI do not map to the Pass-thru services defined in the Object Management Function. Namely, the connection Initialization, Normal_release, and Abort services are performed in ISO by the Association Control Service Element (ACSE). However, certain SQL functions can be mapped to those same services provided by ACSE. The DECLARE SCHEMA statement could be mapped to connection initialization. The DROP SCHEMA in conjunction with COMMIT and ROLLBACK could perform the Normal_release and Abort services.

SQL as it is currently defined, however, is deficient in providing the mechanisms necessary to perform event notification functions (*i.e.*, Unsolicited Receipt in Table VII), replies, and linked replies from perspective network agents. Event notifications are, for the most part, unsolicited on the part of the manager, and SQL provides no means for receipt and notification of unsolicited information. Event Notifications are usually sent by an Agent to alert the Manager about some change in operation or significant event. Event notifications are a very significant part of network management and are crucial to proper management of a network.

Lack of an event notification mechanism is perhaps the most serious defect in the utilization of SQL as a network management query language.

Therefore, some mechanism (perhaps new operands) must be created for the receipt of unsolicited information from external resources. It is proposed that an Event Notification Table be created and that a daemon process monitor access to this table. When an agent sends an event to the manager by performing an SQL INSERT on the Event Notification Table, the notification is placed in the table and the manager (person or process) is alerted by the daemon.

Also not addressed in standard SQL is the ability to perform operations on the physical attributes of a given object at a specified time. SQL performs as it was originally designed and is primarily concerned with the capturing of historical information. In network management, SQL must also be capable of adding, modifying, or removing the values of any working attributes on any given device in the network. Two solutions to this restriction exist. First, an augmented version of the SQL standard can be appended to include new command syntaxes (operands) that execute against "live" attributes of an object in addition to those that address the historical attributes of the object. Second, existing SQL can be utilized by developing standard naming conventions for attributes, tables and schemas in both historical and "live" categories. Augmentation of the existing SQL operands is the more long-term solution, while the development of standard naming conventions is the more short-term solution.

The utilization of embedded SQL over dynamic SQL was chosen because of the excessive memory requirements dynamic SQL might have on smaller, less sophisticated platforms (such as single-tasking PC's).

SQL does provide additional features in the area of security that can provide benefit to the network management community. By utilizing the GRANT and REVOKE commands, in addition to establishing views (as described in Chapter 3), dissemination of network management information can be restricted and controlled. However, since SQL utilized in the network management applications role would function on a network, use of devices such as sniffers and protocol analyzers would have to be carefully controlled to ensure the integrity of the controls in place on the network devices.

Table VII shows that SQL does provide the command syntax necessary for performing most abstract network management functions. SQL presents a more English-style of interface to the end-user than most typical network management applications. SQL provides features for minimal security of management information while resident on the devices, but does not provide any security while the information is in transit. With some augmentation, the operands of SQL can be utilized as a network management query language.

4.2. Question 2: Operators of SQL

How can the operators of SQL be adapted to perform network management operations?

The standard ANSI SQL supports only the following datatypes: character, character varying, integer, floating-point, and date. Network management applications require the ability to store information that does not conform to the current datatypes supported by SQL. For SQL to be accepted as a query language for network management, either some form of adaptation must be performed to make the sometimes lengthy variables used in current and future applications conform to the datatypes currently supported by SQL, or the SQL standard must be amended to include support for these datatypes.

This research attempts to utilize the existing datatypes supported by SQL by converting the 6 octet (*e.g.*, 08-00-2B-AB-CD-EF) Ethernet addresses stored by the LANBridge 100 into some form that SQL recognizes; however, augmenting the ANSI standard SQL to support additional datatypes is highly recommended. Three methods exist for accomplishing this:

- * create two integer fields to contain the Ethernet address,
- * find a mechanism to utilize the existing 64-bit date field to store the Ethernet address, and
- * convert the entire string to character.

The first method appears to be very promising. By utilizing two integer fields (*e.g.*, High-bits and Low-bits), up to 8 octet addresses can be stored as integers within SQL. For example, the Ethernet address 01-02-03-AB-CD-EF would be split into two sections with the hexadecimal digits 0102 occupying the first section (*i.e.*, High-bits) and the hexadecimal digits 03ABCDEF occupying the second (*i.e.*,

Low-bits). Given the Ethernet address above, the High-bits field would contain the integer 258 and the Low-bits field would contain the integer 61,591,023. While this method permits the Ethernet address to be stored in a minimal amount of bits, its major disadvantage is that it prohibits the direct querying of the database using standard on-line SQL (*i.e.*, some form of translation must be performed on the two integer fields before the address can be presented to the user).

The second method of mapping Ethernet addresses to existing SQL datatypes by utilizing the 64-bit encryption algorithm for storing dates also appeared to be very promising. However, no method of utilizing this algorithm has been found and is left for further study.

The third method of storing the addresses as a series of characters may seem to be the least preferred method of the three, since this method is wasteful of disk storage capacity and consumes more primary memory when processing queries. However, it is the only method that provides the advantage of directly accessing the database through on-line SQL and retrieving information without having to translate integer or encrypted date fields into Ethernet addresses.

Utilizing the current SQL standard specification and datatypes, storage of the Ethernet addresses as character strings is the better choice over the other two methods because of the increased flexibility provided. By storing the Ethernet addresses as character strings, the network manager is able to use most of the facilities available with on-line SQL. Neither of the other two methods would

allow for that capability. Either of the other two methods would require the network manager or operator to constantly access a translation application on top of the existing SQL process to retrieve any meaningful information. The choice for storage methods comes down to either using disk space for storing the addresses as characters or using processor time converting the addresses to a presentable form to the user.

One negative aspect concerning the utilization of SQL operators for storing network management data is the inability to define any SQL data structures (a set of operators) that are non-scalar. Non-scalar references are not allowed in the SQL standard due to the inability of the relational database underlying SQL to resolve references. Although this problem is not as major as those presented with the SQL operands, it does effect the definition of the database by causing all table items to be scalar.

4.3. Question 3: Mapping of CMIP to SQL

How will the syntax/semantics of the different network management applications be described in a generic manner? To answer this question, the various syntax/semantics of different network management packages must be examined and reviewed.

4.3.1. Multiple Syntaxes

As stated earlier, network management operations can be categorized into six abstract areas: (1) retrieval of information, (2) altering of information, (3) response

to a request, (4) unsolicited receipt of information, (5) establishment of connection, and (6) releasing of a connection. As part of the early research into the development of a common API for network management applications, the network management protocols CMIP, Digital Equipment Corporation's Network Control Program⁵² (NCP), International Business Machines' Systems Network Architecture⁵³ (SNA) and SNMP were modelled using the six abstract areas presented above. Table V on page 49 details each of these categories. Within each of the six abstract generic categories, the more specific command(s) that relate to the generic category are listed.

Given this abstraction, this research attempts to map individual SQL commands to the abstract network management commands listed in Table VII. Figure 6 shows a layered approach to performing network management applications. In Figure 6, SQL is layered on top of a network management API (NMAPI). The NMAPI provides interfaces to the System Management Functions (SMFs) as defined in the Systems Management Overview, ISO-10040.⁵⁴ The SMFs are then

⁵²Digital Equipment Corporation, "Network Control Program", Digital Press,

⁵³Cole, Leo J., "Network Management as Described in Systems Network Architecture", Proc. IEEE Infocom-86, Miami, FL (April 1986), pp.364-376.

⁵⁴Information Processing Systems - Open Systems Interconnection, "Systems Management Overview", Draft Proposal 10040, (June 16, 1990), pp. 19-20.

layered on top of the ISO/OSI Common Management Information Service⁵⁵ (CMIS). CMIS communicates with the Association Control Service Elements^{56,57} (ACSE) for establishment, release, or abortion of a session with a peer entity. For SQL to provide the functionality necessary to serve as a network management query language there must exist a one-to-many-to-one mapping between SQL, the SMFs, CMIS, and the services provided by ACSE.

Tools exist already (*i.e.*, are already defined within each SMF) for the many-to-one mapping between the SMFs and the services provided by CMIS. As discussed in Section 4.1.2, the Object Management Function provides a direct mapping onto the services provided by CMIS. What must be defined, therefore, is a one-to-many mapping between the SQL statements and the SMFs.

4.3.2. Semantics

Although this research did not address the semantics of the common user interface, the text presented here is an attempt to familiarize the reader with current developments in this field and to, hopefully, promote further research in this area.

⁵⁵Information Processing Systems - Open Systems Interconnection, "Management Information Service Definition - Part 2: Common Management Information Service", Ibid.

⁵⁶Information Processing Organization - Open Systems Interconnection, "Service Definition for the Association Control Service Element", International Standard 8649, August 1988.

⁵⁷Information Processing Systems - Open Systems Interconnection, "Protocol Specification for the Association Control Service Element, International Standard 8650, August 1988.

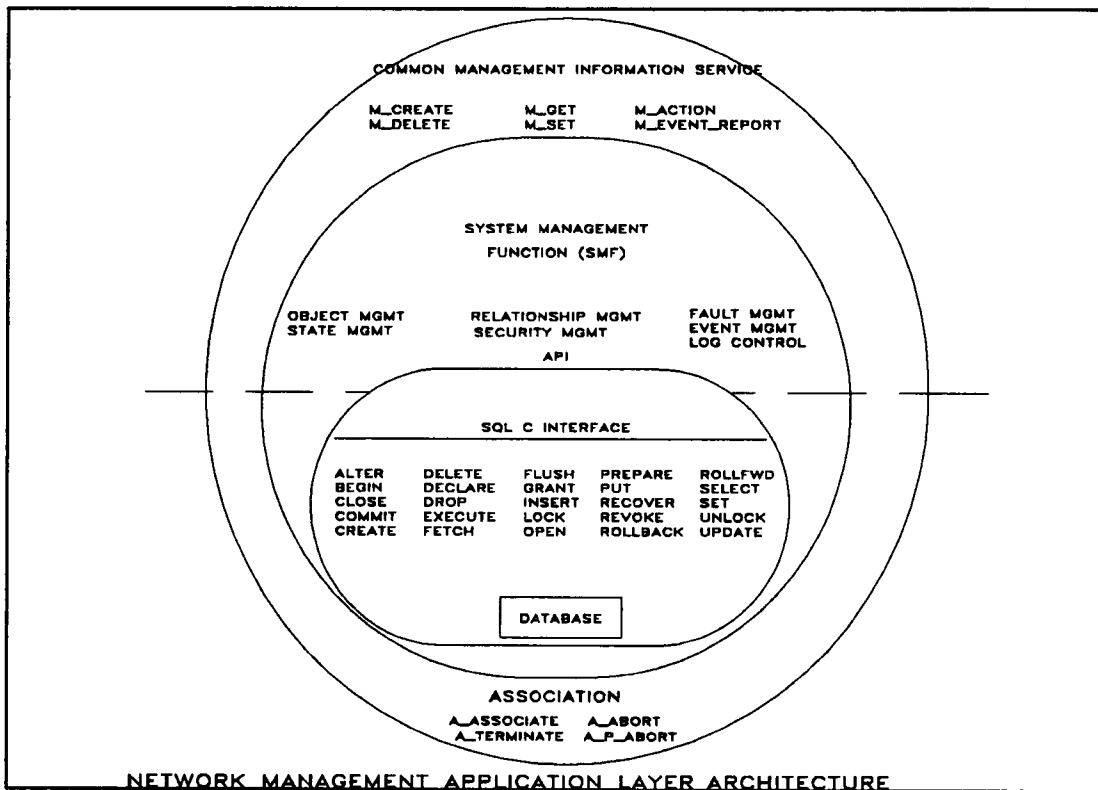


Figure 6 - An SQL API for CMIP

Much research has been accomplished over the past several years concerning how information is presented to a user. In 1986, Foote-Lennox⁵⁸ presented some guidelines concerning prompts, acknowledgements, and the psychology of the user; and he proposed the adoption of these guidelines as an IEEE standard. However in a relatively short time span, significant progress has been made in the incorporation of new technologies into user interfaces. The arrival of windows, icons, mice, and WYSIWYG (what you see is what you get) technologies and the increase in hardware performance required to support these technologies has

⁵⁸Foote-Lennox, Thom, "Ergonomic Guidelines for Computerized User Interfaces", IEEE Proceedings Computer Standards Conference 1986, May 13-15, 1986, San Francisco, CA, pp. 38-41.

allowed the user to have direct manipulation of the objects displayed on the screen (*i.e.*, over the user's mental image of the real system). Despite the advances being made in the intelligence of network management applications (*e.g.*, the use of expert systems⁵⁹, multimedia⁶⁰, and environmental ergonomics⁶¹) and automation of network operations, ultimate decision-making remains with the human network manager or operator.

4.4. Question 4: Common Interface Characteristics

What are the characteristics needed for a common user interface for network management applications? The answer to that question depends generally upon the characteristics of the person using the application. Some individuals prefer the command line style of interface while others prefer mouse-driven, pull-down, pop-up menus with windows spread across the terminal screen. Generally, however, as the performance capacity of the workstation presented to the end-user continues to increase, the trend toward the more graphical style of user interface will also continue to increase. This is evident with the development of X.11 Window

⁵⁹Wang, K., Brady, S., and Gilbert, R., "Expert Systems in Network Management and Control", Expert Systems in Government Symposium, IEEE, 1986, pp. 290-295.

⁶⁰Ganesan, K. and Ganti, M., "A Multimedia Front-End for an Expert Network Management System", IEEE Journal on Selected Areas in Communications, Vol. 6 No 5, June 1988, pp. 788-791.

⁶¹Fletcher, J. and Taylor, R., "Human Factors Engineering in Network Management", British Telecommunications Technology Journal, Vol 6 No 4, (October 1988), pp 44-51.

System⁶² at MIT. As stated previously, the X.11 System was developed to provide identical user interfaces independent of the hardware.⁶³

In an attempt to answer the question presented above, attributes that collectively may quantify the "ideal" interface for network management applications are presented below:

- * support for an API into the application,
- * support for a windowing style of user interface, and
- * support for a command line style of user interface.

Even though the work presented in this research supports only the command line style of user interface, it is strongly stressed that support for the others will soon become a necessity in network management applications. Justifications for this belief are presented below.

4.4.1. API Support

No single network management application can provide support for all the needed functions in all environments. Network management applications must allow the individual customer sites to develop their own methods of interfacing to the various protocols contained within the application. To accomplish this, a network management API is required. Other APIs for network management

⁶²Scheifler, Robert W. and Gettys, Jim S., The X.11 Window System, Ibid.

⁶³Digital Equipment Corporation, Windowing - A Guide to the DECWindows Desktop Environment, Digital Press, (1989), pp. 9-14.

protocols such as Netview, SNMP⁶⁴, MAP⁶⁵ and others already exist. With the SNMP, motivations for the design of an API included independency of machine architecture, the need for a more powerful and flexible interface to the user, and the need for a higher-level and more efficient interface.⁶⁶ With the MAP API, the committee's design motivations included: portability, reduction of programmer training cost, reduction of applications development cost, and attracting applications developers.⁶⁷

Fong and Reinstedler then discussed some of the major issues⁶⁸ in the development of an API including:

- * What level of common specification is needed to characterize an interface between different application layer protocols?
- * What is a reasonable common mechanism that can unambiguously specify all interface functions and be implementable in multiple applications languages?
- * How many resource management details should the API expose to the programmer?
- * How should arbitrarily complex data be presented to and received from a user?

⁶⁴Case, Jeffrey D., Davin, James R., Fedor, Mark S., and Schoffstall, Martin L., "An Applications Programming Interface for the Simple Network Management Protocol", (June 1989).

⁶⁵Fong, K. and Reinstedler, J., Ibid.

⁶⁶Case, Jeffrey D., Davin, James R., Fedor, Mark S., and Schoffstall, Martin L., Ibid., p. 2.

⁶⁷Fong, K. and Reinstedler, J., Ibid., pp. 25-26.

⁶⁸Fong, K., and Reinstedler J., Ibid., pp. 32-59.

The developers of the MAP API resolved the first two questions by performing certain levels of abstraction (*i.e.*, What are the common functions performed by any API? What are the common parameters needed to perform a given function?). One of the goals in the language abstraction was to develop specification procedures that were language independent and yet specific enough to make possible a translation into any given language. Both static and dynamic allocation of memory must be supported within the API. The ability for the API to allocate resources dynamically became apparent when the MAP API committee discovered that instances existed where only the API would know the exact size of certain data structures to be passed to the user. The fourth question presented by Fong and Reinstedler remained unanswered. The authors stated that a model for the interchange of complex representations is needed before an OSI API is developed.

4.4.2. Windowing Support

Bit-mapped workstations, personal computers (PCs), and terminals have become extremely popular in the workplace within recent years. The rationale for the deployment of these devices are threefold: (1) increased ease of use, (2) improved productivity, and (3) ease of data sharing between windowed applications.⁶⁹ As networks continue to proliferate throughout most organizations, the need for high-quality visuals representing management information from multiple network

⁶⁹Hanner, Mark, "Gambling on Window Systems," Unix Review, Vol. 6 No. 12, (1988), p. 52.

resources becomes a priority. Without windowing support, network managers will be severely restricted in performing their everyday job functions.

