UCSF
UC San Francisco Electronic Theses and Dissertations

Title
User interfaces and software architecture in biomedical research

Permalink
https://escholarship.org/uc/item/5mb0817s

Author
Morris, John Hugh

Publication Date
1990

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

User Interfaces and Software Architecture in Biomedical Research

by

John Hugh Morris III

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Medical Information Sciences

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA

San Francisco

Acknowledgement

I would like to thank the members of the Computer Graphics Laboratory for their contributions to and criticisms of this project: Professor Ferrin for a well-managed laboratory; Eric Pettersen, Greg Couch, and Leslie Taylor for their constructive criticisms; Teri Klein, for her encouragement and continuing reminder of the "user's view"; and above all, Conrad Huang for the hours of useful arguments and discussions which contributed greatly to the overall progress of this project.

I thank my research committee: Professors Robert Langridge, Thomas Ferrin, and I.D. (Tack) Kuntz for their instruction and advice, and for serving on my thesis committee. In particular, Professor Langridge for providing a facility where the pursuit of research into biomedical computing can proceed next to and associated with the pursuit of biomedical research; Professor Kuntz for practical suggestions during the course of this research; and Professor Ferrin who made many constructive comments and criticisms which helped guide this research, and who suggested that he may have a thesis project for me all those years ago in "the dungeon" at Genentech.

Many other people supported me during this effort. The members of the Scientific Computing group at Genentech had to survive years of having their manager available on a part-time basis, yet whenever issues of time constraints were raised, no one ever suggested I reduce the time devoted to my dissertation. Also at Genentech, Polly Moore, the Director of Information Resources and my immediate supervisor encouraged me and supported me by allowing me to have an incredibly flexible schedule. As a matter of fact, she even edited all of the versions of this document which was of tremendous assistance. Jarrett Rosenburg and Sarah Bly also provided encouragement and extremely useful criti-

cisms during the early stages of this work.

Mostly, however, I feel I owe a debt to my family, and especially my wife, Suzanne, who has had to pay the price of my absence during my pursuit of my degree. I could not have done it without their love and support.

Finally, I thank the NIH Division of Research Resources (RR-1081), the DARPA University Research Initiative (N00014-86-K-0757), and Genentech, Inc. for providing the facilities and support.

Abstract

Software is becoming an increasingly important part of many research endeavors. This has been increasingly true for biomedical research, yet the software available for the pursuit of biomedical research is deficient in a number of ways. The goal of this project is to derive a new approach to the development of software which is more appropriate to the realities of the biomedical research environment, then test that approach by using it to design and implement a non-trivial system to support research in this area.

The approach proposed differs from traditional techniques by shortening the time to initial prototype, incorporating the user's view directly into the design process, and specifically incorporating an iterative design process with an explicit evaluation step. The goal of the approach is to produce software which is easier to use, and supports an internal consistency with the user's view. This goal is accomplished by the development of an explicit mental model of the system.

The system which was designed and implemented utilizing this approach is the macromolecular workbench (MMWB). This system provides a variety of unique features. First, it provides the ability for users to interact with the system utilizing multiple user interfaces. The learning curve is reduced by providing a consistent "user interface metaphor" across all of the interfaces. Second, the system provides for a separation between programs which require direct user interaction and programs which require very little in the way of user interaction. Third, the ability to have "sessions" which can extend over a period of time which can be much longer than a typical login session is provided. The sessions can be attached to and waiting messages from protocols can be received and replied to. One unique feature of the sessions is that the user can reattach to

a session on a different interface, or using a different device than it was started on without any loss of functionality or communications ability with running protocols. Finally, the system provides a mechanism which is independent of the user interface but can still be extended to support semantic information and feedback.

Table of Contents

1. User Interfaces and Software Architecture in Biomedical Research 1
1.1. Introduction 1
1.2. Design approach 2
1.3. MMWB 3
1.4. Conceptual Model 6
1.4.1. Background 6
1.4.1.1. Mental Models 6
1.4.1.2. Metaphors 8
1.4.1.3. User Interface Management Systems 10
1.4.1.4. User Interface Consistency 11
1.4.2. The Workbench Metaphor 12
1.5. Summary 15
2. The Design Approach for MMWB 15
2.1. Introduction 16
2.2. The Design Environment 17
2.3. The Design Process 18
2.3.1. Traditional Software Design 18
2.3.1.1. Software Development in a Research Environment 20
2.3.1.2. Summary of Problems 22
2.3.2. Proposed Development Approach 22
2.4. Summary 24
3. The MacroMolecular Workbench - MMWB 24
3.1. Introduction 25
3.2. Goals of MMWB 25
3.2.1. Data Sharing 26
3.2.2. Resource Sharing 26
3.2.3. Multiple, Consistent User Interfaces 26
3.2.4. Compatible Computing Services 26
3.2.5. Extensibility 27
3.2.6. Distributed Processing 27
3.2.7. Network Transparency 27
3.2.8. Implementable in Finite Time 28
3.3. MMWB Conceptual Model 28

3.3.1. The Workbench Metaphor 2
3.3.1.1. A Detailed Example 2
3.3.1.2. Extensions 2
3.3.2. The MMWB Workbench 31
3.3.3. Alternatives 32
4. The MMWB Design 33
4.1. MMWB Architectural Overview 33
4.2. Sessions in MMWB 34
4.2.1. Session Management Functions of SM 39
4.2.2. Tool Management Functions of SM 39
4.2.3. SM Design 41
4.2.3.1. SM Common Library 45
4.2.3.2. SM-UI Library 45
4.2.3.3. SM-Tool Library 46
4.2.3.4. SM Process 48
4.3. Dialogs 49
4.3.1. Description and Use 50
4.3.2. Design of MMWB Dialogs 50
4.3.2.1. Simple and Basic Dialogs 53
4.3.2.2. Compound Dialogs 53
4.3.2.3. Notifiers 54
4.3.2.4. Messages 55
4.3.2.5. Options 56
4.3.3. Comparison of MMWB Dialogs with Macintosh Dialogs 56
4.4. User Interface Managers 57
4.4.1. Workbench Metaphor Interface 58
4.4.2. Semantic Dialogs 59
4.4.3. Tool Invocation 61
4.4.4. Interface to Instruments 63
4.4.5. Example User Interface 64
4.5. Foreign Tool Integration 68
4.5.1. FTD Design 69
4.5.2. FTD Rule Syntax 70
4.5.3. Debugging 74
4.6. Discussion 75
4.6.1. Why the Complexity? 75
4.7. Conclusions 76
5. Discussion 77
..... 78

5.1. Introduction 79
5.2. Platform Status 80
5.2.1. Implementations 80
5.2.1.1. Dispatcher 81
5.2.1.2. Data Manager (DM) 81
5.2.1.2.1. Data Manager Library (libofs) 82
5.2.1.3. Session Manager (SM) 82
5.2.1.3.1. Session Manager Library (libsm) 84
5.2.1.4. Dialog Library (libdialog) 85
5.2.1.5. Foreign Tool Driver (FTD) 86
5.2.1.5.1. Foreign Tool Driver Library (libftd) 87
5.2.2. Performance 87
5.2.2.1. Performance Results 88
5.2.3. Reliability 90
5.3. twb - An Example UIM 91
5.3.1. Implementation 92
5.3.1.1. Interface 92
5.3.1.2. Commands 93
5.3.1.2.1. QUIT 94
5.3.1.2.2. STATUS 95
5.3.1.2.3. BEGIN 95
5.3.1.2.4. END 96
5.3.1.2.5. LIST 96
5.3.1.2.6. HELP 96
5.3.1.2.7. ANSWER 97
5.3.1.2.8. CREATE 97
5.3.1.2.9. BROWSE 98
5.3.1.2.10. DELETE 98
5.3.1.2.11. NAME 98
5.3.1.2.12. SELECT 99
5.3.1.2.13. MODIFY 99
5.3.1.2.14. SET 99
5.3.1.2.15. COPY 99
5.3.1.3. Browser Library 100
5.3.1.4. I/O Library 101
5.3.2. Lessons Learned 102
5.4. spasms - Integration of a Foreign Tool 103
5.5. clock - A Simple Example Protocol 107

5.6. Evaluation of the Design Process 109
5.7. User Perceptions and Suggestions 112
5.7.1. Laboratory Notebooks 113
5.7.2. MMWB Scripting Language 114
5.7.3. Object Dialog 114
5.8. Conclusions 115
6. Future Directions 116
6.1. Introduction 117
6.2. Design Approach 117
6.3. MMWB 119
6.4. Conclusions 120
References 121
Glossary 125
Appendix A - Tool Invocation Details 131
Appendix B - Session Manager Communications 134
Appendix C - Unix Manual Pages 138
1. FTD 139
1.1. FTD(1) 139
1.2. FTD(3) 141
1.3. FTD_RULE(5) 144
2. *libsm* 146
2.1. SM(3) 146
2.2. SMT(3) 148
2.3. SMUI(3) 150
2.4. TOOLDES(3) 152
3. *libdialog* 154
3.1. DIALOG(3) 154
3.2. DIALOGTEXT(3) 156
3.3. DIALOGUTIL(3) 157
3.4. DIALOGVALUE(3) 158
Appendix D - FTD Grammar 162
Appendix E - Source Listings 166
1. Session Manager (SM) 167
1.1. libsm 217
2. libdialog 243
3. twb 262
3.1. libbrowse 306
4. ftd 331

List of Tables

Table 1 - Interface Characteristics of Devices 5
Table 2 - Tool Description Properties 43
Table 3 - SM Library Functions 46
Table 4 - SM-UI Library Functions 47
Table 5 - SM-Tool Library Functions 49
Table 6 - Dialog Types 52
Table 7 - Dialog Contents 54
Table 8 - An Example Dialog 54
Table 9 - Minimum UIM Functionality 59
Table 10 - Desirable UIM Features 61
Table 11 - FTD Library Functions 71
Table 12 - FTD Mandatory Options 73
Table 13 - Results of MMWB Performance Tests 89
Table 14 - twb Commands 94
Table 15 - LIST Command Arguments 96
Table 16 - Keystroke Commands in BROWSE Mode 98
Table 17 - Communications Types 134
Table 18 - UIM-SMP Communications Protocol 136
Table 19 - Tool-SMP Communications Protocol 137

List of Figures

Figure 1 - The Traditional Software Development Life Cycle 19
Figure 2 - Software Development in Research 21
Figure 3 - Proposed Development Approach 23
Figure 4 - Example Project Flow 30
Figure 5 – Process Architecture of **MMWB** 35
Figure 6 - Example Parallel Tool Execution 44
Figure 7 - An Example Dialog Interface 55
Figure 8 - An Example Compound Dialog Interface 55
Figure 9 - **FTD** Invocation 72
Figure 10 - Example multi-step **FTD** series 72
Figure 11 - **FTD** Script for SPASMS Integration 106
Figure 12 - *clock* Source 108
Figure 13 - Flowchart for *sm_invoke* 132
Figure 14 - Flowchart for *sm_send_reply* 133
Figure 15 - State Transition Diagram for UIM-SMP Protocol in the UIM 136
Figure 16 - State Transition Diagram for UIM-SMP Protocol in the SMP 136
Figure 17 - State Transition Diagram for Tool-SMP Protocol in the SMP 137
Figure 18 - State Transition Diagram for Tool-SMP Protocol in the Tool 137

Chapter 1

User Interfaces and Software Architecture in Biomedical Research

1.1. Introduction

The user interface of any system is important to that system's eventual success. Because it is the primary means by which the user interacts with a program or system, it is one of the main criteria from which a user forms an impression of the system. A poor user interface can be a serious impediment to learning the system, and to eventual mastery of the functionality the system provides. Unix is an example of a system which has a user interface that provides a great deal of power and functionality, but has been criticized as being "unfriendly" and difficult to learn.

Since the user interface has a major impact on how easy a system is to learn, the user's perception of the functionality of the system is often based on the user interface. This happens because many users will not learn all of the features a system has to offer, or will simply refuse to invest the time if the system is too difficult to learn. The overall functionality of a system, from the user's viewpoint, is then seen to be a combination of the ease-of-use of the system and the actual functionality.

The Macromolecular Workbench, **MMWB**, is an environment being developed at the University of California, San Francisco for supporting research into the structure and function of biological macromolecules. The diverse nature of the work involved in this research suggested that a worthwhile goal for the system design would be the integration of software from a number of sources and, simultaneously, utilization of a variety of different computer display devices.

Traditional software design methodologies are not well suited to this type of project. The traditional software lifecycle and software development methodologies were developed in response to the need to manage very large projects, primarily for the defense and aerospace industries (cf. [4]). A consistent theme with all of the currently accepted design methodologies is the need for very detailed and complete requirements and specification phases. Software development in a research environment, however, is often performed with very small teams of programmers, usually consisting of a large percentage of non-professional programmers (e.g. postdocs, graduate students, etc.). Detailed specification is difficult because of the changing needs of the research and the iterative nature of experimentation. This has led to the current state of software for the research environment: it is poorly integrated, derived from a variety of sources, and neither flexible nor easily maintainable.

1.2. Design approach

A new approach to design of systems in a research environment was used for the development of **MMWB**. This approach shortens the traditional requirements definition and specification stages into a single, more informal goal definition stage. In a research environment with small teams of programmers, detailed specification and requirements documents are seldom, if ever, prepared. After the goals for the system are defined, a conceptual model for the system is derived. This conceptual model will be used to ensure that the user's view of the system and the developer's view of the system are compatible. The conceptual model is used to influence the design process so that the resulting system reflects the user's expectations.

The approach used for the design of MMWB is also highly iterative. After the initial implementation is complete, it is evaluated in terms of the initial goals and the validity of the conceptual model. The results of the evaluation may cause another iteration of the implementation, design, or even refinement of the conceptual model.

1.3. MMWB

The application domain for the Macromolecular Workbench (MMWB) is the determination of the structure and function of biological macromolecules. This is an area which will involve a diverse population of scientists and researchers in a variety of areas including:

- X-ray crystallography
- NMR spectroscopy
- Protein chemistry
- Molecular modeling
- Physical chemistry

The eventual goal is to develop an integrated computer-based environment for investigators to utilize as a tool in the research of organic macromolecules.

The process of determining molecular structure and function will involve a large number of applications. Many of these applications will require very little in the way of user interaction once they have been initiated. Many of the others, however, will take advantage of modern three-dimensional graphics workstations to view the putative structure and to hypothesize about molecular interactions. Modern graphics workstations will be the primary and most natural user interface for most of the work which will be done on the MMWB. However, other user interfaces must also be developed to allow users to interact with some of the non-graphics portions of the system. These user interfaces must be supported by the terminals which are readily accessible by all users, including those

which users have at home. This will allow the user to execute simple status commands, query information of long jobs, perform data-entry, etc., without being dependent on more expensive workstations.

MMWB then, must support user interfaces on three types of output devices: alphanumeric terminals, raster displays, and 3D graphics displays. These devices differ in a variety of ways, including the style of output they will support and the types of input devices they will support. The most powerful display technology from a user interface viewpoint is the 3D graphics display. These displays can support a variety of interfaces, including command line, menus, point-and-click, and manipulations in three-space. The technology also lends itself very well to the presentation of objects or commands as representative pictures, or *icons*. The user can manipulate these icons as if they were, in fact, the objects themselves. This style of interaction has come to be known as *direct manipulation* [26]. The range of input devices for 3D graphics displays is likewise broad. Modern displays allow use of keyboards; *locators,* which are devices for the input of position such as mice, tablets, and joysticks; and *valuators,* which are devices for the input of scalar values such as knobs, dials, and potentiometers.

Raster displays are similar to 3D graphic displays in the types of input devices they can support, but in general it is not feasible to implement three dimensional graphics on displays without some sort of graphics support in the hardware. This limitation seems minor but the advent of low-cost workstations with support for three dimensional graphics has spurred a great deal of interest in user interfaces which take advantage of all three dimensions.

At the other end of the spectrum are alphanumeric terminals, which are typically

used only for menu oriented or command-line displays and which have only a keyboard for input. The interface styles and input devices typically supported by these technologies are summarized in Table 1.

Device	Supported Interface Styles	Supported Input Devices
Display Terminal	Menus, command-line	Keyboard
3D Graphic Display	Direct manipulation, menus, command-line, 3D objects and viewing	Keyboard, locators, valuators
Raster Display	Direct manipulation, menus, command-line	Keyboard, locators, valuators

Table 1 - Interface Characteristics of Devices

MMWB must also support multiple user interfaces on a *single* device. For example, when a researcher is working with a molecular model, it is a common procedure to compute some form of energy minimum using molecular mechanics. It is often desirable to monitor the progress of such programs graphically. On 3D graphical displays, it should be possible to manipulate a molecular model on the display and to have a small window which is monitoring the progress of the molecular mechanics program. This could be added as a feature to the molecular modeling program, but that is probably not optimal because it restricts the number of such displays to what the authors of the modeling program are able to add. A better approach is to allow multiple user interfaces which cooperate with some form of *user interface manager* which provides an interface to the system as a whole, and some mechanism to switch the focus into other user interfaces on the same device. This is not too different in concept from the window managers which are available on modern window systems. The primary difference between a window manager and an **MMWB** user interface manager is in the degree of interaction between the manager and the managed user interface (or window). The **MMWB** user interface

manager cooperates with applications running under it, where window managers on X, for example, are relatively independent of the managed processes [25].

1.4. Conceptual Model

The previous section discussed the application domain and some of the goals for MMWB. This section will begin to explore the idea of conceptual models and suggest an appropriate conceptual model for MMWB.

1.4.1. Background

1.4.1.1. Mental Models

When software engineers or programmers design a computer-based system they form mental models of how the system works. These mental models include various internal data structures that the programmers use, communications protocols, software interfaces, and proposed functionality. These mental models are collectively called the *system model* [18]. For example, the system model of a text processing system may be stated briefly as a hierarchical structure of document components (sections, paragraphs, sentences, words, characters) which is displayed in a linear fashion.

Users of computer-based systems also form mental models. These models typically have little to do with the internals of the system, or with the software interfaces, communications protocols, or any of the things the software engineers or programmers think about. Rather, the *user model* is formed as users interact with the system. Sometimes users will have formed a user model based on their expectations of a system's performance before they even use the system. Different users can also form very different models of how the system functions based upon their own experience with similar

systems, their level of training on the system, their expectations, or the way they happen to have used the system the first time. An example of one user model for the text processing system mentioned above is a simple list of characters in a document. With that model, users approach the system in the same way they might approach the use of a typewriter. The fact that there is a hierarchical organization to a document is meaningless to them. It doesn't change the inherent features and advantages of the hierarchical organization, but because the users have a very different mental model, they may not use them.

The *user interface* to a computer-based system can be considered a mapping from the system model to the user model. It can function to help users form a consistent user model which will continue to operate throughout their use of the system. This is the reason that consistent user interfaces are stressed -- so that once users form the correct mental model of system operation, the system does not do anything to break that model. This reduces the amount of training necessary to feel comfortable using a computer-based system. Two obvious examples of systems that have been designed to support very consistent user interfaces are the Xerox Star [27] and the Apple Macintosh [1].

It is important to note that the user interface is a mapping between the user model and the system model, not an attempt to make the user model conform to the system model. Many user interfaces have been written which take the latter approach (e.g. IBM's JCL). Often the goal is to present a user model which is very different from the system model. The Apple Macintosh, for example, presents users with a series of *icons* (pictorial representations of objects) on the screen. Users form a model of an icon representing an object which may be manipulated in some fashion. In fact, at the system level, each icon is represented by three separate "forks", or file divisions, which include

different information about the data, resources, and position of that object. Under normal circumstances, users never have to deal with the system's concept of what that icon represents.

1.4.1.2. Metaphors

One technique for helping users form a specific mental model of their interaction with the system is to use a metaphor [18]. A metaphor utilizes objects or actions that users are (hopefully) already familiar with and already have mental models of to suggest how they should interact with the system. The desktop metaphor, as an example, relies on the users' familiarity with objects in an office environment such as documents, files and filing cabinets to suggest a user model. This user model maps to the system model of a hierarchical file system, without having to explain the difference between a "file" and a "directory". The users use knowledge that they already possess to deduce how to interact with the system. This reduces training time and provides a basis for the user to quickly begin to use a system, although mastery may still take additional training.

The primary use of a metaphor, then, is to reduce the learning curve and allow a user to quickly become comfortable with a system. One additional use of a metaphor may be to reduce the relearning necessary as a user switches from one system to another. Again taking the desktop metaphor as an example, users who have used the Xerox Star will find that many things about the Apple Macintosh come very easily and naturally to them. In particular, the concept of files and documents and a hierarchical organization of the desktop is something that will be familiar to anyone who has worked on the Star. One potential extension of this is to utilize a consistent metaphor across different user interfaces to the same system to allow users who have used one user interface to quickly

adapt to another with a minimum of retraining.

Another point about metaphors is that at some level they have to be extended. Again taking the desktop metaphor as an example, both Xerox and Apple designers added objects to the "desktop" which don't exist on a real desktop. In the Star, Xerox designers chose to represent a printer as an object on the desktop to which the users copy documents which they wish to print. On the Macintosh, Apple designers implemented a pull down menu across the top of the screen to control system operation. There is obviously no real world analogue to this. In both of these cases, the designers reached the limitation of the metaphor and chose different ways to extend it. At some level, all metaphors will have to be extended in some way. The job of the interface designer is to utilize the metaphor as much as possible, but to recognize when the limits have been reached. It is often a very difficult task to assess at what point the metaphor ceases to be an advantage, and how best to extend it.

There has been much work in the area of linguistics and natural language relating to metaphors [15]. This work has resulted in systems which can interpret metaphors in language. Unfortunately, no such systems have been created which can capture or describe a user interface metaphor. This is not surprising considering that a user interface metaphor goes far beyond a linguistic metaphor. User interface metaphors can take advantage of visual similarities as well as semantic similarities. This makes a formalism for describing user interface metaphors extremely difficult to imagine. In addition, such a formalism would not replace the need for the designer to understand the semantic domain of the potential users of the system.

1.4.1.3. User Interface Management Systems

The common approach to user interfaces within the computer science community is to utilize a *user interface management system*, or UIMS. The primary goal of these systems is to provide a strong separation between the user interface of a program and the algorithms and manipulations provided by the program. The commonly held belief is that this will allow the user interface designers to concentrate on designing good user interfaces, and will allow the system architects to focus on designing a clean, efficient software architecture. A UIMS is also supposed to allow modification of the user interface to suit individual preferences and changing technology.

The difficulty with UIMSs to date has been the difficulty of providing *semantic* feedback to the user interface.[1] Semantic feedback is extremely important in the biomedical environment for such things as molecular modeling, where knowing that you've selected a carbon atom as opposed to a neighboring oxygen is very important. Many UIMSs provide some means of having the application display some semantic feedback, but if this style of interaction is fundamental to many, or most, of the actions users might perform, then the application becomes bound once again to the user interface with the additional overhead of the UIMS. Most UIMS advocates now concede that the "pure" separation of application and user interface is not appropriate (cf. [19], [21], and [17]) and that some portion of the application should exist within the user interface to provide semantic feedback, and some portion of the user interface should exist within the application to provide special user interface functions which might not be present within the UIMS.

[1] The *semantics* of an interface refer to the domain or application specific knowledge which it employs.

1.4.1.4. User Interface Consistency

The importance of user interface consistency in reinforcing the user's mental model was discussed above. The recognition of this important aspect of user interface design has resulted in the adoption of user interface consistency as a primary goal by designers. Unfortunately, exactly what is meant by consistency has yet to be clearly articulated. A recent paper by Jonathan Grudin [7] presents the case against user interface consistency. His central thesis is that understanding the task domain, and designing interfaces which are appropriate for that task domain, are more important than maintaining a set of user interface guidelines across all task domains.

The approach presented here attempts to use a metaphor to achieve consistency at a task, or semantic, level rather than simply depending on the look and feel of the interface. There are several reasons for this approach. First, because researchers utilize a variety of different computer systems and computing devices, "look and feel" consistency could only be achieved by compromising on either a lowest level hardware interface, or attempting to emulate higher level interfaces with less functional hardware. In either case, a sub-optimal user interface often results. Second, as hardware and software improves over time, new user interface technology is likely to emerge also. By enforcing consistency at a semantic level, it will be much easier to incorporate new interfaces and still provide the user with a sense of consistency. As an example, consider the Macintosh interface, which is rigidly consistent. One of the new areas of user interface technology will be the utilization of three dimensional graphics to present the user with a three dimensional interface into the system. This style of interface will be very difficult to incorporate into a set of guidelines which were originally written for a single-tasking

environment on a small display. If, however, the semantic consistency behind the interface were stressed, then the transition might be much easier. In the case of the Macintosh, this would involve focusing on the desktop metaphor, the relationship between objects and applications, and the idea of a standard set of operations. As a final reason, enforcing consistency at a semantic level does not discourage the incorporation of new interface styles and technology into the system. If the semantic consistency is retained, then new interfaces can be easily added.

1.4.2. The Workbench Metaphor

Metaphors are being used in a variety of systems to reduce the learning curve necessary to become comfortable with the systems. In addition to the desktop metaphors of the Xerox Star and the Apple Macintosh as discussed above, other uses of metaphor include the Rooms metaphor of the Interlisp-D environment [10], and the NoteCards metaphor used in both NoteCards [8] and Hypercard. The implementations of the desktop metaphor have some common characteristics. First of all, the system is organized around "documents" which are acted upon by "applications". Second, there tends to be a relationship between an object and the application which acts on that object. On the Macintosh, for example, the application which creates an object is "registered" as part of the information associated with object. When a user "double-clicks" on the object, the associated application is invoked on that object. Third, objects tend to be single-valued; for example, a word processing program operates on a single document, not collections of documents. These characteristics make sense in a text-processing or office environment, but provide certain restrictions in other environments. What is proposed below is a new metaphor based upon a workbench instead of an office desktop. For the purposes of

MMWB, the metaphor is of a chemist's workbench.

The underlying idea of the workbench metaphor is that one utilizes a variety of *tools* to develop, or complete, a *project*. A project can (and usually will) contain multiple objects representing data in various stages of refinement necessary to accomplish a specific goal. For example, if the goal is to determine the three-dimensional structure of the drug *relaxin*, then the project may contain a two-dimensional NMR spectrum which represents the original data, and a variety of objects which represent the progressive refinement steps in determining a predicted structure for relaxin. Thus a project is simply a container for all of the objects which are related to a specific study. There is no limit to the number of tools which may operate on a given project. Tools need to have an understanding of the types of objects within the project that they can operate on, but as long as there is some object within the project that the tool understands, the tool can be used. In the real world, imagine the synthesis of an organic compound. There are a variety of different materials that go into the synthesis: various starting materials, reagents, buffers, etc. If you take a tool, say a magnetic stirrer, from your tool cabinet, the number of things you can use that on is limited to liquids, or those solids which may be dissolved within liquids. A spatula, on the other hand, is more specific to solids. The goal of the project is to synthesize a compound. The tools used to accomplish that goal are at the discretion of the chemist, and are not in any way related to specific objects other than the limitations on what each tool can operate on.

The workbench metaphor has been further defined for MMWB to be a chemist's workbench. This decision was made primarily because most of the system's eventual users would be familiar with this environment. The only change that this entails is a

better metaphorical relationship for *tools* and *user interface tools*, which become *protocols* and *instruments*, respectively. The idea is that one uses instruments to view or manipulate objects, and one uses protocols to change or permute those objects. Both instruments and protocols are considered tools for the purposes of the metaphor.

The actions which operate on projects and tools in the workbench metaphor follow from the metaphor. The user must be able to *select* a project for future tool application. There is an assumption in the metaphor that the project is the major focus, not the tool. Thus one may apply a variety of tools to develop a single project. This is different from the common usage of the desktop metaphor where a user typically uses a single tool on a single document. In the real world, a chemist decides to synthesize a specific compound, and then applies a series of tools to starting materials until that compound is complete. It is less common to work on multiple projects at the same time. It therefore seems reasonable that in the implementation of the workbench metaphor there needs to be a "selected" project upon which all subsequent tools can operate.

One immediate extension of these actions is the ability to link a variety of protocols together so they can be invoked with a single action by the user. This is a simple extension of the idea of a protocol, which may entail a single step, or have multiple steps. This allows protocols to be used together to perform new functions which are combinations of their individual functions. This also provides a way for users to tailor the function of the system in some fashion.

It is also possible to combine instruments in a similar fashion. This might be used for molecular refinement requiring both molecular editing and molecular graphics. Another application might be the use of a series of instruments linked together for

demonstrations or movies.

1.5. Summary

This study takes a new approach to the design of environments for use in biomedical research. This approach involves the definition of a user interface paradigm which is appropriate to, and potentially specific to, the application domain for which the environment is being written. One user interface paradigm, the use of a user interface metaphor, will be tested to determine what impact it has on the overall design and implementation effort, and what impact it may have on the user's perception and use of the system.

Chapter 2

The Design Approach for MMWB

2.1. Introduction

The importance of computers and computer software as tools to aid modern researchers is widely recognized. In biomedical research the use of computer-based tools and techniques has provided scientists with new ways to model biological macromolecules, predict molecular interactions, determine similarity in nucleotide and protein sequences, and search for structural similarities between a series of small molecules [16]. While a wide variety of tools is now available, these tools are poorly integrated and in many cases difficult to maintain [3]. In addition to the state of the software internals, the interface between the software and the user is often crude and difficult to learn. There is little, if any, consistency between the various software tools which might be used by an individual researcher.

A great deal of literature exists which describes software development methodologies which purport to lead to the development of easy-to-use software which is maintainable and to some extent, extensible. Unfortunately, none of these methodologies has significantly penetrated into the biomedical research environment.

One reason that software development methodologies fail to be widely adopted by developers in research environments is that these environments differ substantially from the large industrial and government environments for which most software development methodologies are tailored. The following sections will examine the biomedical research environment as it relates to software, explore the difficulties in adopting existing software development methodologies for use in this environment, and suggest an

alternative approach.

2.2. The Design Environment

The biomedical research environment is typically associated with an academic or government sponsored laboratory. There are several commercial laboratories which are active in this area, but most of these rely on commercial or academic software. Those that develop software themselves hesitate to distribute it because of the competitive nature of biotechnology. For the most part, then, software development for new technologies occurs in a grant-based environment. The nature of most granting environments is that there are *always* resource constraints of one form or another. Immediately after the awarding of a large grant, there is a lack of space and personnel. Later into the grant cycle, after space is found (if it can be) and people are hired, computational resources become constrained. This is the first major characteristic of this environment: it is resource constrained.

Resource constraints are not limited to the academic environment. Most commercial biotechnology research environments also face chronic resource constraints. Most companies involved with biotechnology are still relatively small and are financially constrained. While there is typically no grant cycle in these environments, there is certainly a yearly budget cycle. As a result, most biotechnology companies continue to use academic software and share resources amongst the researchers. In addition, very few biotechnology companies have large development staffs, making them more dependent on commercial offerings and software obtained from academic collaborators.

Another major characteristic of this environment is the nature of the user community. Biomedical researchers are obviously very intelligent individuals who recognize the

computer as a necessary tool. If they perceive that the tool provides them with an avenue to pursue their research, they will use it. If the perceived utility of that tool is offset by a steep learning curve, they may not be willing to use the software. If they do use it, they may be unwilling to expand their use of the software or learn new ways to use it. The result of this is an emphasis on the interface between the user and the software.

The software requirements themselves are different in the research environment also. As experiments are conducted the results are used to refine theories and suggest new experiments. This often results in continually changing software requirements. Even software which was written to support instrumentation and tools may need to be modified as technology advances. The need to modify software is not unique to this environment; however, the number of those changes and the continual flux in requirements are distinct from what might be found in a non-research environment.

Finally, the software which is used by researchers is derived from a variety of sources. Some of the software may be obtained commercially when such solutions are available. Much of the software is obtained from other researchers and local development. Not only are user interfaces quite different, but often the data formats are incompatible. The result of this is software which is poorly integrated, inconsistent, difficult to use, and often difficult to modify to fit a specific researcher's needs.

