



Fermi National Accelerator Laboratory

FERMILAB-Conf-92/268

murmur
A Message Generator and Reporter for
Unix, VMS and VxWorks

G. Oleynik, L. Appelton, B. MacKinnon, C. Moore, G. Sergey, L. Udumula

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

October 1992

Presented at *Computers in High Energy Physics*,
Annecy, France, September 21-25, 1992



murmur
**A Message Generator and Reporter for
Unix, VMS, and VxWorks ¹**

*Gene Oleynik, Laura Appleton, Bryan MacKinnon,
Carmenita Moore, Gary Sergey, Lourdu Udumula*

Online Support Department

Fermi National Accelerator Laboratory

P.O. Box 500
Batavia, IL 60510
Tel: 708-840-3921

This paper describes the software product **murmur**, a message generation, reporting, display and logging software system that we have developed for use in data acquisition systems at Fermilab. **murmur** is a tool for the production and management of message reporting. Its usefulness ranges from software product development and maintenance to system level shakedown and diagnostics.

murmur provides a VMS MESSAGE-like function code generation utility, a client routine package for sending these codes over the network to a central server, and a server which translates the codes into meaningful visual information, writes the information to a logfile and displays it on X windows. Information associated with message codes such as message text, message color, X display configuration and routing information, and other related information are kept in fast access keyed files and can be modified by a set of Motif based configuration editors. As a result, **murmur** provides advanced features such as popping up help when a displayed message is clicked on by the mouse and executing "action" shell scripts when selected messages are received by the server.

The server and editors are written in C++ for extensibility and maintainability, and are currently available on Sun and SGI systems. The client library is written in C, and has a Fortran binding as well. Clients are supported on SGI IRIX, Sun SunOS, IBM AIX, DEC ULTRIX, VAX VMS, and under the VxWorks real-time kernel operating system.

¹Sponsored by DOE contract No. DE-AC02-76CH03000

1 Introduction

The **murmur** message system acts as a reporter and central repository for messages generated from applications across a distributed system. Message generation, reporting, logging, and displaying are part of the **murmur** system. Access to **murmur** is provided via a simple routine interface for applications, and through a set of X-windows/Motif graphical editors for configuration of the system. **murmur** displays messages with X/Motif in color or black and white.

murmur was developed as part of the DART – data acquisition software for the next Fermilab fixed target run – and Sloan Digital Sky Survey projects. It is also being used in the Drift Scan Camera project. While its primary purpose is to provide for integration of such data acquisition systems, its domain of applicability extends to stand alone programs - in particular those integrating several different software packages.

Applications use **murmur** to associate message mnemonics and message text with 32 bit message codes. In a manner similar to the VMS MESSAGE Utility, a message's facility, number, and severity are encoded into these 32 bit numbers. These codes are used by an application's function library to return conditional status. Callers of the function library use the returned **murmur** function codes to determine if the functions completed successfully and can send the codes to a **murmur** server to be translated into associated message text and displayed.

Message text and other associated information are stored in a message "database". One of the major tasks of the **murmur** server is to translate message codes into message text and to log and display the text with information obtained from the message database.

Translated messages can be routed to specific X display windows based on message context (e.g. message severity, message "facility"). In addition, named groups of messages can be associated with display windows for the purpose of routing. This permits display window classifications such as OPERATOR, BEAMLINER, and EXPERT to be created, and the appropriate messages routed to them. Messages can be associated with groups when an application's messages are generated, or during run time.

Display attributes, such as color, can also be associated with individual messages or groups of messages. These attributes determine how an individual message is displayed *within* a display window.

Message displays will also respond to user input. For instance, a message can have help text associated with it, and clicking on the displayed message with a mouse will display the help text in a separate pop-up window.

In addition to these interactive "mouse based" actions, **murmur** supports actions driven by the receipt of messages. Individual messages or groups of messages can have "action" shell scripts associated with them that are executed upon message receipt.

murmur is tailored with a set of X/Motif based Graphical User Interface (GUI) "editors". These editors are used to associate actions with messages, classify messages into groups, configure display windows, and route messages to configured windows.

2 Message Generator

The Unix operating system's `errno` facility is not user extensible, so one must resort to ad hoc mechanisms for returning conditional codes from function calls. This may be acceptable for small programs, but for even moderately sized software systems it is not. To fill this void, **murmur** provides a VMS MESSAGE-like message code generator. This generator enables end users and software product developers to build function libraries which return conditional codes in a well defined manner. The generator stores textual and other information associated with the message codes in random access files keyed on the message codes. This permits more functionality and information, such as help text, to be associated with the message codes than is possible with other systems.

Under the **murmur** system, messages are defined by the application developer with a message definition file. This file associates text, help, groups, and severity attributes with message mnemonics, and specifies a message facility name and number for the messages.²

The Message Generator processes this file and generates unique 32 bit message codes for each message in which are encoded the message's facility, severity, and generated number. As in VMS message codes, an odd number code indicates a success or informative condition, and an even number code indicates an error or warning condition. Up to 2047 messages per facility, and 16383 facilities can be accommodated by **murmur**. The last 4095 **murmur** facilities are reserved for mapping VMS MESSAGE facilities to corresponding **murmur** ones.