Some of the primary windowing systems (*i.e.*, toolkits) in use today are: (1) those based on the X.11 library (*e.g.*, Digital Equipment Corporation's DECWindows, Hewlett Packard's NewWave, and Apollo's Open Dialogue); (2) those based on the X/NeWS library which comes bundled with UNIX System V Release 4 (*e.g.*, AT&T's Xtk+, NeWS Development Environment, and SunView); (3) MicroSoft Windows for MS-DOS based systems; (4) Presentation Manager for OS/2 based systems; and (5) the Mac toolkit for Macintosh systems.⁷⁰

Of the windowing systems mentioned above, those based on the X.11 library are more apt to survive due to the fact that the X.11 library has several advantages over the other windowing libraries.⁷¹ X.11 is highly portable between diverse hardware and software environments, is capable of functioning over any network (*i.e.*, is network independent), and is non-proprietary.

4.4.3. Command Line Support

Support for a command line user interface should always be available to the network management personnel. Command line interfaces allow the network

⁷⁰Ibid., p. 53.

⁷¹Gettys, Jim, "Network Windowing Using the X Window System," Dr. Dobbs Journal, (March 1989), pp. 42-46.

management personnel to query managed devices for information from a multitude of workstations platforms, some of which can not support a windowing environment. The command line interface should also be intelligent enough to support both concise and verbose forms of commands.

Chapter 5

Conclusions and Summary

5.1. Summary

5.1.1. SQL as an NMQL

Research into the utilization of ANSI standard SQL as a network management query language (NMQL) has shown certain feasibility. Don Tow⁷² in a paper on advances and trends within telecommunication's network management, cites the need for an integrated network management operating system (*i.e.*, an integrated network management query language) to provide a global view of and control over the network. By augmenting the existing standard, SQL can provide the functions necessary to perform that role.

5.1.1.1. Operand Limitations

The limitations imposed by the current operands offered by ANSI SQL present some significant obstacles before network management utilizing SQL as a command line interface can occur. The most prominent limitation is the lack of an event notification mechanism. Without some method for devices to alert managers, realtime network management can not occur. The inability of the existing SQL operands to distinguish between realtime queries of a device and

⁷²Tow, Don M., "Network Management -- Recent Advances and Future Trends", IEEE Journal on Selected Areas in Communications, Vol. 6 No. 4, (May 1988), pp. 732-741.

historical queries of the device has been recognized. The inability to schedule queries of devices and set duration and repetition intervals, while important, is not as major as event notification.

5.1.1.2. Operator Limitations

The operator limitations of the current ANSI standard SQL impose some small, but workable problems before SQL's implementation as an NMQL can be realized. The ability to store data in fields larger than 32 bits is needed before network management utilizing SQL (or an augmented form of SQL) can occur. Once the data is stored in an extended (*i.e.*, greater than 32 bit) data field, some method must exist to view the data in optional forms. Also, the limitation pertaining to non-scalar operators within an SQL data structure is minor with respect to the other outstanding issues, incorporation of vectored references would certainly be advantageous.

5.1.1.3. Suggested Solutions to Recognized Limitations

One approach to resolving the limitations of the current SQL is to submit a draft letter to the ANSI X3.135 committee, which oversees the SQL standard, explaining the need and rationale for extension of the datatypes supported by ANSI SQL. Another approach (and probably a better one) is to submit a draft letter to one of the many network management working groups (*e.g.*, NIST OSI Implementors Workshop, European Workgroup for Open Systems (EWOS), IETF, *etc.*) asking for the formation of an NMQL working group based on an augmented

version of ANSI SQL. For SQL to be utilized to perform network management applications, the following modifications to the existing operands (command syntax) and operators (supported datatypes) are needed:

- * **START AT XX:XX** - where **XX:XX** is some military time representing when to initiate a specified query, (Note: This additional command will enable network management applications to schedule queries to other managers/agents and will provide for the ability to distinguish between queries of the historical database and "live" queries of the specific device.)
- * **FOR XX:XX EVERY XX:XX** - where **XX:XX** is some military time representing the duration and repetition intervals of a specific query⁷³,
- * **WIDTH N** - used during database definition, where **N** the width in bytes of the field (Note: The 48-bit Ethernet addresses shown within this prototype are minimal in size when considering the size of some of the ISO addresses. This additional command will eliminate the waste of storing each nybble of an address as an eight-bit American Standard Code for Information Interchange (ASCII) character), and
- * **/HEX** - This qualifier when attached to a specific field or group of fields would display the hexadecimal representation of the bits. (Note: This would eliminate the need for an applications program to have to translate the field(s) and then re-present them to the user.)

The usage of dynamic versus embedded SQL in network management applications has been researched and the results of that research are now presented. Dynamic SQL isolates the end-user from the task of developing applications programs by allowing SQL statements to be entered directly from the command line; whereas embedded SQL requires the commands to be pre-encoded. Dynamic SQL does, however, require more complex programming and is also more

⁷³Warrier, U. and Sunshine, C., Ibid.

demanding of computer resources. Therefore, dynamic SQL is not supported on most smaller implementations of SQL; whereas embedded SQL is supported on all.

Once (If) the ANSI standard SQL is modified or an augmented SQL is referenced, embedded SQL is suggested as the primary form for the development of an NMQL.

5.1.2. Common User Interface

The notion of an "common user interface" has been explored and various styles and formats have been discussed. The idea that a single style of user interface can function as an ideal interface seems impractical, due to the fact that different users prefer different types of interfaces to their computers. However, a user interface which, in reality, presents the availability of multiple styles to the user community, can be considered an "ideal user interface". By allowing the user community to chose the style of interface, the various psychologies of each individual can be addressed.

5.1.3. Future Deployment of an NMQL

The prototype produced by this research layered SQL on top of RBMS, which is layered on top of DECNET and Ethernet. While this is a layered approach to performing network management, it in no way resembles the amount of layering that will be necessary for network communications in the environment that exists today or that will someday exist in the ISO environment of the future. With this in mind, the deployment of an NMQL throughout an entire network must be

thought out carefully. Due to the substantial amount of memory required to implement SQL (1.2Mb⁷⁴), some devices, such as those controlled by Intel 80386 processors and smaller, may not have the processing capacity necessary to perform NMQL services on top of the ISO layers.

The CMIS/CMIP and SNMP models of network management divide devices into two major categories -- (1) those that perform managing functions (*i.e.*, Managers) and (2) those that provide management information (*i.e.*, Agents). Therefore, as one model for the deployment of NMQL systems, it is suggested that the NMQL be deployed only on those systems which can function as managers.

Deployment of NMQL in this manner will reduce the processing and memory requirements placed on agents, but will also prevent the realtime querying of those devices from within the NMQL. However, realtime NMQL communication could be accomplished by developing translation programs between the NMQL and all supported network management protocols. Development of multiple translation programs is not recommended due to the time involved and the maintenance required.

For NMQL to be deployed on both manager and agent systems, the necessary datatype extensions must be added to the SQL standard must become a reality. Otherwise, the utilization of alternative methods of storing management information

⁷⁴Oracle Corporation, "Oracle Applications Tools for MS-DOS User's Guide", Version 5.1B, 1988, p. A-2.

in a format known by the NMQL (*i.e.*, SQL) which require more memory and more processing capacity make the NMQL infeasible, especially on single processor devices with minimal processing capacity.

5.2. Conclusions

The work presented herein has provided the research community with a prototype for the utilization of SQL as a Network Management Query Language (NMQL). The author has presented an abstract model describing the functions of network management, provided a mapping between the abstract model and CMIP, and shown that existing SQL (with some extensions) can provide the necessary syntax to perform network management. The further development and eventual deployment of SQL as a Network Management Query Language is dependent upon: (1) the modification of the existing ANSI specification or formation of NMQL working group within one of the network management standards groups, (2) the willingness of computer manufacturers to implement a NMQL, and (3) the desire of the user community to use a common NMQL within a given enterprise.

5.3. Recommendations for Further Study

One of the major faults of utilizing the current SQL as an NMQL is its inability to receive unsolicited inputs. This is to be expected since SQL was originally developed as a query language for existing databases. This research presented one possible solution to this problem (Section 4.1), but leaves development of unsolicited inputs for the NMQL for further research.

The ability to submit management queries for execution at a later time remains unavailable with ANSI SQL and is left for further study. One approach to solving this problem is to create a new operand similar to that defined in Section 5.1.1.3.

The work presented some of the characteristics necessary for the development of a common user interface, but failed to make any significant contribution in this area of research. However, a recent publication from the Open Software Foundation may have made a significant contribution towards that effort. The OSF/1 operating system will attempt to provide "...a single software technology that will let computers from a variety of vendors transparently work together and share resources such as computing power, data, printers, and other peripheral devices."⁷⁵ Due to the significant amount of vendor and academic interest/involvement in the development of OSF/1, the author recommends that further research be explored into the layering of SQL and NMQL applications on top of OSF/1.

If an NMQL is layered on top of multiple existing (or future) network management protocols, the occasion may arise where one or more devices support more than one of management protocols contained within the NMQL. Implementation of some form of recognizing and ignoring duplicate queries is left for further study.

⁷⁵Fauth, Dr. Dietmaur, et al, OSF Distributed Computing Environment Rationale, Open Software Foundation: Cambridge, MA, May 14, 1990, p. 32.

Bibliography

Bibliography

- ANSI X3.135-1986. *American National Standards for Information Systems - Database Language - SQL*. New York: American National Standards Institute, October, 1986.
- Aronoff, R., M. Chernick, K. Hsing, K. Mills, and D. Stokesberry. "Management of Networks Based on Open Systems Interconnection (OSI) Standards: Functional and Requirements Analysis." NIST Special Publication 55-175. Washington, D.C.: U.S. Department of Commerce, November, 1989.
- Ben-Artzi, Amatzia. "An Architecture for a Multi-Vendor Network Management System." *Integrated Network Management*, I. B. Meandzija and J. Westcott, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989. pp. 445-453.
- Case, J., J. Davin, M. Fedor, and M. Schoffstall. "An Applications Programming Interface for the Simple Network Management Protocol." June, 1989.
- Case, J., M. Fedor, M. Schoffstall, and J. Davin. "A Simple Network Management Protocol." Request For Comments 1098. Menlo Park, CA: DDN Network Information Center, SRI International, March, 1988.
- CCITT Recommendation X.400. "Message Handling Systems: System Model - Service Elements." Red Book, 1984.
- Cerf, V. "IAB Recommendations for the Development of Internet network management standards." Request For Comments 1052. Menlo Park, CA: DDN Network Information Center, SRI International, April, 1988.
- Cole, Leo. "Network Management as Described in the Systems Network Architecture.", *Proceedings of the IEEE Infocom-86*. Miami, FL, April, 1986, pp. 364-376.
- Czejdo, B., V. Reddy, and M. Rusinkiewicz. "Design and Implementation of and Interactive Graphical Query Language for a Relational Database Management System." *IEEE Computer Society - Workshop on Visual Languages*. 1988, pp. 14-19.

- Digital Equipment Corporation. *VAX SQL An Overview*. Maynard, Mass: July, 1988.
- Digital Equipment Corporation. *VAX SQL Reference Manual*. Order #: AA-JM32B-TE. Maynard, Mass: July, 1988.
- Digital Equipment Corporation. *VAX SQL User's Guide*. Order #: AA-JM33B-TE. Maynard, Mass: July, 1988.
- Digital Equipment Corporation, *Windowing - A Guide to the DECWindows Desktop Environment*. Digital Press, 1989.
- Dupuy, A., J. Schwartz, Y. Yemmi, G. Barzilai, and A. Chana. "Network Fault Management: A User's View." *Integrated Network Management, I*. B. Meandzija and J. Westcott, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989. pp. 101-108.
- Emerson, Sally, *et al.* *The Practical SQL Handbook*. Reading, Mass.: Addison-Wesley Publishing, Co., April, 1989.
- Faurer, L. D. "Network Management in an OSI Environment." *Computer Standards and Interfaces* 7. North Holland, 1988, pp. 155-159.
- Fauth, D., J. Gossels, D. Hartman, B. Johnson, R. Kumar, N. Leser, D. Lounsbury, D. Mackey, C. Shue, T. Smith, J. Stiener, and W. Tuvell. *Distributed Computing Environment Rationale*. Cambridge, Mass.: Open Software Foundation, May 14, 1990.
- Fletcher, H. and R. Taylor. "Human Factors Engineering in Network Management." *British Telecommunications Technology Journal*. Vol. 6 No. 4. October, 1988, pp. 44-51.
- Foley, James D., V. Wallace, and P. Chan. "The Human Factors of Computer Graphics Interaction Techniques." *IEEE CG&A*. Vol. 4 No. 11, November 1984, pp. 13-48.
- Fong, K. and J. Reinstadler. "The Development of an OSI Application Layer Protocol Interface." *Computer Communication Review*. 1986. p. 26.

- Footen-Lennox, Thom. "Ergonomic Guidelines for Computerized User Interface." *IEEE Proceedings Computer Standards Conference 1986*. San Francisco, CA: pp. 38-41.
- Ganesan, K. and M. Ganti. "A Multimedia Front-End for an Expert Network Management System." *IEEE Journal on Selected Areas in Communications*. Vol. 6 No. 5, June, 1988, pp. 788-791.
- Gettys, Jim. "Network Windowing Using the X Window System." *Dr. Dobbs Journal*. March, 1988. pp. 42-46.
- Hanner, Mark. "Gambling on Window Systems." *Unix Review*. Vol. 6 No. 12, 1988.
- Howard, H. C. and C. S. Howard. "User Interfaces for Structural Engineering Relational Data Base." *Engineering with Computers*. 4 (1988), pp. 239-249.
- Information Processing Systems - Open Systems Interconnection. "Carrier Sense Multiple Access with Collision Detect." International Standard 8802.3. International Organization for Standardization: Omnicom International, 1988.
- Information Processing Systems - Open Systems Interconnection. "File Transfer, Access, and Management, Part 1: General Introduction." International Standard 8571-1.
- Information Processing Systems - Open Systems Interconnection. "Management Information Service Definition Part 2: Common Management Information Protocol." Draft International Standard 9596-2. International Organization for Standardization: Omnicom International, August 1988.
- Information Processing Systems - Open Systems Interconnection. "Management Information Service Definition Part 2: Common Management Information Service." Draft International Standard 9595-2. International Organization for Standardization: Omnicom International, August 1988.
- Information Processing Systems - Open Systems Interconnection. "Protocol Specification for the Association Control Service Element." International Standard 8650. International Organization for Standardization: Omnicom International, August 1988.

Information Processing Systems - Open Systems Interconnection. "Service Definition for the Association Control Service Element." International Standard 8649. International Organization for Standardization: Omnicom International, August 1988.

Information Processing Systems - Open Systems Interconnection. "Structure of Management Information Part 4: Guidelines for the Definition of Managed Objects." Draft International Standard 10165-4. International Organization for Standardization: Omnicom International, June 1990.

Information Processing Systems - Open Systems Interconnection. "Systems Management Overview." Draft Proposal 10040. International Organization for Standardization: Omnicom International, June 1990.

Information Processing Systems - Open Systems Interconnection. "Systems Management Part 1: Object Management Function." Draft International Standard 10164-1. International Organization for Standardization: Omnicom International, May 1990.

Institute of Electrical and Electronics Engineers, "Carrier Sense Multiple Access with Collision Detect."

International Business Machines Corporation. *NetView/PC Application Program Interface/Communications Services*. Version 1.1. Order #: SC30-3313-1. October, 1987.

Kato, A., M. Togawa, K. Suzuki, and K. Oda. "A Visual Interface for an Object-Oriented Query in Relational Databases." *Systems and Computers in Japan*. Vol. 19 No. 11. 1988, pp. 62-64.

Kobayashi, Yoshikazu. "Standardization Issues in Integrated Network Management." *Integrated Network Management, I*. B. Meandzija and J. Westcott, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989. pp. 79-90

Kroenke, David M. *Database Processing: Fundamentals, Design, Implementation, 2nd Edition*. Chicago: Science Research Associates, 1983, pp. 265-266.

Manufacturing Automation Protocol Version 3.0 - Implementation Release - Subject to Errata Changes. July, 1987.