2.3. The Design Process

2.3.1. Traditional Software Design

The traditional software life cycle as depicted by Wasserman and Stinson [29] is shown in Figure 1. Most of the current design methodologies assume this approach. The

stages of this approach are discussed in sequence.

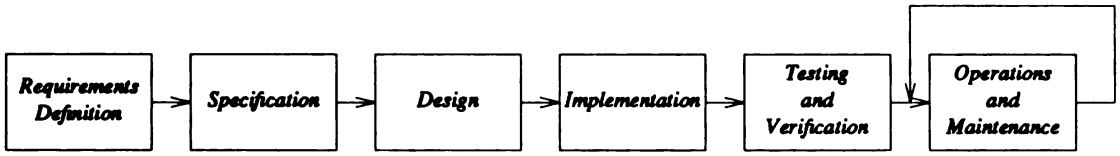


Figure 1 - The Traditional Software Development Life Cycle

The first stage in the life cycle is *requirements definition*. At this stage the requirements for the software are completely defined. A well-prepared requirements document should define exactly what the software is going to do, and why. This has been called the most important stage of the development process because all of the subsequent stages assume that the requirements have been correctly identified.

The requirements document is then used as one of the primary inputs to the next stage, *specification*. At this stage the details of the software are specified. This should include the reports, screens, algorithms and any part of the software which will interact with the user.

Once well specified, the software can be *designed*. Software design involves the documentation of modules and their interactions, as well as any data formats which are not user-visible or otherwise already specified. The design is handed off to the programmers for *implementation*, where programs are actually written which (hopefully) correspond to the design.

The final two stages are *testing and verification* and *operations and maintenance*. At the testing and verification stage the software is tested to ensure that it meets the requirements and specifications, and that it was implemented according to the design. If the software is determined to meet all of the above criteria, it is assumed that the

software will be satisfactory to the user and will fulfill their desire.

Researchers in the area of software engineering have focused on various stages of this development approach and proposed various methodologies and tools for completing stages of this process (cf. [22] , [6] , [24] , [12] , [29] , [30] , [13]). Fewer researchers have proposed methodologies which encompass the entire development process from requirements definition to operations and maintenance. One of these is the User Software Engineering Methodology ([30] , [29]), and another is the DMS methodology [9]. Both of these methodologies focus on the development of large systems by large teams of developers. The DMS methodology, for example, assumes that each development team will have a user interface specialist as part of the team. This is typically not feasible in a research environment.

2.3.1.1. Software Development in a Research Environment

The software development life cycle which is followed by most developers in the research environment is shown beneath the traditional approach in Figure 2. The requirements and specification stages are combined and shortened into a simple statement of the request or need. This is a pragmatic response to the changing software requirements in research. If formal requirements and specification documents were prepared, the time necessary for their preparation would cause them to be outdated before they were ever complete. It is also true that most systems developed within the research environment are smaller in scale than a major banking system, or the controlling software of the space station. However, most practitioners are now recognizing that the rigid, formal approach practiced by government procurement has led to rigid, inefficient software that is typically based on the technology which was prevalent at the time of the contract rather than

at the time of implementation [28].

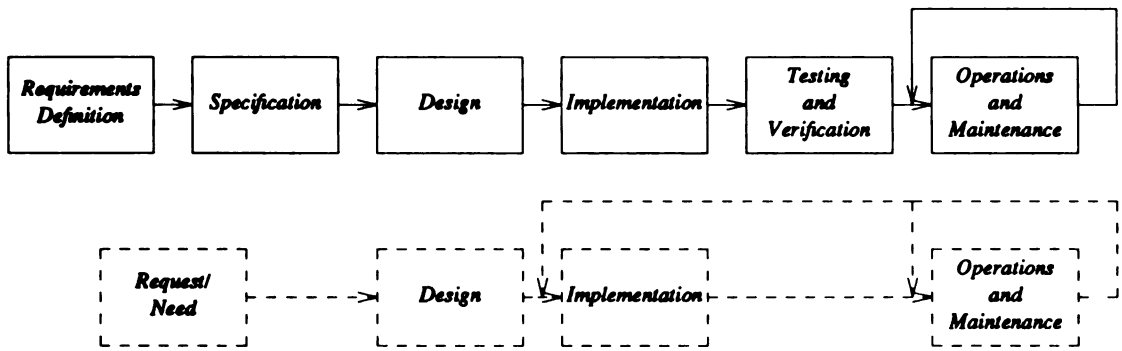


Figure 2 - Software Development in Research

In computer-sophisticated laboratories and institutions, the request is followed by a more traditional design process. The design is often prepared in close collaboration with the researchers to make up for the lack of formal specifications. The design is then implemented and cursorily tested. More formal testing is often not performed because of the expense or difficulties in testing the software. It also may not be efficient to formally test software which might need significant revision after each use.

The operations and maintenance stage is also somewhat different. The continual modifications to the software will often require reimplementation, if not redesign. This results in a more iterative process than the traditional approach.

In many of the discussions of the software development process, it is suggested strongly that the specification stage include specification of the user interface and an iterative cycle of prototyping the user interface for the user. Unfortunately, concerns about the user interface are often completely ignored in development of software for the research environment, even though the user interface is arguably as important, if not more important, in this environment.

2.3.1.2. Summary of Problems

It should not be construed from the previous discussion that there is any inherent problem with the traditional software development approach, nor with the development methodologies which have been proposed based on that approach. The traditional approach has been demonstrated as being very successful in a large number of major developments. The difficulty with the traditional approach is not in the approach itself, but in the application of this approach to an environment which is quite different from that for which it was developed.

The way software is currently developed in a research environment presents some serious problems. First, the user's view is often incorporated late in the development process, often during the actual implementation. This may result in software which is not as easy to use as it might be. Second, because of the constantly changing requirements, the software may quickly become difficult to maintain and "brittle" from too many modifications unless flexibility and extensibility are somehow designed into the software. Finally, if extensibility and flexibility are designed into the software, there is nothing inherent in the design process which provides for consistency for the user. A possible result of this is the addition of new features which never get used because of too steep a learning curve or potentially opaque utility.

2.3.2. Proposed Development Approach

The difficulties in applying traditional approaches to software development to the biomedical research environment have lead to the proposal of a new approach which is tailored to research software development. This new approach is shown in Figure 3. This approach is highly iterative in recognition of the iterative nature of research.

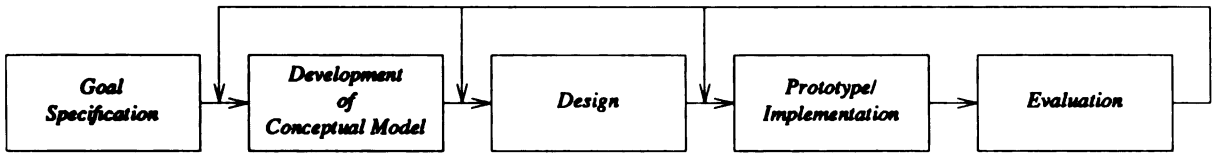


Figure 3 - Proposed Development Approach

The first step in this approach is the specification of the goals which the software is to meet. The goals should include goals for the software (e.g. extensible, network independent, etc.), as well as a statement of the purpose of the software (e.g. "a system to support a variety of tools for investigating biological macromolecules").

The next step is to develop a *conceptual model* which the users can use as an underlying foundation for their understanding of the system [18]. The conceptual model also provides a framework for the developers as they design and implement the system. The goal is to give the developers a framework which corresponds to the *users'* view of the system.

The system is then designed to meet the goals. The design should reflect the conceptual model internally as well as in the users' view. The design is then implemented. The implementation is then evaluated by the developers against the initial goals and by real users to determine the clarity and appropriateness of the conceptual model. If the users understand the model, but are unable to understand the software, then there is a flaw in either the user interface, the software architecture, or the model itself. These flaws are removed by iterating through the process.

The major advantages of this approach are the explicit incorporation of a conceptual model, the recognition of the iterative nature of software development in this environment, and the avoidance of time-consuming specification. The development of the con-

ceptual model *before* the design stage allows the conceptual model to become a key feature of the design. This approach also provides consistency for the user through adherence to the conceptual model. This allows the actual user interface to adapt to changing needs with a minimum of relearning by the users. One corollary of this is that *multiple* user interfaces could also be consistent at this conceptual level. This allows the development of user interface for specific devices or applications which have some degree of consistency.

2.4. Summary

The proposed development approach has several advantages for this environment. The idea of using a conceptual model as an aspect of the design process has been used with success by Xerox during the design of the Star [27] and has been suggested by Rubinstein and Hersh [23] as one part of their design methodology. The Xerox designers, however, never formalized the design process they used, nor did they discuss in any detail how the conceptual model influenced the design. They did strongly suggest that if a conceptual model were to be used, that it should be decided upon before the design process was begun so that the design could more closely reflect the chosen model.

Rubinstein and Hersh's treatment of the selection of a conceptual model is actually limited to choosing a *metaphor*. They also make this an optional part of their design strategy. If a metaphor is to be used, it is selected before the design, as is suggested here, but the overall methodology is extremely complex and the metaphor is not given the central role proposed here.

Chapter 3

The MacroMolecular Workbench - MMWB

3.1. Introduction

The Macromolecular Workbench, MMWB, is an environment being developed at the University of California, San Francisco for supporting research into the structure and function of biological macromolecules. This chapter discusses the issues involved in the design of user interfaces for MMWB. The diverse nature of the work involved in this research suggested that a worthwhile goal for the system design would be the integration of software from a number of sources and the simultaneous utilization of a variety of different devices.

The application domain for MMWB is the determination of the structure and function of biological macromolecules. This is an area which will involve scientists and researchers from a variety of areas including:

- X-ray crystallography
- NMR spectroscopy
- Protein chemistry
- Molecular modeling
- Physical chemistry

The eventual goal is to develop an integrated computer-based environment for investigators to utilize as a tool in the research of organic macromolecules.

3.2. Goals of MMWB

There are several goals that the designers of MMWB are hoping to achieve. The general aim is to design a system which could take maximal advantage of the resources available to the MMWB project, while providing for the flexibility and extensibility

which are necessary for a research platform in a rapidly growing field. The design goals of the system are:

3.2.1. Data Sharing

This is a clearly desirable goal in a cooperative research environment. It is imperative that two different researchers who are attempting to hypothesize the structure of the same protein using different methods have the ability to share data. This should be as seamless as possible without sacrificing the security of the data.

3.2.2. Resource Sharing

In most research environments resources are limited. It is almost always desirable to maximize the ability of researchers to share scarce resources. In particular, the availability of high-resolution graphics devices and compute servers is limited and they should be shared.

3.2.3. Multiple, Consistent User Interfaces

This goal actually follows from the previous goal. When researchers do not have access to the high-resolution graphics workstations, they still would like to be able to get work done. Much of the work can be done on lower resolution workstations and terminals. Having consistent user interfaces for these other devices will reduce the learning curve necessary to use them.

3.2.4. Compatible Computing Services

One of the problems with the current state of software in the computational chemistry area is that there are very few standards for data and interface formats. This

makes it very difficult to write an application which takes the output from other applications as its input. All applications in MMWB should be able to gain access by some well understood mechanism to the data which is output by other MMWB applications. In addition, mechanisms should be made available for applications to cooperate directly, or through an intermediary, to achieve some combined result. For example, a molecular dynamics application and a molecular display application should be able to cooperate to provide users with a visualization of the results as they are being calculated.

3.2.5. Extensibility

As new applications are developed, and older applications improved, they must be able to integrate into the system with a minimum of effort. This should help encourage the development of new applications, or the refinement of existing applications. In addition, the architecture must allow for the integration of new computing and graphics resources as they become available.

3.2.6. Distributed Processing

All of the computational resources in the MMWB environment should be easily available to researchers. For example, the system must allow a researcher on a graphics workstation to utilize the computing power of a supercomputer or minisupercomputer which is available on the same network.

3.2.7. Network Transparency

Ideally, the system should be designed to hide the network from the user. This type of system would allow applications to be run on any available resource in a

transparent fashion, without regard to the system type or the location of the application.

3.2.8. Implementable in Finite Time

It is important that the system be implementable given the available resources. This may seem obvious and trivial, but it is always important to be in touch with the real world constraints surrounding a design effort. It thus seemed worthwhile to list this as a specific goal.

3.3. MMWB Conceptual Model

The next stage in the design process was the selection of a conceptual model. For the purposes of MMWB the conceptual model chosen was a *user interface metaphor*. This is a computer-based analogy to a familiar concept or environment. The users' experience with the real world should aid them in their use and exploration of the computer-based world. The MMWB metaphor is that of a workbench, a chemist's workbench to be precise[2].

3.3.1. The Workbench Metaphor

The workbench metaphor has already been introduced in the first chapter. Briefly, the central concept behind the metaphor is that users utilize a variety of *tools* to develop, or complete, a *project*. For MMWB, the workbench metaphor has been restricted further to be a chemist's workbench. Chemists typically deal with a variety of *objects*, which

[2] Other conceptual models are possible. For example, rather than basing the system design on a metaphorical chemistry workbench, the conceptual model could focus on the idea of clients and servers.

may be chemicals and reagents, and they use *protocols* to mix those chemicals and reagents to achieve the desired final product, which is the completion of the *project*. During this process, a chemist may use a variety of *instruments* to monitor and modify the progress of the chemical reactions dictated by the protocols. *Instruments*, then, are tools which require a user interface, whereas *protocols* have a more restricted interaction with the user. The next section provides a detailed example of how this metaphor is used.

3.3.1.1. A Detailed Example

The central idea behind the concept of a project is that the work of the user is a process, where data are modified, combined, and created over time to produce some final result. This process is depicted graphically in Figure 4, which shows an idealized flow of object and process from primary structure to a putative tertiary structure. To the extent that this holds, the workbench metaphor will map very well into the user's application domain. The modification of data can occur in three different ways: the transformation of data of one type into data of another type (i.e. the object type changes), the addition or modification of data within a single object, and the modification of data within an object which gets written into a new object. In the latter case, we now have two objects of the same type within the project. This is certainly possible, but is not optimal for the workbench metaphor since the user will have to select which object they are interested in for all future tool invocations which operate on that object type. In the first two cases, there remains only one object of each type, and tools will have no need to request further selection information from the user.

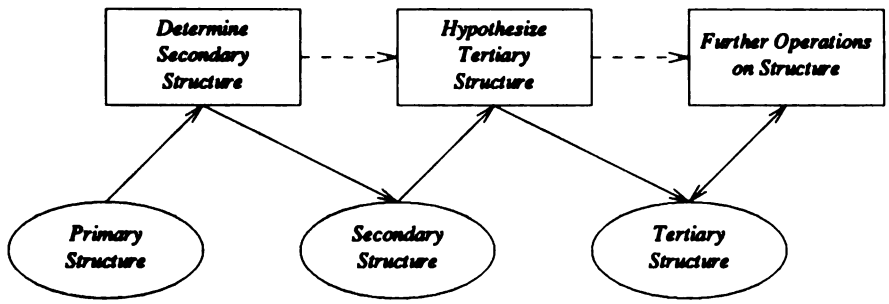


Figure 4 - Example Project Flow

Boxes represent operations performed on structure objects and ellipses represent the structure objects.

Consider, as an example, the process of deriving a model of a protein. The end result, or goal, is a file containing the residues, domains, and the three-dimensional coordinates of the atoms along with information about connectivity or bonding. There could be several starting points for this information: primary sequence, the structure of a related protein, NMR spectra, or partial X-ray crystallographic data. For the purposes of this example, the starting point is a primary sequence which has been determined utilizing genetic technology. The user creates a new project, selects it, and stores within that project an object of type *primary sequence* which contains the sequence data. The first step might be to investigate the secondary structure of the protein. The user selects a tool which (hopefully) can give him/her a predictive secondary structure.[3] The tool should be written to produce a new object, of type *secondary structure*. This object may then be used by the user to hypothesize a three dimensional structure. The resulting three dimensional structure will be a third object and its type might be *molecular data*. This is a more complex object than the other two because it is actually the final form. All of the future manipulations will actually involve modifications of this object, or additions to it.

[3] Several such tools have been written, although their predictive ability remains an area of research and debate.

Note that through the entire process, the user never has to specify the object to operate on. Each of the tools is specific to a single type of object and selects the appropriate object based upon the current project and the list of objects of its type within that project.

This example presents the workbench metaphor in its best light. The user has only to select the project and then all other operations occur without further selection. There are several circumstances when this breaks down and the user interface must provide mechanisms for tools to request further selection information from the user. In the above example, if the user wants to hypothesize a second tertiary structure to compare with the first, he/she could either create a new project, or add a second object of type *molecular data* to the current project. In this case, there would be two objects of that type and the user would have to provide selection information where appropriate. Note that it may not always be appropriate to select just one of these. For example, one operation a user may want to perform is a visual comparison of both of these structures. In this case, the molecular modeling program, MIRAGE, would simply open both of the structures and allow the user to select them using MIRAGE commands.

3.3.1.2. Extensions

The workbench metaphor immediately suggests two extensions which are not strictly part of the metaphor. A metaphor allows users to interact with a variety of different devices and interfaces. If this is to be a feature of the system a mechanism needs to be provided which allows software tools to have some device independence, especially those which are not particularly interactive. For example, a tool which does molecular mechanics in a batch mode only needs to interact with users to obtain initial parameters and (perhaps) provide some occasional status or update information. It is not acceptable

to require the author of such a tool to implement a version of the tool for each different type of interface. **MMWB** provides such a mechanism which is similar in concept to the Apple Macintosh *dialog box*. The similarities are strong enough that the **MMWB** term for this mechanism is a *dialog*. Dialogs provide an interface-independent mechanism for interaction between "mostly" batch tools and the users. This mechanism provides the ability for users to interact with these tools from a variety of devices.

The next extension is to provide a mechanism for users to *change* user interfaces without terminating the running tools. **MMWB** provides the concept of a *session* which allows users to change devices or user interfaces without any loss of interaction with currently active batch tools.

3.3.2. The MMWB Workbench

The workbench metaphor has been implemented as an integral part of the **MMWB** architecture. The applications interface to the data is based upon objects which have associated properties. One type of object is a *project*; projects may contain other objects. Another type of object is a *tool*, which may be either a *protocol* or an *instrument*. These objects contains all of the information necessary to execute that tool. There are two "special" projects which have been implemented in **MMWB**. The first is the *prototype project* which contains all of the default tool descriptions and prototypes of all objects in the system. This provides a mechanism to register objects with a system administrator or system manager to avoid object type clash.

The second "special" project is the user's *default* or *home project*. This project contains the user's versions of prototype tool descriptions. It also contains any additional tools or multi-step protocols that the user desires to have accessible. The user default

project may also contain objects of type *state* which contain information about user interface and session state.

3.3.3. Alternatives

There are several alternatives to using a metaphor to ease the transition between systems. The first is to simply have no linking user interface at all. That is, each component of the system would be written with the user interface that best suited the implementor. This would have the advantage that each user interface could be tailored to best meet the goals of the system, but would have the disadvantage that users would have to relearn each component with no obvious transfer of knowledge or experience from the previous components.

Another alternative is to enforce a least common denominator. This would basically have to be some form of menu-style interface which could be run on any available type of hardware. The obvious exception to this would be the molecular modeling program, **MIRAGE** [11], which would have to have some convenient means of controlling the models being displayed on the screen. This alternative has the advantage of being easiest to implement (a series of library routines to display and process menus), and providing the most consistent user interface to all but **MIRAGE**. It has the disadvantage of not making the best use of more modern workstations. In addition, wading through menus would become very tedious for experienced users.

Chapter 4

The MMWB Design

4.1. MMWB Architectural Overview

The goals for MMWB are reflected in the system design as shown in Figure 5. Resources in the system are controlled by specific resource managers. A session manager, SM, is responsible for all tool initialization and communications; a data manager, DM, is responsible for all data access and concurrency control; a dispatcher is responsible for contacting tools on the network; and a user interface manager, UIM, is responsible for managing the input and output devices, and any graphics or display-intensive tools. The Foreign Tool Driver, FTD, provides an interface to tools which are available only in binary, or are too cumbersome to convert to run under the more functional MMWB architecture.

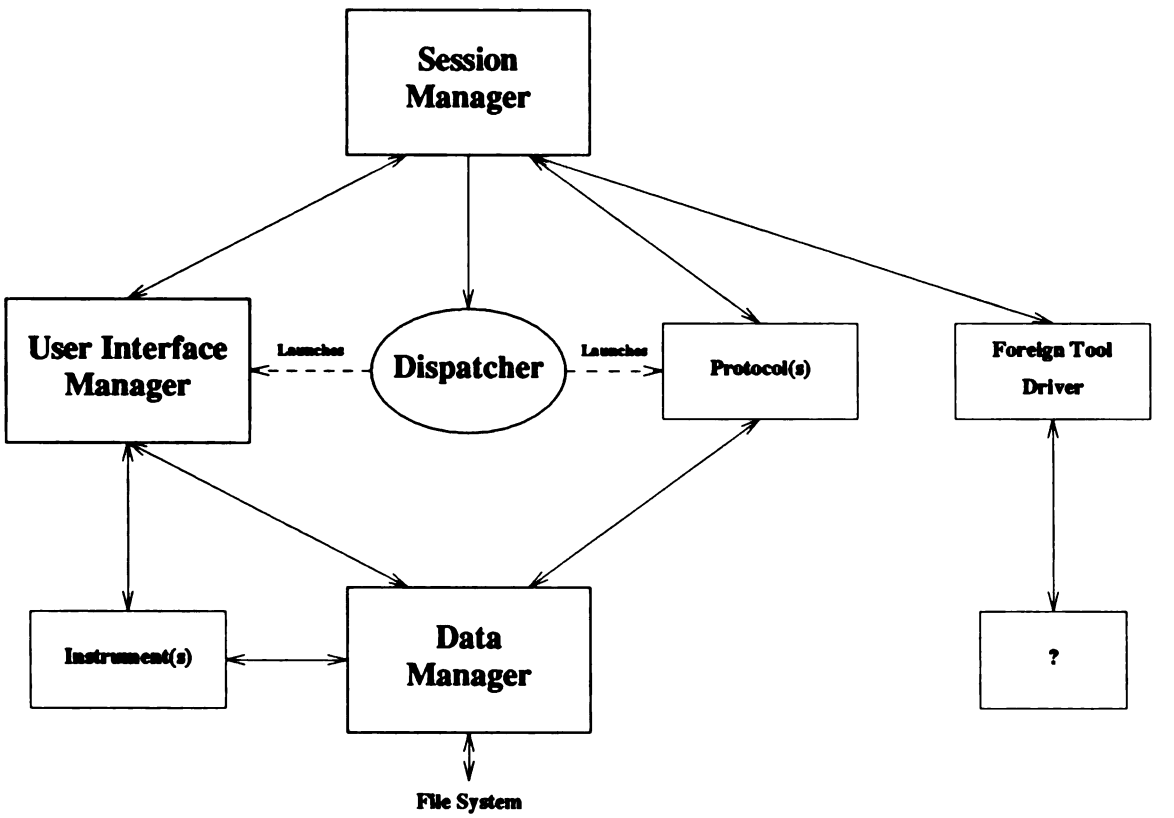


Figure 5 – Process Architecture of MMWB

MMWB data are organized into collections called *projects*. The DM manages all data access and concurrency for a single project. There is one, and only one, DM for any active project. The DM is executed by the dispatcher at the request of the user interface when a project is selected. The DM will accept requests from clients anywhere on the network so that users on different machines may share a project (which may be on yet another machine). This also allows data sharing for machines on the network which may not be linked by a remote file system such as Sun's NFS. The basic data model supported by the DM is a simple table with ordered rows and named columns called *attributes*. To retrieve a data value from a table, the application only needs to specify the row number and the attribute name. A variety of different data value types are supported, including a *table* type, which allows recursive tables. A more complete description of

DM is available in [11].

The dispatcher server is responsible for providing access to services on a network. The dispatcher will launch the service, establish communications with it, and return the communications streams to the calling program. The dispatcher is used not only by SM to launch tools requested by users, but it is also used to find the appropriate DM for the selected project. The use of a dispatching service provides an easy mechanism for launching tools on different machines.

A UIM manages the screen device itself. Under some circumstances, for example an ASCII terminal, the UIM is actually the user interface. For multi-windowed bit-mapped graphics displays, however, the UIM provides the basic user interface functions necessary to implement the workbench metaphor. Other user interfaces may be present in separate windows on the display. For example, on a Silicon Graphics Iris it would be possible to have a molecular modeling program (MIRAGE) and a molecular dynamics display on the screen at the same time. All communications between these user interfaces and the rest of the tools will be handled by the UIM, or through objects in the file system. Note that there is a different UIM for each display device supported by the system. In fact, it is possible to have more than one UIM for a given device to present different user interfaces to users. A UIM utilizes the underlying windowing system and window manager to implement its user interface. This is important to maintain consistency with other applications being executed on that device. A UIM should not be confused with a window manager, however. Window managers typically implement some basic window manipulation policies and functionality and rely on applications to implement the semantics of an interface to the underlying system. A UIM provides a complete inter-

face to a system and manages the *tools* themselves, not simply the windows. The design and functionality of user interface managers will be discussed in more depth below.

The Session Manager, SM, serves two functions. First, the SM manages all communications for a *session*, which is the collection of protocols and (possibly) a user interface manager. Second, as the main interface to the dispatcher, the SM manages the execution of the tools themselves, including controlling the sequential execution of tools in a script, and maintaining the list of available tools for delivery to the user interface.

There are two forms of communication between **MMWB** and users. The first is through an *instrument*. This takes on a well defined meaning in **MMWB** because tools are classified by their need to interact with users and the nature of that interaction. For example, **MIRAGE** would certainly be considered an instrument whereas a tool which only needed to interact to gather initialization conditions, or occasionally to display status, would be considered a *protocol* rather than an instrument, even though it requires occasional input from the users. All instruments run under a **UIM**, and must satisfy the requirements of that **UIM** for interaction and interface. In general, instruments are **device specific**. The second type of user interaction is through *dialogs*. Dialogs are discussed in detail in section 4.3. A dialog is a device independent mechanism for soliciting input from a user. Tools which do not require direct interaction with a device will communicate with users using this mechanism. Dialogs have several distinct advantages over other forms of interaction:

- they are device independent
- they are user interface independent
- they persist across user interfaces
- they are asynchronous
- they can utilize semantic knowledge

Dialogs are presented to users by the current **UIM**. If no **UIM** is running when a tool sends a dialog to a user, it is queued by the **SM** until a **UIM** is available. A **UIM** must also implement the basics of the system interface including project selection, object manipulation, and tool initiation.

The **Foreign Tool Driver** is a tool which interfaces to other tools which have not been written to conform to the **MMWB** architecture. The **FTD** captures all output from the foreign tool and searches for strings which match patterns defined by a series of rules inside of a file. If a match is found, that rule is executed (see Section 4.5). When these strings are detected, **FTD** captures them and sends a dialog to the user. The users enter the appropriate information and **FTD** then sends that information to the tool as if it were typed on a terminal. This mechanism will allow for the integration of a large number of tools, although it is far from perfect.[4]

The architecture described above provides for a number of the **MMWB** goals. The existence of the **Dispatcher** allows distribution of compute services around the network. The **Data Manager** provides a mechanism for compatible data handling across applications. The **Session Manager** allows a user to communicate with a variety of tools in an interface-independent fashion using *dialogs*. It is easy to add tools to the system, and the **Foreign Tool Driver** provides the ability to interface to tools which do not conform to the **MMWB** architecture.

[4] It would be nice, for example, to be able to handle more advanced user interfaces, such as forms or graphics.

4.2. Sessions in MMWB

The SM serves two roles for the MMWB system. First, it acts as a session manager, controlling all communications between tools and the user interface. It is also responsible for insuring delivery of messages to a user interface when one becomes available. Second, SM acts as a tool manager, launching tools when requested and making options available to the tools. Collections of MMWB protocols which are to be executed either in sequence or in parallel (*multi-step protocols*) are processed and controlled by SM.

4.2.1. Session Management Functions of SM

An MMWB session is a collection of an SM, possibly a UIM, and possibly one or more tools. There must be at least one tool or a UIM present; in fact, SM will exit if there are no tools and no UIM active. MMWB does allow users to change user interfaces or to terminate a user interface. The purpose for this is to allow a typical scenario in which a user initiates a user interface from home on an ASCII terminal to start some long jobs, then logs off, drives to the lab, starts a user interface on a graphics display and reattaches to the running session through the session manager. Generally, scarce resources like high resolution graphics displays can be managed more easily if users have the ability to utilize more available resources without terminating any running tools.

SM manages all communications between the protocols and UIM. The communications occur primarily in the form of *dialogs*. As defined above, a dialog is an interface-independent request for information from a user. All user interfaces implement the complete set of dialog types, but they are free to implement them in any fashion. The important consideration for session management is that all communications between tools are implemented as dialogs.

There are three major types of communications which must be made available for MMWB operation. The first of these is user interaction with tools. This is the major use of dialogs and they provide a very simple mechanism for tools to request values from the users. The problem is that there is no guarantee that a user interface will be present when a tool issues a dialog request, nor is there any guarantee that users will respond to a dialog request before terminating their session. It is the responsibility of SM to deliver dialog requests to the user interface, and to deliver the response to the tool. If no user interface is available at the time a dialog request is issued, SM simply accepts the request and queues it up for eventual delivery to a user interface. When a user interface attaches to SM, all queued dialog requests are sent to it. A dialog request is held in the queue until a reply has been received from the user interface.

One special type of user interaction with tools is called a *notifier*. Notifiers are special types of dialogs which notify the user of the completion codes or status of tools. No response is possible to a dialog of this type, and in this special case, the dialog is dequeued as soon as it is delivered to the user interface.

A possible extension of the dialog delivery mechanism might be the ability to "log" all dialogs to provide a transcript of all interaction with a given tool. This functionality could become part of a "lab notebook" which could be utilized by users to record their activities. This could not be used, however, to eliminate interaction with the users. The modal nature of dialogs (other than notifiers) would still require user interaction before tools could continue. This is discussed in more detail in Chapter 5.

The second type of communications supported is that between two cooperating tools. In this case, a very simple form of dialog is used called a *message*. Messages are

used primarily for synchronization, although it is possible to use them to pass values between tools.

The final type of communication supported is the *option*. An option is a value which is passed to a tool at the tools' request which does not require any user interaction. Options are used primarily by UIMs to allow users to set various parameters and defaults for tools before invoking them. Options allow the user a great deal of flexibility. First, potentially all of the dialogs which could be sent by a tool could be answered in advance using the option mechanism. This would allow the users to launch tools and then be free from any interaction until they terminate. Second, required options are processed before tools are actually invoked. This allows users to make sure that all required information is present before tools ever get launched. If they desire to abort at this point until the necessary information is available, they may do so, with an assurance that no data has been touched.

One interesting side note about the design of options: because options are implemented as complex dialogs, the same interface is presented for options as for dialogs. This preserves a consistent look and feel within the same user interface. *Options*, *messages*, and *notifiers* are discussed in more detail in section 4.3.

4.2.2. Tool Management Functions of SM

In addition to its session management functions, SM also serves as the tool manager for MMWB. If all tool invocation is done by SM, then it is easier to establish the communications links necessary for its session management functions and eliminate the need for complicated communications rendezvous schemes. This simplifies the design of the MMWB communications as a whole.

The process of tool invocation is reasonably straightforward. SM validates the invocation request, reads and validates the options, and then sends a request to a server called the *dispatcher* to locate the tool on the host and launch it. The dispatcher sets the appropriate communications in place and provides a stream to the tool.