The Message Generator also generates include files for applications programs, and **murmur** database files for translation of message codes. Include files for C, C++, and FORTRAN contain message mnemonics equated to their 32 bit codes for use by application programs (const's for C++, #defines for C, and PARAMETERS for FORTRAN).

3 Message Concatenator

murmur uses a software development strategy in which software "systems" are built from a number of software "products" or applications. In this scheme, software products include their own message facility - messages and message databases generated with the Message Generator. **murmur** provides a Message Concatenator which combines any number of product databases into a system database for use by the **murmur** server.

This strategy is most easily accomplished within the UPS (Unix Product Support) framework, and **murmur** has been optimized for this environment, though it is not required. The Message Concatenator will accept a list of UPS products as input to build product databases, in addition to absolute database file specifications.

While the system database is not part of the **murmur** product proper, **murmur** defines some rules and conventions on how a system database interfaces to the **murmur** server and Configuration Editor programs. These programs look for three environment variables: one for the path to the database files, one for the logfile path, and the third for a path to a "default" area to locate message action scripts.

²The facility name is usually associated with an application or system, and the associated unique facility number assignment is administered external to **murmur**.

A **murmur** system database can be made into a UPS system database product. This product provides areas for the system database files, server logfiles, and message action scripts, and when setup, sets the three environment variables mentioned above to point to these areas. Startup scripts for the **murmur** server and Configuration Editors take the product name as an optional argument and will setup the product if present.

The concatenator is also used to update a system database with new versions of product databases. Unlike the Message Generator, the database concatenator preserves database tailoring, such as display configuration information.

4 Client Library

The client library consists of a set of simple routines for sending message codes to the **murmur** server for translation and display. The server formats this data for display a la printf; integer, floating point, or character string arguments can be sent along with the code as long as the message text associated with the code has the appropriate printf directives (e.g. %d, %f, %s) embedded in it. The client call can also specify that a 132 character string be appended for display. This is useful to supply specific routine location information for debugging (e.g. the routine name and line number where the error occurred). Clients can also send text messages to the server.

For the VxWorks real-time kernel operating system, client software is provided that allows interrupt service routines, in addition to user processes, to send messages to the **murmur** server.

An application can connect to a single **murmur** server. If a server is not specified, then the client library displays the messages to standard error. In this case, for coded messages, the code is displayed but cannot be translated to meaningful text.

The **murmur** client software is the only part of **murmur** not written in C++. It is written entirely in C so that it can be ported to systems without a C++ compiler. A set of C MACROS is provided to parse the bit fields in the 32 bit message codes. For example, MUR_MSG_SUCCESS returns a 1 if the message code indicates a success, and a 0 for failure.

5 Message Server

The **murmur** message server displays and logs messages sent by remote client applications. Information associated with the messages is fetched from the system database and is used to display and log the messages. With each message, the server can display and log not only message text, but client time and time-zone, server time, client node-name and process ID, a client specified application name, and text appended by the client.

If a message action is associated with a received message, the associated shell scripts are executed. This is especially useful for application maintainers since it provides a mechanism for automatic notification when specific errors occur. (e.g. a script that sends mail or executes a diagnostic snapshot)

If help information is associated with the message, then clicking on the displayed message will pop up a window to display the help text.

murmur is configured and customized through the use of four graphic interface database editors: the Display Setup Editor, the Group Editor, the Routing Editor, and the Message Action Editor. These four editors are available from a single program. All configurable values in the **murmur** database are modified through this program. For instance, you can

- Specify and format the windows to display messages on.
- Route messages to these windows based on facility, severity or group.
- Collect a set of messages that you would like to be handled in the same way into a message group.
- Specify “default” windows to which unrouted messages are sent.
- Specify that receipt of a particular messages initiate execution of “action” scripts (UNIX shell scripts).

6 Observations and Concluding Remarks

murmur was our first project written in C++ that made extensive use of X and Motif, as well as commercial software production tools. We found C++ (ATT based) to be a quality production level language and its object oriented features to be powerful programming tools. We used both the ObjectCenter (CenterLine) and Objectworks (ParkPlace) C++ programming environments on Suns and Silicon Graphic’s native C++ compiler on SGIs. Developers were happy with all three. The SGI compiler was well integrated with dbx, while the other two were not, but had their own debugging environment with some features not available via dbx.

Our experience with Motif and building the GUI Editors was not as pleasant. There were a number of Motif bugs we had to work around, which we did successfully. The GUI builder we used, UIM/X (Visual Edge, Bluestone distributor and consulting), crashed frequently when we first started using it. Bluestone was very cooperative in getting this fixed however, and it crashes less frequently now (attributable to Motif?). In order to use this C based interface builder with C++, we had to construct an environment to use it in, which among other things, included fixing typos and bugs in X, Motif and UIM/X include files that were visible only under C++.

Even with the assistance of the graphical interface builder, there was a steep and fairly continuous learning curve in X and Motif – we would think long and hard before attempting to do graphical interfaces as ambitious as those in **murmur** again. However, it is hard to imagine how to provide the functionality of **murmur** without a graphical interface.

V1.0 of **murmur** has just been released. It provides a server for Sun SunOS and SGI IRIX platforms, and clients for Sun SunOS, SGI IRIX, IBM AIX, DEC ULTRIX, DEC VMS, and VxWorks platforms, and has also been ported to a HP platform by experimenters. We will make the server available on more platforms as we acquire C++ compilers and port the UIM/X run-time library for these platforms.