- McCloghrie, K. and M. T. Rose. "Management Information Base for TCP/IP-based Internets." Internet Engineering Task Force. Request For Comments 1066. Menlo Park, CA: DDN Network Information Center, SRI International, September, 1988.
- Oracle Corporation. "Oracle Application Tools for MS-DOS User's Guide. Version 5.1B. 1988.
- Postel, J. "Transmission Control Protocol." Internet Engineering Task Force. Request For Comments 793. Menlo Park, CA: DDN Network Information Center, SRI International, September, 1981.
- Postel, J. "User Datagram Protocol." Internet Engineering Task Force. Request For Comments 768. Menlo Park, CA: DDN Network Information Center, SRI International, August, 1980.
- Rose, M. and K. McCloghrie. "Structure and Identification of Management Information for TCP/IP-based Internets." Internet Engineering Task Force. Request For Comments 1066. Menlo Park, CA: DDN Network Information Center, SRI International, August, 1988.
- Scheifler, R. and Gettys, J. The X Window System. *ACM Transactions on Graphics*. 5(2):79-109. April 1986. Special Issue on User Interface Software.
- Sloman, Morris and Jonathan Moffett. "Domain Management for Distributed Systems." *Integrated Network Management*, I. B. Meandzija and J. Westcott, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989, pp. 505-516.
- Stonebraker, M., E. Wong, and P. Kreps. "The Design and Implementation of INGRES." *ACM Transactions on Database Systems*. 3(1976), pp. 189-212.
- Tannenbaum, Andrew S. *Computer Networks*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.
- Wakid, S., P. Brusil, and L. LeBarre. "Coming to OSI: Network Resource Management and Global Reachability." *Data Communications*. December, 1987, pp. 137-150.
- Warrier, U. and C. Sunshine. "A Platform for Heterogeneous Interconnection Network Management." *Integrated Network Management*, I. B. Meandzija and

J. Westcott, Ed. Amsterdam, The Netherlands: Elsevier Science Publishers, 1989, pp. 13-24.

Warrier, U. and L. Beesaw. "The Common Management Information Services and Protocol over TCP/IP." Internet Engineering Task Force. Request for Comments 1095. Menlo Park, CA: DDN Network Information Center, SRI International, September, 1981.

Whang, K., S. Brady, and R. Gilbert. "Expert Systems in Network Management and Control." *Expert Systems in Government Symposium*. IEEE. 1986, pp. 290-295.

Appendices

Appendix A

Source Code for Bridge_Schema_Create.sc

```

#include stdio
#include descrip
/*
*
*   This program illustrates how you declare C program variables
*   to match a select group of data types and how you can specify those
*   variables in SQL statements when you store and retrieve column
*   values or nulls.  The program:
*
*   o Creates the BRIDGE schema and table
*   o Stores a row using a reference to a host structure and
*     indicator array occurrences
*   o Retrieves that row using references to separate fields
*     and associated indicators
*   o Displays the stored row on the terminal
*   o Updates some column values in the row
*   o Displays the changed row on the terminal
*   o Drops the schema
*
*   Copyright 1990, Gary W. Haney and Jeffrey D. Case
*   University of Tennessee, Knoxville
*
*   NOTE: VAX SQL will not allow references to array variables within a
*   CURSED FETCH operation.  All variables must be scalar.
*   Added: 14-AUG-89
*
*   This program must be linked with appropriate C support routines.
*   Unless you specify the /NOG_FLOATING qualifier when you
*   precompile this program, be sure to link the program with routines
*   that support the G-floating interpretation of double-precision
*   floating-point values.
*/

#define debug 0

main()
{
/*
*   Declare the variable for monitoring execution status of SQL
*   statements.  If you omit this declaration, you must specify
*   the statement EXEC SQL INCLUDE SQLCA; and must qualify

```

*** references to SQLCODE (SQLCA.SQLCODE).**

***/**

/* int SQLCODE; */

EXEC SQL INCLUDE SQLCA;

char error_txt[301];

\$DESCRIPTOR (error_txt_desc, error_txt);

/* Declare a structure for row-level operations. The structure

*** has subordinate elementary variables for most column-level**

*** operations. For this structure, note the following points.**

*** Any char variables that correspond to CHAR and VARCHAR columns**

*** are defined as the length of the longest valid column value**

*** plus 1. (SQL expects null terminators on all values**

*** you are storing in CHAR and VARCHAR columns, and SQL puts a**

*** null terminator into all character strings you retrieve**

*** from the database.) C does not directly support the varying**

*** character data type; however, the ASCIZ strings you**

*** manipulate for char variables contain null terminators that**

*** SQL uses to determine string length when you are storing data**

*** in columns defined as VARCHAR.**

*** C does not support fixed-point binary numbers that are**

*** scaled, so you declare floating-point variables for any**

*** columns that can contain fixed-point binary values with a**

*** fractional component.**

*** C does not support fixed-point binary values of quadword**

*** size, even if the values are unscaled. For columns defined as**

*** QUADWORD, you can define variables with the double data type,**

*** but you should note that this variable can retrieve only**

*** approximations of the larger values that are valid for**

*** QUADWORD columns. These larger quadword values might be**

*** stored in the database by non-C applications.**

***/**

struct

{

char bridge_id[16];

char bridge_addr[18];

```

/* Line 1 counters.      */
char ln1_bin_date_var[24];
int ln1_bridge_seconds;
int ln1_fwding_state;
int ln1_port_restrt;
int ln1_unkn_dest_addr;
int ln1_frms_filt;
int ln1_mult_frms_filt;
int ln1_frms_addr_filt;
int ln1_frms_proto_filt;
int ln1_tot_frms_sent;
int ln1_tot_frms_rcv;
int ln1_tot_bytes_sent;
int ln1_tot_bytes_rcv;
int ln1_dev_frms_sent;
int ln1_dev_frms_rcv;
int ln1_dev_bytes_sent;
int ln1_dev_bytes_rcv;
int ln1_dev_frms_lost;
int ln1_mult_frms_sent;
int ln1_mult_frms_rcv;
int ln1_mult_bytes_sent;
int ln1_mult_bytes_rcv;
int ln1_mult_dev_frms_sent;
int ln1_mult_dev_frms_rcv;
int ln1_mult_dev_bytes_sent;
int ln1_mult_dev_bytes_rcv;
int ln1_inv_dev_msg_cnt;
int ln1_bad_frms_rcv;
int ln1_bad_bytes_rcv;
int ln1_xmt_frms_lost;
int ln1_rcv_frms_lost;
int ln1_bad_hello_lim_exc;
int ln1_collis_tst_fail;
int ln1_xmt_frms_err;
int ln1_mult_src_frms_rcv;
int ln1_lifetm_exc_frms;
int ln1_lifetm_exc_bytes;
int ln1_xmt_bytes_err;
int ln1_inv_dev_bytes;
int ln1_rcv_overrun;

```

```

    int ln1_oversize_frm;
    int ln1_framing_err;
    int ln1_xmt_mult_collis;
    int ln1_carrier_loss;
    int ln1_collis_lim_exc;
/* Repeat the same stuff for line 2 */
    char ln2_bin_date_var[24];
    int ln2_bridge_seconds;
    int ln2_fwding_state;
    int ln2_port_restrt;
    int ln2_unkn_dest_addr;
    int ln2_frms_filt;
    int ln2_mult_frms_filt;
    int ln2_frms_addr_filt;
    int ln2_frms_proto_filt;
    int ln2_tot_frms_sent;
    int ln2_tot_frms_rcv;
    int ln2_tot_bytes_sent;
    int ln2_tot_bytes_rcv;
    int ln2_dev_frms_sent;
    int ln2_dev_frms_rcv;
    int ln2_dev_bytes_sent;
    int ln2_dev_bytes_rcv;
    int ln2_dev_frms_lost;
    int ln2_mult_frms_sent;
    int ln2_mult_frms_rcv;
    int ln2_mult_bytes_sent;
    int ln2_mult_bytes_rcv;
    int ln2_mult_dev_frms_sent;
    int ln2_mult_dev_frms_rcv;
    int ln2_mult_dev_bytes_sent;
    int ln2_mult_dev_bytes_rcv;
    int ln2_inv_dev_msg_cnt;
    int ln2_bad_frms_rcv;
    int ln2_bad_bytes_rcv;
    int ln2_xmt_frms_lost;
    int ln2_rcv_frms_lost;
    int ln2_bad_hello_lim_exc;
    int ln2_collis_tst_fail;
    int ln2_xmt_frms_err;
    int ln2_mult_src_frms_rcv;

```

```
struct {
    char bridge[16];
    char bdg_address[18];
    char phy_addr[18];
    int on_line;
} lrnd_addr instance;
```

* /

*** /**

*/

/*

```
short int    bdg address ind = 0;
```

```

short int    phy_addr_ind = 0;
short int    on_line_ind = 0;
*/

/* Create the schema and table.
*/
printf ("\n Starting Schema creation...");
fsync(stdout);
EXEC SQL CREATE SCHEMA FILENAME "[HANEYG]BRIDGE.RDB";
    if (SQLCA.SQLCODE < 0)
    {
        printf("\nCreate schema failed %d", SQLCA.SQLCODE);
        SQL$SIGNAL();
    }
printf ("\n Schema created for Bridge...\n");
fsync(stdout);

/* Monitor execution of subsequent SQL statements using a
* WHENEVER statement.
*/
EXEC SQL WHENEVER SQLERROR GO TO sql_error_handler_1;
if (debug) printf ("\n Got to 3");
fsync(stdout);

EXEC SQL CREATE TABLE BRIDGE_TABLE
(
/* IINE 1 COUNTERS.      */
    BRIDGE_ID          CHAR(16)
                        NOT NULL,
    BRIDGE_ADDR        CHAR(18),
    LN1_BIN_DATE_VAR   CHAR(24),
    LN1_BRIDGE_SECONDS INTEGER,
    LN1_FWDING_STATE   INTEGER,
    LN1_PORT_RESTRT    INTEGER,
    LN1_UNKN_DEST_ADDR INTEGER,
    LN1_FRMS_FILT       INTEGER,
    LN1_MULT_FRMS_FILT  INTEGER,
    LN1_FRMS_ADDR_FILT  INTEGER,
    LN1_FRMS_PROTO_FILT INTEGER,
    LN1_TOT_FRMS_SENT   INTEGER,
    LN1_TOT_FRMS_RECV   INTEGER.

```

```

LN1_TOT_BYTES_SENT      INTEGER,
LN1_TOT_BYTES_RECV      INTEGER,
LN1_DEV_FRMS_SENT       INTEGER,
LN1_DEV_FRMS_RECV       INTEGER,
LN1_DEV_BYTES_SENT      INTEGER,
LN1_DEV_BYTES_RECV      INTEGER,
LN1_DEV_FRMS_LOST       INTEGER,
LN1_MULT_FRMS_SENT      INTEGER,
LN1_MULT_FRMS_RECV      INTEGER,
LN1_MULT_BYTES_SENT     INTEGER,
LN1_MULT_BYTES_RECV     INTEGER,
LN1_MULT_DEV_FRMS_SENT  INTEGER,
LN1_MULT_DEV_FRMS_RECV  INTEGER,
LN1_MULT_DEV_BYTES_SENT INTEGER,
LN1_MULT_DEV_BYTES_RECV INTEGER,
LN1_INV_DEV_MSG_CNT     INTEGER,
LN1_BAD_FRMS_RECV       INTEGER,
LN1_BAD_BYTES_RECV      INTEGER,
LN1_XMT_FRMS_LOST       INTEGER,
LN1_RCV_FRMS_LOST       INTEGER,
LN1_BAD_HELLO_LIM_EXC   INTEGER,
LN1_COLLIS_TST_FAIL     INTEGER,
LN1_XMT_FRMS_ERR        INTEGER,
LN1_MULT_SRC_FRMS_RCV   INTEGER,
LN1_LIFETM_EXC_FRMS     INTEGER,
LN1_LIFETM_EXC_BYTES    INTEGER,
LN1_XMT_BYTES_ERR       INTEGER,
LN1_INV_DEV_BYTES       INTEGER,
LN1_RCV_OVERRUN         INTEGER,
LN1_OVERSIZE_FRM        INTEGER,
LN1_FRAMING_ERR         INTEGER,
LN1_XMT_MULT_COLLIS     INTEGER,
LN1_CARRIER_LOSS       INTEGER,
LN1_COLLIS_LIM_EXC      INTEGER,
/* REPEAT THE SAME STUFF FOR LINE 2 */
LN2_BIN_DATE_VAR        CHAR(24),
LN2_BRIDGE_SECONDS      INTEGER,
LN2_FWDING_STATE        INTEGER,
LN2_PORT_RESTRT         INTEGER,
LN2_UNKN_DEST_ADDR      INTEGER,
LN2_FRMS_FILT           INTEGER.

```

LN2_MULT_FRMS_FILT	INTEGER,
LN2_FRMS_ADDR_FILT	INTEGER,
LN2_FRMS_PROTO_FILT	INTEGER,
LN2_TOT_FRMS_SENT	INTEGER,
LN2_TOT_FRMS_RECV	INTEGER,
LN2_TOT_BYTES_SENT	INTEGER,
LN2_TOT_BYTES_RECV	INTEGER,
LN2_DEV_FRMS_SENT	INTEGER,
LN2_DEV_FRMS_RECV	INTEGER,
LN2_DEV_BYTES_SENT	INTEGER,
LN2_DEV_BYTES_RECV	INTEGER,
LN2_DEV_FRMS_LOST	INTEGER,
LN2_MULT_FRMS_SENT	INTEGER,
LN2_MULT_FRMS_RECV	INTEGER,
LN2_MULT_BYTES_SENT	INTEGER,
LN2_MULT_BYTES_RECV	INTEGER,
LN2_MULT_DEV_FRMS_SENT	INTEGER,
LN2_MULT_DEV_FRMS_RECV	INTEGER,
LN2_MULT_DEV_BYTES_SENT	INTEGER,
LN2_MULT_DEV_BYTES_RECV	INTEGER,
LN2_INV_DEV_MSG_CNT	INTEGER,
LN2_BAD_FRMS_RECV	INTEGER,
LN2_BAD_BYTES_RECV	INTEGER,
LN2_XMT_FRMS_LOST	INTEGER,
LN2_RCV_FRMS_LOST	INTEGER,
LN2_BAD_HELLO_LIM_EXC	INTEGER,
LN2_COLLIS_TST_FAIL	INTEGER,
LN2_XMT_FRMS_ERR	INTEGER,
LN2_MULT_SRC_FRMS_RCV	INTEGER,
LN2_LIFETM_EXC_FRMS	INTEGER,
LN2_LIFETM_EXC_BYTES	INTEGER,
LN2_XMT_BYTES_ERR	INTEGER,
LN2_INV_DEV_BYTES	INTEGER,
LN2_RCV_OVERRUN	INTEGER,
LN2_OVERSIZE_FRM	INTEGER,
LN2_FRAMING_ERR	INTEGER,
LN2_XMT_MULT_COLLIS	INTEGER,
LN2_CARRIER_LOSS	INTEGER,
LN2_COLLIS_LIM_EXC	INTEGER /* Last line -- no comma

*/

);

```
printf ("\n Table Bridge_Table created...\n");
fsync(stdout);
```

```
/* Create an index for the BRIDGE_TABLE - DON'T Allow duplicates. */
EXEC SQL CREATE INDEX BRIDGE_IDX
      ON BRIDGE_TABLE (BRIDGE_ID, LN1_BRIDGE_SECONDS);
```

```
printf ("\n Index on Bridge_Table created...");
fsync(stdout);
```

```
EXEC SQL CREATE TABLE LRND_ADDRS
      (BRIDGE_DATE      CHAR(24),
       BRIDGE_ID        CHAR(16)
                               NOT NULL,
       BDG_ADDRESS      CHAR(18)
                               NOT NULL,
       PHY_ADDR         CHAR(18),
       ON_LINE          INTEGER
      );
```

```
printf ("\n Table Lrnd_Addrs created...");
fsync(stdout);
```

```
EXEC SQL CREATE VIEW BRIDGE_LEARNED_ADDRS
      (BRIDGE_DATE, PHY_ADDRESS, ON_LINE) AS
      SELECT BRIDGE_DATE, PHY_ADDR, ON_LINE
      FROM LRND_ADDRS LA, BRIDGE_TABLE BT
      WHERE LA.BRIDGE_ID = BT.BRIDGE_ID
      AND LA.BRIDGE_DATE = BT.LN1_BIN_DATE_VAR
      OR LA.BRIDGE_DATE = BT.LN2_BIN_DATE_VAR;
```

```
printf ("\n View Bridge_learned_addrs created...\n");
fsync(stdout);
```

```
/* Move values to main variables. Usually, a program
 * would first initialize all main variables and indicators
 * before accepting each set of main variable values from a
 * terminal or data file. To keep this program short,
 * simply store a set of literals and rely on 0 initialization
 * in indicator declarations. (A value of 0 in an
```