The role as tool manager for SM extends beyond tool invocation. SM also interfaces with the file system to locate all of the system and user prototype tool information. The information retained for tools is shown in Table 2. This information is stored in two places. The first is in the system prototype project. This project contains information about all of the tools known to the system. Default values are present for each of the options in the option sheet for each tool. This information is read by SM for all known tools, and this information is passed to the user interface. The information in the system prototype project may only be modified by the system administrator. The second place where this information is stored is in the user's home project. Any user modifications to system tool descriptions are saved in the user's home project. If a system tool description has a higher version number than the user's tool description, a warning is issued to the user interface so that the user may update to the system version. This is to allow the system administrator to add options or change the location of tools without having to update all user copies of the tool description. Any tool descriptions for privately developed tools are also stored in the user's home project.

Value	Type	Description
Name	text	The name of the tool
Version	text	The version of the description
nodisplay	boolean	If set, don't display the option sheet
Options	option	The option sheet (collection of all options) for the tool
Subtool	text	A list of subtools to be run
OnError	integer	What to do upon termination. Possible values are: **SM_IGNORE**, **SM_DIE**, and **SM_KILLB**.
Host	text	List of hosts to search for the tool, may include the keywords *default* and *die*.

Table 2 - Tool Description Properties

SM is also responsible for the management of tools which have been chained together into *multi-step protocols*. As can be seen from Table 2, a tool description may contain a list of subtools which actually make up that tool. In this case, the tool description can be considered a type of script which directs SM to execute each of the tools in turn. The option sheet which is present in this case is actually a compound option sheet containing options for all of the tools. Each option sheet is numbered and delivered to the appropriate tool. Option sheet zero, however, is delivered to all tools in the multi-step protocol. This allows a single option to be filled out only once for all tools.

Multi-step protocols are typically sequential executions of tools. The syntax for the **Subtools** property of the tool description is a list of valid tool names separated by commas. Protocols may be executed in parallel, however, and for parallel execution, the syntax must be a little more complicated. Parallel streams are separated by a vertical bar, and sequential streams are grouped within a parallel stream by brackets. For example, to

execute the five tools as shown in Figure 6, the **Subtools** property would contain the text:

Tool1, {Tool2, Tool3} | Tool4, Tool5

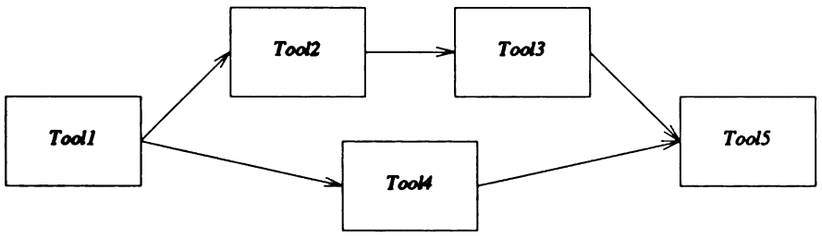


Figure 6 - Example Parallel Tool Execution

To execute six tools with Tool4 and Tool5 running in parallel with Tool2 and Tool3, the **Subtools** property would contain:

Tool1, {Tool2, Tool3} | {Tool4, Tool5}, Tool6

Finally, to execute five tools with Tool2, Tool3, and Tool4 all running in parallel, **Subtools** would be:

Tool1, Tool2 | Tool3 | Tool4, Tool5

It is worthwhile to note that the options for multi-step protocols are derived completely from the tool description, and except for checking versions, the system and user prototype project definitions for the tools are not consulted. This allows tools which are executed as part of a multi-step protocol to have different options than when they are executed by themselves.

Error conditions for multi-step protocols are specified by the **OnError** property of the protocol description. This allows the user to specify whether the protocol should continue after errors have occurred or should abort immediately on errors, and how to deal with parallel branches. SM maintains this information for each running protocol and is the agent responsible for carrying out the appropriate action.

4.2.3. SM Design

SM is implemented as two separate pieces, a server and a library. The SM server is responsible for maintaining the state of all dialogs, including options, messages, and notifiers, and performs most of the tool management functions. The library provides interfaces to the dialog mechanism, the tool invocation mechanism, options sheets, and tool lists. There are three sections to the library: a section for tools (instruments and protocols), a section for User Interface Managers, and a section which contains common routines. Manual pages for each of these libraries are included in Appendix C. The major difference between the tool-specific section and the UIM-specific section is in the communications protocols. The SM-Tool library implements a strict client-server protocol, while the SM-UI protocol is peer-peer. The protocols are presented more fully in Appendix B. The rest of this section discusses each of these components of SM.

4.2.3.1. SM Common Library

The SM library contains a list of routines which are common to both UIMs and protocols or instruments. These routines are shown in Table 3. Several of these routines implement the interface between the SM and the underlying object file system, OFS [11]. These routines allow the caller to retrieve the current project, as well as the user's home project and a path to the system project. These routines, and the corresponding routines which allow the setting of these projects, are seldom used directly by the client program.

Function	Description
sm_error	Return the last error message
sm_get_curr_proj	Return the current project
sm_get_home_proj	Return the home project
sm_get_sys_proj	Return the system project
sm_kill	Kill a tool (protocol or instrument)
sm_launch	Launch a tool (protocol or instrument)
sm_read_dialog	Read a dialog from the SM
sm_read_reply	Read a reply from the SM
sm_send_dialog	Send a dialog to the SM
sm_send_reply_tool	Send a reply to a tool (from a tool)
sm_session_status	Return the session status (list of tools)
sm_set_curr_proj	Set the current project
sm_set_home_proj	Set the home project
sm_set_sys_proj	Set the system project
sm_set_file	Set the channel used for SM communication

Table 3 - SM Library Functions

Other routines in the SM library provide a mechanism to launch other tools (protocols or instruments), determine what tools are running, or kill a running tool. The routine to launch a tool, *sm_launch*, does not provide any access to the option sheet mechanism, so should only be called by programs which know the exact nature of the tool to be launched. It is provided in this library for use by programs which desire to launch and establish communications with subordinate tools such as the possible execution of a protocol by an instrument. One of the dialog interfaces, *sm_send_reply_tool*, is provided specifically for communication between tools.

4.2.3.2. SM-UI Library

The SM-UI library, SMUI, is the only interface between the UIM and the SM server. The goal of SMUI is to provide a simple, clean interface for UIMs to both the session management and the tool management functions of SM. The relationship between the UIM and SM is a peer to peer relationship, that is, either the UIM or SM can initiate

communications. The implications of this are that messages may originate from SM or UIM at any time. Care must be taken to insure that the communications protocol avoids any race conditions resulting from this. The functions which are provided by SMUI are shown in Table 4.

Function	Description
sm_attach_session	Attach to an existing session
sm_cancel	Cancel a tool invocation
sm_detach_session	Detach from a session
sm_dispatch	Handle SM input
sm_invoke	Initiate tool invocation, including option sheet handling
sm_list_sessions	Return the current active sessions
sm_send_reply	Send a reply to a dialog

Table 4 - SM-UI Library Functions

The design of the communications protocol between SMUI and the SM server must deal with the peer to peer relationship between SM and the user interface manager, UIM. The difficulty with peer to peer communications is the avoidance of race conditions caused by messages which are initiated simultaneously by each end of the connection. In the case of communications between SM and the SMUI library of a UIM, this is handled by forcing the SM server to be intelligent and handle any incoming requests from UIM processes, regardless of the current state of SM. When a UIM receives a request from the SM server which is not in answer to its request, it simply drops it. The SM server is responsible for resending its request after responding to the UIM. This approach makes a great deal of sense in the context of the role of SM as a session manager. SM is already required to maintain queues of all pending dialogs, so that all SM must do is mark the dialog in some way to make sure that it gets redelivered after SM answers the user's request. This is also consistent with good user interface design because it lets the user

have control.

Most of the functions implemented by the SMUI library are actually simple interfaces to the communications protocols which connect a UIM to the SM server. Much of the processing for tool invocation, however, is actually handled by the SMUI routine *sm_invoke*. When the user interface requests that a tool be invoked, *sm_invoke* will validate the options and insure that all required options have values by calling the dialog handling routine which is supplied by the UIM, *uim_dialog*, as needed. This results in a reduction of communications time and a quicker response to the user.

One additional function provided by SMUI is the ability to reattach to a running session. The SM server will store an object in the user's prototype project which contains all of the information necessary to reattach. This information includes a process identifier for the SM server and the machine the server is running on. This information is used to establish communications with the running SM server. Once communications are established, all pending dialogs are sent to the user interface, and processing continues. Users may only connect to their own running session, and can only connect if they have access to the session object. This limits some functionality (e.g. it would be nice to have a "group" working on a single session), but avoids the obvious security concerns. Any other state information about screen displays or locations are the responsibility of the user interface.

4.2.3.3. SM-Tool Library

The SM-Tool library, SMT, is very similar to SMUI. It implements a similar set of functions as shown in Table 5. The major differences are that specific functions are provided for option requests, and the relationship to the SM server is pure client-server. All

requests are made by the SMT library, and no communications are expected from SM unless they have been requested.

Function	Description
sm_dialog	Send a dialog to the UIM
sm_exit	Terminate tool execution
sm_init	Initialize the SM
sm_notify	Send a notification dialog to the UIM
sm_request_int_option	Get an integer valued option
sm_request_text_option	Get a text valued option
sm_request_float_option	Get a floating point valued option
sm_request_bool_option	Get a boolean valued option

Table 5 - SM-Tool Library Functions

4.2.3.4. SM Process

The SM Process, SMP, is the central focus of both session management and tool management. SMP maintains all of the queues of dialog requests and monitors the status of all tools, including the user interface. This monitoring is more complicated than it may first appear. Tools are actually initiated by the Dispatcher, as discussed in the overview, not by SMP. This means that the normal exit status which is returned by the operating system to parent processes is not available to SMP when a tool exits. To overcome this problem, SMP monitors the status of the communication streams. Upon a normal completion, a dialog is sent by the tool notifying the user (and coincidentally, SMP) of the completion code. An abnormal completion may, or may not, result in a notifier. If a tool terminates with no completion code, however, the communications channel will be closed by the operating system. This will be used by SMP to determine that a tool has terminated. SMP will then issue a notifier to the user interface stating that the tool has terminated with an unknown completion code.

SMP utilizes this technique to monitor all processes in a multi-step protocol. Unknown completions are assumed to be errors, and SMP will take the appropriate action as described by the **OnError** variable in the tool description for the multi-step protocol. After SMP receives a request to invoke a multi-step protocol it interprets the tool specification and prepares a graph of tool execution. The first set of tools are invoked and their options are extracted from the combined option sheet and are prepared for delivery as required by the tools. Tools within a multi-step protocol may launch other tools and establish communications with them using the dialog mechanisms described above. As each tool completes, SMP determines which tool or tools to invoke next, extracts their options and launches them. This process continues until a fatal error occurs or until the final tool or tools complete. The final completion code is returned to the user interface.

Another function provided by SMP is the ability to rendezvous with user interfaces which need to reattach. When a user interface terminates, SMP writes an object into the user prototype project which contains a contact location for the session. When a user interface wishes to reattach, it reads the object out of the user prototype project, and attaches via the contained location. Once communications are established, the object is removed to prevent other user interfaces from attempting to attach to the same session.

4.3. Dialogs

4.3.1. Description and Use

Dialogs are primarily interface-independent requests for information from the user. They can also be used for inter-process communication and to provide information to the

user without requiring any response. Because dialogs are independent of the actual user interface which is displaying them and interacting with the user, tools become completely independent of the user interface.

Dialogs are an important part of the provision for multiple user interfaces. The major aspects of dialogs which allow for multiple user interfaces are:

- Dialogs are interface-independent, so that a dialog can be presented by any user interface.
- The SM queues dialogs for delivery, so a user interface manager does not have to be active for a tool to request input from the user.
- User interface managers can reconnect to a running SM and receive all of the queued dialogs. This provides for guaranteed delivery of the information.

Interface independence for dialogs is achieved by defining a rather limited set of dialog *types*. The type of a dialog is defined to be the type of the value being requested or sent to the user or tool. The set of dialog types is shown in Table 6. All user interface managers for MMWB are required to implement the entire set of dialog types. This provides a mechanism for any MMWB tool to retrieve information from the user without regard to the user interface the user is actually interacting with. This is an important criterion because, as discussed above, the user interface may change during the tool's execution time.

Type	Returned Value
IValuator	Returns an integer value
FValuator	Returns a floating point value
Text	Returns a text string
Select	Returns an index into the selection values. The selection list is passed in the **List** component.
Boolean	Returns a simple TRUE or FALSE, often used for Continuation requests
YesNo	Returns a TRUE for yes or a FALSE for no
Notifier	No return, **Value** contains the integer completion code

Table 6 - Dialog Types

One of the disadvantages of a small set of dialog types is that user interfaces and tools which share knowledge about the semantics of objects are not able to utilize that semantic information to implement intelligent dialog response interfaces. For example, a tool which requires a starting residue number for some computation could only send a **Text** or **IValuator** dialog to be filled in by the user. A nicer user interface for this request would be to have an image of the molecule on the display and to be able to select the starting residue with a pointing device of some form. This would be possible if the current user interface shared the semantics of **residue** with the tool. To overcome this disadvantage, **MMWB** dialogs have a *semantic type* field which can be used by a tool to pass semantic information to a potentially intelligent user interface. User interfaces which do not implement that semantic type present the dialog to the user as if that field did not exist. In the example above, the dialog would be presented as a normal **Text** or **IValuator** dialog. If the user interface does implement that semantic type, however, more sophisticated user input and error detection mechanisms can be used, such as using a mouse to select a residue from a molecular model on the screen.

4.3.2. Design of MMWB Dialogs

4.3.2.1. Simple and Basic Dialogs

The internal data structure of **MMWB** dialogs is shown in Table 7. The primary use of a dialog is to request some form of information from the user. For this use of a dialog the **Name** field contains the name of the value, the **Type** and **SType** are set to the **Value**'s type and semantic type, respectively. If **SType** is set to a semantic type, the text value **SInfo** may contain information which is necessary for the understanding of the semantic type. For example, if the semantic type is **object**, the **SInfo** value might contain the name of the project which contains the object. **Upper** and **Lower** are used for primitive bounds checking on input. These provide a means by which the tool can pass bounds information to the user. The **Prompt** and **Help** information may be provided for possible use by the user interface for presentation of the dialog and context-sensitive help. The **Source** and **Dest** fields are filled in by SM. The **ToolName** is filled in by the tool, or by SM if it's blank. One additional field used by simple dialogs of type *Select* is the **List** field which contains a list of values which provide the selection list. This type of dialog is called a *simple* dialog. If **SType** is zero, then it is also a *basic* dialog. A sample simple dialog for the **residue** example discussed above is shown in Table 8. An example showing how a user interface manager might present this dialog to the user is shown in Figure 7.

Field	Type	Description
Name	**text**	The name of the value
Type	**integer**	The type of **Value**
Stype	**integer**	The semantic type of **Value**
SInfo	**text**	Any information necessary for semantics
Prompt	**text**	A prompt to be displayed by the user interface
Help	**text**	Context help text
List	**text**	A list of values for a selection
Value	**value**	The returned value
Upper	**value**	The upper bound of **Value**
Lower	**value**	The lower bound of **Value**
Flags	**integer**	Various flags
Source	**integer**	The tool ID of the dialog source
Dest	**integer**	The tool ID of the dialog destination
ToolName	**text**	The name of the tool generating the dialog
Option	**integer**	For use by option sheets

Table 7 - Dialog Contents

Field	Contents
Name	*Starting Residue Number*
Type	**IValuator**
Stype	**RESIDUE**
SInfo	*Human Relaxin*
Upper	**55**
Lower	**1**
Prompt	*Enter the starting residue*
Help	*The calculations performed may be bounded by selecting starting and ending residues. The default value is the beginning of the molecule (e.g. residue 1).*
List	**NULL**
Source	**toolID**
Dest	**0** (0 is the tool id of the UIM)
ToolName	*Calculate hydrophobic moment*
Option	**0**

Table 8 - An Example Dialog

4.3.2.2. Compound Dialogs

A problem with dialogs arises when a tool requires the user to input multiple values. Presenting the user with a box for every value would be unwieldy and tiresome. To

overcome this, **MMWB** dialogs may be *compound*. A compound dialog is identical to multiple, simple dialogs except that they are joined together, and all have the same **Source, Dest,** and **ToolName** fields. No restriction is placed on the UIM about how to implement the user interface for this, but it is the current suggestion that compound dialogs should be presented together. Taking again the **residue** example above, if the tool sent a compound dialog which contained requests for the starting and ending residues, one possible user interface is shown in Figure 8.

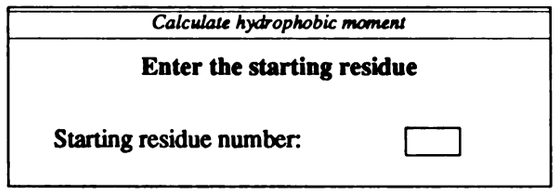


Figure 7 - An Example Dialog Interface

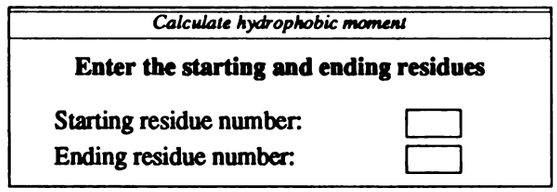


Figure 8 - An Example Compound Dialog Interface

4.3.2.3. Notifiers

In addition to the normal use of dialogs discussed above, there are also some special forms of dialogs which are used for specific purposes. One special form of dialog is called a *notifier*. Notifiers are used to notify the user of the completion codes or status of tools. No response is possible to a dialog of this type, and in this special case, the dialog is dequeued as soon as it is delivered to the user interface. A Notifier is designated by a value of **Notifier** in the **Type** field of the dialog.

4.3.2.4. Messages

The second special form of dialog supports communications between two cooperating tools. In this case, a very simple form of a dialog is used called a *message*. Messages are dialogs which do not have **Prompt**, **Help**, or **List** fields. Messages are used primarily for synchronization, although it is possible to pass values between tools in the **Value** field of the dialog. Because there are many tools running which may desire communications of this sort, the tool is responsible for filling in the **Dest** field with the tool ID of the tool it wishes to send the dialog to. Currently, the only ways for a tool to determine the tool ID is as a return value from an invocation request to SM, or from the response of a call to *sm_session_status*. In addition, the special tool ID of zero is returned for the current UIM. SM will guarantee message delivery as long as the tool referenced is present. If the referenced tool terminates, an error is returned to the sending tool. A message dialog contains a **Flags** value **DI_MESSAGE**, which is primarily used to determine what action to take if there is no recipient.

4.3.2.5. Options

The final special type of dialog is an *option*. An option is a value which is passed to a tool at the tool's request which does not require any user interaction. Options are used primarily by UIM to allow users to set various parameters and defaults for tools before invoking them. When a UIM invokes a tool, it sends a very large compound dialog called an *option sheet* to SM which contains the values for all options which may be requested. SM accepts these options, and validates them to insure that all required options are present and have values. A required option is a dialog which has the **DI_REQUIRED** flag set. Any required options which are not present are sent back to

the user interface in the form of dialog requests which can actually be answered directly by the user. After all required options are present, the tool is invoked. Each dialog request that comes from the tool is checked against the list of options. If the dialog request can be answered by an option, the response is sent; otherwise the dialog is forwarded to the UIM.

4.3.3. Comparison of MMWB Dialogs with Macintosh Dialogs

MMWB dialogs are patterned after Macintosh dialog boxes; however there are several important differences. These differences are outlined below:

- There are no modal dialogs in MMWB, as there are in the Macintosh. Macintosh modal dialogs require the user to dismiss them before any other action can happen on the machine. There is no need for modal dialogs in MMWB because of the multi-tasking nature of the underlying environment. In fact, the tool sending the dialog may not even be running on the same machine.
- MMWB dialogs may be compound. Macintosh dialogs may have multiple input items, but they are part of the same dialog. A compound dialog in MMWB is actually just a list of simple dialogs. Each MMWB dialog may input only one item of data.
- MMWB dialogs are user-interface independent. A Macintosh dialog includes information about the way to display that dialog, and may even include graphics calls and objects. This is consistent with the Macintosh model of a single, consistent interface. One of the goals of MMWB, however, is to allow multiple user interfaces.
- MMWB dialogs make provisions for semantic types. Macintosh dialogs are restricted to the provided types.

- **MMWB** dialogs may be used for display only, without any input requirements. This is possible on the Macintosh (i.e. no control items), but it is not part of the intent of dialogs.
- **MMWB** dialogs may be exchanged between two tools as a means of inter-process communication. Dialogs always interact with the display on the Macintosh.

4.4. User Interface Managers

The previous sections have referred repeatedly to **MMWB** user interfaces. Actually the user's interface to the basic system is implemented by a process called a User Interface Manager (UIM). User interface tools (*instruments*) in **MMWB** differ from user interface managers in a variety of ways. A UIM must provide certain functionality. First, it must implement the workbench metaphor and allow users to select and manipulate projects, objects, and tools. Second, it must implement interfaces for *all* dialog types, and interface with SM for exchange of dialog information. Finally, it must provide mechanisms to launch tools, and communicate to SM the tool description information.

An instrument runs under a specific user interface manager. The instrument is typically specific to a given application or task. Instruments communicate only with the user interface manager and the data manager, and are not required to implement any dialog handling except the semantic dialogs which they choose to support. The protocol between the user interface manager and an instrument is undefined because it may be specific to the window system or environment in which the instrument and user interface manager are implemented. This allows implementors of UIMs to take advantage of the capabilities of the target workstation or window system. The functionality which must be provided between UIM and instruments is defined and discussed below.

4.4.1. Workbench Metaphor Interface

The UIM is the user's primary interface to the workbench. Table 9 lists the functions which must be supported. Table 10 lists some additional functions which should be supported by UIMs whenever possible. Whether a UIM implements the functions in this set depends on the features of the user interface and whether or not it will handle instruments underneath it.

Function	Description
Project Lists	The UIM must provide the capability to list all of a user's projects
Project Contents	The UIM must provide the capability to list all of the objects within the selected project.
Project Selection	The UIM must provide a means to select a project.
Project Maintenance	The UIM must provide mechanisms for project creation, naming, and deletion.
Object Maintenance	The UIM must provide mechanisms for object creation, naming, and deletion.
Tool Lists	The UIM must provide a means to list all of the available tools, both *protocols* and *instruments*.
Tool Maintenance	The UIM must provide a means to modify local copies of a tool description. This includes setting options in the tool's option sheet.
Dialog Handling	The UIM must implement user interfaces to all of the basic dialog types.
Tool Invocation	The UIM must provide a means to invoke a tool.
Session Connection	A UIM must be able to connect to a running session and retrieve all active dialogs for display and user input.

Table 9 - Minimum UIM Functionality

The role of the UIM as the interface to the Workbench is critical to the system's success. The degree to which the interface is faithful to the metaphor will to some extent

determine the amount of cognitive transfer that a new user will be able to apply from experience with previous UIMs. UIMs which have been developed for graphical interfaces should take advantage of icons to reinforce the metaphor. For example, a simple user interface on a bit-mapped graphics workstation may start up with three icons, one representing the project list, another representing the list of protocols, and the last representing the list of instruments. The user would open these, perhaps using a double-click with a mouse to view the list of projects or one of the lists of tools, respectively. When the user views the tool and project lists, the iconic representations should be informative. Different object types in the project should be distinguished by their icons. Tools might also be distinguished by having part of their icon be the icon of the object(s) on which they operate. This may turn out to be somewhat cluttered, but gives a good example of how icons might be used.

The example user interface described above has some interesting features. The distinction between tools and projects does not align with the system model in which tools are simply objects within a project just like any other object. It does, however, seem reasonable to separate this particular object out from any of the others because more tools are available than are actually present in the current project. This is because tools which are described in the user's default project and the system project are always available, regardless of which project is currently selected. This allows users to define their own default values for commonly used tools and store them in their default project. These new tool descriptions will be available no matter what project is currently selected. Users may also define tool defaults on a project-specific basis. This actually results in a hierarchical system in which definitions in the current project take precedence over definitions in users' default project which in turn take precedence over the system's

defaults.

Function	Description
Instrument Launching	An instrument is much like a protocol, but the UIM may distinguish between them for purposes of invocation and communication.
Instrument Communication	If the UIM supports instruments, there must be a means of communicating with them. This needs to be defined by the UIM designer.
Semantic Dialogs	If instruments are supported, the UIM must provide a means of informing user interfaces about semantic dialogs and sending them to interested instruments.
Workbench Semantic Types	The UIM should implement the semantic types which are relevant to the domain of the workbench itself. For example: **WObject** is a semantic type for an object in the workbench.
Multi-step Protocols	The ability to create multi-step protocols should be cleanly integrated into the workbench user interface.

Table 10 - Desirable UIM Features

4.4.2. Semantic Dialogs

Semantic dialogs are the means by which tools can take full advantage of shared application domains with a running instrument. All UIMs implement the basic set of dialog types. This set cannot be extended without modification of all UIMs, or without inducing potential user-interface dependencies in the tools. The set of semantic dialogs, however, is easily extensible because the UIM does not have to implement a semantic type to allow communications between the tool and the user.

Semantic dialogs are the same as a basic dialog except that the **SType** field contains a value which reflects the semantic type of the dialog. These values are assigned by the systems administrator to guarantee uniqueness. The **Type** field still contains the type of

the returned **Value**. If neither the UIM nor any of the current set of instruments have implemented the semantic type, the UIM handles the dialog as a basic dialog of type **Type**. This insures that the user will always be able to respond to the information and that the dialog mechanism is always user interface independent.

When a UIM receives a semantic dialog, it must first determine if it or any of the current instruments have implemented this semantic type. To accomplish this, the UIM sends a message[5] to each of the current instruments. Any instruments which have implemented that semantic type respond with a message to the UIM. The UIM maintains a list of instruments which may answer this dialog. The UIM then displays a dialog box containing the name of the dialog, the prompt, and any other information consistent with the user interface implemented by the UIM. Users are also free to ignore the semantic type and enter the appropriate value in the dialog box. If users wish to take advantage of the power of user interfaces which share semantics with tools requesting the dialog, they notify the UIM that they are going to respond to the dialog, and then perform whatever action is consistent with the user interface to answer the request. When they are satisfied with the answer, they can notify the UIM to accept the current answer and send it.

For example, if the **residue** dialog used above had an **Stype** value of **RESIDUE** it would be a semantic dialog. Assume that the current UIM is running on a Silicon Graphics Iris and that a molecular modeling program (**MIRAGE**) is currently running as a instrument underneath it. The UIM would send the dialog to **MIRAGE** to see if it has implemented the semantics of the **RESIDUE** type. **MIRAGE** does implement this type, but *would* first check the **SInfo** field, which for type **RESIDUE** should contain the name

[5] Actually, it sends the entire dialog, for convenience. This avoids having to send it again if the instrument implements it.

of the molecule the tool is interested in. If MIRAGE has this molecule on the display, it would respond that it could answer the dialog. The UIM then presents the dialog on the screen. When the user is ready to respond to the dialog, he or she may use the mouse to click into the dialog box. The UIM then sends a message to MIRAGE that the dialog is now active. The user may then use a mouse to select the desired residue directly by clicking on the correct residue. When a residue is selected, the value (in **Type** terms) is sent to the UIM. This value is displayed in the dialog box. When the user sees that they have the correct value, they may accept it by pressing the RETURN key, or clicking on an OK or Accept field. The UIM then sends the completed dialog back to the tool (via the SM) just as it would a basic dialog.

4.4.3. Tool Invocation

Tool invocation is one of the most important functions of a UIM. There are several aspects to tool invocation. The first is the user interface, in other words, how does the user indicate that they wish to invoke a tool? The MMWB philosophy is to allow UIM implementors to implement tool invocation in a manner consistent with the device and user interface they are writing for. There are a variety of possibilities, from double-clicking with a mouse, to menu picks, to command line interfaces. Each of these is appropriate for certain types of user interfaces and it is expected that there will be a UIM for each one of these. Some UIMs will implement more than one user interface to allow users to select their preference.

The second aspect of tool invocation is the actual initiation of the tool[6]. The user

[6] A more detailed description of tool invocation may be found in Appendix A.

designates in some fashion that a tool is to be invoked, and it is now up to the UIM to make that happen. This functionality is implemented for the most part by the SMUI library. The list of tools is initially determined by a call to the SMUI library function *sm_list_protocols* or *sm_list_instruments*. When a user designates one of these tools to be invoked, the UIM calls the function *sm_invoke*. This function only actually launches the tool if the option sheet for the tool has been previously filled out and marked as complete by the user or UIM. Otherwise, it marks the tool for eventual invocation and sends the option sheet to the UIM dialog handler for interaction with the user. Once all of the options have been completed, the UIM attempts to return the option sheet to the sender. In this case, however, the UIM itself is the sender and it determines that this option sheet corresponds to a tool which is awaiting invocation. Then the tool is actually launched.

The last aspect of tool invocation was briefly mentioned above: option sheet handling. The user interface implemented by the UIM must provide a means for modifying and filling out option sheets. Option sheets are actually part of the tool description which is maintained as part of the object file system, OFS [11]. The user must be able to modify the default values for options and to designate that the option sheet is complete and that they do not wish to change it before invoking the tool. This allows a user to set the option values for a tool, and then launch the tool repeatedly with the same values, without interacting with the tool at all.

4.4.4. Interface to Instruments

User Interface Managers (UIMs) which function on graphics workstations are assumed to support multiple instruments in a windowed environment. An example showing why this is necessary is easy to construct. Assume the user is using a Silicon

Graphics Iris workstation, and is working with a molecular model of *relaxin*. In addition, assume that user wishes to perform some molecular dynamics computations on the model that he/she is working with and wishes to view the results of those calculations. Now we have a problem, since the dynamics display may update once every minute, and the user is not going to want to have his or her entire display devoted to this. Nor is the user going to want to switch UIMs whenever he/she wishes to check on the status of the calculations. What a user requires is the ability to put the dynamics display program in a window on his or her current display. MMWB provides for this by allowing multiple instruments to be active under a single UIM.

The separation between UIM and instruments is straightforward. The UIM provides an interface to the workbench, to dialogs, to tools and option sheets, and to the SM. An instrument provides a user interface to a specific task or applications domain. Example user interfaces are:

- A molecular modeling program (MIRAGE)
- A molecular display program (for dynamics, etc.)
- A molecular editing program
- A protein sequence analysis program

In general, an instrument will be written whenever there is a need for direct interaction with the user which cannot be met by the dialog mechanism.

Most instruments must be launched directly by the UIM and not as part of the normal SM scheme for two basic reasons. First, it is critical that an instrument run directly on the machine on which the display resides. This is primarily for bandwidth considerations, but also because there are not yet any three dimensional graphics standards which

are network independent. Second, the instruments and the UIM should provide as consistent a user interface as possible. Unlike protocols, which use the dialog mechanism, there is no simple way to separate the user interface from an instrument. The instrument and the UIM must interact to provide such consistency. The UIM will be written for a specific windowing system, often for a specific hardware platform. The interaction between an instrument and the UIM will be dependent on that platform and on the facilities provided by the window system.

Some UIMs, however, may be implemented for window systems which may be connected to from a network, such as X or NeWS. For these UIMs, all of the information necessary to connect to the UIM may be communicated to an instrument using the standard dialog mechanism. This makes it possible to utilize the facilities provided by the SM to launch instruments. The SM makes a distinction, however, between instruments and protocols so that if a UIM terminates, all associated instruments will be terminated also.

When the UIM is not implemented on a network-based window system, it must provide a mechanism to launch instruments which is independent of the SM. This mechanism must include all of the normal option sheet handling and option delivery. There is to be a default library available which implements instrument launching and option sheet handling. Typically, instrument launching will involve device-dependent environmental setup (e.g. for the window system) followed by the function the operating system uses to launch tools.[7]

---

[7] In Unix this would be a **fork** system call followed by an **exec** system call.

The UIM and the instrument must maintain a communications channel for other information which needs to be exchanged during the session and at start-up. This information includes: the currently selected project, semantic dialog information, option values, and probably access to the dialog mechanism of the UIM. The instrument needs to know about the current project so that it can interact directly with the Data Manager. MIRAGE, for example, will access the Data Manager directly to read in the molecular structure information for display. When the user selects a new project, there are two possible schemes. First, the new project could be sent to the instrument. Current lists of available objects and future selections would be updated to reflect the new project. A second alternative is to not update running instruments with the new project information, but to force the user to initiate a new instrument to work within the new project. While the second alternative seems more restrictive, it is also more consistent with user's tasks, as well as easier to implement. As a result, this is the alternative which has been selected for implementation at this time.

The need for communicating semantic dialog information was discussed above in some detail. The instrument implementor must decide which semantic types they wish to handle and determine how the user will interact to respond to dialogs of those types. Note that it is not necessary for the semantic interaction to be graphical. It would be very useful to have the instrument simply do type- and range-checking based upon the semantic domain.

It would also be extremely useful to allow instrument implementors to take advantage of the UIM's ability to handle dialogs. For example, assume an instrument is run which requires some initial information which was not supplied in the option sheet. The

instrument could display a dialog-type message, or it could take advantage of the UIM to do so. The need for the UIM to handle options for the instrument makes this facility a simple addition.

4.4.5. Example User Interface

This section describes an example user interface which meets all of the functionality described above. Not every feature of a UIM will be discussed, but hopefully enough actions and images will be presented to allow the reader to imagine the look-and-feel of the interface. For the purposes of example, let us assume that this user interface will be run on a Silicon Graphics Iris 4D/GT. When a user sits down at the Iris, he or she logs in using their normal Unix name and password. When the user has gained access to the system, he or she may initiate the UIM by typing the name of the appropriate program. When the UIM begins running, the first function which must be performed is a search of the user's default project to determine if there are any sessions currently running without a UIM. For our Iris interface, if any detached sessions are detected, the UIM displays a series of session icons on the screen, one for each detached session. The color or shape of the icon is different if there are pending dialogs for the session. The user then select one of these sessions by double-clicking on it with the mouse. This connects the user to that session and any pending dialogs will be read and displayed.

Continuing our example, let us assume that there are not yet any dialogs waiting. When the UIM is connected to the session, two icons appear on the screen. The first is a little picture of a toolbox, and the second is a little picture of a workbench. The user opens the workbench (by double-clicking) and selecting a project to work on, **Relaxin**. The **Relaxin** project contains five objects: the primary sequence for *relaxin*, an energy

minimization tool description, and three putative structures. The user now opens the toolbox icon (again, by double-clicking), and selects the molecular graphics tool, **MIRAGE**. **UIM** invokes **MIRAGE** as a user interface and passes the information about the currently selected project to it. **MIRAGE** initializes and presents its user interface in a window, including three icons which represent the three putative molecular structures. The primary sequence data is not presented because **MIRAGE** is not designed to display that object type. The user now decides to perform a minimization on one of the structures. Again the user opens the toolbox and selects the minimization tool. This is the version of the tool which is in the current project, because it has precedence over the system version. The user had previously selected the normal defaults for performing energy minimizations on this project, so no information is required to initially launch the tool. Almost immediately, however, a dialog window pops up on the display which lists the three structures and asks the user to select which one is of interest. The user selects the correct one by double-clicking on the proper string in the dialog box, and the minimization begins.

4.5. Foreign Tool Integration

The previous section describes the role of the **UIM** which is to present an interface to the user. This section will discuss the role of the Foreign Tool Driver, **FTD**, which is to present an interface to applications which were originally developed outside of the **MMWB** environment.

One of the goals of the **MMWB** architecture is to provide an extensible environment. New protocols, instruments, and objects can be defined and integrated into the system. In addition, new user interface managers can be written to be consistent with the

workbench metaphor, but present new or alternative interaction styles to the user. Libraries, interfaces, and tools are provided to aid the software developers in the development and integration of their code. It has already been suggested, however, that much of the scientific software which is currently in use comes through exchanges with other laboratories or is highly modified and reworked code which has found popular acceptance. It would be extremely costly and unwise to assume that all software will be rewritten to be compatible with the MMWB environment. The Foreign Tool Driver, FTD, was developed to provide a mechanism to integrate tools which are too difficult or too costly to convert to the MMWB environment.

The goal of the design of FTD is to provide the maximum MMWB functionality such as access to *dialogs*, *sessions*, and *options* while minimizing the amount of programming necessary to provide the integration. One constraint imposed on the design is the assumption that the foreign tool cannot be modified; for example, only the binary might be available. The approach taken by FTD is to "script" the interaction between the user and the foreign tool, and translate the tool's output and the user's input into notifiers and dialogs. The script takes the form of a series of rules, each rule containing a set of preconditions, a regular expression to be matched by the output from the tool, an action to perform if the preconditions are met and the pattern matches, and a set of postconditions to be set. The design of the FTD and the syntax of the rules are discussed in the following sections.

4.5.1. FTD Design

The FTD is a program which takes as its input a series of rules and arguments which describe how to initiate the foreign tool. Most of the functionality of the FTD is

implemented as a series of library calls which compile the rules and control rule processing. In addition, the library includes some intrinsic functions to be used as actions for sending dialogs and notifiers to the user. Table 11 lists each of the functions available in the FTD library and their purpose. A version of the FTD program exists which can be modified by integrators to provide special functionality such as conversion of data file formats, special tool invocation procedures, or user-defined actions for the completion of a rule.

Function	Description
ftd_parserules	Parse the ruleset
ftd_launch	Launch the foreign tool
ftd_process	Process output from the foreign tool
ftd_term	Terminate the foreign tool
ftd_dialog	Send the dialog to the user and the response to the tool
ftd_notify	Send the dialog to the user
ftd_print	Output the prompt (debugging only)
ftd_input	Output the prompt and get input from the user (debugging only)
ftd_sink	Discard the input

Table 11 - FTD Library Functions

The FTD program is launched from the SM as a normal MMWB protocol as shown in Figure 9. It must be run on the same machine that the foreign tool is to be invoked on. The FTD may be part of a multi-step protocol. A common use of the FTD as part of a multi-step protocol might be as the middle protocol shown in Figure 10. The first step converts from an MMWB object into the native format expected by the foreign tool. Then the FTD is called to execute the foreign tool on that data file. Finally, another protocol runs which converts the foreign tool's output file into another MMWB object.

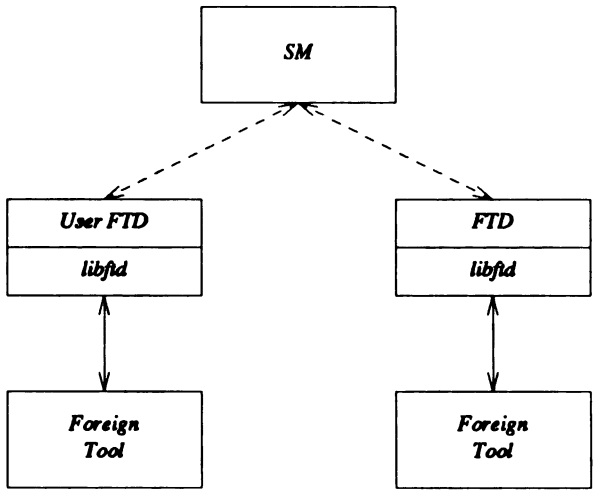


Figure 9 - FTD Invocation

The dotted lines show the path of normal MMWB communications through the dialog mechanism and the solid lines show the communications between either version of FTD and the foreign tool.

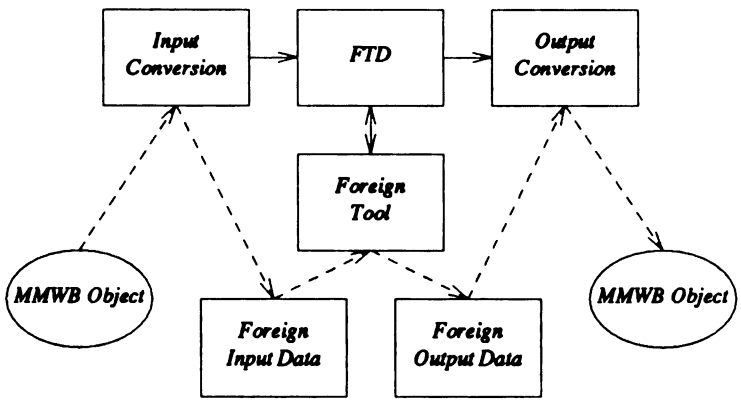


Figure 10 - Example Multi-step FTD Series

Dotted lines show the flow of data from an MMWB object into the appropriate foreign format, and then back into an MMWB object. Solid lines show the time sequence through each of the steps in the multi-step protocol.

Options may be passed to the FTD like any MMWB protocol and the options are made available to the foreign tool through the use of flags in the script ruleset. Three options must be passed to the FTD as shown in Table 12: the path on the local machine to

the foreign tool, the object containing the rule set,[8] and the command-line arguments to be used to invoke the foreign tool.

Option	Description	Example
program	Path to the foreign tool	*/usr/local/mmwb/bin/ftool*
args	Command line arguments	*-i input -o output*
rules	Path to the ruleset	*/usr/local/mmwb/rules/ftool.ftd*

Table 12 - **FTD** Mandatory Options

If all mandatory options are available, the **FTD** then opens the ruleset and compiles the rules into an internal format. If no syntax errors are detected the foreign tool is started and input and output channels are opened to it. **FTD** then accepts input from the tool and scans the ruleset for meeting preconditions and a matching pattern. If a match is found, and all preconditions are met, the rule is activated and any specified actions are performed. After the actions have completed, any postconditions are set, and more input is read from the foreign tool. This continues until either the foreign tool terminates or an explicit termination is requested as part of one of the rules. The exact nature of the output and prompting conventions of the foreign tool make it impossible to rely on newline terminated strings. As a result, characters are read one at a time, until either a newline is reached or no characters are received for 1 second. Then the accumulated characters are processed through the rules to see if any rules match. If no match is found, the characters are discarded. Another approach would be to specify the termination character of the prompt. This approach was considered, and was discarded for two reasons. First, it would require either that the termination character be unique, throughout all prompts

[8] Actually, in the current implementation, the rulesets are not implemented as objects but as files in the native file system.

from the tool, or that the ruleset be scanned upon every character. This leads not only to additional overhead, but causes problems with embedded termination characters and premature matches of shorter strings.

4.5.2. FTD Rule Syntax

The FTD is a simple rule-based system, which has preconditions, actions, and postconditions. The basic syntax is shown below:

```
name: precondition ? pattern =>
            action [flags],
            {dialogs}
            postconditions;
```

Preconditions are boolean expressions and the patterns are quoted regular expressions. The syntax for the dialog section of the rules is patterned after the one used by the MMWB dialog compiler [5]. Postconditions are assignment expressions. Variables can be set in a postcondition and tested in the preconditions section of another rule. In this manner, state may be retained to selectively enable and disable rules.

The detailed FTD grammar is presented in the appendix D. A short example is given below:

```
# This is a test
testname:
      a==1?
      "Enter .* output file:"=>
            ftd_dialog [FTD_QUIT|FTD_OPTION],
            {
                  text; # no default, this is required
                  prompt "output object";
                  help "name of resulting molecule object";
            }
      a=2;
```

There are currently three intrinsic functions which are provided for normal use,

ftd_dialog, *ftd_notify*, and *ftd_sink*. The action of *ftd_dialog* and *ftd_notify* depends on the setting of the flag **FTD_COPY_PROMPT**. If this flag is set, then the output from the tool is copied into the prompt field of the dialog and sent to the user. If *ftd_dialog* is called then the return value is passed to the tool for further processing. If *ftd_sink* is called then the text from the tool is discarded.

4.5.3. Debugging

For the purposes of debugging **FTD** rulesets, it is also possible to execute **FTD** directly from the command line without utilizing any of the features of **MMWB**. If **FTD** is executed in this manner, then all occurrences of *ftd_dialog* should be replaced with *ftd_input*, and all occurrences of *ftd_notify* should be replaced with *ftd_print*. In all other ways the actions of the **FTD** and the processing of the **FTD** rulesets are the same. This allows quick testing and modification of the foreign tool interface before it is integrated fully into **MMWB**.

4.6. Discussion

The goal of the design of **MMWB** was to create an environment for the investigation of the structure and function of biological macromolecules which provided the following features:

- Data Sharing
- Resource Sharing
- Multiple, Consistent User Interfaces
- Compatible Computing Services

- Extensibility
- Distributed Processing
- Network Transparency
- Implementability

The architecture presented provides an environment which includes all of these features, with the possible exception of network transparency, which has not quite been achieved in a real sense. From the *user's* point of view, one is able to execute tools on a variety of machines without regard to which machine they will run on or how the data is returned. The network ceases to be transparent only to the programmer who must port the software to each platform, and provide tool descriptions which list the available platforms.

Much of this architecture has been implemented under the Unix operating system on VAXen, Suns, and Silicon Graphics Irises using TCP/IP as the underlying network transport. All of the features described here have been implemented and tested, including *sessions*, *dialogs* (both simple and compound), *option sheets*, *semantic dialogs*, *multi-step protocols*, **DM**, **SM**, and **Dispatcher**, and multiple SMUIs. The entire system was implemented in three months by two programmers and one graduate student.

4.6.1. Why the Complexity?

The overall architecture as shown in Figure 5 is reasonably complex. It involves the use of multiple processes for performing any task, and often will involve more than six processes for doing any real work. The reason for this complexity is simple -- it made the system much easier to implement. By segregating the **Session Manager** as a separate process, for example, the concepts of *sessions* and *dialogs* were very straight-

forward to implement and have proven themselves to be robust. The complexity of the **MMWB** architecture reflects this separation of the overall functionality into separate processes which logically implement each individual function. This eases not only implementability, but also maintainability, because functional lapses can be easily traced to their source.

4.7. Conclusions

The user interface paradigm chosen for **MMWB** has proven itself to be valuable, not only in helping to structure the user interfaces, but also as a useful component of the design process. Much of the overall architecture of **MMWB** resulted from the choice of the workbench metaphor, and from allowing that choice to influence the software architecture throughout the design process. Some extremely valuable design elements resulted from this influence, in particular the concepts of *sessions*, *dialogs*, and the separation of *instruments* and *protocols*. Our experience through this design process argues strongly that the user interface design must be an integral part of the overall design process, although for our design, a potential *family* of user interfaces was considered. This approach allows user interface designers to tailor user interfaces to specific applications, or even to specific users.

There still remain some unanswered questions about this approach. First, and perhaps foremost, is the success of the workbench metaphor in transferring training between two different user interfaces. This is a very difficult question to answer, primarily because of the size of the study group which would be necessary to statistically verify the effect. What is being planned is a small series of subjective experiments to get some feeling for the importance of the metaphor. The design of this experiment is

discussed in detail in Section 6.2.

Another question which will be answered with practice is the ratio of tools which require a user interface (*instruments*) to those which do not (*protocols*). If the majority of modern tools for molecular modeling require access to a bit-mapped graphics display, then the architecture discussed above is not optimal. Our experience suggests that this will not be the case, and that while the importance of scientific visualizations will continue to increase, so will the need for large-scale computation which requires little or no interaction until the results are available.

Chapter 5

Discussion

5.1. Introduction

The final test of any approach to design is to produce and implement a system using that approach. The approach suggested in Chapter 2 involves a specific evaluation step. To evaluate the design described in Chapter 4 requires several components. First, the underlying architectural components must be implemented and tested. For **MMWB** this required implementation of the **Session Manager**, the **Data Manager**, the **Dispatcher**, the **Foreign Tool Driver**, and libraries which provide access to them from **UIMs**, **protocols**, and **instruments**. In addition, support libraries were implemented to provide access to *dialogs* and other objects used by the system.

After the architectural platform has been implemented, the next step is to implement a **UIM**, some **protocols**, and to integrate foreign tools using the **FTD**. The implementation and integration of these tools provides experience with the libraries, architecture, and syntax of **MMWB**. This experience can then be used to evaluate both the design and implementation of the architecture in terms of the design goals. In addition, these tools provide a platform for testing new implementations and design changes in the underlying architecture. The initial set of tools can also be used to provide examples for implementation of new tools by developers who were not members of the original design team. In effect, these prototype implementations become an important part of the "documentation" for the system, giving examples of how the designers perceived that the libraries and architecture would be used.

Obviously the final evaluator is the target user population. This is especially true when the architecture involves novel user interface concepts and designs. There are several possible approaches to performing this evaluation, from formal tests and experiments to the more informal interviews and discussions with users. In addition, invaluable information can be gained by simply observing users utilizing (or attempting to utilize) the system at various stages during implementation.

The rest of this chapter presents the current status of MMWB and discusses some of the lessons which have been learned by the implementation of the architecture, the libraries, and some prototype tools. Also discussed is the design process itself, how that contributed to the resulting architecture, and how it detracted. Finally, some initial user comments and perceptions are presented and discussed.

5.2. Platform Status

5.2.1. Implementations

The initial implementations of all platform components were done on a Vax 8650 running Berkeley Unix. The networking layer is provided by the Berkeley socket library and the TCP/IP protocol suite was used. In the current implementation, TCP is the sole transport protocol.

The language chosen for the initial implementations was C. The design of the MMWB architecture suggests that an object-oriented language such as C++ or Objective C might have been a more appropriate choice to implement MMWB. C was chosen at the time primarily because of its portability, the degree of standardization which has already occurred, and the programming experience of the developers. Future implemen-

tations might benefit from C++ as a language choice, particularly since C++ now supports multiple inheritance. A DM cell, for example, can inherit the properties of both the instance and the attribute. Another example where inheritance would be useful is in the implementations of option sheets, which should inherit attributes of the tool, and attributes of the dialogs which make up the options. The current implementation is very modular, taking advantage of ANSI C's information hiding capabilities, so substituting the existing modules for modules written in C++ should be straightforward.

5.2.1.1. Dispatcher

The dispatcher is an important part of the overall MMWB architecture, although its design and implementation are not directly a part of this research. The dispatcher has been implemented and is currently in use. All of the functionality discussed in chapter 4 has been included. In addition to the Vax implementation the dispatcher has also been ported to a Silicon Graphics Iris workstation, a Convex C1, NeXT machines, and a Sun Microsystems workstation running the 4.0.3 version of SunOS. The implementation currently depends on the Berkeley socket library and utilizes TCP/IP as the underlying networking protocol. The only significant modification necessary to the dispatcher after its initial implementation was the addition of a KILL command to allow the dispatcher to kill processes that it had started. The dispatcher has proven to be reliable with acceptable performance.

5.2.1.2. Data Manager (DM)

The DM provides a table-based data storage system to the MMWB architecture. The initial implementation of DM utilized a library to provide an object based file archi-

tecture. The DM itself only provided access to tables and cells within the tables. Performance measurements and experience with this scheme demonstrated that this design was very inefficient. The DM is in the process of being reimplemented. The new design moves the object and project functionality into the server rather than leaving it in the library. The initial design of the DM was the subject of previous work [11].

5.2.1.2.1. Data Manager Library (libofs)

The data manager library, *libofs*, provides the interface to the object and project layers of the DM. Experience utilizing *libofs* has demonstrated that it provides a complete and effective interface to the underlying object layer. The current implementation requires that the calling program preallocate all memory before calls to *libofs*. This requires that the calling program first determine the number and size of the objects to be returned, then allocate the memory for those objects, and then finally call the routine to read the objects. This unfortunately can result in a race condition in which it is possible for a routine to return more data than was allocated. If, for example, one routine is reading data from an object while another is writing it is possible for the writing routine to add a field between the time that the reading routine got a count of fields and the time that it reads the fields. As part of the reimplementation of the DM, the *libofs* interface will be modified to allocate all memory internally. A new routine will be added to free the allocated memory. As with the effort progressing on DM and the dispatcher this work is being accomplished by others and is included here only for completeness.

5.2.1.3. Session Manager (SM)

The Session Manager, SM, has been implemented and provides all of the functional-

ity described in Section 4.2, including session management, dialog queueing, session reattachment, tool invocation, and management of multi-step protocols. One of the early modifications to the design of SM was to provide support for UIMs to use the facilities of SM to initiate the execution of instruments. This required the SM to distinguish between instruments and protocols so that all instruments may be terminated when the UIM terminates, while protocols may continue. This modification proved to be a straightforward extension of the tool description by the addition of a **Type** field which describes whether the tool is a protocol or an instrument.

The other major problem that was discovered with the existing design of SM involves the implementation of multi-step protocols. In the current implementation, all information necessary for the execution of the multi-step protocol is sent as part of the tool invocation as a tool description. The tool description includes the **Subtool** field which contains the names of all of the tools to be executed and the manner of their execution as described in Chapter 4. In addition, all of the option sheets from the tools are concatenated and sent. This requires either the user (or developer) to manually concatenate all of the option sheets together when the multi-step protocol is defined, or it requires that the library be intelligent enough to concatenate the option sheets together at the time of invocation. Additionally, the tools in the **Subtool** field are the internal names of the tools as known by the Dispatcher. Thus either a special "multi-step protocol building" tool must be built which would perform all of these tasks, or the creation of multi-step protocols would be limited to developers. A final problem has to do with the extension described above. It would be extremely advantageous to be able to include instruments as parts of multi-step protocols. This is impossible under the current implementation because there is no way to identify them as instruments since only the tool description for

the entire multi-step protocol is sent at tool invocation time.

An alternative implementation would be to pass only the name of the object containing the multi-step protocol to SM and have SM read the object. The object could then contain the names of the other objects which comprise the multi-step protocol in the **Subtool** field, plus a **Type** field which states that this is a multi-step protocol. SM could then read each of the tool description objects as part of the execution. While this seems a feasible solution it requires the SM to maintain the current project and the user's home project so that it can read the correct object. In addition, because the UIM must read the tool descriptions anyway to present the user with the options before invocation, there would be a great deal of duplication. The current plan is to implement a multi-step protocol tool and to devise an extension to the **Subtool** specification syntax which would designate any instruments which might be included.

5.2.1.3.1. Session Manager Library (libsm)

libsm has been implemented and provides all of the functionality described in Section 4.2. The interface to *libsm* provides functions for tool invocation, dialog dispatching, option sheet handling, getting status from SM, killing running tools, sending and receiving dialogs, setting and getting the current, home, and system projects, and displaying errors. In addition, routines have been provided which simplify the task of implementing protocols. These routines provide for initialization, option handling, a simplified dialog interface, and a simplified interface to send notification dialogs. The current implementation and design have proven to be quite stable. Experience using the library (see below) has reinforced this impression. The library is divided into three parts: a common portion, a portion used only by protocols, and a portion used only by UIMs. The

manual pages and documentation for each of these portions are included in Appendix C.

5.2.1.4. Dialog Library (libdialog)

Dialogs are an important underpinning to the MMWB architecture. As a result, the dialog data structures were originally public so that all functions could access them directly. This soon proved to be problematic because whenever the dialog data structures needed to change for any reason, all of the code which used dialogs had to be checked and recompiled. There were also many functions which were commonly used by most, if not all, programs which used dialogs. As a result *libdialog* was implemented as an interface to dialogs. The internal data structure of a dialog is now hidden completely from all but the dialog routines. The implementation provides routines for the creation, deletion, copying, and linking of dialogs. Other routines are provided for accessing the various components of dialogs, along with some convenience routines to simplify getting and setting commonly used parameters. The new implementation of *libdialog* has proven to be extremely useful and robust. The one feature of the interface which has caused difficulties is that the set of routines which are provided to return values of a given type from the dialog (e.g. *dialog_get_int*, *dialog_get_text*, and *dialog_get_float*) return the value requested and the success or failure is passed back through a formal argument. This requires the caller to separately test the success value for possible problems in a statement after the function call. The requirement of a second statement to test for the success of the call was deemed to be less important than the difficulties of passing the actual result back in a formal argument. No clear case has been made favoring one approach over the other so until more experience is gained, no changes will be made to the interface.

Included in *libdialog* is a pair of routines to convert dialogs from the internal form to a textual equivalent and from this textual form to the internal form. These routines are heavily used by the SM protocol processing routines to send dialogs back and forth between tools, UIMs, and SM. The routine which converts to text can also be a useful debugging tool for discovering problems with dialog communications.

5.2.1.5. Foreign Tool Driver (FTD)

FTD was the last of the platform components implemented. This was primarily because **FTD** is an **MMWB** protocol and relies on the presence of all of the other features of the platform in order to function. Both of the versions of **FTD** described in Section 4.5 have been implemented. Because **FTD** is the most immature of the components of the **MMWB** platform it has not undergone much revision. One feature of **FTD** that was included in the standalone version was the ability to run independently of the **MMWB** environment. This was done to provide a mechanism for testing and debugging **FTD** scripts. In the current implementation **FTD** scripts are text files in the operating system, rather than **MMWB** objects. This provides an easy mechanism for creating, modifying, and printing **FTD** scripts. In future implementations, when **MMWB** instruments are available for the creation and editing of text files, **FTD** scripts will become **MMWB** objects in addition to optionally being normal text files. One additional problem that was discovered is the need to provide a capability to use **MMWB** options to pass command line options to foreign tools. This is described in more detail below as part of the discussion of the integration of **spasms** (Section 5.4). The Unix manual page for **FTD** is included in Appendix C.

5.2.1.5.1. Foreign Tool Driver Library (libftd)

The standalone version of **FTD** and the **FTD** skeleton are both implemented on top of the **FTD** library, *libftd*. *libftd* provides a very simple interface to the features of **FTD**. Functions are provided to execute and terminate the foreign tool, to parse the ruleset, and to process strings received from the foreign tool. No modifications have been made to the interface since its original implementation.

5.2.2. Performance

One of the concerns about the **MMWB** architecture is whether the complexity of the architecture has an adverse effect on the performance and responsiveness of the system. To address this concern, performance measurements were made to determine the throughput of dialogs from a protocol to a **UIM**, and from a program through the **FTD** to a **UIM**. Nine tests were performed. Test number 1 tested the maximum throughput from a protocol to **SM**. This was accomplished by sending 500 notification dialogs with the **DI_SUPERCEDES** flag set. This flag directs **SM** to supercede any previous notification dialogs from this protocol with this dialog. In this manner only the time between the protocol and **SM** affected the throughput. Test number 2 sent 500 integer dialogs with a semantic type of **ECHO**; these were interpreted by the **UIM** in the *uim_dialog* callback function and resulted in immediate replies to the dialog. This tested the total round trip time between a protocol and a **UIM**. Test number 3 was similar to test 1 except that the **DI_SUPERCEDES** flag was not set. Tests 4 through 6 were the same as tests 1 through 3, respectively, except that the protocol and the **UIM** were executed on different machines. This was done to test the impact of dispersing protocols on a local area network. Tests 7 and 8 are similar to tests 1 and 2 except that the protocol is actually a

small program running under the **FTD**. In order to test the impact of the 1 second timeout described in section 4.5.1 test 9 is the same as test 8 except that the promp given by the program does not terminate in a newline.

The protocol used to perform tests 1 through 6 was a modification of the **clock** protocol described in Section 5.5 below. An option was added to specify which test was to be performed, and the **interval** option was modified to collect the number of dialogs to be sent. The actual dialogs sent were either notification or integer dialogs (depending on the test) which contained the date and time in the prompt field. For tests 7 through 9 the protocol was a small program which either sent a short string (test 7) or prompted for an input and waited until a newline terminated string was received (tests 8 and 9). The **FTD** ruleset was very simple and utilized on the intrinsic functions *ftd_notify* for test 7 and *ftd_dialog* for tests 8 and 9. Timings were recorded by utilizing the *syslog* mechanism of Berkeley Unix. The **UIM** used was **twb** which is described in Section 5.3 below. The only necessary modification was to add a test for the semantic type **ECHO** to the *uim_dialog* function and instructions to reply to that dialog immediately.

5.2.2.1. Performance Results

The performance results are shown below in Table 13. The host *socrates* is a Vax 8650 running Berkeley Unix 4.3. *cgl* is a MicroVax 3000 running Berkeley Unix 4.3. The hosts are located on the same segment of a 10MB ethernet network.

Test	Protocol Host	UIM Host	Number of dialogs	Time (seconds)	Milliseconds/ dialog
1	socrates	socrates	500	7	14
2	socrates	socrates	500	100	200
3	socrates	socrates	500	8	16
4	cgl	socrates	500	16	32
5	cgl	socrates	500	100	200
6	cgl	socrates	500	17	34
7	socrates	socrates	500	10	20
8	socrates	socrates	500	100	200
9	socrates	socrates	500	521	1042

Table 13 - Results of MMWB Performance Tests

The timings are consistent with expectations and well within reasonable limits for the primary uses of MMWB dialogs. The throughput demonstrated by test 1 is quite acceptable for a protocol which is sending update information to a UIM or an instrument. Likewise, the throughput exhibited by the round-trip timings of test 2 is quite acceptable for interactive use. In fact, 200 milliseconds is considered acceptable for character-oriented response time, and dialogs contain a great deal more information than single characters. The results of test 3 are quite interesting and demonstrate the cacheing ability of SM. All the dialogs are eventually delivered to the UIM, but not until long after the 8 seconds that it took the protocol to send them.

The network times are much as would be expected for tests 4 and 6. Basically it takes about twice as much time to deliver the dialogs to SM over the network. Interestingly, there was no measurable difference between test 2 and test 5. This is an indication that the bottleneck is not in the communications, but in some internal handling of the dialogs in either SM or the UIM. While it is certainly possible to profile both SM and the UIM to increase the performance, it seems that given the current 200ms round trip time the potential gains might not be worth the effort.

The results for the tests 7 and 9 are also consistent with expectations. Test 7 demonstrates that the performance impact of the **FTD** is negligible for notifications. Test 9 is predictably much slower than any of the other tests. Of the 521 seconds, 500 seconds can be attributed to the one second timeout which is built into the **FTD** in cases where no newline is detected. Several possibilities exists for improving this. Probably the best approach would be to use a much shorter timer, on the order of 200 milliseconds. This would improve the performance by a factor of 5. This approach was not taken initially due to portability concerns, but will be revisited as more experience is gained with applications which are to be integrated into **MMWB** through the **FTD**. The results for test 8 are a surprise, however, although they are consistent with test 2 and test 5. These results suggest that there is *no* performance overhead associated with the **FTD** in this case. This result corroborates that there is a bottleneck somewhere in the **SM** or the **UIM** which masks the other components of dialog delivery.

5.2.3. Reliability

It is extremely difficult to assess the reliability of any system which has not yet gone into production use. Early indications are that because of the modular nature of the system, it will prove to be very reliable and maintainable. The rewrite of the **DM** has gone on with little effect on the rest of the architecture. Certainly when the new *libofs* interface is completed, a fair amount of work will be necessary to modify all components which utilize it. However, it is important to note that while the specifics of the interface will be changed, the conceptual foundations have not changed. It is hoped that this will allow tools to be modified for the new interface with little or no change in logic or flow. In fact, the early discussions on the interface have indicated that the most profound

changes to tools, other than changes in calling syntax, will be to delete a fair amount of code which handles memory allocation.

Another feature of the separation of components has been the ease with which developers have been able to target a particular component when problems do arise. The separation of function has been invaluable in pinpointing which component of the system is exhibiting problems.

The **MMWB** platform has also demonstrated the ability to fail gracefully. Typically when failures do occur, some message is received describing the problem and/or failure condition. Rarely do problems result in complete failure of the system. The notable exception to this rule is a failure in the communications protocol between UIMs, protocols and SM. The communications protocol between SM and its tools is written assuming that all communications are reliable and that tools correctly implement the protocol. While this is true once tools have been implemented and debugged, protocol errors do occur during the debugging cycle. SM correctly identifies a protocol failure when it does happen, but the response in the current implementation is to terminate all **MMWB** protocols, instruments, and UIMs, post an error, and then exit itself. One possible improvement would be to log the error, and then send some form of a **RESET** command to the offending tool to reset the protocol back to an initial state.