*** indicator ensures that a value in an associated main variable
* is stored.)
*/**

```
strcpy (bridge_instance.bridge_id, "MVPC LAN");  
strcpy (bridge_instance.bridge_addr, "08-00-2B-08-38-F9");  
strcpy (bridge_instance.ln1_bin_date_var, "12-AUG-1989 12:17:56.00");  
strcpy (bridge_instance.ln2_bin_date_var, "12-AUG-1989 12:17:58.00");  
/* Put dummy information in for line 1 */  
bridge_instance.ln1_bridge_seconds = 123456789;  
bridge_instance.ln1_fwding_state = 12;  
bridge_instance.ln1_port_restrt = 1234;  
bridge_instance.ln1_unkn_dest_addr = 56;  
bridge_instance.ln1_frms_filt = 1231411110;  
bridge_instance.ln1_mult_frms_filt = 1222137;  
bridge_instance.ln1_frms_addr_filt = 1248601;  
bridge_instance.ln1_tot_frms_sent = 1821113460;  
bridge_instance.ln1_tot_frms_rcv = 1237890;  
bridge_instance.ln1_tot_bytes_sent = 1999871156;  
bridge_instance.ln1_tot_bytes_rcv = 9899996;  
bridge_instance.ln1_dev_frms_sent = 1821113460;  
bridge_instance.ln1_dev_frms_rcv = 1237890;  
bridge_instance.ln1_dev_bytes_sent = 1999871156;  
bridge_instance.ln1_dev_bytes_rcv = 9899996;  
bridge_instance.ln1_dev_frms_lost = 6;  
bridge_instance.ln1_mult_frms_sent = 1821113460;  
bridge_instance.ln1_mult_frms_rcv = 1237890;  
bridge_instance.ln1_mult_bytes_sent = 1999871156;  
bridge_instance.ln1_mult_bytes_rcv = 9899996;  
bridge_instance.ln1_mult_dev_frms_sent = 1821113460;  
bridge_instance.ln1_mult_dev_frms_rcv = 1237890;  
bridge_instance.ln1_mult_dev_bytes_sent = 1999871156;  
bridge_instance.ln1_mult_dev_bytes_rcv = 9899996;  
bridge_instance.ln1_inv_dev_msg_cnt = 96;  
bridge_instance.ln1_bad_frms_rcv = 9;  
bridge_instance.ln1_bad_bytes_rcv = 3989;  
bridge_instance.ln1_xmt_frms_lost = 98;  
bridge_instance.ln1_rcv_frms_lost = 198;  
bridge_instance.ln1_bad_hello_lim_exc = 3;  
bridge_instance.ln1_collis_tst_fail = 11;  
bridge_instance.ln1_xmt_frms_err = 49;  
bridge_instance.ln1_mult_src_frms_rcv = 1298;
```

```

bridge_instance.ln1_lifetm_exc_frms = 393;
bridge_instance.ln1_lifetm_exc_bytes = 2698;
bridge_instance.ln1_xmt_bytes_err = 0;
bridge_instance.ln1_inv_dev_bytes = 0;
bridge_instance.ln1_rcv_overrun = 0;
bridge_instance.ln1_oversize_frm = 938;
bridge_instance.ln1_framing_err = 1222;
bridge_instance.ln1_xmt_mult_collis = 17501;
bridge_instance.ln1_carrier_loss = 0;
bridge_instance.ln1_collis_lim_exc = 0;
/* Now for line 2 */
bridge_instance.ln2_bridge_seconds = 123456789;
bridge_instance.ln2_fwding_state = 12;
bridge_instance.ln2_port_restrt = 1234;
bridge_instance.ln2_unkn_dest_addr = 56;
bridge_instance.ln2_frms_filt = 1231411110;
bridge_instance.ln2_mult_frms_filt = 1222137;
bridge_instance.ln2_frms_addr_filt = 1248601;
bridge_instance.ln2_tot_frms_sent = 1821113460;
bridge_instance.ln2_tot_frms_rcv = 1237890;
bridge_instance.ln2_tot_bytes_sent = 1999871156;
bridge_instance.ln2_tot_bytes_rcv = 9899996;
bridge_instance.ln2_dev_frms_sent = 1821113460;
bridge_instance.ln2_dev_frms_rcv = 1237890;
bridge_instance.ln2_dev_bytes_sent = 1999871156;
bridge_instance.ln2_dev_bytes_rcv = 9899996;
bridge_instance.ln1_dev_frms_lost = 6;
bridge_instance.ln2_mult_frms_sent = 1821113460;
bridge_instance.ln2_mult_frms_rcv = 1237890;
bridge_instance.ln2_mult_bytes_sent = 1999871156;
bridge_instance.ln2_mult_bytes_rcv = 9899996;
bridge_instance.ln2_mult_dev_frms_sent = 1821113460;
bridge_instance.ln2_mult_dev_frms_rcv = 1237890;
bridge_instance.ln2_mult_dev_bytes_sent = 1999871156;
bridge_instance.ln2_mult_dev_bytes_rcv = 9899996;
bridge_instance.ln2_inv_dev_msg_cnt = 96;
bridge_instance.ln2_bad_frms_rcv = 9;
bridge_instance.ln2_bad_bytes_rcv = 3989;
bridge_instance.ln2_xmt_frms_lost = 98;
bridge_instance.ln2_rcv_frms_lost = 198;
bridge_instance.ln2_bad_hello_lim_exc = 3;

```

```

bridge_instance.ln2_collis_tst_fail = 11;
bridge_instance.ln2_xmt_frms_err = 49;
bridge_instance.ln2_mult_src_frms_rcv = 1298;
bridge_instance.ln2_lifetm_exc_frms = 393;
bridge_instance.ln2_lifetm_exc_bytes = 2698;
bridge_instance.ln2_xmt_bytes_err = 0;
bridge_instance.ln2_inv_dev_bytes = 0;
bridge_instance.ln2_rcv_overrun = 0;
bridge_instance.ln2_oversize_frm = 938;
bridge_instance.ln2_framing_err = 1222;
bridge_instance.ln2_xmt_mult_collis = 17501;
bridge_instance.ln2_carrier_loss = 0;
bridge_instance.ln2_collis_lim_exc = 0;

```

```

if (debug) printf ("\n Got to 5");
fsync(stdout);

```

```

/* The following if statements evaluate contents of main variables
 * and then set indicators as appropriate. The conditional
 * evaluation is superfluous in this particular program, but is
 * appropriate for programs that store multiple rows and permit
 * nulls for columns associated with the main variables.
 */

```

```

if (debug) printf ("\n Got to 6");
fsync(stdout);

```

```

/* The following INSERT statement transfers an entire record as
 * a row. Because the INSERT statement does not list columns in
 * BRIDGE_TABLE in the order that they correspond to
 * elementary fields in BRIDGE_INSTANCE statement
 * assumes that the current number and order of columns in the
 * table correspond correctly to number and order of fields
 * in the record. This is appropriate when a table definition
 * is stable (as is true here because the program creates the table).
 */

```

```

EXEC SQL INSERT INTO BRIDGE_TABLE
VALUES (:bridge_instance:indicator_item);

```

```

printf ("\n\n LAN Bridge Test values have been inserted into the database.");
fsync(stdout);
/*
 * Now retrieve and display the row.
 */
EXEC SQL DECLARE CURSED_CONVERSIONS CURSOR FOR
      SELECT LN1_BRIDGE_SECONDS FROM BRIDGE_TABLE
      WHERE BRIDGE_ID = "MVPC LAN";

if (debug) printf ("\n Got to 8");
fsync(stdout);

EXEC SQL OPEN CURSED_CONVERSIONS;

if (debug) printf ("\n Got to 9");
fsync(stdout);

printf("\n Test stored values by retrieving a single row from the database.");
printf("\n Value inserted into the database for Bridge seconds on Line 1 =
%d",bridge_instance.ln1_bridge_seconds);
fsync(stdout);

EXEC SQL FETCH CURSED_CONVERSIONS INTO
:bridge_instance.ln1_bridge_seconds;

/*
 * Display the value of ln1_bridge_seconds stored in the first row of the
 * database.
 */

printf("\n\n Value returned from the Historical Database: %d",
bridge_instance.ln1_bridge_seconds);
fsync(stdout);

/* End the transaction so the schema closed. First change
 * error handling so that actions are appropriate for failure of
 * the ROLLBACK or CLOSE SCHEMA statement.
 */
EXEC SQL WHENEVER SQLERROR GO TO sql_error_handler_2;

EXEC SQL COMMIT;

```

```

printf ("\n\n Changes have been committed to the Bridge_Table Schema");
printf ("\n\n Creation of the Historical Database for the LANBridge completed
normally...");
fsync(stdout);
SYS$EXIT(1);

```

```

/* Error-handling sections
*/

```

```

sql_error_handler_1:
    SQL$GET_ERROR_TEXT( &error_txt_desc);
    printf ("\n Error text: %s", &error_txt);
    printf ("\n SQLCODE is %d", SQLCA.SQLCODE);
    printf ("\n Error on attempt to create table, or on");
    printf ("\n attempt to insert, update, or retrieve data.");
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK;
    if (SQLCA.SQLCODE < 0)
    {
        printf ("\n ROLLBACK failed. Delete database files manually.");
        SQL$SIGNAL();
    }

```

```

sql_error_handler_2:
    SQL$GET_ERROR_TEXT( &error_txt_desc);
    printf ("\n Error text: %s", &error_txt);
    printf ("\n SQLCODE is %d", SQLCA.SQLCODE);
    printf ("\n Schema could not be dropped.");
    printf ("\n Delete database files manually.");
    SQL$SIGNAL();
}

```

Appendix B
Source Code for NMQL_Manager.sc

```

/*****
*
* NMQL_MANAGER.SC
*
* This module is the first in a series of five that are used to demonstrate
* the capabilities of a Network Management Query Language (NMQL) that uses
a
* derivative of SQL. This program implements dynamic SQL through the C
* Language.
*
* The first external module is a routine to query the devices
* (i.e. bridge) to update the database periodically or whenever realtime
* information is requested.
*
* The second module performs SQL non-SELECT stmts (e.g. DECLARE
* DATABASE, etc.)
*
* The third module performs SQL SELECT stmts.
*
* The fourth module (this one) is responsible for:
*   (1) accepting an NMQL stmt from the user,
*   (2) parsing the statement to determine which routine to call
*       (DO_SELECT, DO_NONSEL, DO_REALTIME),
*   (3) sending the statement string via the appropriate module,
*   (4) accepting data or messages returned by the modules, and
*   (5) displaying the data or messages to the user
*
* The fifth module performs NMQL statements on the agent.
*
* AUTHORS:
*   Gary W. Haney and Jeff D. Case, University of Tennessee
*   Copyright 1989, Gary W. Haney, All rights reserved.
*   Permission to use or copy this software is granted provided
*   copies are not made or distributed for direct commercial
*   advantage, the copyright notice and its date appear, notice
*   is given that copying or use of is by permission of Gary W.
*   Haney. To copy or use otherwise requires a fee and/or
*   specific permission.
*
* CREATION DATE:

```

```

*      17-Aug-89
*
*  MODIFICATION HISTORY:
*
*/
#include stdio
#include ctype
#include file
#include unixio
/* VMS Header files */
#include descrip
#include ssdef
#include lnmdf
#include jpidef

/* Declare the SQLCA */
EXEC SQL INCLUDE SQLCA;
int SQLCODE;

/*
*  NOTE: VAX SQL will not allow references to array variables within a
*  CURSED FETCH operation. All variables must be scalar.
*      Added: 14-AUG-89
*
*  This program must be linked with appropriate C support routines.
*  Unless you specify the /NOG_FLOATING qualifier when you
*  precompile this program, be sure to link the program with routines
*  that support the G-floating interpretation of double-precision
*  floating-point values.
*/

/* Declare a structure for row-level operations. The structure
* has subordinate elementary variables for most column-level
* operations. For this structure, note the following points.
*
* Any char variables that correspond to CHAR and VARCHAR columns
* are defined as the length of the longest valid column value
* plus 1. (SQL expects null terminators on all values
* you are storing in CHAR and VARCHAR columns, and SQL puts a
* null terminator into all character strings you retrieve
* from the database.) C does not directly support the varying

```

```

* character data type; however, the ASCIZ strings you
* manipulate for char variables contain null terminators that
* SQL uses to determine string length when you are storing data
* in columns defined as VARCHAR.
*
* C does not support fixed-point binary numbers that are
* scaled, so you declare floating-point variables for any
* columns that can contain fixed-point binary values with a
* fractional component.
*
* C does not support fixed-point binary values of quadword
* size, even if the values are unscaled. For columns defined as
* QUADWORD, you can define variables with the double data type,
* but you should note that this variable can retrieve only
* approximations of the larger values that are valid for
* QUADWORD columns. These larger quadword values might be
* stored in the database by non-C applications.
*/

```

```

#define      maxpar      31
#define maxfldlen 100
#define maxtblen 100
#define false 0
#define true 1
#define error 0
#define ss$normal 1

#define PRC$M_HIBER 32
#define LNM$M_TERMINAL 512
#define LNM$_STRING 2
#define LNM$_ATTRIBUTES 3

```

```

char ret_string[256], search=27, result, cmd_line_input='Y';
int ret_attr=0,start_pos=5;
short ret_len;

```

```

FILE *rbmstxt, *fopen();
/* Define a type for quadword variables */
typedef int quadword[2];

```

```
typedef union
```

```
{  
    struct  
    {  
        short buffer_length;  
        short item_code;  
        int buffer_addr;  
        int ret_len_addr;  
    } items;  
    long end_of_list;  
} list;
```

```
list lnm[3];
```

```
list jpi[2];
```

```
quadword current_time, interval, daytim, reptim;
```

```
int bridge_process_id,sql_proc_id;
```

```
int proc_status;
```

```
char  ascii_int[24], asc_str[24], asc_end[24];
```

```
char  def_intvl[7];
```

```
char  stmt[1024];
```

```
char  agent[30];
```

```
/*
```

```
*/
```

```
$DESCRIPTOR(output_sql, "CS$USERS:[HANEYG]SQLOUT.DAT");
```

```
$DESCRIPTOR(input_sql, "USER_INPUT.DAT");
```

```
$DESCRIPTOR(table_name, "LNM$PROCESS_TABLE");
```

```
$DESCRIPTOR(terminal, "SYS$OUTPUT");
```

```
$DESCRIPTOR(sub_name, "NMQL_AGENT");
```

```
$DESCRIPTOR(cluster_name, "NMQL");
```

```
$DESCRIPTOR(output,ret_string);
```

```
$DESCRIPTOR(sql_sub, "NMQL_SQL");
```

```
$DESCRIPTOR(sql, "SQL");
```

```
$DESCRIPTOR(dcl,"DCL");
```

```
$DESCRIPTOR(rbms_add,"RBMS_ADD.COM");
```

```
$DESCRIPTOR(rbms_del,"RBMS_DEL.COM");
```

```
struct dsc$descriptor_s intvl_dsc;
```

```

struct dsc$descriptor_s strt_dsc;
struct dsc$descriptor_s end_dsc;
struct dsc$descriptor_s def_dsc;

```

```

/* Define procedures. */

```

```

int get_real();
int do_history();
int add_to_agent();
int del_from_agent();
int err();
int usage();

```

```

short  user_interval = 0;
short  realtime;
short  begintime;
short  endtime;
short  stop = 0;

```

```

short  debug = 1;
short  selstmt = 0;
short  insertstmt = 0;
short  deletestmt = 0;
FILE   *nmql_log,*sql_output,*sql_input;
short  version = 1;

```

```

char datebuf[8];

```

```

main(argc,argv)

```

```

int argc;
char *argv[];
{

```

```

unsigned long clocktime;
int i,j,k,l=0,tmp;
register status;

```

```

/*

```

```

* Declare the default schema to be used from the NMQL_Manager.
* When other agents are supported the FILENAME parameter
* would be the logical SQL$DATABASE and the logical would change
* depending on the agent.