5.3. twb - An Example UIM

twb, or Terminal WorkBench, is a UIM which has been designed and implemented for use on ASCII terminals. **twb** was the first UIM which was written for production use as an interface to **MMWB** and had several design goals. First, it was designed to be a production interface to **MMWB** for use by scientists from their homes or other terminals

when access to a graphics device was unavailable. The intent of the user interface for **twb** was to provide an easy to use interface designed specifically for ASCII terminals. Second, **twb** was intended as a prototype UIM to serve as an example implementation of the functionality and services required for an MMWB UIM as described in Section 4.4.

5.3.1. Implementation

twb was implemented on a Vax running Berkeley Unix 4.3 in the C language. It has since been ported to a Sun Microsystems Sun 3/50 and SparcStation I. **twb** uses the **curses** [2] screen management package which is common on Unix systems. The implementation was accomplished by a single programmer and all interfaces to the MMWB platform were accomplished through the public libraries. No internal data structures or assumptions were used or made.

5.3.1.1. Interface

twb provides a menu-assisted command interface. This style of interface provides experts or experienced users with the power and efficiency of a command interface, but allows less experienced users to receive the guidance and help provided by menus. Each command is a verb followed by a series of arguments. An experienced user can enter the complete command or any initial part of the command. If an incomplete command is entered, the user is presented with a menu consisting of all of the arguments available at that point. The user may then select the desired argument, if more arguments are available, subsequent menus are presented. If the argument is an MMWB object of some form, then the "menu" presented is actually a list of the appropriate objects within the current project. Included within the list of objects are the parent project to the current

project, and all projects which are contained within the current project. The user may then "browse" through the tree of projects to select the desired object. This functionality is provided by a library which is described in more detail below.

twb divides the screen into five separate areas or windows, three of which are scrolling areas. The top line of the screen is reserved as a *header* window. The header displays the current project, the **twb** version number, and the date and time. By default the bottom two lines of the screen display single line messages announcing the arrival of dialogs. When a dialog arrives the field is scrolled up and the bottom line is cleared. A line is then displayed showing the tool that sent the dialog followed by the prompt string from the dialog.

The lines above the dialog window comprise the command window. By default the command window contains two lines. The user enters commands on the bottom line of this window. As commands are entered the window is scrolled up one line.

The lines above the command window display errors and messages. The errors are signified by an audible bell, while messages are displayed with no audible signals. This window is also a scrolling window.

The lines between the top of the message window and the bottom of the header window comprise the display window. This window is used to display all of the menus, forms, and browsers which result from user commands. These are described below in more detail.

5.3.1.2. Commands

twb has a relatively limited command set as shown below in Table 14. Each of these commands and their functions are described in detail below. When **twb** is executed

it connects to DM and opens up the user's home project. If there are any session objects in the home project a menu is displayed on the user's screen with the name of each session object and a menu item for a new session. Once the user selects the desired session *sm_attach_session* is used to connect to the SM, the screen is cleared and the windows are displayed. Any dialogs which have been queued up for that session are dispatched to **twb** and are displayed.

Command	# of arguments	Description
QUIT	1	Exit **twb**
STATUS	0	Display the status of the session
BEGIN	1	Begin a protocol
END	1	End a protocol
LIST	1	List various things
HELP	1	Display information about commands
ANSWER	1	Respond to a dialog
CREATE	1	Create a new project
BROWSE	0	Browse the project tree
DELETE	1	Delete an object
NAME	2	Name an object
SELECT	1	Select a project to be "current"
MODIFY	1	Modify an option sheet
SET	3	Set various internal values (not implemented)
COPY	2	Make a copy of an object (not implemented)

Table 14 - **twb** Commands

5.3.1.2.1. QUIT

The QUIT command uses the *sm_dettach_session* call to exit the session. The optional argument to QUIT is the string to use to name the object which stores the session information. The terminal is then reset to the status it was in before **twb** was invoked.

5.3.1.2.2. STATUS

The STATUS command displays the status of all of the tools associated with the running session. The information displayed includes the name of the tool, the current state of the tool, and the current project of the tool.

5.3.1.2.3. BEGIN

BEGIN begins the execution of a protocol. The single argument to this command is the name of the protocol to invoke. If the user does not provide the name of a protocol object, the list of available protocols in the current project, the user's home project, and the system project is displayed in a row-column format and the first protocol on the list is highlighted. The user may then use keys to move the cursor around until the desired project is highlighted. The SPACE key is then used to select the project. The ESCAPE key may also be depressed to abort the command. If there are two protocols with the same name in different projects, then the object in the current project is given precedence, followed by the object in the user's home project. The option sheet for that protocol is then posted as a dialog and must be filled in using the ANSWER command (see below). After the option sheet has been filled in, the tool description and the option sheet are sent to SM and the protocol is initiated.

Note that BEGIN is specific to the execution of a protocol. In its current implementation, **twb** does not support instruments. If instrument support were added to **twb** START and STOP commands would be added which would be the equivalent of BEGIN and END, respectively.

5.3.1.2.4. END

END takes a single argument, which is the number of the protocol (as assigned by SM) to terminate. If the user does not provide an argument then a list of the names of all running protocols is displayed. The user can then highlight the protocol to end and select it.

5.3.1.2.5. LIST

The single argument to LIST is the name of the item the user wishes to list. The possible arguments are shown in Table 15.

Argument	Description
PROTOCOLS	List known protocols
INSTRUMENTS	List known instruments
PROJECTS	List projects
OBJECTS	List all objects within this project
DIALOGS	List all unanswered dialogs

Table 15 - LIST Command Arguments

If the argument is PROTOCOLS, INSTRUMENTS, PROJECTS, or OBJECTS then a simple list of the objects is displayed on the screen in a row-column format. If the argument is DIALOGS, then a list of all pending dialogs is displayed, one per line. The tool which sent the dialog and the name of the dialog are displayed along with the dialog number.

5.3.1.2.6. HELP

The HELP command takes one argument as its option which is the command to display help for. If no argument is given, a list of all of the commands is displayed in the display window and the user is allowed to select the desired command.

5.3.1.2.7. ANSWER

ANSWER allows the user to respond to dialogs. If no argument is given, then a list of pending dialogs is displayed and the user may select the dialog he/she wants to answer by highlighting it. If the user knows the number of the dialog already, then it may be given as an argument. After the desired dialog has been selected a form is displayed showing each field in the dialog. The user may then highlight the desired field and press the SPACE key to select it. If the field is an integer, text, or floating point field, the field is then cleared and the user is allowed to type in the value. If the field is a yes/no or true/false, then the field is highlighted and the values are toggled as the user depresses the SPACE key. If the field is a selection list, the list is displayed and the user may highlight the desired item. In any case, when the user types RETURN the values are accepted. If the user does not change a field, then the default value is used. Once the user has entered all of the desired fields then he/she may press RETURN and the dialog is returned.

While a dialog prompt is highlighted there are a couple of additional keys the user may press. If the user presses a '?' key the help text (if any) which was sent with the dialog is displayed. An '=' key will display the type of dialog and values for the upper and lower bounds. And finally, if the user types ESCAPE, the ANSWER procedure is terminated and the dialog remains unanswered.

5.3.1.2.8. CREATE

CREATE allows the user to create a new project within the current project. If the user does not supply the name of the project to be created, he/she is prompted for it.

5.3.1.2.9. BROWSE

The BROWSE command is the basis for many of the commands in **twb**. There are no arguments to BROWSE. Once the user has entered browse mode, all of the objects in the current project are displayed on the screen, along with an item for the parent project, in a row-column format. The first object on the list (usually the parent project) is highlighted. The user may then move the cursor to highlight the other objects in the display. If the highlighted object is a project, then the SPACE key will move the browser into that project, and display the objects within that project. The other commands available are shown in Table 16 below.

Key	Description
<ESCAPE>	Exit from the browser
c	Copy the object
d	Delete the object
n	Name, or rename the object
+	Select this project as the current project
=	Display brief information about the object
?	Display detailed information about the object

Table 16 - Keystroke Commands in BROWSE Mode

5.3.1.2.10. DELETE

DELETE allows the user to delete an object. If the user does not provide an object then a browser interface is displayed similar to the BROWSE command. The user may then move to the desired object and press the 'd' key to delete it. The other BROWSE commands available are '?' and '='.

5.3.1.2.11. NAME

NAME allows the user to name or rename an object. If the user gives no argu-

ments, a browser interface is displayed similar to the BROWSE command. The user may select the object to be named by highlighting it and typing 'n'. The user is then prompted for the new name of the object. The other BROWSE commands available are '?' and '='. The user may also provide on the command line the name of the object followed by the new name for that object.

5.3.1.2.12. SELECT

SELECT allows the user to select a project to become the current project. If the user does not provide the name of the project then a browser interface similar to the BROWSE command is displayed. The user may then find the desired project and use the '+' key to select it.

5.3.1.2.13. MODIFY

The MODIFY command allows the user to modify the default values in an option sheet. The interface is identical to the dialog interface described for the ANSWER command above. MODIFY takes one argument, which is the name of the protocol or instrument object to modify. If no argument is given, then all protocol and instrument objects are displayed and the user is allowed to select the desired object.

5.3.1.2.14. SET

SET allows the user to set various parameters which affect the function of **twb**. This command has not yet been implemented.

5.3.1.2.15. COPY

COPY allows the user to copy an object. If the user gives no arguments, a browser

interface is displayed similar to the BROWSE command. The user may select the object to be copied by highlighting it and typing 'c'. The user is then prompted for the new name of the object. The other BROWSE commands available are '?' and '='. The user may also provide on the command line the name of the object to be copied followed by the new name for that object. This command is unimplemented until the new **DM** is complete.

5.3.1.3. Browser Library

It should be clear from the description of the **twb** commands above that many of them utilize the display window to display menus or browsers or forms. Part of the design of **twb** included a library to provide a menu/browser interface. This library is called *libbrowse* and is used by **twb** to create all of the menus, browsers, and forms. *libbrowse* provides a generalized object known as a BROWSER. There are currently two types of browsers: form browsers and row-column browsers. Menus and forms are implemented as form browsers, while the browser interfaces used by the BROWSE command and other row-column object displays are implemented as row-column browsers.

Browsers contain information about the type of the browser, the window the browser is contained within, and the number of screens in the browser. In addition, a browser contains two lists of FIELDs (defined below). One list contains static fields like labels, titles, and prompts, and one contains active fields such as buttons, text entry fields, and toggle fields. Fields contain information about their type; usually a string used as a label; some formatting information such as left, right, or centered justification; display flags such as inverse video; x and y locations; and a function to be called if this field is selected. Most of these fields need not be specified and have reasonable default values.

Functions are provided for the creation of browsers and the addition of fields to a browser. Once all of the desired fields have been added to a browser, a function call is provided to format it, and another is available to display it. Once a browser has been displayed, a function is provided to activate it. For row-column browsers, *libbrowse* will automatically lay out all of the fields in the browser which have provided the default value (-1) for the x, y, and screen values.

In addition to the basic functions provided by the library several convenience routines are provided. One routine allows the user to create a small browser which prompts for input from the user and allows the user to enter a text string. One common occurrence in **twb** displays is that a pair of fields in a browser are somehow related. For example, in the display for the ANSWER command, the dialog prompts are displayed as buttons each, followed by a field which contains the value of the dialog. To support this common usage, a function is provided which allows pairs of fields to be specified together, along with the relationship between the two fields. All of the functions in *libbrowse* are described in detail in the manual pages in Appendix C.

5.3.1.4. I/O Library

The browser library provides a series of high-level functions for the creation and execution of a certain style of interface. While *libbrowse* was written to support interfaces on ASCII terminals, there is nothing specific about the library which limits it to one device. To maintain and insure this independence, *libbrowse*, and in fact, the rest of **twb**, was written on top of an I/O interface to allow easy portability to different screen packages and/or window systems.

The I/O interface presents a simple series of function calls for the creation and destruction of windows, the input and output of characters and strings, and the manipulation of various display functions such as inverse video. The interface was intentionally kept simple enough to make porting it easy, but functional enough to provide a complete I/O interface. The only I/O interface that has been implemented is one for the *curses* library described above. The manual pages for the I/O interface are contained in Appendix C.

5.3.2. Lessons Learned

The implementation of **twb** was an invaluable experience for learning about the strengths and weaknesses of the **MMWB** architecture. Most of the development time associated with **twb** was in the implementation of the user interface. Very little time was needed to implement the interface to **MMWB**. This demonstrated that the interface provided by the public libraries supplied a sufficient set. In fact, the interface proved very easy to use and quite robust.

Some deficiencies of the various interfaces did become clear, however. First, there was no way to implement the COPY command without opening up the object and actually copying each cell. A *libofs* level copy function will be provided in the new *libofs*. Another issue which was discovered was the need to be able to specify a path to an object. In the early designs, there was no restriction on how the projects might be organized. It became clear, however, that the **MMWB** needed to have a syntax for specifying a path to an object. This implied that the project hierarchy needs to be able to be traversed as a tree. The exact syntax and implementation strategy for this within the **DM** have not been decided upon, but it will be a feature of the new **DM**.

A problem was also discovered with the option sheet handling mechanism. In the current design when a protocol or instrument is launched, a special dialog is added to the dialog queue which represents the option sheet. This dialog is always given dialog number zero (0) to avoid conflicting with the dialog numbers assigned by SM. Unfortunately, it's possible to begin the invocation of one tool before completing the option sheet from a previous invocation. This results in multiple dialog number zeros and considerable confusion. Two possible solutions to this problem are either to extend the SM-UIM protocol to get the SM to assign a dialog number, or to use a scheme where a range of numbers (e.g. 1 - 100, or all negative numbers) are reserved for option sheets. No decisions have been made yet about which approach will be taken. The protocol extension seems to be the cleaner of the two approaches, but has the potential disadvantage of decreasing the responsiveness of the interface. The reserved numbering approach may confuse the users, but is the easiest to implement.

5.4. spasms - Integration of a Foreign Tool

One of the more important features of MMWB is the ability to integrate foreign tools. The Foreign Tool Driver, FTD, was written for this purpose. The first tool to be integrated using FTD was a molecular mechanics package known as SPASMS which was developed at UCSF. The first task was to modify SPASMS so that its options and arguments were prompted for rather than collected as command line options. This pointed to an early difficulty with FTD and the option mechanism. What was needed in this case was the ability to transform MMWB dialogs into command line options to SPASMS. This is not possible in the current design[9]. So SPASMS was modified to prompt for its

[9] One possible way around this problem would be to have a special dialog flag, or more likely a semantic type which specifies a command-line option. This could be interpreted by the FTD and

options. The resulting **FTD** script is shown below in Figure 11. Once implemented, the results were very promising. The interaction with the foreign tool through the option sheet mechanism was quite smooth with no noticeable performance degradation. In addition, it now became quite feasible to modify SPASMS to send periodic update messages in the form of notification dialogs. Such messages were impractical before integration into **MMWB** because the length of the runs was long enough to prohibit directly writing to the terminal throughout the run.

---

the result could be included on the command line of the tool. This will be evaluated as a future extension of **FTD**

```
#
# FTD Ruleset for David Spellmeyer's SPASMS
#
# NOTE: this assumes the MMMB version of SPASMS which prompts
#       for all input files
#
#
input_file_name:
    " Input file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "mdin";
            prompt "Input file name";
            help "Name of the SPASMS input file";
        }
     state=2;

output_file_name:
    state==2?
    " Output file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "mdout";
            prompt "Output file name";
            help "Name of the SPASMS output file";
        }
    state=3;


start_file_name:
    state==3?
    " Starting coordinates file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "inpcrd";
            prompt "Starting Coordinate file";
            help "Name of the file containing the starting coordinates";
        }
    state=4;

parm_file_name:
    state==4?
    " Parameter Topology file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "prmtop";
            prompt "Parameter Topology file name";
            help "Name of the SPASMS topology file";
        }
    state=5;

restart_file_name:
    state==5?
    " Restart file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "restrt";
            prompt "Restart file name";
            help "Name of the SPASMS Restart file";
        }
    state=6;

reference_file_name:
    state==6?
    " Reference coordinates file name.*"=>
```

```
        ftd_dialog[FTD_SEND],
        {
            text default "refc";
            prompt "Reference file name";
            help "Name of the SPASMS reference file";
        }
    state=7;

velocity_file_name:
    state==7?
    " Velocity dump file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "mdvel";
            prompt "Velocity dump file name";
            help "Name of the SPASMS Velocity dump file";
        }
    state=8;

coordinate_file_name:
    state==8?
    " Coordinate dump file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "mdcrd";
            prompt "Coordinate dump file name";
            help "Name of the SPASMS Coordinate dump file";
        }
    state=9;


energy_file_name:
    state==9?
    " Energy dump file name.*"=>
        ftd_dialog[FTD_SEND],
        {
            text default "mden";
            prompt "Energy dump file name";
            help "Name of the SPASMS Energy dump file";
        }
    state=10;

freq_dump:
    state==10?
    " Frequency.*"=>
        ftd_dialog[FTD_SEND],
        {
            integer default 10;
            prompt "Number of steps between status";
            help "Number of steps between status";
        }
    state=11;

ftd_def:
    ".*"=>
        ftd_notify,
        {
            notifier;
            prompt "SPASMS Output";
        }
    ;
```

Figure 11 - FTD Script for SPASMS Integration

5.5. clock - A Simple Example Protocol

In order to test the interface to SM which is provided for protocols, a simple tool was written to send a notification dialog every *interval* seconds, where *interval* is the one option in the option sheet for this protocol. As can be seen from the source code in Figure 12, the interface between a protocol and SM is extremely straightforward and easy, requiring very few lines of code.

The use of **clock** as a prototype protocol was tested by modifying it to serve as the protocol for the performance tests. This turned out to be trivial, requiring only that a new option be read to determine the test number, and that a case statement be added to send the proper dialog type. This protocol should be easy to use as a starting point for the development of any MMWB protocol.

```
/*
 * clock:
 *      A simple (very) protocol to test libsm
 */

#include        <stdio.h>
#include        <syslog.h>
#include        "sm.h"
#include        "dm.h"
#include        "dialog.h"

static  char    *cid;
static  char    *tool;

main(argc,argv)
int     argc;
char    **argv;
{
        extern  int     optind;
        extern  char    *optarg;
        int             c, err, interval;
        long    clock;
        DIALOG          d;

        tool = argv[0];         /* Get our tool name */

        openlog(tool, LOG_PID, LOG_LOCAL4);

        while ((c = getopt(argc,argv,"C:")) != EOF)
                switch(c) {
                  case 'C':
                        cid = optarg;
                        break;
                }

        /*
         * The dispatcher will provide us with a connection
         * to the Session Manager on fds 0,1, and 2.  We need
         * to begin by initializing the library.  Note that
         * this will wait for the session manager to send us our
         * options before returning.
         */
        if (sm_init(cid) == NULL) {
                syslog(LOG_INFO, "sm_init failed - err %d", sm_errno);
                syslog(LOG_INFO, "(just in case) dm_errno %d: %s",
                        dm_errno,dm_errlist[dm_errno]);
                abort();
        }

        if (err = sm_request_option("interval", &interval, DI_IVAL) < 0)
                syslog (LOG_INFO, "sm_request option failed (%d)",err);

        while (1) {
                sleep(interval);
                clock = time(0);
                d = dialog_create(DI_NOTE,0,"Clock",ctime(&clock));
                (void) dialog_set_flag(d,DI_SUPERCEDES,1);
                (void) sm_send_dialog(d);
                dialog_free(d);
        }
}
```

Figure 12 - *clock* Source

5.6. Evaluation of the Design Process

The design for MMWB is complete and the platform has been implemented, along with a prototype UIM, a prototype protocol, and the integration of a foreign tool using the FTD. As yet, however, MMWB has not been put into production use. This makes the evaluation of the system somewhat difficult. However, some evaluation of the design process may be appropriate at this point.

The MMWB architecture was designed primarily by two individuals, with support and input from two others. In order to evaluate the process, the two primary designers (one of whom was the author) discussed the design process to answer the following questions:

- How closely was the approach outlined in Chapter 2 followed?
- What is the current impression of the quality of the design?
- How did the conceptual model affect the design?
- Was the lack of a formal specification missed?
- How did the interactive process impact the design?

Both of the designers felt that the approach outlined in Chapter 2, and depicted in Figure 3, was followed quite closely during the entire design process. There was a slight difference of opinion about the overall quality of the design. Both designers felt overall that it was a good design with some very nice features; however one of the designers expressed concern that because of the complexity (i.e. many interacting components) there were potential integration difficulties. In particular, the fact that the individual components were not testable by the users during development was raised as a concern. The other designer felt that this was characteristic of an architectural design and that the

architectural foundations had to be laid before user testing could begin.

The use of the conceptual model as a design aid was discussed in detail. Both designers felt that the design process paid a lot of attention to the conceptual model, and that the design itself strongly reflected the underlying conceptual model. Interestingly, the implementation of that design was somewhat more independent of the conceptual model, largely because the design was so closely tied to it. Clearly, error messages and terminology needed to reflect the appropriate conceptual model, but other implementation decisions could be made on the basis of good computing practice rather than adherence to the conceptual model.

One of the major advantages of using the conceptual model to drive the design was that it gave designers an external criteria for evaluating design alternatives. Often design decisions are made based upon performance, modularity, or ease of implementation, none of which reflects what the user may perceive.[10] This external criteria was seen as forcing the user's view of the system into the design process, without requiring a user to be a formal part of the design team. An immediate question which might arise is whether it would be better simply to include a user as part of the design team. For MMWB, this was neither practical nor would it have achieved the same ends. MMWB is to be used by a variety of users, from several different disciplines. While potential users of the system were involved in the design process, requiring any single user to take time from their research to be involved in the details of the design would not have been feasible. Rather, the designers were able to use verbal descriptions of the conceptual model to communi-

[10] Performance may be an exception to this statement. However the user's view of performance is probably based more on a threshold of acceptability than an appreciation of whether this algorithm is 500 milliseconds faster than another one.

cate to the user community the goals and functionality of the system. In this manner, the conceptual model functioned not only as a design aid, but also as a mechanism to help communicate that design to the target user community. It is also important to note that the designers of **MMWB** both had significant experience in working with the research community and had both developed a good understanding of the application domain.

The use of the conceptual model was perceived as extremely valuable, but the costs of the development of that model were also apparent. A great deal of effort was expended in refining the conceptual model to ensure that it would be consistent with the model of the system formed by the users. This refinement involved a great deal of attention to terminology. For example, the terms *instrument* and *protocol* were the topics of several design sessions, as was the importance of differentiating between them. While both designers recognized the time and effort required in the formulation of the conceptual model for the system, both felt that the potential benefits far outweighed the costs, and both felt that they would use this technique again.

Neither designer felt that the lack of a formal specification negatively impacted the design. In the words of one of the designers, "we would never have done it anyway". One interesting point that was made, however, is that as the design progressed and was documented, a formal specification *was* created. Each of the library interfaces, and the communications protocols between each of the components, was clearly documented and specified. In a fashion, a formal (i.e. written) specification resulted from the design, rather than having the design result from the specification.

The interactive nature of the design process seemed a natural extension of the normal development cycle used within this environment. It was felt that the iterations added

more constraints on the design, which was seen as being quite useful. For example, the design for the rewrite of the DM discussed above is constrained by the rest of the MMWB architecture. This allows the DM designer to further optimize the implementation for the expected use. Another iterative component that was stressed was the importance of the feedback between the design stage and the development of the conceptual model. For example, the need to clearly distinguish between *instruments* and *protocols* arose during a design session after the basic conceptual model had already been formulated. This is a natural point of iteration, but modifications of the conceptual model should not be taken lightly. As the process continues, changes in the conceptual model could have dramatic effects on the implementation and design. This stresses the importance of choosing a good conceptual model for the system, and devoting time and effort early in the process to refine and solidify that model.

5.7. User Perceptions and Suggestions

The current state of MMWB makes the collection of user perceptions extremely difficult. Several presentations have been made to the user community, and some users were asked to "try out" **twb** to gain their reactions. In order to actually solicit meaningful user reaction, however, enough instruments and protocols would have to be written for, or integrated into, MMWB to allow the users to conduct meaningful work. In the absence of that the only responses have to do with desired features which are not currently offered, and brief impressions users have offered about **twb**. The features which have been requested so far which were not included in the design are: the inclusion of some form of notebook, some user-oriented scripting language, and the need for an object dialog. Each of these features, along with possible design and implementation

strategies, is discussed below.

5.7.1. Laboratory Notebooks

As researchers goes about their work, it is extremely important for them to be able to keep track of what manipulations they made to which objects. In the laboratory, this is done by making entries into a laboratory notebook. Often these entries are handwritten, but they may included output from various computerized or automated systems. Users can still utilize the same method if their "workbench" is **MMWB** by keeping a laboratory notebook opened on their desk and making the appropriate entries. Many potential users of the system requested that this be automated, and become an integral part of **MMWB**.

The implementation of a notebook feature should be straightforward due to the fact that all communications between users and their protocols occurs using dialogs. A first step in the implementation would be to create a new **MMWB** object, *notebook*. The notebook object could be added to any project, and when present, it implies that the user desires to track interactions with protocols.

A new library would need to be implemented to support these interactions and to provide functions for the entering of protocol invocations and dialog values. It would not be prudent to retain all of the information contained within a dialog. The only necessary information would be the name of the dialog, the name of the protocol, the dialog prompt, and the value. Functions would also have to be included for adding user's comments into the text of the notebook. In addition, some form of extension mechanism needs to be included so that instruments and protocols can insert various entries into the notebook.

5.7.2. MMWB Scripting Language

Another request that users made was the ability to "script" various interactions with the UIM. One approach to this would be to make this a UIM dependent function which would be dependent on the UIM implementor to design and implement. The user would need to learn the scripting language of the UIM they used most often, and for other UIMs would probably just avoid scripting.

Another approach would be to divide the scripting language into two sections. One would be a UIM dependent section, which would include screen manipulations, potentially the creation of "buttons" which would invoke scripts, and other user interface features. The other section would be a UIM independent section which would provide mechanisms for looping, flow control, and interactions with MMWB. A scripting language, *tcl* [20], which has recently become available may be a very good model for this type of interface, and may make a good basis for a general MMWB scripting language. Such a language would also provide an important prototyping facility and would augment the MMWB multi-step protocol capabilities nicely.

5.7.3. Object Dialog

One of the common activities for protocols and instruments will be the determination of which objects in the current project they should use for input. As was discussed in Chapter 3, in the best case there would be only one object of appropriate type. Clearly, this will not always be true. When there is more than one object of the type required by the protocol or instrument, then some form of resolution needs to occur to allow users to choose between the objects. The browser interface used by **twb** would provide a very nice interface for resolving between objects of the same type. Other UIMs

can be expected to have some mechanism for browsing the project/object hierarchy. An early request by potential implementors of protocols on behalf of their users was the ability to access the browser interface of a UIM.

One way to access the browsing mechanism provided by a UIM would be to implement a semantic dialog. The semantic option for this dialog would contain the current project and the list of "interesting" object types. When the user answers the semantic dialog, the UIM could provide the browsing interface so that the user could select the desired object. This semantic dialog should be implemented by all UIMs, but if it is not implemented by a UIM, what should the syntactic type of the dialog be? This problem led to the realization that there needs to be a syntax for specifying a path to an MMWB object. If this were the case, then the syntactic type of the dialog could be DI_TEXT.

5.8. Conclusions

The MMWB development team consists of five individuals, none of whom spends more than twenty (20) percent of their time on MMWB. In fact, three of the developers spend considerably less than that. A reasonable upper-limit estimate of the time devoted to the development of MMWB to date would be between four and six man-months spread out over an actual time of two years. During that period of time, the workbench metaphor was developed, the MMWB architecture designed and implemented, a prototype UIM was written, prototype protocols produced, and a foreign tool was integrated. In addition, the developers met with users, presented the design and architecture formally to the user community at large, and met with potential protocol and instrument developers to familiarize them with the architecture. This reflects a major accomplishment for such a small group of developers with so many other service responsibilities.

The design approach contributed greatly to the outcome of this effort. The conceptual model of a chemist's workbench provided a user's view of the system throughout the design process and resulted in a system which, although complicated, is quite consistent and simple from the user's viewpoint. The prototype UIM, protocol, and foreign tool demonstrate that all of the architectural components are in place to develop production applications and tools. They also demonstrate that the development of those applications and tools is not extraordinarily complicated by the MMWB architecture and that, in fact, it can be quite simple.

Reflecting back on the goals for MMWB outlined in Section 3.2, the architecture and design clearly provide for: data sharing through the DM; resource sharing and multiple, consistent user interfaces through the SM and multiple UIMs; compatible computing services and distributed processing through a combination of the dispatcher, SM, and DM; extensibility through the implementation of new protocols, instruments, and integration of foreign tools; and implementation in finite time. The only goal which has not been completely met is the goal of network transparency. The designers are currently looking at ways to expand the role of the dispatcher to actually locate services on the network, rather than simply execute them. If this were the case, then the tool description for protocols and instruments would not need to specify the list of hosts where that tool could be found. The result would be an important step in making the network transparent to the user.

Chapter 6

Future Directions

6.1. Introduction

The previous chapters of this document have outlined the work which has been done to date on **MMWB**, and in defining an approach to software architectural design which is tailored to research environments. A great deal of work has been accomplished in both designing and implementing **MMWB**. Unfortunately, a great deal of work remains to test the design approach, and enhance **MMWB**. This chapter will explore these future directions, both in terms of the process and in terms of **MMWB** itself. Much of the work outlined here is already in progress, either in an early discussion stage, or in a resource allocation stage. This is particularly true for the work remaining on **MMWB**. The further work described for the design approach has not yet been started, and the resources needed have not been identified.

6.2. Design Approach

The experience of the designers of **MMWB** strongly supports the contention that the proposed design approach provides a considerable improvement over traditional approaches. The statement that this approach would be used again for similar systems is encouraging. The next step would be to utilize this approach for the design of another system, with a different set of designers, users, and goals. If those designers also believed that they benefited from the approach, it would add a great deal of credence to the assertion that the approach can be generalized. One potential source of such a test is the Scientific Computing Group at Genentech, Inc. which is involved in the design and

implementation of similar systems for biomedical researchers.

One of the major stated benefits of the proposed design approach is that the user plays a more central role in the design by the formulation of a conceptual model. This raises several questions. First, does the development and adherence to a conceptual model really make the system better from the user's perspective? Second, does the adherence to a conceptual model make the system easier to learn and more consistent? Third, is the use of a conceptual model simply a stop-gap measure to make up for the lack of a user on the design team? Further work on this approach to design should attempt to answer each of these questions.

An experiment has been designed to answer the second question. This experiment would use **MMWB** as the test platform. Two sets of users, and three different UIMs would be used. One of the UIMs would be the graphical UIM developed for **MMWB**. Both of the user groups would be trained on that UIM and the workbench metaphor would be fully explained. The other two UIMs would both be implemented on ASCII terminals. The first would be **twb** as described above. The second would be a UIM with similar functionality, but the terminology would be changed to intentionally avoid any reference to the components of the metaphor. One of the groups would perform a set of tasks on **twb**, then on the other terminal UIM. The other group would first perform the tasks on the other terminal UIM, then on **twb**. In this manner, each group serves as its own control and any order effects may be tested by having the groups start on different interfaces. The users will be interviewed to determine which interface they found preferable, and their performance on the tasks will be evaluated. The execution of this experiment will have to wait until a graphical UIM is available.

The other questions will have to be answered by experience. The first question will only be able to be answered when MMWB goes into production use. The patterns of use and questions and problems which users ask and encounter will tell much about how well the system has been accepted. The third question will be answered by designing systems with users as members of the design team. A project is underway at Genentech to design a system to document and keep track of DNA sequence constructions. A potential user of the system is a member of the design team. The experiences from this design team should be compared with the experiences of the MMWB design to determine the impact of having a user on the team.