```

```

*
* Set the transaction to READ/WRITE.
*/
EXEC SQL DECLARE SCHEMA FILENAME 'BRIDGE';
EXEC SQL SET TRANSACTION READ WRITE;

printf ("Program:  %s  --  Version:  %d  \nType  NMQL  ?  for
usage\n\n",argv[0],version);

if (argc == 1) {
    cmd_line_input = 'N';
    do_history();
    exit(0);
}

if ((j=strcmp(argv[1],"-d")) == 0) {
    debug = 1;
    if (argc < 3) {
        cmd_line_input = 'N';
        do_history();
        exit(0);
    }
}

if ((j=strcmp(argv[1],"?")) == 0) {
    usage();
    exit(0);
}
/*
* Set up the tmlst, the item list for SYS$TRNLNM
*/
lnm[1].items.buffer_length = sizeof(ret_string) - 1;
lnm[1].items.item_code = LNM$_STRING;
lnm[1].items.buffer_addr = ret_string;
lnm[1].items.ret_len_addr = &ret_len;
lnm[2].items.buffer_length = 4;
lnm[2].items.item_code = LNM$_ATTRIBUTES;
lnm[2].items.buffer_addr = &ret_attrib;
lnm[2].items.ret_len_addr = 0;
lnm[3].end_of_list = 0;

```

```

/*
 * Translate SYS$OUTPUT to its lowest equivalence name using the
 * SYS$TRNLNM system service.
 */
do {
    if (((status = SYS$TRNLNM(0,&table_name,&terminal,0,&lnm[1])) &1) != 1)
        LIB$STOP(status);
    /* Put result translation into a descriptor. */
    terminal.dsc$a_pointer = ret_string;
    terminal.dsc$w_length = ret_len;
    } while (!(LNM$M_TERMINAL & ret_attrib));

/* Check if process permanent file */
ret_string[ret_len] = '\0';
if ((result = strchr(ret_string, search)) != 0)
    {
        /* take off first four bytes. */
        if (((status = STR$RIGHT(&output, &terminal, &start_pos))
            &1) != 1)
            LIB$STOP(status);
        /* adjust length field of descriptor */
        output.dsc$w_length = output.dsc$w_length - 4;
    }

/* Open a log file for this application. */
nmql_log = fopen("nmql.log", "a+");
if (nmql_log == NULL) {
    printf("Bad log filename\n");
    err("log file");
    exit(-1);
}

/* Open a file to contain the user's input. */
sql_input = fopen("nmql.usr_input", "w");
if (sql_input == NULL) {
    printf("Bad input filename\n");
    err("sql user input file");
    exit(-1);
}

/*

```

```

* Check current time and print informational message to the log file.
*/

clocktime = time(0);
fprintf(nmq1_log, "\nNMQL started at %s", ctime(&clocktime));
fflush(nmq1_log);

/*
* Parse the command line input to determine if the statement is Standard
* SQL or if it is an NMQL Statment.
*/
stmt[0] = ' ';
stmt[1] = '\0';
ascii_int[0] = '0'; /* Place a "0 :" at the start so that VMS is happy */
ascii_int[1] = ' '; /* with the interval time string entered by the user. */
ascii_int[2] = '\0';
asc_strt[0] = '0'; /* Place a "0 :" at the start so that VMS is happy */
asc_strt[1] = ' '; /* with the interval time string entered by the user. */
asc_strt[2] = '\0';
asc_end[0] = '0'; /* Place a "0 :" at the start so that VMS is happy */
asc_end[1] = ' '; /* with the interval time string entered by the user. */
asc_end[2] = '\0';
def_intvl[0] = '0'; /* Build a default interval string of "0 :30:" mins. */
def_intvl[1] = ' ';
def_intvl[2] = ':';
def_intvl[3] = '3';
def_intvl[4] = '0';
def_intvl[5] = ':';
def_intvl[6] = '\0';
for (i=1; i < argc; i++) {

    tmp = str_toupper(argv[i]); /* Convert the string to upper case */

    if (j=strcmp(argv[i], "SELECT") == 0) {
        selstmt = 1;
        if (debug) printf("DEBUG: Found string select in command line\n");
    }
    if (j=strcmp(argv[i], "INSERT") == 0) {
        insertstmt = 1;
        if (debug) printf("DEBUG: Found string insert in command line\n");
    }
}

```

```

if (j==strcmp(argv[i],"DELETE") == 0) {
    deletestmt = 1;
    if (debug) printf("DEBUG: Found string delete in command line\n");
}

if (j==strcmp(argv[i],"START") == 0) {
    if (i < (argc - 1))
        tmp = str_toupper(argv[i+1]); /* Convert the string to upper case */
    else
    {
        printf ("\n*** Invalid syntax...\n\n");
        usage();
        exit(0);
    }
    if ((j==strcmp(argv[i+1],"NOW") == 0) ||
        (j==strcmp(argv[i+1],"NOW;") == 0))
    {
        realtime = 1;
        if (debug) printf("DEBUG: Realtime query is requested.\n");
    }
    else
    {
        if ((j = strchr(argv[i+1], ':')) == 0) {
            err("time string (need VMS Delta or Absolute time)");
        }
        k = strlen(argv[i+1]);
        j = strncat(asc_strt,argv[i+1],k);
        if (debug) printf ("\n DEBUG: Start Time = %s",asc_strt);
        realtime = 2;
    }
}

if (j==strcmp(argv[i],"REMOVE") == 0)
{
    if (i < (argc - 1))
    {
        tmp = str_toupper(argv[i+1]); /* Convert the string to upper case */
        j = strcpy (agent, *argv[i+1]); /* agent to be spoken with... */
        if (debug) printf ("\n DEBUG: Removal of Agent %s selected\n",agent);
        realtime = 3;
    }
    else

```

```

    {
        printf ("\n*** Invalid syntax...\n\n");
        usage();
        exit(0);
    }
}
if (j==strcmp(argv[i],"EVERY") == 0) {
    if (i < (argc - 1))
    {
        k = strlen(argv[i+1]);
        j = strncat(ascii_int,argv[i+1],k);
        if (debug) printf ("\n DEBUG: Interval Time = %s\n",ascii_int);
        user_interval = 1;
    }
    else
    {
        printf ("\n*** Invalid syntax...\n\n");
        usage();
        exit(0);
    }
}
if (j==strcmp(argv[i],"FROM") == 0) {
    if (i < (argc - 1))
    {
        tmp = str_toupper(argv[i+1]); /* Convert the string to upper case */
        j = strcpy (agent, argv[i+1]); /* agent to be spoken with... */
        if (debug) printf ("\nDEBUG: Agent = %s\n",agent);
    }
    else
    {
        printf ("\n*** Invalid syntax...\n\n");
        usage();
        exit(0);
    }
}
if (j==strcmp(argv[i],"INTO") == 0) {
    if (i < (argc - 1))
    {
        tmp = str_toupper(argv[i+1]); /* Convert the string to upper case */
        j = strcpy (agent, argv[i+1]); /* agent to be spoken with... */
    }
}

```

```

    else
    {
        printf ("\n*** Invalid syntax...\n\n\n");
        usage();
        exit(0);
    }
}

l = strcat(stmt,argv[i]);
l = strcat(stmt," ");
if (debug) printf("\nDEBUG: Constructed Stmt: %s\n",stmt);
}

fprintf(sql_input,"%s\n",stmt);
fclose(sql_input); /* close the input file so interactive SQL can read it */

/*****
* Create the sub process to query the bridge. *
* if this were a true manager, multiple      *
* processes might have to be created.        *
* If the process exists, continue.,.         *
*****/
jpi[1].items.item_code = JPI$_ACCOUNT;
    printf("\n%s Agent process created with default wake-up interval.\n",
        agent);
jpi[1].items.buffer_length = sizeof(ret_string) - 1;
jpi[1].items.buffer_addr = ret_string;
    printf("\n%s Agent process created with default wake-up interval.\n",
        agent);
jpi[1].items.ret_len_addr = &ret_len;
jpi[2].end_of_list = 0;
if (((proc_status = SYS$GETJPI( 0,0, &sub_name, &jpi[1], 0,0,0)) &1)
    != SS$_NORMAL)
{
    if (debug) printf("\n DEBUG: proc_status = %d", proc_status);
    if (((proc_status = SYS$CREPRC(&bridge_process_id, &sub_name, &output,
        &terminal, &output, 0, 0, &sub_name, 4,
        0, 0, PRC$_M_HIBER)) &1) != 1) LIB$STOP(proc_status);
    printf("\n%s Agent process created...\n",
        agent);
    if (debug) printf("\n DEBUG: Process ID of prototype NMQL_Agent is: %x\n",
        bridge_process_id);
}

```

```

    fprintf(nmql_log, "\n Process ID of prototype NMQL_Agent is: %x\n",
        bridge_process_id);
    fflush(nmql_log);
}

switch (realtime) {
case 1:
    /******
    * Realtime query of the specified agent was selected.      *
    * If user provided a specified "wake up" interval, use it.*
    * Strip the semicolon off of the "ascii_int" and send it. *
    * Otherwise use the default time of 30 minutes.           *
    *****/

/* Depending on the agent specified in the query wake up the proper agent */
if (j==strcmp(agent,"BRIDGE") == 0) {
    if (user_interval) {
        j = strlen(ascii_int);
        if (ascii_int[j-1] = ';') ascii_int[j-1] = '\0';
    }
    else j = strncpy(ascii_int,&def_intvl,7);
    if (debug) printf("DEBUG: Realtime query was selected,interval: %s\n",
        ascii_int);
    fprintf(nmql_log, "Realtime query was selected, interval: %s\n",ascii_int);
    fflush(nmql_log);
    if (insertstmt) {
        i = add_to_agent();
    }
    if (deletestmt) {
        i = del_from_agent();
    }
    if (selstmt) {
        i = get_real(ascii_int);
    }
    if (i < 0) err("get_real function");
}
else {
    printf ("\n\n*** Specified Agent %s not currently supported...\n",agent);
    fsync(stdout);
}
break;

```

case 2:

```
/******  
 * Change the next scheduled wakeup time for the specifed agent *  
 * and, if necessary, the repetition interval. *  
*****/  
  
/* Depending on the agent specified in the query wake up the proper agent */  
if (j==strcmp(agent,"BRIDGE") == 0) {  
    strt_dsc.dsc$w_length = strlen( asc_strt );  
    strt_dsc.dsc$a_pointer = asc_strt;  
    strt_dsc.dsc$b_class = DSC$K_CLASS_S;  
    strt_dsc.dsc$b_dtype = DSC$K_DTYPE_T;  
    if (((proc_status = SYS$BINTIM(&strt_dsc,daytim)) &1) !=1)  
        LIB$STOP(proc_status);  
  
    if (user_interval) {  
        intvl_dsc.dsc$w_length = strlen( ascii_int );  
        intvl_dsc.dsc$a_pointer = ascii_int;  
        intvl_dsc.dsc$b_class = DSC$K_CLASS_S;  
        intvl_dsc.dsc$b_dtype = DSC$K_DTYPE_T;  
        if (((proc_status = SYS$BINTIM(&intvl_dsc,reptim)) &1) !=1)  
            LIB$STOP(proc_status);  
        if (((proc_status = SYS$SCHDWK(&bridge_process_id,0,&daytim,&reptim  
    ))  
        &1) != 1) LIB$STOP(proc_status);  
        printf("\n%s Agent next wake-up time: %s.\n",  
            agent,asc_strt);  
        printf("\n%s Agent process wake-up interval: %s.\n",  
            agent,ascii_int);  
        fsync(stdout);  
    }  
    else {  
        if (((proc_status = SYS$SCHDWK(&bridge_process_id,0,&daytim,0))  
            &1) != 1) LIB$STOP(proc_status);  
        printf("\n%s Agent next wake-up time: %s.\n",  
            agent,asc_strt);  
        printf("\n%s Agent process wake-up interval has been set to null.\n",  
            agent);  
        fsync(stdout);  
    }  
}
```

```

else {
    printf ("\n\n*** Specified Agent %s not currently supported...\n",agent);
    fsync(stdout);
}
break;

case 3:
    /******
    * Remove the installed Agent image from the system. *
    * Cancel scheduled wakeup for the subprocess and    *
    * force it to exit.                                  *
    *****/
    /* Depending on the agent specified in the query wake up the proper agent */
    if (j==strcmp(agent,"BRIDGE") == 0) {
        if (((proc_status = SYS$CANWAK(&bridge_process_id, 0 )) &1) != 1)
            LIB$STOP(proc_status);

        /******
        * Force the subprocess to exit. *
        *****/
        if (((proc_status = SYS$FORCEX(&bridge_process_id,0,0)) &1) != 1)
            LIB$STOP(proc_status);
        printf("\n%s Agent process has been uninstalled.\n",
            agent);
        fsync(stdout);
    }
    else {
        printf ("\n\n*** Specified Agent %s not currently supported...\n",agent);
        fsync(stdout);
    }
    break;

default:
    /******
    * Otherwise, assume a valid SQL statement was *
    * entered and pass the command to SQL. If no *
    * command was entered spawn SQL.              *
    *****/
    if(debug)printf("DEBUG: Assumed a SQL stmt was entered: \nStmt: %s\n",

```

```

        stmt);
    fprintf(nmql_log,"Assumed a SQL statement was entered: \nStmt: %s\n",stmt);
    fflush(nmql_log);
    i = do_history();
    if (i < 0) err("do_history function");
}

sys$exit(1);
} /* End Main driver for NMQL */

```

```

err(string)
char *string;
{
    fprintf(stderr,"** Error: %s in NMQL **\n",string);
    fprintf(nmql_log,"** Error: %s in NMQL **\n",string);
    fflush(nmql_log);
    return;
}

```

```

usage()
{
    printf("Usage: NMQL [-d] NMQL_statement\n");
    printf("\n\n    d    - Turns on the NMQL DEBUG Statments");
    printf("\n    ?    - Types this message");
    printf("\n    DELETE - Used to specify which parameters to delete from a
domain");
    printf("\n                (Realtime deletion not supported)");
    printf("\n                (LANBridge Learning would have to be turned off)");
    printf("\n    EVERY - Used to specify the repetition interval for an agent");
    printf("\n                Time value form: HH:MM:SS");
    printf("\n    FROM - Used in conjunction with DELETE to specify the");
    printf("\n                management domain");
    printf("\n                Currently, the following domains are supported:");
    printf("\n                BRIDGE");
    printf("\n    INSERT - Used to specify which parameters to add to a domain");
    printf("\n                (Realtime insertion not supported)");
    printf("\n                (LANBridge Learning would have to be turned off)");
}

```

```

    printf("\n      INTO    - Used in conjunction with INSERT to specify the
management domain");
    printf("\n      Currently, the following domains are supported:");
    printf("\n      BRIDGE");
    printf("\n      NOW    - Only has meaning as a parameters to the START verb");
    printf("\n      Used to specify realtime queries");
    printf("\n      REMOVE - Used to remove an agent task from memory");
    printf("\n      SELECT - Used to specify which parameters to list");
    printf("\n      (Only * is supported with realtime queries)");
    printf("\n      START  - Used to specify when to initiate a query to an agent");
    printf("\n      Time value form: HH:MM:SS\n\n\n");
    printf("For further information, please consult any SQL manual\n");
    printf("or the NMQL thesis report\n");
    fsync(stdout);
    return(0);
}

```

```

get_real(ascii_int_addr)
char  *ascii_int_addr;

```

```

{
int j;

```

```

    $DESCRIPTOR(ascii_time,ascii_int_addr);
    if (debug) printf ("DEBUG: Ascii time interval: %s",ascii_int_addr);

```

```

/* Depending on the agent specified in the query wake up the proper agent */
if (j==strcmp(agent,"BRIDGE") == 0) {
    /*****
    * Schedule the subprocess to wake up NOW and every "ascii_time"  *
    * intervals                                                    *
    *****/
    if (((proc_status = SYS$BINTIM(&ascii_time, interval)) &1) != 1)
        LIB$STOP(proc_status);
    if (((proc_status = SYS$GETTIM(&current_time)) &1) != 1)
        LIB$STOP(proc_status);
    if (((proc_status = SYS$SCHDWK(&bridge_process_id, 0, current_time,
        interval)) &1) != 1) LIB$STOP(proc_status);

```

```

/*****
* Associate to cluster NMQL and wait for EFN 70. *
*****/
if (((proc_status = SYS$ASCEFC(64, &cluster_name, 0,0)) &1) != 1)
    LIB$STOP(proc_status);

if (debug)
    printf("\nDEBUG: Waiting for hibered process to set Flag 70...\n");

fprintf(nmql_log,
        "Waiting for hibernated process to set Event Flag 70...\n");
if (((proc_status = SYS$WAITFR(70)) &1) != 1) LIB$STOP(proc_status);
if (debug) {
    printf("DEBUG: Event flag 70 has been set by the sub-process.\n");
}
fprintf(nmql_log,"Event flag 70 has been set by the sub-process.\n");
}
else {
    printf ("\n\n*** Specified Agent %s not currently supported...\n",agent);
    fsync(stdout);
}
return(0);
}

```

```

do_history()
{
$DESCRIPTOR(nmql_prompt, "NMQL>");
register status;
FILE *temp_fd;
char *cptr;
int i;
/*
* Spawn the SQL_SUB process to collect historical information from
* the database.
*/
if (debug) printf("DEBUG: cmd_line_input = %c\n",cmd_line_input);
/*
* User put nothing on cmd_line

```

```

*/
if ((i = strcmp(cmd_line_input,"N")) == 0) {
    printf ("\nEnter SQL... Type HELP for help.\n\n");
    if (((status = LIB$SPAWN(&sql,&terminal,&terminal,0,&sql_sub,
        0,0,0,0,0,&nmql_prompt,0)) &1) != 1) LIB$STOP(status);
}
else {
/*
* Write a temporary file and pass that file as input to SQL.
*/
    temp_fd = fopen("USER_INPUT.DAT", "w");
    if (temp_fd == NULL) err("could not open USER_INPUT.DAT");
    cptr = fputs (stmt, temp_fd);
    fclose(temp_fd);
    printf ("Entering the NMQL historical database; type HELP for Help.\n");
    if (((status = LIB$SPAWN(&sql,&input_sql,&terminal,0,&sql_sub,
        0,0,0,0,0,0)) &1) != 1) LIB$STOP(status);
    /*******
    * Delete the user-input temporary file. *
    *****/
    temp_fd = fopen("USER_INPUT.DAT", "r", "fop=dlt");
    fclose(temp_fd); /* Delete the temporary file. */
}
return(status);
} /* End of do_history */