6.3. MMWB

There is a great deal of work to be done on MMWB itself. First, the projects already described in the previous chapter need to be completed. Most important of these is the completion of the new DM and its associated library. Once this important component is available, the rest of the software written to date will need to be modified to utilize the new library.

The nature of the application domain for MMWB will place a very large emphasis on objects which contain information about molecules. An MMWB object has been designed which will contain information about molecules, including large biological macromolecules. A library will need to be designed and implemented for access to these molecule objects. This will provide a link into the MMWB object layer for applications which use molecules as their base object. These applications include molecular mechanics programs, molecular dynamics programs, molecular display programs, and molecular modeling programs. These applications will need to be modified to function as protocols

and instruments in the MMWB architecture. Once this is done, MMWB can go into production use.

MMWB will need to support a variety of instruments on graphical display devices. The UIM used on these devices should utilize the graphical capabilities to provide the best user interface possible. The first such graphical UIM will be implemented on a Silicon Graphics Iris Workstation. Other graphical UIMs should eventually be written for other commonly used graphical devices.

Finally, a variety of support tools will need to be implemented both for administrators of the MMWB system, and for users. Tools need to be provided for the archiving and dearchiving of MMWB objects; for editing and displaying text files within MMWB; for registering object types, semantic dialog types, and new users; and for creation and modification of multi-step protocols. All of these will be needed to support the production use of MMWB. Fortunately, the architectural foundations provided by MMWB make the creation of these tools easier than they would otherwise be. However, the time and resources still need to be allocated.

6.4. Conclusions

The design approach proposed by this research and the resulting design of MMWB have both been implemented and tested. Work remains to be done on the MMWB before it can be put into production use. Once MMWB is in production use some of the concepts behind the design approach can be more fully tested from the user's view. In addition, more experience with this design approach needs to be gained from other design teams in other institutions to determine how generalizable it is.

References

1. Apple Computer Company, *Inside Macintosh*, vol. I, II, III. Addison-Wesley Publishing Company, Inc..

2. Arnold, K. C. R. C., "Screen Updating and Cursor Movement Optimization: A Library Package," *Unix Programmer's Manual Supplementary Documents*, vol. 1. Usenix Association, November 1986.

3. Board, C. S. T., *The National Challenge in Computer Science and Technology*, National Academy Press, Washington, D.C., 1988.

4. Boehm, B. W., "Software and Its Impact: A Quantitative Assessment," *Datamation*, May 1973.

5. Couch, G., "Dialogc -- A Dialog Compiler for the MacroMolecular WorkBench," *MacroMolecular WorkBench Programmer's Documentation*, UCSF Computer Graphics Lab, 1988.

6. Gane, C. P., "Data Design in Structured Systems Analysis," *Infotech State of the Art Report on Data Design*, 1980.

7. Grudin, J., "The Case Against User Interface Consistency," *Communications of the ACM*, 32(10):1164-1173, October 1989.

8. Halasz, F. G., T. P. Moran, and R. H. Trigg, "NoteCards in a Nutshell," *Proceedings of SIGCHI+GI '87*, p. 137-142, ACM, New York, 1987.

9. Hartson, H. R. and D. Hix, "Human-Computer Interface Development," *ACM Computing Surveys*, 21(1):5-92, 1989.

10. Henderson., D. A. and S. Card, "Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphics User Interface," *ACM Transactions on Graphics*, 5:211-243, July, 1986.

11. Huang, C., "Designing a Modern Molecular Modeling Environment," *Ph.D. Thesis*, University of California, San Francisco, San Francisco, CA, 1989.

12. Jackson, M. A., "Constructive Methods of Program Design," *Proceedings of the First Conference of the European Cooperation in Informatics*, 44:1976.

13. Jacob, R. J. K., "A Specification Language for Direct-Manipulation User Interfaces," *ACM Transactions on Graphics*, 5(4):283-317, 1986.

14. Johnson, S. C., "Yacc: Yet Another Compiler-Compiler," *Unix Programmer's Manual Supplementary Documents*, vol. 1. Usenix Association, November 1986.

15. Martin, J. H., "A Computational Theory of Metaphor," *Ph.D. Thesis*, University of California, Berkeley, Berkeley, CA, 1988.

16. Modeling, C.-C.-A., *Computer-Assisted Modeling*, National Academy Press, Washington, D.C., 1987.

17. Myers, B. A., "User-Interface Tools: Introduction and Survey," *IEEE Software*, 6(1):15-24, 1989.

18. Norman, D. A., "Some Observations on Mental Models," in: *Mental Models*, Stevens, G., ed., p. 7-14, Erlbaum, Hillsdale, NJ, 1983.

19. Olsen, D. R., "ACM SIGGraph Workshop on Software Tools for User-Interface Management," *Computer Graphics*, p. 71-147, 1987.

20. Ousterhout, J. K., "Tcl: An Embeddable Command Language," *Proceedings of the*

Winter 1990 USENIX Conference, p. 133 - 146, USENIX Association, Berkeley, CA, 1990.

21. Rosenburg, J., R. Hill, J. Miller, A. Schuler, and D. Shewmake, "UIMSs: Threat or Menace? - A CHI '88 Panel Session," *Proceedings of CHI '88*, ACM, New York, 1988.

22. Ross, D. T. and K. E. S. Jr., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, January 1977.

23. Rubinstein, R. and H. Hersh, "Design Philosophy," *The Human Factor: Designing Computer Systems for People*, p. 12-22, Digital Press, Burlington, MA, 1984.

24. Salter, K. G., "A Methodology for Decomposing System Requirements into Data Processing Requirements," *Proceedings, 2nd International Conference on Software Engineering*, October 1976.

25. Scheifler, R. W. and J. Gettys, "The X Window System," *ACM Transactions on Graphics*, 5(2):79-109, April 1986.

26. Shneiderman, B., "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer*, 16:57-62, 1983.

27. Smith, D. C., C. Irby, R. Kimball, W. Verplank, and E. Harslem, "Designing the Star User Interface," *Byte*, p. 242-282, 1982.

28. Waldrop, M. M., "Will the Hubble Space Telescope Compute?" *Science*, 243:1437-1439, March 1989.

29. Wasserman, A. I. and S. K. Stinson, "A Specification Method for Interactive Information Systems," *Proceedings of the IEEE Conference on Specification of Reliable Software*, IEEE, New York, 1979.

30. Wasserman, A. I., P. A. Pircher, D. T. Shewmake, and M. L. Kersten, "Developing Interactive Information Systems with User Software Engineering Methodology," *IEEE Transactions on Software Engineering*, SE-12(2):326-345, 1986.

Glossary

attributes

MMWB *data tables* contain columns called attributes. MMWB attributes are typed and the attribute type controls the type of the data within each *cell*. Attributes may be referred to by name, and new attributes may be added to a table without restructuring the table. One particular attribute type is the TABLE type, which is a pointer to another MMWB table.

cell

A cell is the intersection of a row and an *attribute* in an MMWB *data table* which contains data. Cells may contain text, integers, floats, characters, or other MMWB *data tables*.

compound dialogs

Compound dialogs are *dialogs* which have more than one value to be prompted for and entered by the user. Compound dialogs allow the UIM to combine multiple *dialogs* into a single presentation to the user.

current project

The current project is the *project* which the user has selected. All operations involving MMWB *objects* occur in the current project. *Protocols* and *instruments* will enumerate *objects* in the current project to search for candidate inputs and any output *objects* will be written into the current project.

data manager

The data manager is a server which provides access to all MMWB tables. It is used by the *UIMs*, the *Session Manager*, and most *protocols* and *instruments* to access data and objects. The data manager allows for sharing of data and provides for all necessary concurrency control.

data table

The data model underlying all MMWB objects is that of a table, similar to the relational model. Unlike the relational model, however, the rows in MMWB tables are ordered. The columns in MMWB data tables are called *attributes* and are strongly typed.

dialog

A dialog is a single device independent message between a *protocol* and the user. The message contains a prompt, help text, the type of the dialog and the *semantic type* of the dialog. The primary use for dialogs is to request some input from the user, and return that input to the *protocol* for further processing. Dialogs can also be used to display status information to the user without requiring any user action.

instrument

An MMWB instrument is a program which requires direct access to the screen device. Instruments are specific to the *User Interface Manager* (UIM) that they are written for. Instruments may be launched via the *Session Manager*, or directly by the *UIM*.

object

An object in **MMWB** is the lowest-level entity which the user manipulates. All **MMWB** information is stored in an object of some form. Objects have a CLASS associated with them as well as a TYPE within that CLASS. Examples of object CLASSes are: STATE and DATA. TYPEs further distinguish the object. Examples of object TYPEs for the DATA CLASS are: MOLECULE and NMR Spectra. Object names need not be unique so *protocols* and *instruments* distinguish objects by CLASS, TYPE, and instance number when there are multiple objects of the same name. All object CLASSes and TYPEs are registered in the *system project* to avoid duplication.

object properties

MMWB objects have a list of properties associated with them which give information about the object. Example properties are: Location, Icon, and Comment. Most objects also have a property which is a pointer to the actual data table.

option sheet

An option sheet is a series of *dialogs* which describes the initial conditions for launching a *protocol* or an *instrument*. The option sheet is filled out before the *protocol* or *instrument* is actually started and all of the values are made available to the program when it is launched. Options sheets may be customized and saved within the user's *home project* or the *current project* to simplify future invocations of the *protocol* or *instrument*.

protocol

A protocol is the MMWB term for a program, or series of programs, which do not require direct access to the screen device. An MMWB protocol, like a chemical process, may be a simple transformation (a single program), or may contain many steps (multiple programs).

project

An MMWB project contains all of the objects necessary to complete the purpose of the research. For example, if the goal is to determine the structure of a large protein, the project might contain the primary sequence, any option sheets for protocols and instruments which are tailored for this project, predicted secondary structures, and any predicted tertiary structures. Projects may also contain other projects. For example, the structure of the AIDS virus project may contain a sub-project for deriving the structure of the coat protein.

semantic dialog

A semantic dialog is a dialog which has an entry in the SEMANTIC TYPE field. This allows protocols and instruments (or the UIM) to attribute some semantic information about the value to be returned. For example, if a protocol needs to receive the name of a residue to perform some processing on it, then it can set the semantic type RESIDUE in the dialog. If the UIM or any running instruments understand that type, they can utilize more advanced user interface techniques to allow the user to answer the dialog (e.g. by picking it on the molecule). If neither the UIM nor any of the instruments understand that

semantic type, then the UIM will present the dialog using the normal type. Semantic types are registered in the *system project* to avoid duplication.

session

An **MMWB** session is a collection of protocols, instruments, a UIM, and the Session Manager itself. At a minimum, a session will consist of the Session Manager and either one protocol or a UIM. Users may detach and reattach to sessions without any loss of information or communication with any running protocols. Instruments, because they are dependent on the UIM, terminate when the user detaches from a session.

session manager

The **MMWB** Session Manager is responsible for actually launching all protocols and providing the communications between protocols and the UIM. All dialogs sent via the Session Manager are maintained in a queue until they are answered by the user. This allows the user to change UIMs without any loss of information or communication with the protocol.

system project

The system project is an **MMWB** *project* which contains *objects* for all available *protocols*, *instruments*, *object types*, *object classes*, and *semantic types*. The *objects* are used as prototypes when new objects are created.

user interface manager (UIM)

An **MMWB** User Interface Manager is the program which presents the work-

bench metaphor to the user. All manipulation of projects, objects, instruments, and protocols is implemented by the UIM. UIMs are typically written for a specific device or window system. A UIM is also responsible for implementing all dialog types.

Appendix A

Tool Invocation Details

Tool invocation is a common and complicated process which must be performed by the Session Manager and the Session Manager libraries. This appendix describes the steps involved in tool invocation in more detail.

There are three library routines involved in tool invocation: *sm_invoke*, *sm_send_reply*, and *sm_launch*. In addition, the dialog handler supplied by the UIM, *ui_dialog*, is called to handle option sheets. The relevant sections of *sm_invoke* and *sm_send_reply* are shown below in Figures 13 and 14, respectively.

sm_invoke is called to initiate the process of tool invocation. If **nodisplay** is set in the tool description, it is assumed that all options are correct and *sm_launch* is called with the current option values to actually launch the tool (see Figure 13). When the tool is launched, the tool description is updated to include the *tool_id* of the tool. Because this is updated, the entire tool description is copied upon entry into *sm_invoke*. A pointer to this copy of the tool description is returned.

If **nodisplay** is not set, then the user must be prompted for the values to the options for this tool. This is much more complicated because the mechanism which services both dialogs and options is essentially asynchronous. There are no assumptions about when a user will respond to dialog requests or in what order the user will perform any action. As a result, some mechanism must detect when an option sheet is complete and continue with the tool's invocation. This is handled by putting a copy of the tool description in a queue of pending tool invocations. The destination field of the option sheet is set to the queue slot of the tool description, and the source field is set to zero to indicate

that the user interface manager itself is the source. The option sheet which is pointed to by the copy of the tool description is then provided to *ui_dialog* for display and handling by the UIM. *sm_invoke* then returns a pointer to the queued tool description.

After the user provides any necessary values for the option sheet, the UIM will call *sm_send_reply* to return the dialog to the sender (see Figure 14). In this case, however, the sender is set to tool zero, the UIM. *sm_send_reply* notes this and checks the list of pending tool launches for a tool in the queue at the slot specified by the destination field. If the dialog identifiers match, *sm_launch* is called.

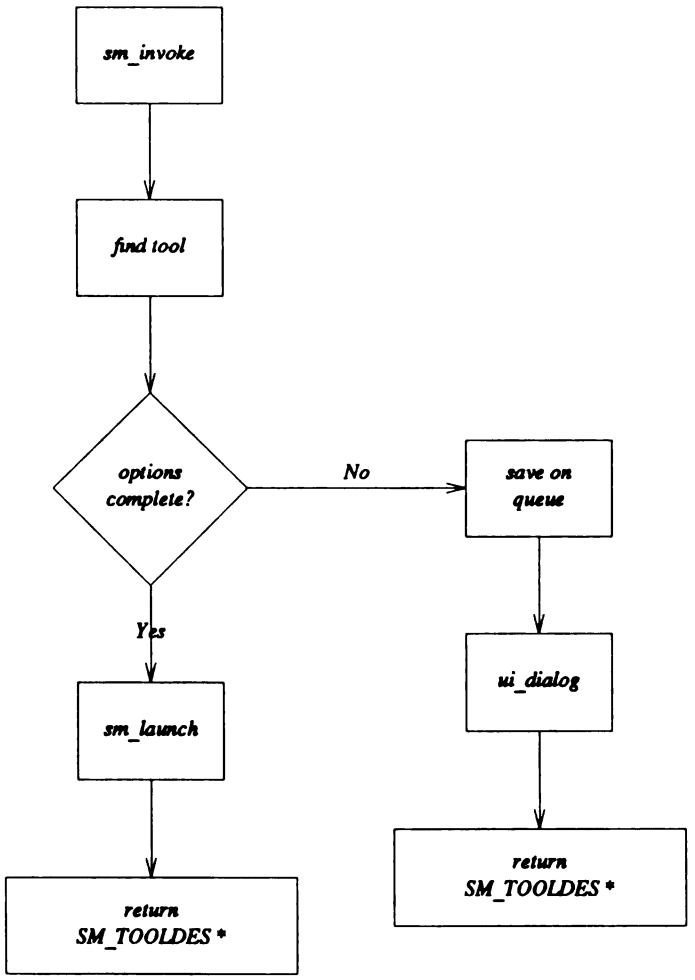


Figure 13 - Flowchart for *sm_invoke*

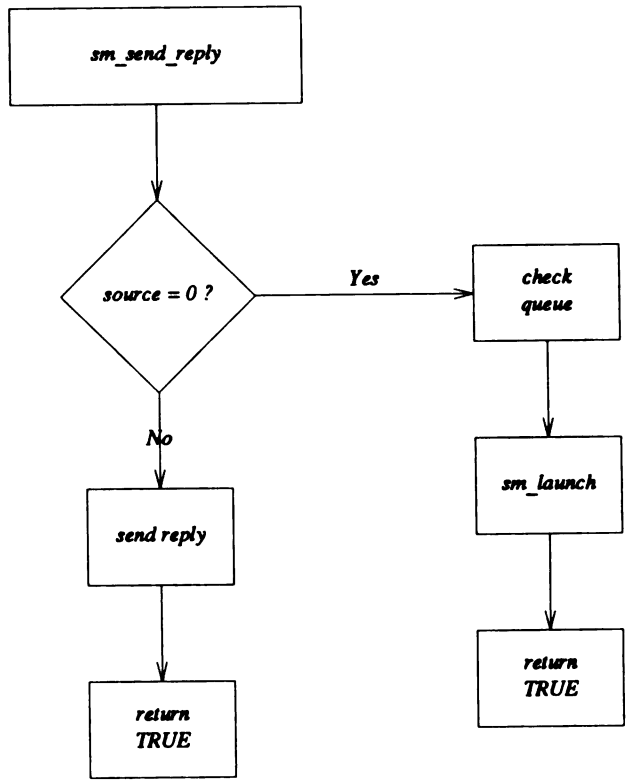


Figure 14 - Flowchart for *sm_send_reply*

Appendix B

Session Manager Communications

This section describes the communications protocols between the Session Manager and the User Interface Manager, and between the Session Manager and a tool in the session. Table 17 contains a list of the types utilized by the protocols described below. Note that all of the protocols described follow the low-level negotiation and response codes outlined in the document **Network Communications in MMWB**. Aggregate types are colon separated and newline terminated. If more than one aggregate type is sent, for example multiple *status list* replies, they are sent on subsequent lines.

Type	Definition
<status list>	[tool_name : tool_id : status NL]*
<dialog>	[name : type : stype : prompt : help : list : value : flags : source : destination : id NL]*
<tool des>	[Version : Location : Subtool : OnError : Host NL]
<success>	**OK**
<tool_id>	Integer

Table 17 - Communications Types

UIM-SMP Communications

Table 18 and Figures 15 and 16 describe the communications between a user interface manager (UIM) and the session manager process (SMP). This protocol is very different from other MMWB protocols because of the peer-peer nature of the interaction between a UIM and the SMP. This peer-peer relationship is manifested by the presence of two different initiators of communications (see Table 18). Contrast this with the protocol described in Table 19 between the SMP and tools, where only tools can initiate communication. It is necessary for the Session Manager to initiate communications to inform

the user of messages from tools at the earliest possible time. In a true client-server communications model the delivery of incoming dialogs would have to wait until the client (in this case the UIM) queried the server for any queued dialogs. While this would be much easier from a communications standpoint, it might disrupt the interactive feel of the interface.

The impact of the peer-peer relationship is that it is necessary to detect and avoid race conditions which may result when both processes initiate a communication at the same time. This is avoided in the UIM-SMP protocol by allowing the UIM to ignore any unsolicited input if it is expecting a reply to a command that it sent. This is shown by the arc labeled **any command** in the transition diagram for the protocol (Figure 15). Note that the action (boxed label) directs the protocol to ignore any commands which are received when the UIM is in state 2. This implies that the SMP must have all of the logic necessary to recover from a situation in which its command is ignored. The transition diagram for the SMP side of the protocol (Figure 16) must have an arc from state **1** back to the start state, **S**. This transition will occur when SMP has sent a command, and instead of receiving a reply, it receives a new command. The action associated with this arc causes the command to be requeued for sending to the UIM as soon as the command received from the UIM is processed. In this fashion, there is no possibility of a race condition, and all of the logic necessary to avoid race conditions is localized in the session manager process.

Initiator	Command	Arguments	Replies
UIM	**Status**		<status list>
UIM	**Send_reply**	<dialog>	<success>
UIM	**Launch_tool**	<tool_des>	<tool_id>
SM	**Send_dialog**	<dialog>	<success>

Table 18 - UIM-SMP Communications Protocol

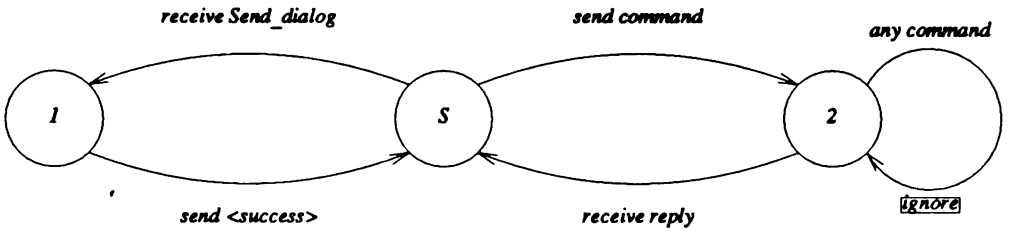


Figure 15 - State Transition Diagram for UIM-SMP Protocol in the UIM

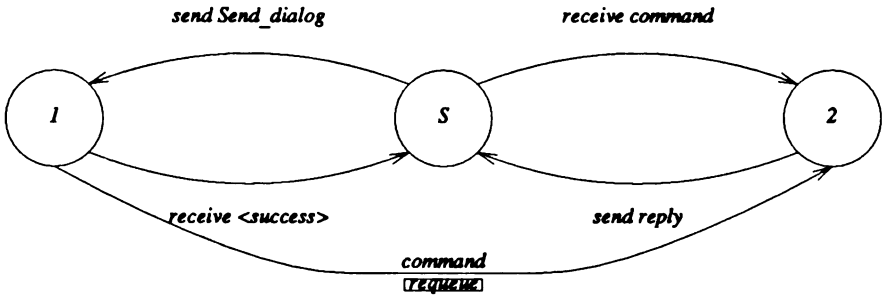


Figure 16 - State Transition Diagram for UIM-SMP Protocol in the SMP

Tool-SMP Communications

The communications protocol between a tool and the SMP is a simple client-server protocol as shown by the transition diagrams in Figures 17 and 18. This implies that two tools which desire to communicate through the SMP must do so in a synchronous fashion. In fact, this mechanism can be used to synchronize two tools which are passing data through the Data Manager.

Initiator	Command	Arguments	Replies
Tool	**Status**		<status list>
Tool	**Send_dialog**	<dialog>	<success>
Tool	**Send_reply**	<dialog>	<success>
Tool	**Read_reply**		<dialog>
Tool	**Read_dialog**		<dialog>
Tool	**Launch_Tool**	<tool_des>	<tool_id>
Tool	**Dialog_Pending**		<success>

Table 19 - Tool-SMP Communications Protocol

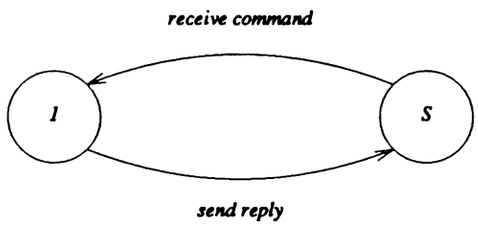


Figure 17 - State Transition Diagram for Tool-SMP Protocol in the SMP

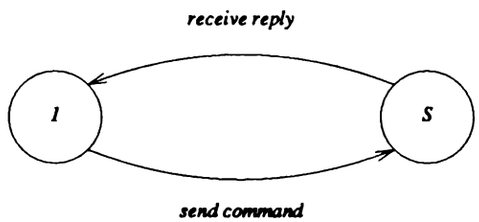


Figure 18 - State Transition Diagram for Tool-SMP Protocol in the Tool

Appendix C

Unix Manual Pages

The Unix manual is divided up into several sections. Section 1 of the Unix manual contains descriptions of programs which are executed directly by the users on the command line or in command procedures. Section 3 contains descriptions of library routines, including calling conventions, return values, and function descriptions. Section 5 contains descriptions of the file formats utilized by various programs. The documentation for **MMWB** was written to comply with the Unix manual page conventions since all of the current users are familiar with them. By convention, the manual entry is described by a name followed by the section number in parenthesis. For example, the manual entry for the Unix command to list directories, **ls**, is found in Section 1 of the Unix manual and is written: **ls(1)**. For clarity, the manual entries included in this appendix have been grouped by the subsystem or library to which they belong rather than by the manual section. Included are manual entries for **FTD**, *libsm* , and *libdialog* .

NAME

ftd – MacroMolecular Workbench Foriegn Tool Driver

SYNOPSIS

ftd [-C cid] [-r ruleset -p program -a arguments]

DESCRIPTION

ftd reads a set of rules, *ruleset* which describe the inputs and outputs required by a foreign tool, *program*. **ftd** can be run in one of two modes. It can be run under the Session Manager, **sm**(1) as a full MMWB application, or it can be run directly from the shell. In the first mode, all of the information necessary to execute the foreign tool must be available as options in the option sheet. The name of the program is passed as option **program**, the argument is list is passed as option **args**, and the path to the rules is passed as option **rules**. If **ftd** is being run from the shell, this information is passed via command line arguments. The path to the rules is specified after the –r flag. The path to the program is specified after the –p flag. And the argument list is specified after the –a flag. Note that all three of these arguments must be specified. The –C command line argument is only used when **ftd** is being run under the Session Manager.

ftd will collect all of the needed arguments and then read and interpret the specified rule file. If no errors exist in the rule file, the requested program is executed with the specified arguments. As input is received from the program it is collected until either an end-of-line, or a one-second timeout with no input occurs. The collected buffer is then passed through the rules until a match is found. Two conditions must be met in order for a match to be accepted. First, any preconditions specified in the rule must be met, and second, the regular expression specified must match the input buffer. If these conditions are both met, the action specified in the rules is performed according to the flags and dialog specified. After the action has returned, any postconditions are set and **ftd** will again wait for input from the tool.

FTD RULE EXAMPLE

```
# This is a test

testname:
        a==1||^4?       # Precondition using variable "a"
        "a*\"junk"=>    # Regular expression
                        # ftd_dialog is called with flags
                ftd_dialog [FTD_QUIT|FTD_OPTION],
                # a four-part dialog
                {{
                        text;     # no default, this is required
                        prompt "output object";
                        help "name of resulting molecule object";
                }{
                        text;     # no default, this is required
                        prompt "pdb filename";
                        help "name of pdb file";
                }{
                        text default "/usr/local/lib/midas/connect";
                        prompt "Atomic radii file";
                }{
                        real default 0.85;
                        prompt "default radius";
                        help "radius used for atom types missing in radii file";
                }}
        a=2+(4&7);
```

DIAGNOSTICS

Functions return NULL on success and a pointer to an error message on failure.

CAVEATS

Foreign tools which attempt to do terminal ioctls will not work correctly under a driver.

This approach does not work very well for a forms-based interface.

SEE ALSO

ftd(3), ftd_rule(5)

NAME

ftd_launch, ftd_parserules, ftd_process, ftd_term – Foriegn Tool Driver library interface

SYNOPSIS

```
#include <mmwb/ftd.h>

err = ftd_launch(path, program, args, readfd, writefd)
        char            *err;
        char            *patch;
        char            *program;
        char            *args;
        int             *readfd;
        int             *writefd;

err = ftd_parserules(rulefile, command_tab)
        char            *err;
        char            *rulefile;
        FTD_CMDTAB *cmdtab;

err = ftd_process(input)
        char            *err;
        char            *input;

err = ftd_term(readfd, writefd)
        char            *err;
        int             readfd;
        int             writefd;
```

FTD RULE GRAMMAR

```
rule ::= name ':'
                conditions '=>' execution ';'

conditions ::=  [precondition '?'] pattern

execution ::=   [action ','] '{' dialog '}' [postcondition]

precondition ::= expression

pattern ::=     '"' regular_expression '"'

action ::=      [function_name] [ '[' flags ']' ]

postcondition ::= var_name '=' expression
```

FTD RULE EXAMPLE

```
# This is a test

testname:
        a==1||~4?       # Precondition using variable "a"
        "a*\"junk"=>    # Regular expression
                        # ftd_dialog is called with flags
                ftd_dialog [FTD_QUIT|FTD_OPTION],
                # a four-part dialog
                {{
                        text;   # no default, this is required
                        prompt "output object";
                        help "name of resulting molecule object";
```

```
        }{
                text;     # no default, this is required
                prompt "pdb filename";
                help "name of pdb file";
        }{
                text default "/usr/local/lib/midas/connect";
                prompt "Atomic radii file";
        }{
                real default 0.85;
                prompt "default radius";
                help "radius used for atom types missing in radii file";
        }}
a=2+(4&7);
```

DESCRIPTION

These routines represent a library interface to the MMWB Foreign Tool Driver routines. They are meant to be used with **ftd(1)**, or **ftd_skel(1)**, but may be called directly outside of these programs. A Foreign Tool Driver for MMWB is a program which processes output from a program which was not written for the MMWB environment, and handles it in accordance with the MMWB. This may mean that the output is formatted into an MWMB dialog and delivered to the user or some other user-specified action taken. This is done by matching the output against a set of *rules*. The rules contain information about preconditions that must be met, patterns to search for, actions to execute if those patterns match, dialogs to be sent to the user, and postconditions which may be set upon completion of a rule. When input is received from the foreign tool, the list of rules is searched for a match and the appropriate action is taken.

ftd_parserules must be called before any of the other routines in this library. Its arguments are a pointer to the file containing the rules and a pointer to a user-constructed table which relates action names to user-provided functions which perform those actions. *ftd_parserules* will read the rule file, and compile all of the rules into an internal table for later processing by *ftd_process*.

ftd_launch is then called to actually start the foreign tool using **path** and **prog** and pass arguments **args**. The read and write file descriptors are returned in **readfd**, and **writefd**, respectively.

ftd_process is called repeatedly with input from the tool to process the input with the rules provided in **rulefile**.

ftd_term is called to terminate the connection to the tool and close the file descriptors.

Some actions are intrinsically defined by the system and are always available for use in rules. These actions are: *ftd_dialog*, *ftd_notify*, *ftd_sink*, *ftd_print*, and *ftd_input*. *ftd_dialog* will send the dialog specified in the rule to the user and wait for the reply. The reply will then be sent to the foreign tool. This is used in response to a prompt by the foreign tool. If the flag **FTD_COPY_PROMPT** is set in the *flags* field of the rule, the input from the tool is used as the prompt for the dialog.

ftd_notify will send the dialog as a notification to the user, and will immediately continue. This can be used to pass on informational messages from the tool to the user.

ftd_sink will discard the input from the tool and return. *ftd_print* and *ftd_input* are used to debug **ftd** rules and roughly correspond to *ftd_notify* and *ftd_dialog*, respectively. *ftd_print* will print the buffer or prompt string on the user's terminal, depending on the **FTD_COPY_PROMPT** flag. *ftd_input* will print the appropriate prompt string and collect input from the user's terminal to be delivered to the tool. These routines can be used without the presence of a User Interface Manager or a Session Manager.

Other flags which effect the actions are:

FTD_QUIT which requests that the driver be terminated upon completion of this action.

FTD_OPTION which suggests that the desired value may have been passed as an option to the driver. This allows values which the user knows will be requests to be entered into the tools option sheet.

FTD_SEND writes the result of the action to the program.

DIAGNOSTICS

Functions return **NULL** on success and a pointer to an error message on failure.

CAVEATS

Foreign tools which attempt to do terminal ioctls will not work correctly under a driver.

This approach does not work very well for a forms-based interface.

SEE ALSO

ftd(1), ftd_rule(5)

NAME

ftd_rule – Foreign Tool Driver rules

SYNOPSIS

name:
preconditions?
"*pattern*" =>
function [*flags*],
{ *dialogs* }
postcondition;

DESCRIPTION

The MMWB Foreign Tool Driver interprets a set of rules which describe the output from a foreign tool and the actions to be performed when that input is detected. The complete syntax for a rule file is shown below.