```

```

add_to_agent()
{
char *spos,*epos,addr[25],line;
int j;
register status;
    /* open a file that will contain the actual command string executed
    * by DCL.
    */
    if (agent == "BRIDGE") {
        printf("\n\n*** INSERT function not supported on bridges...");
        printf("\n      Learning must be turned off on bridges");
        fsync(stdout);
    }
/*

```

```

    rbmstxt = fopen("RBMS_ADD.COM", "w");
    spos = strchr(stmt, '-');
    j = strncpy(addr, spos - 2, 18);
    spos = strchr(stmt, ' ');
    j = strncpy(line, spos - 1, 1);
*/
/* put the command string into the file */
/*
    fprintf(rbmstxt, "$RBMS  ADD  PHYSICAL  ADDRESS  %s  LINE
%s\n", addr, line);
    fclose(rbmstxt);
    if (((status = LIB$SPAWN(&dcl, &rbms_add, &terminal, 0, 0, 0, 0, 0, 0, 0)) & 1)
!= 1) LIB$STOP(status);
    printf("The command has been sent to the agent for execution:\n");
    printf("command:  RBMS  ADD  PHYSICAL  ADDRESS  %s  LINE
%s\n", addr, line);
*/
}
else {
    printf("\n\n*** Agent not supported...");
    fsync(stdout);
}
return(0);
}

del_from_agent()
{
char *spos, *epos, addr[25], line;
int j;
register status;
/* open a file that will contain the actual command string executed
* by DCL.
*/
if (agent == "BRIDGE") {
    printf("\n\n*** DELETE function not supported on bridges...");
    printf("\n    Learning must be turned off on bridges");
    fsync(stdout);
}
/*
    rbmstxt = fopen("RBMS_DEL.COM", "w");
    spos = strchr(stmt, '-');

```

```

        j = strncpy(addr,spos - 2,18);
        spos = strchr(stmt,')');
        j = strncpy(line,spos-1,1);
    */
    /* put the command string into the file */
    /*
        fprintf(rbmstxt,"$RBMS  REMOVE  PHYSICAL  ADDRESS  %s  LINE
%s\n",addr,line);
        fclose(rbmstxt);
        if (((status = LIB$SPAWN(&dcl,&rbms_del,&terminal,0,0,0,0,0,0,0,0,0)) &1)
!= 1) LIB$STOP(status);
        printf("The command has been sent to the agent for execution:\n");
        printf("command:  RBMS  REMOVE  PHYSICAL  ADDRESS  %s  LINE
%s\n",addr,line);
    */
    }
    else {
        printf("\n\n*** Agent not supported...");
        fsync(stdout);
    }
    return(0);
}

```

```

str_toupper(string) /* Converts a string to uppercase */
char *string;
{
    char c,uc;
    int i,j,k;

        j = strlen(string);
        for (i=0; i < j; i++) {
            c = *(string+i);
            uc = toupper((int)c);
        /*
            printf("\n C= %c UC = %c\n",c,uc);
        */
            k = strncpy((string+i),&uc,1);
        }
    }
}

```

Appendix C
Source Code for NMQL_Agent.sc

```
/** NMQL_AGENT.SC
```

```
*
```

```
* This sub-process is created by NMQL.C in a hibernating state with a
```

```
* reschedule interval determined either by default or by command line.
```

```
* Event flag 70 is used to indicate completion of subprocess functions.
```

```
*
```

```
* The job of this process is to get the LANBridge counters and address data,
```

```
* parse the data and insert it into an SQL database.
```

```
*
```

```
* Copyright 1990, Gary W. Haney and Jeffrey D. Case
```

```
* University of Tennessee, Knoxville
```

```
*/
```

```
#include stdio
```

```
#include descrip
```

```
#include ssdef
```

```
#include ctype
```

```
#include unixio
```

```
#include file
```

```
#include prcdef
```

```
FILE *nmql_agent_log, *fopen();
```

```
char aline[90];
```

```
char *spos,*epos,*lpos;
```

```
int vms_return_status;
```

```
FILE *lb_out_fd, *lb_addr_fd, *temp_fd;
```

```
int open();
```

```
/* Declare the SQLCA */
```

```
EXEC SQL INCLUDE SQLCA;
```

```
int SQLCODE;
```

```
/*
```

```
* Declare the Schema to be used for storing of the LANBridge 100
```

```
* information
```

```
*/
```

```
EXEC SQL DECLARE SCHEMA FILENAME 'BRIDGE';
```

```
/* Structure for the lan bridge 100 counters query */
```

```

/*
*NOTE: This type of structure (below) can not be used when interfacing to SQL
*      All variables referenced by SQL must be scalar.  What a bummer.
*
*struct lb100_counters {
*  char id[20];
*  char addr[20];
*  char bin_date[24];
*  struct line {
*    int seconds;
*    int inv_prot_msgs;
*    int fil_frms;
*    int bad_frms;
*    int life_excd;
*    int no_fwd_frms;
*    int fwd_trans_cnt;
*    int collis;
*    struct xmt_rcv {
*      int frms;
*      int bdg_frms;
*      int frms_lost;
*    } x_r[2];
*  } ln[2];
*}lb100_cnt;
*/

```

```

struct
{
  char bridge_id[16];
  char bridge_addr[18];
/* Line 1 counters.      */
  char ln1_bin_date_var[24];
  int ln1_bridge_seconds;
  int ln1_fwding_state;
  int ln1_port_restrt;
  int ln1_unkn_dest_addr;
  int ln1_frms_filt;
  int ln1_mult_frms_filt;
  int ln1_frms_addr_filt;
  int ln1_frms_proto_filt;
  int ln1_tot_frms_sent;

```

```

int ln1_tot_frms_rcv;
int ln1_tot_bytes_sent;
int ln1_tot_bytes_rcv;
int ln1_dev_frms_sent;
int ln1_dev_frms_rcv;
int ln1_dev_bytes_sent;
int ln1_dev_bytes_rcv;
int ln1_dev_frms_lost;
int ln1_mult_frms_sent;
int ln1_mult_frms_rcv;
int ln1_mult_bytes_sent;
int ln1_mult_bytes_rcv;
int ln1_mult_dev_frms_sent;
int ln1_mult_dev_frms_rcv;
int ln1_mult_dev_bytes_sent;
int ln1_mult_dev_bytes_rcv;
int ln1_inv_dev_msg_cnt;
int ln1_bad_frms_rcv;
int ln1_bad_bytes_rcv;
int ln1_xmt_frms_lost;
int ln1_rcv_frms_lost;
int ln1_bad_hello_lim_exc;
int ln1_collis_tst_fail;
int ln1_xmt_frms_err;
int ln1_mult_src_frms_rcv;
int ln1_lifetm_exc_frms;
int ln1_lifetm_exc_bytes;
int ln1_xmt_bytes_err;
int ln1_inv_dev_bytes;
int ln1_rcv_overrun;
int ln1_oversize_frm;
int ln1_framing_err;
int ln1_xmt_mult_collis;
int ln1_carrier_loss;
int ln1_collis_lim_exc;
/* Repeat the same stuff for line 2 */
char ln2_bin_date_var[24];
int ln2_bridge_seconds;
int ln2_fwding_state;
int ln2_port_restrt;
int ln2_unkn_dest_addr;

```

```

int ln2_frms_filt;
int ln2_mult_frms_filt;
int ln2_frms_addr_filt;
int ln2_frms_proto_filt;
int ln2_tot_frms_sent;
int ln2_tot_frms_rcv;
int ln2_tot_bytes_sent;
int ln2_tot_bytes_rcv;
int ln2_dev_frms_sent;
int ln2_dev_frms_rcv;
int ln2_dev_bytes_sent;
int ln2_dev_bytes_rcv;
int ln2_dev_frms_lost;
int ln2_mult_frms_sent;
int ln2_mult_frms_rcv;
int ln2_mult_bytes_sent;
int ln2_mult_bytes_rcv;
int ln2_mult_dev_frms_sent;
int ln2_mult_dev_frms_rcv;
int ln2_mult_dev_bytes_sent;
int ln2_mult_dev_bytes_rcv;
int ln2_inv_dev_msg_cnt;
int ln2_bad_frms_rcv;
int ln2_bad_bytes_rcv;
int ln2_xmt_frms_lost;
int ln2_rcv_frms_lost;
int ln2_bad_hello_lim_exc;
int ln2_collis_tst_fail;
int ln2_xmt_frms_err;
int ln2_mult_src_frms_rcv;
int ln2_lifetm_exc_frms;
int ln2_lifetm_exc_bytes;
int ln2_xmt_bytes_err;
int ln2_inv_dev_bytes;
int ln2_rcv_overrun;
int ln2_oversize_frm;
int ln2_framing_err;
int ln2_xmt_mult_collis;
int ln2_carrier_loss;
int ln2_collis_lim_exc;
} lb_cnt;

```



```
short int non_blank;  
char vms_date_string[24];
```

```
char error_txt[301]; /* Used for SQL Error processing. */  
$DESCRIPTOR(error_txt_desc, error_txt);
```

```
$DESCRIPTOR (t_dsc,get_error_buffer);
```

```
printf("\nDetached Process NMQL_AGENT for LANBridge is executing\n");
```

```
EXEC SQL SET TRANSACTION READ WRITE;
```

```
/* Open a log file for this application. */  
nmql_agent_log = fopen("LB_AGENT.LOG", "a+");  
if (nmql_agent_log == NULL) {  
    printf("Bad log filename\n");  
    err("log file");  
    exit(-1);  
}
```

```
/*  
 * Check the current time and print an informational message to the log file.  
 */  
fprintf(nmql_agent_log, "\nLANBridge Agent received request at %s",  
        ctime(&clocktime));  
fflush(nmql_agent_log);
```

```
/*  
 * Associate to NMQL cluster and set Event flag 70 to let parent process  
 * know that the creation was a success.  
 */  
if (((status = SYS$ASCEFC(64,&cluster_name,0,0)) &1) != 1)  
    LIB$STOP(status);  
if (((status = SYS$SETEF(70)) &1) != 1) LIB$STOP(status);
```

```
/*  
 * Currently Digital Equipment Corporation's RBMS will be used to query the  
 * LANBridge for information. In a later release, ISO 8802.1 is intended  
 * to be utilized. The command file "BRIDGE_DUMP.COM" is used to query the  
 * LANBridge and dump the information to a file. Note: This process must
```

```

* be ran from a privileged account before RBMS can be utilized. Output is
* written the file "lb_output.dat"
*/
/***** Wanted to do this, but something is really kludgy about RBMS *****/
***** Had to resort to submitting the RBMS stuff as a batch job, *****
***** and re-submitting it every 30-minutes. *****
*/
printf("\n Creating a Detached RBMS...");
if (((status = SYS$CREPRC(pidadr, &image_name, &counter_dump, &rbms_log,
    &output,
    0, 0, &counter_sub, 6, 0, 0, PRC$M_DETACH)) &1) != 1)
    LIB$STOP(status);

/*
* Start a timer to display a message to the end user.
* When the file FINISHED.DAT is created, we can continue.
*/
init_timer:
    k=0;
try_again:
    temp_fd = fopen("FINISHED.DAT", "r", "rat=cr", "rfm=var", "fop=dlt");
    if (temp_fd == NULL) /* Not finished yet... */
    {
        k++;
        if (k == 1)
        {
            printf("Querying known LANBridges... ");
            goto try_again;
        }
        else
        {
            for (i=0; i < 150000; i++) i=i; /* Delay */
            if (k == 10)
            {
                printf(".");
                goto init_timer;
            }
            else goto try_again;
        }
    }
}

```

```

else
{
    fclose(temp_fd); /* Closing the file will also delete it */
}
if (debug) printf("\n\n Got past the Spawn \n");

/*
 * Open the output file produced by the SPAWN. Read the input, display to
 * the user, and insert it into the database. Let C know that this file is
 * a stream type file with line feed terminators, tell C to delete this f
 * file on closure.
 */
lb_out_fd = fopen("LB_OUTPUT.DAT","r","rat=cr","rfm=stmlf","fop=dlt");
if (lb_out_fd == NULL) {
    printf("No lb_output.dat file.\n");
    err("lb file");
    exit(-1);
}
if (debug) printf(" Got past output file open \n");
/*
 * Start the main loop. Get the first line (a blank line) for each of the
 * LANBridges in the file and check to see if we've reached the End of File.
 */
printf("\n Start processing known bridges counters...\n");

while ((cptr = fgets(aline, 80, lb_out_fd)) != NULL) /* First Blank Line */
{
    if (debug) printf("\n\naline = %s",&aline[0]);

    /* Get the date and the time. */
    cptr = fgets(aline, 80, lb_out_fd);
ln1: if (debug) printf("\n\naline = %s",&aline[0]);
    /*
     * Remember the INDEX function in C returns a pointer to Char not an
     * integer as with fortran. Here, because of VMS, usage of strchr is
     * mandatory.
     */
    spos = (strchr( &aline[0],'-')); /* Get the starting */
    spos = spos -2; /* position of the date. */
    j = strncpy(&date,spos,11);
    date[11] = '\0';

```

```

if (debug) printf("\nScanned date is: %s",&date);

spos = strchr(&aline[0],':');
j = strncpy (&time,spos - 2,8);          /* Get the starting */
time[8] = '\0';                          /* position of the time. */
j = strcpy(&vms_date_string,&date);       /* Convert the date and */
j = strcat(&vms_date_string," ");        /* time into one string. */
j = strcat(&vms_date_string,&time);
if (debug) printf("\nVMS Date STring: %s",&vms_date_string);
/*
* Convert an ASCII Date string into Binary date format for use
* within an SQL Database.
*/
j = strlen(&vms_date_string);
printf("\nDate String length: %d",j);
j = strcpy (&lb_cnt.ln1_bin_date_var, &vms_date_string);

cptr = fgets(aline, 80, lb_out_fd);       /* Read bridge ID and address */
if (debug) printf("\naline = %s",&aline[0]);
spos = strchr(&aline[0],' ');
epos = strchr(&aline[0],',');
j = strncpy (&lb_cnt.bridge_id, (spos + 1), (epos-spos));
lb_cnt.bridge_id[19] = '\0';
spos = (strchr(&aline[0],'-')) - 2;
j = strcpy (&lb_cnt.bridge_addr,spos);

printf("\n\nReading bridge statistics for %s Bridge: %s Line 1",
       lb_cnt.ln1_bin_date_var, lb_cnt.bridge_id);

cptr = fgets(aline, 80, lb_out_fd);       /* Blank Line */
cptr = fgets(aline, 80, lb_out_fd);       /* Blank Line */

cptr = fgets(aline, 80, lb_out_fd);       /* Get Bridge Seconds */
i = get_int(&aline,&lb_cnt.ln1_bridge_seconds);
if (debug) printf("\ni = %d Seconds= %d\n",i,lb_cnt.ln1_bridge_seconds);

cptr = fgets(aline, 80, lb_out_fd);       /* Get Forwarding State */
i = get_int(&aline,&lb_cnt.ln1_fwding_state);
if (debug) printf("\ni = %d Forwarding State = %d\n",
       i,lb_cnt.ln1_fwding_state);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Port Restarts */
i = get_int(&aline,&lb_cnt.ln1_port_restrt);
if (debug) printf("\ni = %d Port Restarts = %d\n",
    i,lb_cnt.ln1_port_restrt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Unknown Dest. Address */
i = get_int(&aline,&lb_cnt.ln1_unkn_dest_addr);
if (debug) printf("\ni = %d Unknown Dest. Address = %d\n",
    i,lb_cnt.ln1_unkn_dest_addr);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Filtered */
i = get_int(&aline,&lb_cnt.ln1_frms_filt);
if (debug) printf("\ni = %d Frames Filtered = %d\n",
    i,lb_cnt.ln1_frms_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multi. Frames Filtered */
i = get_int(&aline,&lb_cnt.ln1_mult_frms_filt);
if (debug) printf("\ni = %d Multicast Frames Filtered = %d\n",
    i,lb_cnt.ln1_mult_frms_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Addr. Filtered */
i = get_int(&aline,&lb_cnt.ln1_frms_addr_filt);
if (debug) printf("\ni = %d Frames Address Filtered = %d\n",
    i,lb_cnt.ln1_frms_addr_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Proto. Filtered */
i = get_int(&aline,&lb_cnt.ln1_frms_proto_filt);
if (debug) printf("\ni = %d Frames Protocol Filtered = %d\n",
    i,lb_cnt.ln1_frms_proto_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Frames Sent */
i = get_int(&aline,&lb_cnt.ln1_tot_frms_sent);
if (debug) printf("\ni = %d Total Frames Sent = %d\n",
    i,lb_cnt.ln1_tot_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Frames Received */
i = get_int(&aline,&lb_cnt.ln1_tot_frms_rcv);
if (debug) printf("\ni = %d Total Frames Received = %d\n",
    i,lb_cnt.ln1_tot_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Bytes Sent */