FTD RULE GRAMMAR

```
rule ::= name ':'
                    conditions '=>' execution ';'

conditions ::=      [precondition '?'] pattern

execution ::=       [action ','] '{' dialog '}' [postcondition]

precondition ::= expression

pattern ::=         '"' regular_expression '"'

action ::=          [function_name] [ '[' flags ']' ]

postcondition ::= var_name '=' expression
```

FTD RULE EXAMPLE

```
# This is a test

testname:
        a==1||4?        # Precondition using variable "a"
        "a*\"junk"=>    # Regular expression
                        # ftd_dialog is called with flags
                ftd_dialog [FTD_QUIT|FTD_OPTION],
                # a four-part dialog
                {{
                        text;     # no default, this is required
                        prompt "output object";
                        help "name of resulting molecule object";
                }{
                        text;     # no default, this is required
                        prompt "pdb filename";
                        help "name of pdb file";
                }{
                        text default "/usr/local/lib/midas/connect";
                        prompt "Atomic radii file";
                }{
                        real default 0.85;
                        prompt "default radius";
```

```
                help "radius used for atom types missing in radii file";
            }}
        a=2+(4&7);
```

SEE ALSO

ftd(1), ftd(3)

NAME

sm_error, sm_get_curr_proj, sm_get_hom_proj, sm_get_sys_proj, sm_kill, sm_launch, sm_read_dialog, sm_read_reply, sm_send_dialog, sm_send_reply_tool, sm_session_status, sm_set_curr_proj, sm_set_file, sm_set_home_proj, sm_set_sys_proj – MMWB functions for all tools which interface to the Session Manager

SYNOPSIS

```
#include <mmwb/sm.h>

err = sm_error()
        char            *err;

proj = sm_get_curr_proj()
        OFS_OBJECT  proj;

proj = sm_get_home_proj()
        OFS_OBJECT  proj;

proj = sm_get_sys_proj()
        OFS_OBJECT  proj;

result = sm_kill(tool)
        SM_TOOL             tool;
        BOOLEAN             result;

result = sm_launch(toold)
        SM_TOOLDES *toold;
        BOOLEAN             result;

dialog = sm_read_dialog()
        DIALOG              dialog;

dialog = sm_read_reply(id)
        int             id;
        DIALOG              dialog;

dialog_id = sm_send_dialog(dialog)
        DIALOG              dialog;
        int             dialog_id;

result = sm_send_reply_tool(dialog)
        DIALOG              dialog;
        BOOLEAN             result;

statuslist = sm_session_status()
        SM_SESSION_STATUS *statuslist;

sm_set_curr_proj(proj)
        OFS_OBJECT  proj;

sm_set_file(file)
        FILE    *file;

sm_set_home_proj(proj)
        OFS_OBJECT  proj;

sm_set_sys_proj(proj)
        OFS_OBJECT  proj;
```

DESCRIPTION

sm_error returns the text equivalent of the current value of **sm_errno**, the error returned by the **sm** library. The text will also contain any error generated by the **ofs** library or the **dispatcher** library.

sm_get_curr_proj, *sm_get_home_proj*, and *sm_get_sys_proj* return the currently set **current**, **home**, and **system** projects as **ofs** objects (**OFS_OBJECT**).

sm_kill sends a request to the **Session Manager** to terminate a tool. The value of **tool** is the tool identifier which is part of the tool description after a successful return from *sm_launch*.

sm_launch sends a request to the Session Manager to invoke the tool described by the argument, and, if successful, sends all options to be delivered to the tool when it requests them. **Toold** must be a valid tool description with at least one option. If **result** is TRUE then the tool was invoked and the options were delivered. The tool identifier will be set in **toold**. If **result** is FALSE, then the value of **sm_errno** will contain the error condition.

sm_read_dialog will read a pending dialog from the Session Manager.

sm_read_reply will read a reply to a dialog from the Session Manager. The desired reply is indicated by **id**, which is the identifier of the dialog that was sent. If **id** is -1, the first available reply is returned.

sm_send_dialog sends the dialog **dialog** to the Session Manager for delivery to the UIM. **dialog_id** is the dialog identifier assigned by the Session Manager.

sm_send_reply_tool is used to send a reply from a tool to another tool. **dialog** is the reply to be sent.

sm_session_status returns the status of the connected session. The status contains information about the state of the UIM, and the names and states of all tools running under this session.

sm_set_file sets the stream file pointer which is used for interaction with the Session Manager to **file**.

sm_set_curr_proj, *sm_set_home_proj*, and *sm_set_sys_proj* all take an OFS_OBJECT, **proj** as an argument and set the libraries view of the current, system, and home projects, respectively. These values are sent to all tools as part of the invocation process.

DIAGNOSTICS

Functions returning BOOLEAN will return FALSE on errors. Functions returning pointer or handle will return NULL on errors.

SEE ALSO

smt(3), smui(3), mmwb(3), tooldes(3), object(3)

NAME

sm_dialog, sm_exit, sm_fetch_objs, sm_init, sm_init_instrument, sm_notify, sm_request_option – MMWB functions for protocols and instruments which interface to the Session Manager

SYNOPSIS

```
#include <mmwb/sm.h>

reply = sm_dialog(dialog)
        DIALOG          dialog;
        DIALOG          reply;

result = sm_exit(message)
        char            *message;
        int             result;

result = sm_fetch_objs(project, name, class, type, min, max, prompt, help)
        OFS_OBJECT      *results;
        OFS_OBJECT      project;
        char            *name;
        int             class;
        int             type;
        int             min;
        int             max;
        char            *prompt;
        char            *help;

proj = sm_init(cid)
        char            *cid;
        OFS_OBJECT      proj;

proj = sm_init_instrument(cid,semlist)
        char            *cid;
        int             *semlist;
        OFS_OBJECT      proj;

result = sm_notify(message)
        char            *message;
        int             result;

result = sm_request_option(name, value, type)
        char            *name;
        void            *value;
        int             type;
```

DESCRIPTION

sm_dialog sends **dialog** to the Session Manager and awaits a reply. The reply is returned as a dialog.

sm_exit is called to terminate the tool and, if desired, to send a notification dialog to the UIM containing the text supplied in **message**. Note that the user must still call the appropriate termination routine, for most protocols and instruments this is exit(1).

sm_init should be the first routine called by any protocol. It performs all handshaking with the **dispatcher**, initializes the **ofs**, and retrieves all options from the Session Manager. The **cid** is the conversation identifier used for handshaking, and the resulting **proj** is a handle to the currently selected project.

sm_init_instrument like *sm_init* should be the first routine called by any **instrument**. It performs all handshaking, as above, including notifying the UIM of any **semantic dialogs** which this instrument will service. **semlist** is an integer array of the semantic dialog types understood by this instrument.

sm_notify is called to send a notification dialog to the UIM. The content of the dialog is the text supplied in **message**.

sm_request_option returns the value of the option **name** into the parameter **value**. The type of the option is specified by the parameter **type**, which should be a legal dialog type. **value** should be a pointer to a variable of the appropriate type.

All options are read from the Session Manager by *sm_init* and are retained in a buffer local to the sm library. The call to request an option is therefor very inexpensive, and the same option can be returned multiple times. Care should be taken with calls to *sm_request_test_option* to not free the returned text string. If any manipulation of the string is desired, a copy should be made.

DIAGNOSTICS

Functions returning **int** will return -1 on errors. Functions returning pointer or handle will return NULL on errors.

SEE ALSO

exit(1), sm(3), smui(3), mmwb(3), tooldes(3), object(3)

NAME

sm_attach_session, sm_cancel, sm_dettach_session, sm_dispatch, sm_invoke, sm_list_sessions, sm_send_reply – MMWB functions the User Interface Manager - Session Manager interface

SYNOPSIS

```
#include <mmwb/sm.h>

file = sm_attach_session (session, host)
        SM_SESSION  *session;
        char        *host;
        FILE        *file;

success = sm_cancel (options)
        DIALOG      options;
        int     success;

success = sm_dettach_session (session_name)
        char    *session_name;
        int     success;

success = sm_dispatch ()
        int     success;

tooldes = sm_invoke (td)
        SM_TOOLDES *td;
        SM_TOOLDES *tooldes;

sessions = sm_list_sessions ()
        SM_SESSION  *sessions;

success = sm_send_reply (dialog)
        DIALOG      dialog;
        int     success;
```

DESCRIPTION

sm_attach_session is called by a UIM to attach to an MMWB session. The first argument, **session** either contains the session structure of the desired session or NULL if a new session is desired. The second argument, **host** designates the name of the host to look for the session manager on. The return value, **file** is the stream handle to the session manager. This should be used to set the internal stream pointer for the session manager libraries by calling *sm_set_file(3)*.

sm_cancel provides a mechanism to cancel a pending tool invocation during the process of filling out the option sheet. The argument, **options**, is the pointer to the option sheet portion of the tool description. This is used to remove the tool from the list of pending invocations.

sm_dettach_session is called to dettach the UIM from the current session. It is expected that this will be called before the UIM exits. If **session_name** is not NULL, the session will be saved under that name. Otherwise, the current *cid* is used.

sm_dispatch is called by the UIM to handle any input from the session manager. Currently, all it does is call the user supplied routine *uim_dialog* when a dialog request is detected. The argument to uim_dialog is the incoming dialog, and it is expected to return an **int**.

sm_invoke is called to initiate the invocation of a tool. The sole argument, **td** is the tool description of the tool to invoke. If no options need to be presented to the user (because the *nodisplay* flag is set) then the session manager is asked to invoke the tool. If options need to be presented to the user, the tool description is placed on a list of pending invocations, and *uim_dialog* is called with the option sheet. After the user responds to the options, the UIM will call *sm_send_reply* which checks to see if we're replying to an option sheet for a tool on the pending list. If we are, the tool is invoked and the options are provided. In either case, *sm_invoke* returns a copy of the tool description. Note that after the tool is actually invoked, the tool identification field in the tool description will be filled out.

sm_list_sessions returns a list of all sessions active for the current user.

sm_send_reply is called by the UIM to send a dialog reply to the session manager (and thence hopefully to the appropriate tool). The argument, **dialog**, is the reply to be sent.

DIAGNOSTICS

Functions returning **int** will return -1 on errors. Functions returning pointer or handle will return NULL on errors.

SEE ALSO

smt(3), sm(3), mmwb(3), tooldes(3)

NAME

sm_free_tooldes, sm_list_instrument, sm_list_protocol, sm_read_instrument, sm_read_protocol, sm_write_instrument, sm_write_protocol – MMWB functions for dealing with tool descriptions

SYNOPSIS

```
#include <mmwb/sm.h>

(void) sm_free_tooldes(tooldes)
        SM_TOOLDES *tooldes;

tooldes = sm_list_instrument(project)
        OFS_OBJECT  project;
        SM_TOOLDES *tooldes;

tooldes = sm_list_protocol(project)
        OFS_OBJECT  project;
        SM_TOOLDES *tooldes;

tooldes = sm_read_instrument(project, name, find_first)
        OFS_OBJECT  project;
        char        *name;
        BOOLEAN             find_first;
        SM_TOOLDES *tooldes;

tooldes = sm_read_protocol(project, name, find_first)
        OFS_OBJECT  project;
        char        *name;
        BOOLEAN             find_first;
        SM_TOOLDES *tooldes;

result = sm_write_instrument(project, tooldes)
        OFS_OBJECT  project;
        SM_TOOLDES *tooldes;
        BOOLEAN             result;

result = sm_write_protocol(project, tooldes)
        OFS_OBJECT  project;
        SM_TOOLDES *tooldes;
        BOOLEAN             result;
```

DESCRIPTION

MMWB tools are of two varieties: *instruments*, and *protocols*. Instruments are tools which require some direct access to the screen or display device. Examples of instruments are molecular display, molecular edit, and molecular modeling programs. Protocols, on the other hand, are tools which require limited interaction with the user and may rely on the *dialog* mechanism. Protocols and instruments are described by *tool description* objects in a project. The tool description contains information about the name of the tool, the host to run the tool on, and any *option sheet* associated with the tool.

sm_free_tooldes will free the tool description, **tooldes** which was allocated by a call to *sm_read_instrument*, or *sm_read_protocol*.

sm_list_instrument and *sm_list_protocol* will return the list of all tool descriptions of the desired type in **project**.

sm_read_instrument and *sm_read_protocol* both return all tool descriptions of the appropriate type called **name** in the specified **project**. If **project** is NULL, then all tool descriptions matching **name** which are currently available in the system, home, and current projects are returned. If the **find_first** flag is set, only the first object matching **name** and **project** is returned.

sm_write_instrument and *sm_write_protocol* create new objects in the specified **project** of the appropriate type which contain the tool description, **tooldes**. If **project** is NULL, the object is written into the current project.

DIAGNOSTICS

Functions returning BOOLEAN will return FALSE on errors. Functions returning pointer or handle will return NULL on errors.

SEE ALSO

sm(3), smt(3), smui(3), mmwb(3), object(3)

NAME

dialog_create, dialog_copy, dialog_free, dialog_next, dialog_append, dialog_delete – MMWB functions for managing, manipulating, and converting dialogs.

SYNOPSIS

#include <mmwb/dialog.h>

Dialog memory management:

```
dialog = dialog_create(type, destination, name, prompt);
        DIALOG      dialog;
        int         type;
        int         destination;
        char        *name,*prompt;

newdialog = dialog_copy(old_dialog, flag);
        DIALOG      newdialog;
        DIALOG      old_dialog;
        int         flag;

        dialog_free(dialog, flag)
        DIALOG      dialog;
        int         flag;
```

Dialog list management:

```
dialog = dialog_next(dialog_list);
        DIALOG      dialog;
        DIALOG      dialog_list;

dialog = dialog_append(list_head, list_tail);
        DIALOG      dialog;
        DIALOG      list_head;
        DIALOG      list_tail;

dialog = dialog_delete(list, dialog_to_delete);
        DIALOG      dialog;
        DIALOG      list;
        DIALOG      dialog_to_delete;
```

DESCRIPTION

dialogs are the primary mechanism for MMWB protocols to communicate with the user. The primary features of this mechanism are that it is user-interface independent, i.e. no matter what device the user is using, the dialog can be interpreted, and that a protocol and a cooperating instrument can extend the mechanism.

dialog_create allocates a new dialog of type **type**. The dialog destination, name, and prompt are set to **destination**, **name**, and **prompt**, respectively. The return value is a handle to the new created dialog. This handle must be used to modify or send the dialog.

dialog_copy returns a copy of the dialog pointed to by the dialog handle **old_dialog**. If **old_dialog** is a complex dialog, and **flag** is set to **DI_THISDI** then only the first dialog in the chain is copied. Otherwise the entire dialog is copied.

dialog_free returns the space allocated by the dialog **dialog**. If **flag** is set to **DI_THISDI** then only the first dialog of the chain is freed, otherwise the entire chain is freed.

dialog_next returns a handle to the next dialog in the chain. *dialog_append* appends **list_tail** to the end of the dialog chain specified by **list_head** and returns a handle to the beginning of the list. Note that neither dialog list is copied so a subsequent call to *dialog_free* of either **list_head** or **list_tail** will actually free the dialogs in the chain, and a dangling pointer will result.

dialog_delete will delete (and free) a dialog, **dialog_to_delete** from a dialog list, **list**. A handle is returned to the beginning of resulting list, or NULL if **dialog_to_delete** was the only dialog in the list.

DIALOG TYPES

```
DI_IVAL     /* Integer evaluator */
DI_FVAL     /* Float */
DI_TEXT     /* Text string */
DI_SELECT   /* Selection list, return value is list of indexes */
DI_BOOL     /* A true or false */
DI_YESNO    /* Same as above, but yes/no */
DI_NOTE     /* No return value */
```

DIAGNOSTICS

Functions which return a dialog handle return NULL on error.

SEE ALSO

dialog_text(3), dialog_value(3)

NAME

dialog_from_char, dialog_to_char – MMWB functions for converting dialogs to and from character representations

SYNOPSIS

```
#include <mmwb/dialog.h>

dialog = dialog_from_char(text);
        DIALOG      dialog;
        char    *text;

text = dialog_to_char(dialog);
        DIALOG      dialog;
        char    *text;
```

DESCRIPTION

These routines convert dialogs from their internal structure to a more portable character representation. The intended use is for communications protocols where is it desirable to send a dialog to another program in a machine independent manner.

dialog_from_char takes as its argument a text string, **text**, and returns the dialog which has been build from that string, or **NULL** if there is a format error with the string.

dialog_to_char is the inverse, it accepts a dialog and returns the character equivilent in the **text** argument. The space for the text is allocated by *dialog_to_char* and should be freed by the caller.

DIAGNOSTICS

Functions which return a dialog handle, or a character pointer return NULL on error.

SEE ALSO

dialog(3), dialog_value(3)

NAME

dialog_form_sellist – MMWB function for converting string array into selection list string

SYNOPSIS

```
#include <mmwb/dialog.h>

char *dialog_form_sellist(count, strings);
        int     count;
        char    *strings[];
```

DESCRIPTION

This routine takes an array of *count* strings and returns a string that can be passed as the SELLIST field of a selection list dialog. This string, when tokenized, will yield back the original array of strings.

SEE ALSO

tokenize(3)

NAME

dialog_semtype, dialog_type, dialog_source, dialog_dest, dialog_id, dialog_name, dialog_prompt, dialog_get_semantics, dialog_set_semantics, dialog_set_dest, dialog_set_source, dialog_set_id, dialog_get_text, dialog_set_text, dialog_get_int, dialog_set_int, dialog_get_float, dialog_set_float, dialog_get_bool, dialog_set_boot, dialog_get_flag, dialog_set_flag – MMWB functions for reading and setting dialog values

SYNOPSIS

```
#include <mmwb/dialog.h>

type = dialog_type(dialog);
        int     type;
        DIALOG  dialog;

semantic_type = dialog_semtype(dialog);
        int     semantic_type;
        DIALOG  dialog;

source = dialog_source(dialog);
        int     source;
        DIALOG  dialog;

destination = dialog_dest(dialog);
        int     destination;
        DIALOG  dialog;

id = dialog_id(dialog);
        int     id;
        DIALOG  dialog;

name = dialog_name(dialog);
        char    *name;
        DIALOG  dialog;

prompt = dialog_prompt(dialog);
        char    *prompt;
        DIALOG  dialog;

semtype = dialog_get_semantics(dialog,semopt,success)
        int     semtype;
        DIALOG  dialog;
        char    **semopt;
        int     *success;

success = dialog_set_semantics(dialog,semtype,semopt)
        int     success;
        DIALOG  dialog;
        int     semtype;
        char    *semopt;

success = dialog_set_dest(dialog, destination);
        int     success;
        DIALOG  dialog;
        int     destination;

success = dialog_set_source(dialog, source);
        int     success;
        DIALOG  dialog;
        int     source;
```

```
success = dialog_set_id(dialog, id);
        int     success;
        DIALOG      dialog;
        int     id;

result = dialog_get_text(dialog, element, success)
        char    *result;
        DIALOG      dialog;
        int     element;
        int     *success;

success = dialog_set_text(dialog, element, text)
        int     success;
        DIALOG      dialog;
        int     element;
        char    *text;

result = dialog_get_int(dialog, element, success)
        int     result;
        DIALOG      dialog;
        int     element;
        int     *success;

success = dialog_set_int(dialog, element, value)
        int     success;
        DIALOG      dialog;
        int     element;
        int     value;

result = dialog_get_float(dialog, element, success)
        double  result;
        DIALOG      dialog;
        int     element;
        int     *success;

success = dialog_set_float(dialog, element, value)
        int     success;
        DIALOG      dialog;
        int     element;
        float   value;

result = dialog_get_bool(dialog, element, success)
        int     result;
        DIALOG      dialog;
        int     element;
        int     *success;

success = dialog_set_bool(dialog, element, yesno)
        int     success;
        DIALOG      dialog;
        int     element;
        int     yesno;

result = dialog_get_flag(dialog, flag)
        int     result;
        DIALOG      dialog;
        int     flag;
```

```
success = dialog_set_flag(dialog, flag, value)
        int     success;
        DIALOG      dialog;
        int     flag;
        int     value;

result = dialog_isnt_zero(dialog, element)
        int   result;
        DIALOG dialog;
        int   element;
```

DESCRIPTION

dialog_type, *dialog_semtype*, *dialog_source*, *dialog_dest*,*and dialog_id* return the **type**, **semantic type**, **source**, **destination**, and **identifier** of the dialog, respectively. Each of these functions return an integer value. *dialog_name*, and *dialog_prompt* return the name and prompt string associated with the dialog passed as an argument.

*dialog_get_semantics*and *dialog_set_semantics* return and set the variables associated with semantic extensions to the dialog. These variables are the **semantic type** and the **semantic option**. The semantic type is used by a cooperating protocol and instrument to infer some higher semantic meaning associated with the **result** of the dialog. For example, a dialog of type **int** may have a semantic type of **atom number**. In this fashion, the instrument could allow the user to pick the desired atom off the display. In this example, the semantic option field may contain the name of the molecule. Note that *dialog_get_semantics* returns the semantic type. Errors are returned through the **success** variable.

dialog_set_dest, *dialog_set_source*, and *dialog_set_id* set the **destination**, **source**, and **identifier** associated with the dialog **dialog**.

dialog_get_text, and *dialog_set_text* allow the caller to extract and set text values from the dialog **dialog**. The argument **element** contains the element number to either set or return.

dialog_get_int, and *dialog_set_int* allow the caller to extract and set integer values from the dialog **dialog**. The argument **element** contains the element number to either set or return.

dialog_get_float, and *dialog_set_float* allow the caller to extract and set floating point values from the dialog **dialog**. The argument **element** contains the element number to either set or return.

dialog_get_bool, and *dialog_set_bool* allow the caller to extract and set boolean values from the dialog **dialog**. The argument **element** contains the element number to either set or return.

dialog_get_flag, and *dialog_set_flag* allow the caller to test and set flags in the dialog **dialog**. *dialog_set_flag* takes two arguments in addition to the dialog handle, the **flag** to change and whether to set the flag or clear it. If **value** is 1 the flag is set, if it is 0 the flag is cleared.

dialog_isnt_zero tests whether the specified element has a non-zero value.

ELEMENT TYPES

```
DI_V_NAME    /* The dialog name */
DI_V_PROMPT/* The dialog prompt */
DI_V_LBOUND/* The lower bounds */
DI_V_UBOUND/* The upper bounds */
DI_V_RESULT /* The result */
DI_V_SEMOPT/* The semantic option */
DI_V_HELP    /* The help text */
DI_V_SELLIST/* A selection list */
DI_V_OPTION /* The option sheet number */
DI_V_TOOL    /* The tool name */
DI_V_SOURCE/* The source of the dialog */
DI_V_TYPE    /* The dialog type */
DI_V_SEMTYPE        /* The dialog semantic type */
```

DIAGNOSTICS

dialog_set functions return 0 on success, -1 on error. *dialog_get* functions return 0 in the success argument (if provided) on success, -1 on error.

SEE ALSO

dialog_text(3), dialog(3)

Appendix D

FTD Grammar

The YACC [14] grammar for an FTD ruleset appears below. In order to simplify the grammar, the C action calls have been removed.

```
fti:     rules

rules:   rules rule
       | rule

rule:    name conditions execution postcondition SEMICOLON
       | name conditions execution SEMICOLON

name:    fti_name COLON

conditions
       : precondition pattern
       | pattern

execution
       : action dialog

precondition
       : expr QUESTION

pattern: QUOTED_TEXT IMPLIES

action: func_name COMMA
       | func_name LEFTBRACKET flag_expr RIGHTBRACKET COMMA
       | LEFTBRACKET flag_expr RIGHTBRACKET COMMA
       |

postcondition
       : postcondition COMMA assignment
       | assignment

assignment
       : var_name EQUALS expr

var_name: IDENTIFIER

fti_name: IDENTIFIER

func_name: IDENTIFIER

expr:  LEFTPAREN expr RIGHTPAREN
       | INTEGER
       | var_name
       | bool_expr     %prec NOT
       | arith_expr    %prec UMINUS
       | rel_expr      %prec AND

bool_expr: expr B_AND expr
       | expr B_OR expr
       | NOT expr

rel_expr
       : expr AND expr
       | expr OR expr
       | expr GT expr
       | expr LT expr
       | expr GE expr
       | expr LE expr
       | expr R_EQUAL expr
       | expr NOT_EQUAL expr

arith_expr
       : expr ADD expr
       | expr SUBTRACT expr
       | expr MULTIPLY expr
```

```
        | expr DEVIDE expr
        | SUBTRACT expr         %prec UMINUS

rule_flag: F_QUIT
        | F_OPTION
        | F_SEND
        | F_CPYPRMPT

flag_expr:      rule_flag
        | flag_expr B_AND flag_expr
        | flag_expr B_OR flag_expr
        | NOT flag_expr

dialog  : LEFTBRACE dialog_statement_list RIGHTBRACE
        | LEFTBRACE dialog_list RIGHTBRACE

dialog_list
        : LEFTBRACE dialog_statement_list RIGHTBRACE
        | dialog_list LEFTBRACE dialog_statement_list RIGHTBRACE

dialog_statement_list
        : dialog_statement_list dialog_statement SEMICOLON
        | dialog_statement_list error SEMICOLON
        |

dialog_statement
        : K_INTEGER opt_integer_range opt_default_integer
        | K_REAL opt_real_range opt_default_real
        | K_TEXT opt_integer_range opt_default_text
        | select_from_head RIGHTBRACE opt_default_integer
        | K_MULTI select_from_head RIGHTBRACE opt_integer_range opt_default_int_list
        | K_BOOLEAN opt_default_boolean
        | K_YESNO opt_default_yesno
        | K_NOTIFIER
        | K_LABEL ident_or_string
        | K_PROMPT QUOTED_TEXT
        | K_HELP QUOTED_TEXT
        | K_SEMANTIC ident_or_string opt_option_string
        | flag_list

select_from_head
        : K_SELECT K_FROM LEFTBRACE QUOTED_TEXT
        | select_from_head COMMA QUOTED_TEXT

flag_list
        : K_FLAGS ident
        | flag_list COMMA ident

opt_integer_range
        : INTEGER DOTDOT INTEGER
        |

opt_default_integer
        : K_DEFAULT INTEGER
        |

opt_real_range
        : real DOTDOT real
        |

opt_default_real
        : K_DEFAULT real
        |

opt_default_text
```

```
        : K_DEFAULT QUOTED_TEXT
        |

opt_default_boolean
        : K_DEFAULT K_TRUE
        | K_DEFAULT K_FALSE
        |

opt_default_yesno
        : K_DEFAULT K_YES
        | K_DEFAULT K_NO
        |

opt_option_string
        : K_OPTIONS QUOTED_TEXT
        |

opt_default_int_list
        :       K_DEFAULT int_list
        |

int_list
        :       INTEGER
        |       int_list COMMA INTEGER

real    : REAL
        | INTEGER

ident_or_string
        : ident
        | QUOTED_TEXT

ident   : IDENTIFIER
        | K_INTEGER
        | K_REAL
        | K_TEXT
        | K_MULTI
        | K_SELECT
        | K_FROM
        | K_BOOLEAN
        | K_YESNO
        | K_NOTIFIER
        | K_LABEL
        | K_FLAGS
        | K_PROMPT
        | K_HELP
        | K_SEMANTIC
        | K_DEFAULT
        | K_OPTIONS
        | K_TRUE
        | K_FALSE
        | K_YES
        | K_NO
```

Appendix E

Source Listings

The sources for the initial implementations of the Session Manager, both versions of the Foreign Tool Driver, twb, the dialog library, the session manager library, and the foreign tool driver library are included within this appendix.