```

```

i = get_int(&aline,&lb_cnt.ln1_tot_bytes_sent);
if (debug) printf("\ni = %d Total Bytes Sent = %d\n",
    i,lb_cnt.ln1_tot_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Bytes Received */
i = get_int(&aline,&lb_cnt.ln1_tot_bytes_rcv);
if (debug) printf("\ni = %d Total Bytes Received = %d\n",
    i,lb_cnt.ln1_tot_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Sent */
i = get_int(&aline,&lb_cnt.ln1_dev_frms_sent);
if (debug) printf("\ni = %d Device Frames Sent = %d\n",
    i,lb_cnt.ln1_dev_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Rcv */
i = get_int(&aline,&lb_cnt.ln1_dev_frms_rcv);
if (debug) printf("\ni = %d Device Frames Received = %d\n",
    i,lb_cnt.ln1_dev_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Bytes Sent */
i = get_int(&aline,&lb_cnt.ln1_dev_bytes_sent);
if (debug) printf("\ni = %d Device Bytes Sent = %d\n",
    i,lb_cnt.ln1_dev_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Bytes Received */
i = get_int(&aline,&lb_cnt.ln1_dev_bytes_rcv);
if (debug) printf("\ni = %d Device Bytes Received = %d\n",
    i,lb_cnt.ln1_dev_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Lost */
i = get_int(&aline,&lb_cnt.ln1_dev_frms_lost);
if (debug) printf("\ni = %d Device Frames Lost = %d\n",
    i,lb_cnt.ln1_dev_frms_lost);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Frames Sent */
i = get_int(&aline,&lb_cnt.ln1_mult_frms_sent);
if (debug) printf("\ni = %d Multicast Frames Sent = %d\n",
    i,lb_cnt.ln1_mult_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Frames Rcv */
i = get_int(&aline,&lb_cnt.ln1_mult_frms_rcv);

```

```
if (debug) printf("\ni = %d Multicast Frames Received = %d\n",  
i,lb_cnt.ln1_mult_frms_rcv);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Multicast Bytes Sent */  
i = get_int(&aline,&lb_cnt.ln1_mult_bytes_sent);  
if (debug) printf("\ni = %d Multicast Bytes Sent = %d\n",  
i,lb_cnt.ln1_mult_bytes_sent);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Multicast Bytes Recv */  
i = get_int(&aline,&lb_cnt.ln1_mult_bytes_rcv);  
if (debug) printf("\ni = %d Multicast Bytes Received = %d\n",  
i,lb_cnt.ln1_mult_bytes_rcv);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Multi. Dev. Frms Sent */  
i = get_int(&aline,&lb_cnt.ln1_mult_dev_frms_sent);  
if (debug) printf("\ni = %d Multicast Device Frames Sent = %d\n",  
i,lb_cnt.ln1_mult_dev_frms_sent);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Mult. Dev. Frms. Recv */  
i = get_int(&aline,&lb_cnt.ln1_mult_dev_frms_rcv);  
if (debug) printf("\ni = %d Multicast Device Frames Received = %d\n",  
i,lb_cnt.ln1_mult_dev_frms_rcv);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Mult. Dev. Bytes Sent */  
i = get_int(&aline,&lb_cnt.ln1_mult_dev_bytes_sent);  
if (debug) printf("\ni = %d Multicast Device Bytes Sent = %d\n",  
i,lb_cnt.ln1_mult_dev_bytes_sent);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Mult. Dev. Bytes Recv */  
i = get_int(&aline,&lb_cnt.ln1_mult_dev_bytes_rcv);  
if (debug) printf("\ni = %d Multicast Device Bytes Received = %d\n",  
i,lb_cnt.ln1_mult_dev_bytes_rcv);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Invalid Dev. Msg. Cnt. */  
i = get_int(&aline,&lb_cnt.ln1_inv_dev_msg_cnt);  
if (debug) printf("\ni = %d Invalid Device Message Count = %d\n",  
i,lb_cnt.ln1_inv_dev_msg_cnt);
```

```
cptr = fgets(aline, 80, lb_out_fd); /* Get Bad Frames Received */  
i = get_int(&aline,&lb_cnt.ln1_bad_frms_rcv);  
if (debug) printf("\ni = %d Bad Frames Received = %d\n",
```

```

        i,lb_cnt.ln1_bad_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bad Bytes Received */
i = get_int(&aline,&lb_cnt.ln1_bad_bytes_rcv);
if (debug) printf("\ni = %d Bad Bytes Received = %d\n",
        i,lb_cnt.ln1_bad_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Trans. Frms Lost */
i = get_int(&aline,&lb_cnt.ln1_xmt_frms_lost);
if (debug) printf("\ni = %d Transmitted Frames Lost = %d\n",
        i,lb_cnt.ln1_xmt_frms_lost);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Recv. Frames Lost*/
i = get_int(&aline,&lb_cnt.ln1_rcv_frms_lost);
if (debug) printf("\ni = %d Received Frames Lost = %d\n",
        i,lb_cnt.ln1_rcv_frms_lost);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bad Hello Limit */
i = get_int(&aline,&lb_cnt.ln1_bad_hello_lim_exc);
if (debug) printf("\ni = %d Bad Hello Limit Exceeded = %d\n",
        i,lb_cnt.ln1_bad_hello_lim_exc);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Collision Test Fails */
i = get_int(&aline,&lb_cnt.ln1_collis_tst_fail);
if (debug) printf("\ni = %d Collision Test Fails = %d\n",
        i,lb_cnt.ln1_collis_tst_fail);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Trans. Frame Errors */
i = get_int(&aline,&lb_cnt.ln1_xmt_frms_err);
if (debug) printf("\ni = %d Transmitted Frame Errors = %d\n",
        i,lb_cnt.ln1_xmt_frms_err);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Mult. Source Frms Recv */
i = get_int(&aline,&lb_cnt.ln1_mult_src_frms_rcv);
if (debug) printf("\ni = %d Multicast Source Frames Received = %d\n",
        i,lb_cnt.ln1_mult_src_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Lifetime Exceeded Frms */
i = get_int(&aline,&lb_cnt.ln1_lifetm_exc_frms);
if (debug) printf("\ni = %d Lifetime Exceeded Frames = %d\n",
        i,lb_cnt.ln1_lifetm_exc_frms);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Lifetime Exceed Bytes */
i = get_int(&aline,&lb_cnt.ln1_lifetm_exc_bytes);
if (debug) printf("\ni = %d Lifetime Exceeded Bytes = %d\n",
    i,lb_cnt.ln1_lifetm_exc_bytes);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Transmitted Bytes Err */
i = get_int(&aline,&lb_cnt.ln1_xmt_bytes_err);
if (debug) printf("\ni = %d Transmitted Bytes Error = %d\n",
    i,lb_cnt.ln1_xmt_bytes_err);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Invalid Device bytes */
i = get_int(&aline,&lb_cnt.ln1_inv_dev_bytes);
if (debug) printf("\ni = %d Invalid Device Bytes = %d\n",
    i,lb_cnt.ln1_inv_dev_bytes);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Receive Overrun */
i = get_int(&aline,&lb_cnt.ln1_rcv_overrun);
if (debug) printf("\ni = %d Receive Overrun = %d\n",
    i,lb_cnt.ln1_rcv_overrun);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Oversize Frame */
i = get_int(&aline,&lb_cnt.ln1_oversize_frm);
if (debug) printf("\ni = %d Oversize Frame = %d\n",
    i,lb_cnt.ln1_oversize_frm);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Framing Error */
i = get_int(&aline,&lb_cnt.ln1_framing_err);
if (debug) printf("\ni = %d Framing Error = %d\n",
    i,lb_cnt.ln1_framing_err);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Transmit Mult. Collis */
i = get_int(&aline,&lb_cnt.ln1_xmt_mult_collis);
if (debug) printf("\ni = %d Transmit Multiple Collision = %d\n",
    i,lb_cnt.ln1_xmt_mult_collis);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Carrier Loss */
i = get_int(&aline,&lb_cnt.ln1_carrier_loss);
if (debug) printf("\ni = %d Carrier Loss = %d\n",
    i,lb_cnt.ln1_carrier_loss);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Collision Limit Exc */

```

```

i = get_int(&aline,&lb_cnt.ln1_collis_lim_exc);
if (debug) printf("\ni = %d Collision Limit Exceeded = %d\n",
    i,lb_cnt.ln1_collis_lim_exc);
/*
 * Read, at most, 4 blank lines. On the first
 * non-blank character start processing the counters line 2
 * of the bridge.
 */
j = 1;
non_blank = 0; /* reset blank line flag */
cptr = fgets(aline, 80, lb_out_fd); /* ?? Have to prime the loop ?? */
while ((cptr = fgets(aline, 80, lb_out_fd)) != NULL && j <= 4)
{
    j++;
    if (debug) printf("\nJ = %d\nAline= %s",j,&aline[0]);
    spos = &aline[0];
    if ((non_blank=isalpha((int)*spos)) != 0)goto ln2; /* Leave on 1st alpha */
}

/*****
 *
 * Now repeat the same stuff for the bridge's second line.
 *
 *****/

/* Get the date and the time. */
cptr = fgets(aline, 80, lb_out_fd);
ln2: if(debug)printf("\n\nThis line should have date & time...\naline = %s",
    &aline[0]);

spos = (strchr( &aline[0],'-')); /* Get the starting */
spos = spos -2; /* position of the date. */
j = strncpy(&date,spos,11);
date[11] = '\0';
if (debug) printf("\nLine 2 Scanned date is: %s",&date);

spos = strchr(&aline[0],':');
j = strncpy (&time,spos - 2,8); /* Get the starting */
time[8] = '\0'; /* position of the time. */
j = strcpy(&vms_date_string,&date); /* Convert the date and */

```

```

j = strcat(&vms_date_string, " ");          /* time into one string. */
j = strcat(&vms_date_string, &time);
if (debug) printf("\nVMS Date SString: %s", &vms_date_string);
/*
 * Convert an ASCII Date string into Binary date format for use
 * within an SQL Database.
 */
j = strcpy (&lb_cnt.ln2_bin_date_var, &vms_date_string);

cptr = fgets(aline, 80, lb_out_fd);          /* Read bridge ID and address */
cptr = fgets(aline, 80, lb_out_fd);          /* Blank Line */
cptr = fgets(aline, 80, lb_out_fd);          /* Blank Line */

printf("\n\nReading bridge statistics for %s Bridge: %s Line 2",
       lb_cnt.ln2_bin_date_var, lb_cnt.bridge_id);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bridge Seconds */
i = get_int(&aline, &lb_cnt.ln2_bridge_seconds);
if (debug) printf("\ni = %d Seconds= %d\n", i, lb_cnt.ln2_bridge_seconds);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Forwarding State */
i = get_int(&aline, &lb_cnt.ln2_fwding_state);
if (debug) printf("\ni = %d Forwarding State = %d\n",
                  i, lb_cnt.ln2_fwding_state);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Port Restarts */
i = get_int(&aline, &lb_cnt.ln2_port_restrt);
if (debug) printf("\ni = %d Port Restarts = %d\n",
                  i, lb_cnt.ln2_port_restrt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Unknown Dest. Address */
i = get_int(&aline, &lb_cnt.ln2_unkn_dest_addr);
if (debug) printf("\ni = %d Unknown Dest. Address = %d\n",
                  i, lb_cnt.ln2_unkn_dest_addr);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Filtered */
i = get_int(&aline, &lb_cnt.ln2_frms_filt);
if (debug) printf("\ni = %d Frames Filtered = %d\n",
                  i, lb_cnt.ln2_frms_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multi. Frames Filtered */

```

```

i = get_int(&aline,&lb_cnt.ln2_mult_frms_filt);
if (debug) printf("\ni = %d Multicast Frames Filtered = %d\n",
    i,lb_cnt.ln2_mult_frms_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Addr. Filtered */
i = get_int(&aline,&lb_cnt.ln2_frms_addr_filt);
if (debug) printf("\ni = %d Frames Address Filtered = %d\n",
    i,lb_cnt.ln2_frms_addr_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Frames Proto. Filtered */
i = get_int(&aline,&lb_cnt.ln2_frms_proto_filt);
if (debug) printf("\ni = %d Frames Protocol Filtered = %d\n",
    i,lb_cnt.ln2_frms_proto_filt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Frames Sent */
i = get_int(&aline,&lb_cnt.ln2_tot_frms_sent);
if (debug) printf("\ni = %d Total Frames Sent = %d\n",
    i,lb_cnt.ln2_tot_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Frames Received */
i = get_int(&aline,&lb_cnt.ln2_tot_frms_rcv);
if (debug) printf("\ni = %d Total Frames Received = %d\n",
    i,lb_cnt.ln2_tot_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Bytes Sent */
i = get_int(&aline,&lb_cnt.ln2_tot_bytes_sent);
if (debug) printf("\ni = %d Total Bytes Sent = %d\n",
    i,lb_cnt.ln2_tot_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Total Bytes Received */
i = get_int(&aline,&lb_cnt.ln2_tot_bytes_rcv);
if (debug) printf("\ni = %d Total Bytes Received = %d\n",
    i,lb_cnt.ln2_tot_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Sent */
i = get_int(&aline,&lb_cnt.ln2_dev_frms_sent);
if (debug) printf("\ni = %d Device Frames Sent = %d\n",
    i,lb_cnt.ln2_dev_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Recv */
i = get_int(&aline,&lb_cnt.ln2_dev_frms_rcv);

```

```

if (debug) printf("\ni = %d Device Frames Received = %d\n",
    i,lb_cnt.ln2_dev_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Bytes Sent */
i = get_int(&aline,&lb_cnt.ln2_dev_bytes_sent);
if (debug) printf("\ni = %d Device Bytes Sent = %d\n",
    i,lb_cnt.ln2_dev_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Bytes Received */
i = get_int(&aline,&lb_cnt.ln2_dev_bytes_rcv);
if (debug) printf("\ni = %d Device Bytes Received = %d\n",
    i,lb_cnt.ln2_dev_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Device Frames Lost */
i = get_int(&aline,&lb_cnt.ln2_dev_frms_lost);
if (debug) printf("\ni = %d Device Frames Lost = %d\n",
    i,lb_cnt.ln2_dev_frms_lost);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Frames Sent */
i = get_int(&aline,&lb_cnt.ln2_mult_frms_sent);
if (debug) printf("\ni = %d Multicast Frames Sent = %d\n",
    i,lb_cnt.ln2_mult_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Frames Rcv */
i = get_int(&aline,&lb_cnt.ln2_mult_frms_rcv);
if (debug) printf("\ni = %d Multicast Frames Received = %d\n",
    i,lb_cnt.ln2_mult_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Bytes Sent */
i = get_int(&aline,&lb_cnt.ln2_mult_bytes_sent);
if (debug) printf("\ni = %d Multicast Bytes Sent = %d\n",
    i,lb_cnt.ln2_mult_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multicast Bytes Rcv */
i = get_int(&aline,&lb_cnt.ln2_mult_bytes_rcv);
if (debug) printf("\ni = %d Multicast Bytes Received = %d\n",
    i,lb_cnt.ln2_mult_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Multi. Dev. Frms Sent */
i = get_int(&aline,&lb_cnt.ln2_mult_dev_frms_sent);
if (debug) printf("\ni = %d Muticast Device Frames Sent = %d\n",

```

```

        i,lb_cnt.ln2_mult_dev_frms_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Mult. Dev. Frms. Recv */
i = get_int(&aline,&lb_cnt.ln2_mult_dev_frms_rcv);
if (debug) printf("\ni = %d Multicast Device Frames Received = %d\n",
        i,lb_cnt.ln2_mult_dev_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Mult. Dev. Bytes Sent */
i = get_int(&aline,&lb_cnt.ln2_mult_dev_bytes_sent);
if (debug) printf("\ni = %d Multicast Device Bytes Sent = %d\n",
        i,lb_cnt.ln2_mult_dev_bytes_sent);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Mult. Dev. Bytes Recv */
i = get_int(&aline,&lb_cnt.ln2_mult_dev_bytes_rcv);
if (debug) printf("\ni = %d Multicast Device Bytes Received = %d\n",
        i,lb_cnt.ln2_mult_dev_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Invalid Dev. Msg. Cnt. */
i = get_int(&aline,&lb_cnt.ln2_inv_dev_msg_cnt);
if (debug) printf("\ni = %d Invalid Device Message Count = %d\n",
        i,lb_cnt.ln2_inv_dev_msg_cnt);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bad Frames Received */
i = get_int(&aline,&lb_cnt.ln2_bad_frms_rcv);
if (debug) printf("\ni = %d Bad Frames Received = %d\n",
        i,lb_cnt.ln2_bad_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bad Bytes Received */
i = get_int(&aline,&lb_cnt.ln2_bad_bytes_rcv);
if (debug) printf("\ni = %d Bad Bytes Received = %d\n",
        i,lb_cnt.ln2_bad_bytes_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Trans. Frms Lost */
i = get_int(&aline,&lb_cnt.ln2_xmt_frms_lost);
if (debug) printf("\ni = %d Transmitted Frames Lost = %d\n",
        i,lb_cnt.ln2_xmt_frms_lost);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Recv. Frames Lost*/
i = get_int(&aline,&lb_cnt.ln2_rcv_frms_lost);
if (debug) printf("\ni = %d Received Frames Lost = %d\n",
        i,lb_cnt.ln2_rcv_frms_lost);

```

```

cptr = fgets(aline, 80, lb_out_fd);          /* Get Bad Hello Limit */
i = get_int(&aline,&lb_cnt.ln2_bad_hello_lim_exc);
if (debug) printf("\ni = %d Bad Hello Limit Exceeded = %d\n",
    i,lb_cnt.ln2_bad_hello_lim_exc);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Collision Test Fails */
i = get_int(&aline,&lb_cnt.ln2_collis_tst_fail);
if (debug) printf("\ni = %d Collision Test Fails = %d\n",
    i,lb_cnt.ln2_collis_tst_fail);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Trans. Frame Errors */
i = get_int(&aline,&lb_cnt.ln2_xmt_frms_err);
if (debug) printf("\ni = %d Transmitted Frame Errors = %d\n",
    i,lb_cnt.ln2_xmt_frms_err);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Mult. Source Frms Recv */
i = get_int(&aline,&lb_cnt.ln2_mult_src_frms_rcv);
if (debug) printf("\ni = %d Multicast Source Frames Received = %d\n",
    i,lb_cnt.ln2_mult_src_frms_rcv);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Lifetime Exceeded Frms */
i = get_int(&aline,&lb_cnt.ln2_lifetm_exc_frms);
if (debug) printf("\ni = %d Lifetime Exceeded Frames = %d\n",
    i,lb_cnt.ln2_lifetm_exc_frms);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Lifetime Exceed Bytes */
i = get_int(&aline,&lb_cnt.ln2_lifetm_exc_bytes);
if (debug) printf("\ni = %d Lifetime Exceeded Bytes = %d\n",
    i,lb_cnt.ln2_lifetm_exc_bytes);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Transmitted Bytes Err */
i = get_int(&aline,&lb_cnt.ln2_xmt_bytes_err);
if (debug) printf("\ni = %d Transmitted Bytes Error = %d\n",
    i,lb_cnt.ln2_xmt_bytes_err);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Invalid Device bytes */
i = get_int(&aline,&lb_cnt.ln2_inv_dev_bytes);
if (debug) printf("\ni = %d Invalid Device Bytes = %d\n",
    i,lb_cnt.ln2_inv_dev_bytes);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Receive Overrun */

```

```

i = get_int(&aline,&lb_cnt.ln2_rcv_overrun);
if (debug) printf("\ni = %d Receive Overrun = %d\n",
    i,lb_cnt.ln2_rcv_overrun);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Oversize Frame */
i = get_int(&aline,&lb_cnt.ln2_oversize_frm);
if (debug) printf("\ni = %d Oversize Frame = %d\n",
    i,lb_cnt.ln2_oversize_frm);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Framing Error */
i = get_int(&aline,&lb_cnt.ln2_framing_err);
if (debug) printf("\ni = %d Framing Error = %d\n",
    i,lb_cnt.ln2_framing_err);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Transmit Mult. Collis */
i = get_int(&aline,&lb_cnt.ln2_xmt_mult_collis);
if (debug) printf("\ni = %d Transmit Multiple Collision = %d\n",
    i,lb_cnt.ln2_xmt_mult_collis);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Carrier Loss */
i = get_int(&aline,&lb_cnt.ln2_carrier_loss);
if (debug) printf("\ni = %d Carrier Loss = %d\n",
    i,lb_cnt.ln2_carrier_loss);

cptr = fgets(aline, 80, lb_out_fd);          /* Get Collision Limit Exc */
i = get_int(&aline,&lb_cnt.ln2_collis_lim_exc);
if (debug) printf("\ni = %d Collision Limit Exceeded = %d\n",
    i,lb_cnt.ln2_collis_lim_exc);

/* Insert the values into the database by transferring an */
/* entire record into the SQL Database. */
EXEC SQL WHENEVER SQLERROR GO TO sql_err_handler;
EXEC SQL INSERT INTO BRIDGE_TABLE VALUES
    (:lb_cnt);

/*
* Read, at most five blank lines. These lines should all be
* blank and should be at the bottom of each bridge file.
* Terminate on a NULL.
*/
j = 0;

```

```

non_blank = 0; /* reset blank line flag */
cptr = fgets(aline, 80, lb_out_fd); /* ?? Have to prime the loop ?? */
while ((cptr = fgets(aline, 80, lb_out_fd)) != NULL && j < 4)
{
    j++;
    if (debug) printf("\nJ = %d\nAline= %s",j,&aline[0]);
    spos = &aline[0];
    if ((non_blank = isalpha((int)*spos)) != 0)
    {
        goto ln1; /* Leave on 1st alpha */
    }
}
} /* End of Counter Loop */

fclose(lb_out_fd); /* Close the counters input file produced by RBMS */

printf("\n Bridge Counters have been read...");
printf("\n Now reading learned addresses...");
fsync(stdout);

/*
 * Open the output file produced by the SPAWN. Read the input, display to
 * the user, and insert it into the database. Let C know that this file is
 * a variable length type and tell C to delete this file on closure.
 */
lb_addr_fd = fopen("LB_ADDR.DAT", "r", "rfm=var", "fop=dlt");
if (lb_addr_fd == NULL)
{
    printf("No lb_addr.dat file.\n");
    err("lb addr file");
    exit(-1);
}
if (debug) printf(" \n\nGot past addr file open... \n");

cptr = fgets(aline, 85, lb_addr_fd) ;/* One Blank line at start of file */
if (debug) printf("\n\naline = %s***",&aline[0]);

/* Start process the learned addresses file */
while ((cptr = fgets(aline, 85, lb_addr_fd)) != NULL)
{
    if (debug) printf("\n\naline = %s",&aline[0]);

```

```

/*
 * Remember the INDEX function in C returns a pointer to Char not an
 * integer as with fortran. Here, because of VMS, usage of strchr is
 * mandatory.
 */
spos = (strchr( &aline[0],'-'));          /* Get the starting      */
spos = spos -2;                          /* position of the date. */
j = strncpy(&date,spos,11);
date[11] = '\0';
if (debug) printf("\nScanned date is: %s",&date);
spos = strchr(&aline[0],':');
j = strncpy (&time,spos - 2,8);          /* Get the starting      */
time[8] = '\0';                          /* position of the time. */
j = strcpy(&vms_date_string,&date);      /* Convert the date and  */
j = strcat(&vms_date_string," ");        /* time into one string. */
j = strcat(&vms_date_string,&time);
if (debug) printf("\nVMS Date STring: %s",&vms_date_string);
/*
 * Convert an ASCII Date string into Binary date format for use
 * within an SQL Database.
 */
j = strcpy (&lb_addrs.cbindate, &vms_date_string);

cptr = fgets(aline, 85, lb_addr_fd);      /* Read bridge ID and address */
if (debug) printf("\naline = %s",&aline[0]);
spos = strchr(&aline[0], ' ');
epos = strchr(&aline[0],',');
j = strncpy (&lb_addrs.cid, (spos + 1), (epos-spos));
lb_addrs.cid[19] = '\0';
spos = (strchr(&aline[0],'-')) - 2;
j = strcpy (&lb_addrs.cbaddr,spos);

printf("\n Reading physical addresses for Bridge: %s",&lb_addrs.cid);

cptr = fgets(aline,85,lb_addr_fd);        /* Don't care data */
if (debug) printf("\naline = %s",&aline[0]);
cptr = fgets(aline,85,lb_addr_fd);        /* Don't care data */
if (debug) printf("\naline = %s",&aline[0]);
cptr = fgets(aline,85,lb_addr_fd);        /* Don't care data */
if (debug) printf("\naline = %s",&aline[0]);
cptr = fgets(aline,85,lb_addr_fd);        /* Don't care data */

```

```

if (debug) printf("\naline = %s",&aline[0]);

/*
 * For each bridge, read in the learned addresses and store them as
 * a row in the Lrnd_Addrs Table.
 */
non_blank = 0; /* reset the non_blank flag */
while ((cptr = fgets( aline, 85, lb_addr_fd)) > 0)
{
    spos = &aline[0];
    if ((non_blank = isalpha((int)*spos)) != 0 ||
        (non_blank = isdigit((int)*spos)) != 0)
    {
        strncpy(lb_addrs.laddr,&aline[0],17);
        i = get_int(&aline[18], &lb_addrs.on_line);
        if (debug) {
            printf ("\n Aline: %s",&aline[0]);
            printf ("\n Learned Address: %s on line %d\n",
                &lb_addrs.laddr,lb_addrs.on_line);
        }
        EXEC SQL INSERT INTO LRND_ADDRS VALUES
            (:lb_addrs);
        if (SQLCODE < 0)
        {
            printf("\nINSERT failed. Attempting ROLLBACK.");
            SQL$SIGNAL();
            EXEC SQL ROLLBACK;
            if (SQLCODE < 0)
            {
                printf("\nROLLBACK failed. Delete database files manually.");
                SQL$SIGNAL();
            }
        }
    }
    else
    {
        if (debug) printf("\nNon_blnk= %d\naline=%s***",non_blank,&aline[0]);
        break; /* Leave on 1st line with space as 1st char */
    }
}
} /* end of learned addresses while loop */

```

```

EXEC SQL COMMIT;
if (SQLCODE < 0)
{
    printf ("\nCOMMIT failed. Attempting ROLLBACK.");
    SQL$SIGNAL();
    EXEC SQL ROLLBACK;
    if (SQLCODE < 0)
    {
        printf("\nROLLBACK failed. Delete Database files manually.");
        SQL$SIGNAL();
    }
}

/*
 * Close the remaining open files and
 * exit normally.
 */
fprintf(nmql_agent_log, "\nLANBridge Agent completed request at %s",
        ctime(&clocktime));
fflush(nmql_agent_log);

fclose(lb_addr_fd);
fclose(nmql_agent_log);
/*
 * Check to see if any stragglers remain (i.e., the batch job has
 * executed more than once since the last agent). If so, delete them.
 */
while ((lb_out_fd = fopen("LB_ADDR.DAT", "r", "rfm=var", "fop=dlt"))
        != NULL) fclose(lb_out_fd);
while ((lb_addr_fd = fopen("LB_ADDR.DAT", "r", "rfm=var", "fop=dlt"))
        != NULL) fclose(lb_addr_fd);
while (((temp_fd = fopen("FINISHED.DAT", "r", "rat=cr", "rfm=var", "fop=dlt"))
        != NULL) fclose(temp_fd);

printf("\n Database files updated... NMQL_Agent exiting...\n\n\n");
fsync(stdout);
SYS$EXIT(1); /* Bye, Bye!!! */

/*
 * Error handler to print out any potential problem from either
 * SQL or RDB.

```

```

*/
sql_err_handler:

    SQL$GET_ERROR_TEXT (&error_txt_desc);
    printf ("/n/n Error text: %s", &error_txt);
    SQL$SIGNAL();

} /* End of Main */


err(string)
char *string;
{
    fprintf(stderr,"** Error: %s in NMQL **\n",string);
    fprintf(nmql_agent_log,"** Error: %s in NMQL **\n",string);
    fflush(nmql_agent_log);
    return;
}


int
get_int(afile,intvalue)
char *afile;
int *intvalue;
{
    int istat = 0, i = 0;

    while (istat == 0) {
        istat = isdigit((int)*afile);
        if (i > 80)break;
        *afile++;
        i++;
    }
    i = i - 1;
    *afile--;
    *intvalue = atoi(afile);
}
/*

```

```
    if (debug) printf("\nintvalue= %d  aline[%d]= %s",*intvalue,i,aline);  
*/  
return(i);  
}
```

Appendix D
Necessary Command Files

```

$!
$! Bridge_Schema_Create.com
$!
$! Command procedure to compile and link the program that creates the
$! SQL database definitions for the NMQL LANBridge agent.
$!
$! Copyright 1990, Gary W. Haney and Jeffrey D. Case
$!           University of Tennessee, Knoxville
$!
$ SCC == "$SYS$SYSTEM:SQL$PRE/CC" !Define a symbol for the SQL
Pre-compiler
$ scc bridge_schema_create.sc
$ link bridge_schema_create,sys$library:sql$user.olb/lib,-
  sys$library:vaxcrtl.lib,sys$library:vaxcrtl.olb/lib

```

```

$!
$! NMQL.com
$!
$! Command Procedure to compile and link the NMQL Manager and the
$! NMQL Agent for the LANBridges from Digital Equipment Corporation.
$!
$! Copyright 1990, Gary W. Haney and Jeffrey D. Case
$!           University of Tennessee, Knoxville
$!
$!
$! Compile and Link the NMQL Manager
$!
$ SCC == "$SYS$SYSTEM:SQL$PRE/CC" !defines the SQL pre-compiler.
$ scc nmql_manager.sc/list/noopt
$ link nmql_manager, SYS$LIBRARY:SQL$USER.OLB/LIBRARY, -
  sys$library:vaxcrtl.lib,sys$library:vaxcrtl.olb/lib
$!
$! Compile and Link the LANBridge Agent
$!
$ scc nmql_agent.sc/list/noopt
$ link nmql_agent, SYS$LIBRARY:SQL$USER.OLB/LIBRARY, -
  sys$library:vaxcrtl.lib,sys$library:vaxcrtl.olb/lib
$!

```

```

$! Define a symbol so that NMQL is interpreted as a VMS foreign
$! command.
$!
$ nmql ::= $ cs$users:[haneyg]nmql_manager.exe

$!
$! Bridge_Dump.com
$!
$! DCL command file to query the Remote Bridge Management System (RBMS)
$! and get the runtime information for the end user.
$!
$ Set proc/priv=(all,nobypass) !First set up the necessary privs.
$!
$! Next, get the counters and learned addresses from all known
$! LANBridges.
$!
$ assign/user NLA0: TT:           !RBMS requires some physical device.
$ assign/user command.dat sys$input:
$ assign/user temp.dat sys$output:
$ rbms
$ create finished.dat
This file is needed to provide a means of communication with the NMQL_AGENT
Kludgy, but it works...
$! End of Bridge_Dump.com

```

(Note: The following two lines is the file "Command.dat" used in Bridge_Dump.com, above)

show known bridges known lines counters to "lb_output.dat"

show known bridges known lines known physical addresses to "lb_addr.dat"

Appendix E
User's Guide to the NMQL

NMML User's Guide

The NMML consists of two software programs layered on top of RBMS and SQL. The command structure is very similar to that of traditional SQL, but some extensions were needed in order to facilitate realtime communication with the device. The syntactical structure of the NMML commands resembles one of the following:

- * NMQL *SQL_stmt*
- * NMQL *SQL_stmt NMQL_extensions*
- * NMQL *NMQL_extensions*

If the first form is used and the statement is valid, a query of the historical information collected from the LANBridge 100 is initiated and a VAX SQL sub-process is spawned. From there, the user may repeatedly enter new SQL queries. Until the user exits this sub-process, only historical information can be retrieved.

If the second form is used and the statement is valid, a realtime query to the LANBridge 100 is initiated, information returned from the query is inserted into the historical database, and the query is displayed to the user. Valid NMQL extensions are:

- * START XX:XX - where XX:XX is some military time. If an immediate query is desired, the word "NOW" replaces the time variable.
- * EVERY XX:XX - where XX:XX is some military time. This extension specifies the repetition interval of a given query.

If the third form is used and the statement is valid, the manager performs one of the following functions:

- * ADD agent - where agent is the name of an agent process to be supported by the NMQL Manager.
- * REMOVE agent - where agent is the name of an agent process to be removed from the list of currently supported agents.

Vita

Gary Willard Haney was born in Blountville, Tennessee on March 28, 1962. He attended East Tennessee State University in Johnson City, Tennessee from 1980 to 1984 and was very active in the cooperative education program. He graduated from East Tennessee State University in May 1984 with a B. S. Degree in Computer Science.

After two years of employment with Martin Marietta Energy Systems in Oak Ridge, Tennessee, the author entered the University of Tennessee, in May 1986 to pursue his Master's degree while continuing employment. He received the Master of Science degree with a major in Computer Science in May 1991.

The author is a member of Alpha Lambda Delta, Gamma Beta Phi, and is actively involved in the OSI Implementor's Workshop sponsored by the IEEE Computer Society and hosted by the National Institute of Standards and Technology in Gaithersburg, Maryland.