sm/client.h

```
/*
 * $Id: client.h,v 1.12 89/10/25 14:29:53 gregc Exp $
 *
 * client.h:
 *      definitions for the Session Manager clients.  Note that
 *      SM clients come in two flavors: Tools, and User Interface
 *      Managers (UIMs).  These two clients have two different
 *      protocols, but share the client structure.
 *
 * $Log:        client.h,v $
 * Revision 1.12  89/10/25  14:29:53  gregc
 * add extern for cl_find().
 *
 * Revision 1.11  89/08/07  12:57:17  scooter
 * Changed $Header: /usr.MC68020/src/local/mmwb/src/platform/sm/RCS/client.h,v 1.
 *
 * Revision 1.10  89/07/31  11:06:11  scooter
 * Changed SM_PROCESS to SM_PROTOCOL
 *
 * Revision 1.9  89/07/19  11:00:41  scooter
 * Replace cl_rdunlock with CL_READABLE to be more consistent with
 * CL_WRITEABLE.
 *
 * Revision 1.8  89/07/19  10:54:05  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.7  89/03/01  17:58:19  pett
 * uses UIDTYPE/GIDTYPE
 *
 * Revision 1.6  89/02/21  23:39:28  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.6  89/02/21  23:38:12  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.5  88/11/21  11:35:13  scooter
 * Modified sm and libsm to understand the differences between
 * processes and instruments
 *
 * Revision 1.4  88/09/19  17:11:10  conrad
 * Use new dispatcher library interface
 *
 * Revision 1.3  88/09/01  16:57:57  scooter
 * Cleaned up DIALOG protocol somewhat:  added READ_DIALOG and
 * READ_REPLY commands.  Also modified tool launching to include
 * the project IDs.
 *
 * Revision 1.2  88/08/29  20:26:30  scooter
 * Added write locking to clients and finished protocol
 * by adding deliv and external state switch to
 * SM_SENTD.
 *
 * Revision 1.1  88/08/25  11:50:49  scooter
 * Initial revision
 *
 *
 */

#ifndef  CLIENT_INCLUDE
#define  CLIENT_INCLUDE

#include        "smdefs.h"
#include        "protocol.h"
#include        "dispatcher.h"
#include        "tool.h"

#define  COMMAND_LENGTH 255
#define  MAXTOKENS      20

typedef  struct    dlist {
         DIALOG    dl_dialog;
         int       dl_type;
         int       dl_status;
         struct    dlist     *dl_next;
} DLIST;

/*
 * Dialog types
 */
#define  DL_DIALOG      0
#define  DL_REPLY       1
#define  DL_OPTION      2

/*
 * Dialog status
 */
#define  DL_WAITING     0
#define  DL_DELIVERED   1
#define  DL_PENDING     2

#define  DL_NULL(DLIST *)NULL

typedef  struct    client {
```

sm/client.h

```
        DISPATCHER_SERVER   cl_server;          /* Damn idiot name.  Than
                                                   Eric, and Leslie */
        int         cl_id;          /* The tool identifier */
        int         cl_fd;          /* The client file descriptor */
        int         cl_state;       /* The protocol state */
        int         cl_type;        /* The client type (0 = UI) */
        char        *cl_name;       /* The name of the client */
        UIDTYPE     cl_uid;         /* The uid of the user */
        GIDTYPE     cl_gid;         /* The gid of the user */
        DLIST       *cl_dqueue;     /* The dialog queue for this client */
        DIALOG      cl_current;     /* An area to collect a dialog */
        int         cl_flags;       /* For write locking */
        char        cl_cbuf[COMMAND_LENGTH];        /* The input buffer */
        int         cl_indx;        /* Index into the command buffer */
        STATE       *cl_protocol;   /* The protocol for this client */
        SM_TOOLDES  *cl_tooldes;    /* The tool description for this client */
        SM_TOOLTREE *cl_tooltree;   /* The tool tree */
        char        *cl_system;     /* The system project for this client */
        char        *cl_home;       /* The home project for this client */
        char        *cl_curr;       /* The current project for this client */
        struct      client      *cl_next;/* Pointer to the next client */
        struct      client      *cl_child;/* Pointer to the first child */
} CLIENT;

#define  CL_NULL(CLIENT *)NULL
#define  CL_FILENO(c)       (c)->cl_fd
#define  CL_UIM             0
#define  CL_PROTOCOL        1
#define  CL_INSTRUMENT      2
#define  CL_WRITEABLE(c)    FD_SET ( (c)->cl_fd, &sm_writefds )
#define  CL_READABLE(c)     FD_SET ( (c)->cl_fd, &sm_readfds )
#define  CL_WRLOCK          0x1000  /* Write is disabled */
#define  CL_WAITING         0x100   /* Client is awaiting input */
#define  CL_OPWAIT          0x10    /* Client is awaiting option sheet */
#define  CL_WRITESET        1       /* Write bit in fd is set */

extern   CLIENT  *cl_new();
extern   CLIENT  *cl_find();
extern   void    cl_quit();
extern   void    cl_parse();
extern   void    cl_process();
extern   void    cl_notify();

extern   CLIENT  *cl_newuim();
extern   CLIENT  *cl_newtool();
extern   CLIENT  *cl_oldtool();

extern   CLIENT  *clist;

extern   void    cl_greet();
extern   void    cl_free();
extern   void    cl_input();
extern   void    cl_output();
extern   void    cl_addqueue();
extern   void    cl_writelock();
extern   void    cl_wrunlock();
extern   void    cl_rdunlock();

#endif   CLIENT_INCLUDE
```

sm/config.h

```
/*
 * $Id: config.h,v 1.4 89/08/07 12:57:45 scooter Exp Locker: scooter $
 * $Log:        config.h,v $
 * Revision 1.4  89/08/07  12:57:45  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.3  88/09/15  23:06:18  conrad
 * Use machine dependent header file instead of ifdefs
 *
 * Revision 1.2  88/08/28  20:37:00  scooter
 * Efficiency modifications - buffered writes.
 *
 * Revision 1.1  88/08/25  11:51:04  scooter
 * Initial revision
 *
 * Revision 1.1  88/07/25  13:15:20  conrad
 * Initial revision
 *
 */

#ifndef  CONFIG_SM

#include <stdio.h>
#include <syslog.h>
#include <sys/types.h>
#include <machdep.h>

#define  SM_LOGFILE      "/usr/tmp/sm.log"

#define  BPB     8       /* Bits Per Byte */

#define  IO_SIZE  1023
#define  BUFLEN  (IO_SIZE + 1)

#ifndef  TRUE
#define  TRUE    1
#define  FALSE   0
#endif

#define  CONFIG_SM
#endif
```

sm/error.h

```
/*
 * $Id: error.h,v 1.2 89/08/07 12:57:49 scooter Exp Locker: scooter $
 * $Log:		error.h,v $
 * Revision 1.2  89/08/07  12:57:49  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.1  88/08/25  11:51:13  scooter
 * Initial revision
 *
 * Revision 1.1  88/07/25  13:15:26  conrad
 * Initial revision
 *
 */

#ifndef	ERROR_INCLUDE

#include "config.h"
#include "client.h"

/*
 * List of access functions
 */
void	error_internal();		/* Internal server error */
void	error_handshake();		/* Protocol handshake error */
void	error_reply();			/* Reply with an error */

#define	ERROR_INCLUDE
#endif
```

sm/protocol.h

```
/*
 * $Id: protocol.h,v 1.4 89/08/07 12:52:31 scooter Exp $
 *
 * protocol.h - the definitions for the state transition handler
 *
 * $Log:                protocol.h,v $
 * Revision 1.4  89/08/07  12:52:31  scooter
 * Added new state to the tool protocol: SM_TYPE, which
 * identifies the tool type: INSTRUMENT or PROTOCOL
 *
 * Revision 1.3  89/07/19  10:54:38  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.2  88/09/01  16:57:53  scooter
 * Cleaned up DIALOG protocol somewhat:  added READ_DIALOG and
 * READ_REPLY commands.  Also modified tool launching to include
 * the project IDs.
 *
 * Revision 1.1  88/08/25  11:51:22  scooter
 * Initial revision
 *
 *
 */

#ifndef   PROTOCOL_INCLUDE

#include          <stdio.h>

#define   MAXTRANS        10

typedef   struct    {
          char      *token;
          int       next_state;
          int       (*action)();
} TRANSITION;

#define   TR_NULL(TRANSITION *)NULL;

typedef struct      {
          int       state;
          TRANSITION        trans[MAXTRANS];
} STATE;

#define   ST_NULL(STATE *)NULL;

/*
 * Handshaking States
 */

#define   SM_VERS 0
#define   SM_HELO 1
#define   SM_USER 2

/*
 * UIM States
 */

#define   SM_UIM  3
#define   SM_SENTD 4
#define   SM_REPLY 5
#define   SM_GETOPT 6

/*
 * Tool States
 */

#define   SM_TYPE 0
#define   SM_PROJ 1
#define   SM_TOOL 5
#define   SM_DIALOG 6
#define   SM_WAIT 7

#define   ENDSTATE -1

#define   PROTOCOL_INCLUDE
#endif PROTOCOL_INCLUDE
```

sm/reply.h

```
/*
 * $Id: reply.h,v 1.4 89/08/07 12:52:01 scooter Exp $
 *
 * $Log:                reply.h,v $
 * Revision 1.4  89/08/07  12:52:01  scooter
 * Added RB_NOUIM to the reply status
 *
 * Revision 1.3  88/09/15  23:52:14  scooter
 * Added routine cl_quituim and cleaned up termination code
 * for UIMs to allow an interrupt or a close to act as if
 * the UIM had sent a QUIT.
 *
 * Revision 1.2  88/08/28  20:36:58  scooter
 * Efficiency modifications - buffered writes.
 *
 * Revision 1.1  88/08/25  11:51:37  scooter
 * Initial revision
 *
 * Revision 1.1  88/07/25  13:15:30  conrad
 * Initial revision
 *
 */

#ifndef REPLY_INCLUDE

#include "config.h"
#include <sys/uio.h>

#define  MACHINE_TYPE   "vax"

#define  REPLY_IN_ASCII  0
#define  REPLY_IN_BINARY1

#define  REPLY_LENGTH   250

#define  RS_INFO         1
#define  RS_ACCEPT       2
#define  RS_MORE         3
#define  RS_FAILED       4
#define  RS_REJECT       5
#define  RS_END          6

#define  RT_OTHER        0
#define  RT_DM           1
#define  RT_CS           2
#define  RT_SM           3
#define  RT_EXTENSION    8
#define  RT_DEBUG        9

#define  RG_OKAY         0       /* Everything OK */
#define  RG_OKAY2        1       /* The same, but different */

#define  RB_UNKNOWN      0       /* Unknown error */
#define  RB_PROT         1       /* Protocol error */
#define  RB_SYNTAX       2       /* Syntax error (in tool or dialog) */
#define  RB_SERVER       3       /* Server error */
#define  RB_HANDSHAKE    4       /* Handshaking error (byel) */
#define  RB_NOSUCH       5       /* No such dialog or tool */
#define  RB_NOUIM        6       /* No uim is present */
#define  RB_NOTIMP       9       /* Not implemented (yet) */

/*
 * Access functions
 */
void    reply();                        /* Send a reply to a file descriptor */
void    reply_end();                    /* Send end-of-reply message */
void    reply_begin_ASCII();            /* Begin sending ASCII data */
void    reply_ASCII();                  /* Send an ASCII string */
void    reply_end_ASCII();              /* End sending ASCII data */
void    reply_noperm();                 /* Send Permission Denied message */
void    reply_data();                   /* Send data to client */
void    reply__write();                 /* "write" data to buffer */
void    reply__putchar();               /* "write" char to buffer */

#define  REPLY_INCLUDE
#endif
```

sm/sm.h

```
/*
 * $Id: sm.h,v 1.4 89/08/07 12:57:42 scooter Exp Locker: scooter $
 *
 * sm.h - include file for sm server
 *
 * $Log:        sm.h,v $
 * Revision 1.4  89/08/07  12:57:42  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.3  89/07/19  10:54:45  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.2  88/09/15  23:06:17  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
 * Revision 1.1  88/08/25  11:51:45  scooter
 * Initial revision
 *
 *
 */

#include        <stdio.h>
#include        <sys/types.h>
#include        "config.h"

/*
 * Globals
 */

extern  char    *sm_user;
extern  char    *sm_cid;

extern  int     db_flag;
extern  int     sm_count;
extern  int     ui_fd;
extern  int     dilast;
extern  fd_set  sm_readfds,sm_writefds;
extern  void    sm_writestate();
extern  void    sm_deletestate();
```

sm/smdefs.h

```
/*
 * $Id: smdefs.h,v 1.5 89/08/07 12:57:41 scooter Exp Locker: scooter $
 *
 * This is the include file for the MMWB Session Manager, sm, and
 * all of the sm libraries.
 *
 * $Log:            smdefs.h,v $
 * Revision 1.5  89/08/07  12:57:41  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.4  89/07/19  10:54:48  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.3  88/11/21  11:35:32  scooter
 * Modified sm and libsm to understand the differences between
 * processes and instruments
 *
 * Revision 1.2  88/08/29  20:29:06  scooter
 * Removed location from tool description
 *
 * Revision 1.1  88/08/25  11:51:56  scooter
 * Initial revision
 *
 *
 */

#ifndef  SMDEFS_INCLUDE                    /* Avoid multiple includes */

#include          "dialog.h"

/*
 * Typedefs
 */

typedef  int      SM_TOOL;        /* A "tool" is just an accession number */

typedef  struct   sm_session_struct {
         char     *ss_name;       /* The name of the tool */
         SM_TOOL  ss_tool;        /* The ID of the tool */
         int      ss_state;       /* The state of the tool */
         struct   sm_session_struct          *ss_next;          /* Next tool */
} SM_SESSION_STATUS;

typedef  struct   {
         char     *ses_cid;       /* The name (Conversation ID) of the session
         int      ses_date;       /* The date and time of the session */
         int      ses_sock;       /* The socket to rendezvous */
} SM_SESSION;

typedef struct {
         char     *Name;          /* Name of the tool */
         char     *Version;       /* Version number */
         int      nodisplay;      /* Display options ? */
         DIALOG   Options;        /* The option sheet */
         char     *Subtools;      /* Subtools (if any) */
         int      OnError;        /* If subtools, how to handle errors */
         char     *Host;          /* The hosts to execute */
         int      Type;           /* The type of tool: 1 = PROCESS,
                                                       2 = INSTRUMENT */
         SM_TOOL  tool_id;        /* The resulting tool id */
}        SM_TOOLDES;

/*
 * Flags for Tool Descriptions - OnError
 */

#define  SM_IGNORE 1     /* Ignore errors */
#define  SM_DIE    2     /* Immediately die */
#define  SM_KILLB 3      /* Kill branches */


#define  SMDEFS_INCLUDE
#endif   SMDEFS_INCLUDE
```

sm/tool.h

```
/*
 * $Id: tool.h,v 1.3 89/08/07 12:57:39 scooter Exp Locker: scooter $
 *
 * $Log:        tool.h,v $
 * Revision 1.3  89/08/07  12:57:39  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.2  89/07/19  10:54:53  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.1  88/08/25  11:52:16  scooter
 * Initial revision
 *
 *
 */

#ifndef  INCLUDE_TOOL
#define  INCLUDE_TOOL

typedef struct      tooltree {
        char        *tt_tool;           /* The tool to launch */
        char        *tt_host;           /* The hosts it runs on */
        int         tt_tnum;            /* The tool number */
        int         tt_join;            /* The join count if this is a join point */
        struct client *tt_client; /* A back_pointer to the client struct */
        struct      tooltree *tt_next;          /* The next tool */
        struct      tooltree *tt_fork;          /* A parrallel tool */
} SM_TOOLTREE;

SM_TOOLDES      *tooldes_create();
void            tooldes_free();
void            tool_free();
struct client   *tool_launch();     /* Initial tool launch */
struct client   *tool_steplaunch();
int             tool_ismulti();
DIALOG          tool_myoption();

#endif  INCLUDE_TOOL
```

sm/client.c

```
#ifndef lint
static char *RCSid = "$Id: client.c,v 1.29 90/02/20 17:14:54 scooter Exp $";
#endif lint

/*
 * client.c -		this modules contains all of the client code which
 *			is independent of the client type.
 *
 * $Log:		client.c,v $
 * Revision 1.29  90/02/20  17:14:54  scooter
 * Fixed bug in the handling of dialogs to allow dialogs to be
 * retransmitted if the DIALOG command was missed.
 *
 * Revision 1.28  90/02/16  10:42:40  scooter
 * Fixed bugs found during performance testing
 *
 * Revision 1.27  89/10/25  14:31:04  gregc
 * use mmwbmisc.h, ansi-C-ize.
 *
 * Revision 1.26  89/08/07  12:54:17  scooter
 * Added named sessions support in cl_quituim
 *
 * Revision 1.25  89/07/31  11:06:20  scooter
 * Changed SM_PROCESS to SM_PROTOCOL
 *
 * Revision 1.24  89/07/25  16:29:41  scooter
 * Changed cl_oldtool to get the tool type from the tool description
 *
 * Revision 1.23  89/07/24  23:45:33  scooter
 * Fixed problems with forked tools joining back prematurely
 *
 * Revision 1.22  89/07/24  15:15:12  scooter
 * Fixed bugs in fork implementation
 *
 * Revision 1.21  89/07/19  11:01:16  scooter
 * Replace cl_rdunlock with CL_READABLE to be more consistent with
 * CL_WRITEABLE.
 *
 * Revision 1.20  89/07/19  10:54:55  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.19  89/03/10  14:22:00  scooter
 * Various bug fixes to get everything working again
 *
 * Revision 1.18  89/03/02  22:21:25  pett
 * declared emalloc to return void * and used strchr instead of index
 *
 * Revision 1.17  89/02/21  23:39:39  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.17  89/02/21  23:38:20  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.16  88/11/21  11:35:35  scooter
 * Modified sm and libsm to understand the differences between
 * processes and instruments
 *
 * Revision 1.15  88/10/24  13:11:54  conrad
 * multiple dialogs sent towards the UIM are properly queued
 *
 * Revision 1.14  88/10/22  16:46:52  conrad
 * Release allocated string space
 *
 * Revision 1.13  88/10/17  14:27:01  conrad
 * Make multiple dialog delivery work
 *
 * Revision 1.12  88/10/13  11:39:29  scooter
 * Changed size of msg_buf to reflect DI_BUFLEN
 *
 * Revision 1.11  88/10/13  11:33:45  scooter
 * Modified to use emalloc insted of malloc
 *
 * Revision 1.10  88/10/12  17:27:33  conrad
 * Don't let UIM respond to unack'ed dialogs
 *
 * Revision 1.9  88/10/12  16:24:00  conrad
 * Properly ifdef state logging
 *
 * Revision 1.8  88/10/11  13:52:54  conrad
 * ifdef out state-change logging
 *
 * Revision 1.7  88/09/19  17:11:17  conrad
 * Use new dispatcher library interface
 *
 * Revision 1.6  88/09/15  23:53:41  scooter
 * Added routine cl_quituim and cleaned up termination code
 * for UIMs to allow an interrupt or a close to act as if
 * the UIM had sent a QUIT.
 *
 * Revision 1.5  88/09/15  23:08:11  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
```

sm/client.c

```
 * Revision 1.4  88/09/01  16:57:55  scooter
 * Cleaned up DIALOG protocol somewhat: added READ_DIALOG and
 * READ_REPLY commands.  Also modified tool launching to include
 * the project IDs.
 *
 * Revision 1.3  88/08/29  20:26:26  scooter
 * Added write locking to clients and finished protocol
 * by adding deliv and external state switch to
 * SM_SENTD.
 *
 * Revision 1.2  88/08/28  20:36:51  scooter
 * Efficiency modifications - buffered writes.
 *
 * Revision 1.1  88/08/25  11:53:16  scooter
 * Initial revision
 *
 *
 */

#include        "sm.h"
#include        "client.h"
#include        "dialog.h"
#include        "reply.h"
#include        "error.h"
#include        "mmwbmisc.h"
#include        <syslog.h>
#include        <sys/socket.h>
#include        <netinet/in.h>

CLIENT          *clist = CL_NULL;
void            cl_addqueue();
void            dl_free();
extern  void    *emalloc();
extern  char    *strchr();
extern  char    *strncpy();
static  char    msg_buf[DI_BUFLEN+16];  /* Enough for 1 dialog + command */

/*
 * cl_new:
 *      allocate a new client
 */

CLIENT *
cl_new (server, fd, type, protocol)
DISPATCHER_SERVER       server;
int     fd;
int     type;
STATE   *protocol;
{
        CLIENT *cl = (CLIENT *)emalloc(sizeof (CLIENT));

        cl->cl_server = server;
        cl->cl_fd = fd;
        cl->cl_state = 0;                       /* Start State */
        cl->cl_type = type;
        cl->cl_protocol = protocol;
        cl->cl_indx = 0;
        return cl;
}

/*
 * cl_free:
 *      remove a client from the list
 *
 */

void
cl_free (cl)
CLIENT  *cl;
{
        register CLIENT     *clp, *prev;

        /*
         * OK, there are no dialogs, so just remove the
         * structure.
         */

        prev = CL_NULL;

        mmwb_stringfree(cl->cl_name);
        mmwb_stringfree(cl->cl_system);
        mmwb_stringfree(cl->cl_home);
        mmwb_stringfree(cl->cl_curr);

        for (clp = clist; clp != NULL; clp = clp->cl_next)
        {
                if (clp == cl)
                {
                        if (prev == CL_NULL)
                                clist = clp->cl_next;
                        else
                                prev->cl_next = clp->cl_next;
                        if (cl->cl_server != NULL)
                                (void) dispatcher_server_close(cl->cl_server);
```

```
                else
                        (void) close(cl->cl_fd);
                free((char *)cl);
                return;
        } else
                prev = clp;
    }
    return;
}

/*
 * cl_input:
 *      read the client's data.  Note that no parsing
 *      or processing is done until a carraige return
 *      is seen at the input.
 *
 */

void
cl_input(cl)
CLIENT  *cl;
{
    int     n;
    void    cl_quituim();

    n = read(cl->cl_fd, &cl->cl_cbuf[cl->cl_indx],
            COMMAND_LENGTH - cl->cl_indx);

    /*
     * Hack city!  If we get any response from the UIM, then
     * we assume that it is now ready for more dialog handling
     */
    if (cl->cl_type == CL_UIM && cl->cl_state >= SM_UIM) {
        cl_wrunlock(cl);
        cl->cl_flags |= CL_WAITING;
    }

    if (n <= 0) {
        syslog(LOG_ERR, "read error from client %s -- %m", cl->cl_name
        if (cl->cl_type == CL_UIM)
            cl_quituim(cl,(char *)NULL);
        else
            cl_quit(cl,1);
        return;
    }

    cl->cl_indx += n;

    if (strchr(cl->cl_cbuf,'\n'))                   /* Newline? */
    {
        cl->cl_cbuf[cl->cl_indx] = '\0';
        cl_process(cl);             /* Yes, process the data */
        cl->cl_indx = 0;
    }

    return;
}

/*
 * cl_output:
 *      send data (almost always a dialog) to a client
 */

void
cl_output (cl)
CLIENT  *cl;
{
    DLIST       *dl = cl->cl_dqueue;
    static void dl_sendd();
    static void dl_respd();

    switch (cl->cl_type) {

    case    CL_UIM:
        /*
         * Send one dialog at a time, waiting for success
         * report from UIM
         */

        while (dl != NULL) {
            if (dl->dl_status == DL_WAITING)
                break;
            if (dl->dl_status == DL_PENDING &&
              cl->cl_state != SM_SENTD)
                break;
            dl = dl->dl_next;
        }
        if (dl == NULL)
            return;             /* Should not happen */
        dl_sendd(cl, dl);
#ifdef LOG_STATE_CHANGE
        syslog(LOG_DEBUG, "%d: State %d -> %d",cl->cl_id,cl->cl_state,
            SM_SENTD);
#endif
```

```
                cl->cl_flags &= ~CL_WAITING;
                cl->cl_state = SM_SENTD;
                dl->dl_status = DL_PENDING;
                cl_writelock(cl);
                return;

        case    CL_PROTOCOL:
        case    CL_INSTRUMENT:

                /*
                 * Check for queued dialogs
                 */
                while (dl != DL_NULL) {
                        dl_respd(cl,dl);
                        (void) cl_dequeue (cl, dl->dl_dialog);
                        dl = dl->dl_next;
                }

                /*
                 * Remove the fd from the write list
                 */
                FD_CLR (cl->cl_fd, &sm_writefds);
                cl->cl_flags &= ~CL_WAITING;
                return;
        }
}


/*
 * cl_process:
 *      process the data for a client.  This involves tokenizing
 *      the input stream, then calling the appropriate protocol
 *      handler.
 *
 */

void
cl_process(cl)
CLIENT  *cl;
{
        char    *tokens[MAXTOKENS];
        int     nargs, n;

        /*
         * Tokenize the input
         */
        nargs = tokenize(cl->cl_cbuf, tokens, MAXTOKENS);

        if (nargs < 0)
        {
                if (nargs == -1)
                        syslog(LOG_INFO,"trailing garbage ignored");
                else {
                        error_reply(cl, "tokenize failure", RB_PROT);
                        error_handshake(cl);
                        return;
                }
        }

        tokens[nargs] = NULL;           /* Make sure */

        /*
         * Call the state engine
         */

        if ( (n = cl_trans( cl, nargs, tokens) ) > 0 )
        {
#ifdef LOG_STATE_CHANGE
                syslog(LOG_DEBUG,"%d: State %d -> %d",cl->cl_id,cl->cl_state,n);
#endif
                cl->cl_state = n;
        }
        else if (n == -1)
                error_reply(cl, "Protocol error", RB_PROT);

        return;
}


/*
 * cl_trans:
 *      a protocol engine.  Returns -1 on protocol error,
 *      0 to hold state constant, >0 change to state indicated
 *      by return value.
 */

int
cl_trans ( cl, nargs, args)
CLIENT  *cl;
int     nargs;
char    *args[];
{
    STATE   *cstate = &cl->cl_protocol[cl->cl_state];
```

```
                cl->cl_flags &= ~CL_WAITING;
                cl->cl_state = SM_SENTD;
                dl->dl_status = DL_PENDING;
                cl_writelock(cl);
                return;

        case    CL_PROTOCOL:
        case    CL_INSTRUMENT:

                /*
                 * Check for queued dialogs
                 */
                while (dl != DL_NULL) {
                        dl_respd(cl,dl);
                        (void) cl_dequeue (cl, dl->dl_dialog);
                        dl = dl->dl_next;
                }

                /*
                 * Remove the fd from the write list
                 */
                FD_CLR (cl->cl_fd, &sm_writefds);
                cl->cl_flags &= ~CL_WAITING;
                return;
        }
}


/*
 * cl_process:
 *      process the data for a client. This involves tokenizing
 *      the input stream, then calling the appropriate protocol
 *      handler.
 *
 */

void
cl_process(cl)
CLIENT  *cl;
{
        char    *tokens[MAXTOKENS];
        int     nargs, n;

        /*
         * Tokenize the input
         */
        nargs = tokenize(cl->cl_cbuf, tokens, MAXTOKENS);

        if (nargs < 0)
        {
                if (nargs == -1)
                        syslog(LOG_INFO,"trailing garbage ignored");
                else {
                        error_reply(cl, "tokenize failure", RB_PROT);
                        error_handshake(cl);
                        return;
                }
        }

        tokens[nargs] = NULL;           /* Make sure */

        /*
         * Call the state engine
         */

        if ( (n = cl_trans( cl, nargs, tokens) ) > 0 )
        {
#ifdef LOG_STATE_CHANGE
                syslog(LOG_DEBUG,"%d: State %d -> %d",cl->cl_id,cl->cl_state,n);
#endif
                cl->cl_state = n;
        }
        else if (n == -1)
                error_reply(cl, "Protocol error", RB_PROT);

        return;
}


/*
 * cl_trans:
 *      a protocol engine. Returns -1 on protocol error,
 *      0 to hold state constant, >0 change to state indicated
 *      by return value.
 */

int
cl_trans ( cl, nargs, args)
CLIENT  *cl;
int     nargs;
char    *args[];
{
    STATE   *cstate = &cl->cl_protocol[cl->cl_state];
```

```
        int     i;

        for (i = 0; i < MAXTRANS; i++)
        {
                if (strcmp(cstate->trans[i].token, args[0]) == 0)
                        return ( (cstate->trans[i].action)
                                (cl, nargs, args, cstate->trans[i].next_state) );
        }

        /*
         * Error - illegal command.
         */

                (void) sprintf (msg_buf, "Illegal command '%s'", args[0]);
                error_reply( cl, msg_buf, RB_PROT );
                return -1;
}

void
cl_greet(cl)
CLIENT  *cl;
{
        (void) sprintf(msg_buf,"\"%s\"",sm_cid);

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, msg_buf);
        reply_end(cl->cl_fd);

        return;
}

void
cl_quit(cl,errflag)
CLIENT  *cl;
int     errflag;
{
        DLIST           *dl,*d;
        SM_TOOLDES      *smtd;
        SM_TOOLTREE     *smt;

        if (cl == CL_NULL)
                return;

        smtd = cl->cl_tooldes;

        /*
         * First, close the connections to the old client
         */

        if (cl->cl_server != NULL)
                (void) dispatcher_server_close(cl->cl_server);
        else
                (void) close(cl->cl_fd);

        FD_CLR(cl->cl_fd, &sm_readfds);
        FD_CLR(cl->cl_fd, &sm_writefds);

        dialog_free(cl->cl_current,0);
        /*
         * Free all of the dialogs pending for this tool
         */
        dl = cl->cl_dqueue;
        while (dl != DL_NULL) {
                d = dl->dl_next;
                dialog_free(dl->dl_dialog,0);
                dl = d;
        }

        /*
         * Depending on whether there was an error or not,
         * handle multi-step protocols according to the
         * tooldes flags
         */

        if (tool_ismulti(cl,errflag)) {
                /*
                 * Have a multistep protocol -- check for a join and
                 * if there is one, wait until the joincount = 0
                 */
                smt = cl->cl_tooltree->tt_next;
                if (smt->tt_join != 0)
                        smt->tt_join--;
                else
                        (void) tool_steplaunch(cl,smt, cl->cl_tooldes, 0,
                                        cl->cl_system,cl->cl_home,cl->cl_curr);
        } else {
                sm_count--;
                if (smtd->OnError == SM_DIE)
                        /* Kill all children */
                        if (cl->cl_child != CL_NULL)
                                cl_quit(cl->cl_child,1);
                tooldes_free(smtd);
                cl_free(cl);
        }
        return;
```

```
}

/*
 * cl_quituim:
 *      This routine is called when a UIM sends a QUIT or when a
 *      UIM abnormally terminates.
 */

void
cl_quituim (cl,session_name)
CLIENT  *cl;
char    *session_name;
{
        register        int     fd;
        register        CLIENT  *c;
        struct  sockaddr_in     sa;
        int                     salen;
        DLIST                   *dl,*dn,*lastdl;
        DIALOG                  d;
#ifndef ntohs
        unsigned short      ntohs();
#endif

        if (clist == cl && clist->cl_next == CL_NULL)
                exit (0);

        cl_writelock(cl);
        cl->cl_flags &= ~CL_WAITING;        /* Make sure to turn off waiting */

        if ((fd = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        syslog(LOG_ERR, "socket: %m");
        exit(1);
        }
        if (listen(fd, 1) < 0) {
                syslog(LOG_ERR, "listen: %m");
                exit(1);
        }
        salen = sizeof sa;
        if (getsockname(fd , (struct sockaddr *) &sa, &salen) < 0) {
                syslog(LOG_ERR, "getsockname: %m");
                exit(1);
        }

        /*
         * NOTE: this is closed here insted of in the calling
         * routines to prevent fd#0 from getting returned by
         * the socket call above.
         * We don't use dispatcher_server_close() here because the
         * UIM is not started via the dispatcher from here.
         */

        (void) close (cl->cl_fd);

        if (dup2(fd,0) < 0) {
        syslog(LOG_ERR, "dup2: %m");
        exit(1);
        }

        (void) close (fd);

        /*
         * Kill any instruments we are running for the UIM
         */
        for (c = clist; c != CL_NULL; c = c->cl_next) {
                if (c->cl_type == CL_INSTRUMENT) {

                        /*
                         * Free any dialogs which are queued up for this
                         * instrument.  We do this because instrument
                         * dialogs are not retained across UIMs.
                         */
                        dl = cl->cl_dqueue; /* Note: this the the UIM queue */
                        lastdl = DL_NULL;
                        while (dl != DL_NULL) {
                                dn = dl->dl_next;
                                d = dl->dl_dialog;
                                if (d == NULL)
                                        continue;
                                if (dialog_source(d) == c->cl_id) {
                                        dl_free(dl);
                                        if (lastdl == DL_NULL)
                                                cl->cl_dqueue = dn;
                                        else
                                                lastdl->dl_next = dn;
                                }
                                lastdl = dl;
                        }

                        /*
                         * Kill the instrument
                         */
                        dispatcher_server_kill (c->cl_server);
                        cl_quit(c,1);
        }
```

```
        }

        /*
         * Mark all the pending dialogs as waiting so that they will
         * be retransmitted to a new UIM that connects up
         */
        for (dl = cl->cl_dqueue; dl != NULL; dl = dl->dl_next)
                dl->dl_status = DL_WAITING;

        /*
         * Write the state into the user's home project
         */
        sm_writestate(ntohs(sa.sin_port),session_name);

        syslog (LOG_DEBUG, "Session Manager waiting on port %d",
                ntohs(sa.sin_port));

        ui_fd = -1;
        sm_count--;
        return;
}

CLIENT  *
cl_newuim (fd)
int     fd;
{
        CLIENT  *c;
        extern  STATE   uim_prot[];

        if (clist == CL_NULL)
        {
                clist = cl_new ((DISPATCHER_SERVER) NULL, fd, 0, uim_prot);
        } else {
                c = clist;
                clist = cl_new ((DISPATCHER_SERVER) NULL, fd, 0, uim_prot);
                clist->cl_next = c;
        }

        clist->cl_id = 0;
        clist->cl_name = mmwb_cpystring("UIM");

        return clist;
}

static  int     lastid = 0;

CLIENT *
cl_newtool(server, td)
DISPATCHER_SERVER       server;
SM_TOOLDES      *td;
{
        CLIENT  *c;
        extern  STATE   tool_prot[];

        if (clist == CL_NULL)
        {
                clist = cl_new (server, dispatcher_server_fd(server),
                        td->Type, tool_prot);
        } else {
                c = clist;
                clist = cl_new (server, dispatcher_server_fd(server),
                        td->Type, tool_prot);
                clist->cl_next = c;
        }

        clist->cl_id = ++lastid;

        clist->cl_name = mmwb_cpystring(td->Name);
        clist->cl_tooldes = td;
        CL_READABLE(clist);
        return clist;
}

/*
 * cl_oldtool:
 *      reuse an existing tool
 */

/* ARGSUSED */
CLIENT *
cl_oldtool(client,server,td)
CLIENT          *client;
DISPATCHER_SERVER       server;
SM_TOOLDES      *td;
{
        client->cl_server = server;
        client->cl_fd = dispatcher_server_fd(server);
        client->cl_state = 0;           /* Reset the state */
        client->cl_flags = 0;           /* Clear the flags */
        client->cl_cbuf[0] = '\0';
        client->cl_indx = 0;
        client->cl_current = NULL;
        client->cl_dqueue = NULL;
        client->cl_type = td->Type;
```

```
        CL_READABLE(clist);
        return  client;
}

/*
 * cl_find:
 *      find a client by tool_id
 *
 */

CLIENT  *
cl_find (cp, id)
CLIENT  *cp;
int     id;
{
        CLIENT  *c = cp;

        while (c != CL_NULL) {
                if (c->cl_id == id)
                        return c;

                c = c->cl_next;
        }

        return CL_NULL;
}

/*
 * cl_dequeue:
 *      Remove a dialog from a client's queue
 *      Treat unacknowledged dialogs as non-existent
 */

int
cl_dequeue(client, dialog)
CLIENT  *client;
DIALOG  dialog;
{
        DLIST   *dl;
        DLIST   *lastdl = DL_NULL;
        DIALOG  d;

        for (dl = client->cl_dqueue; dl != DL_NULL; dl = dl->dl_next) {
                if (dl->dl_status == DL_PENDING)
                        continue;
                d = dl->dl_dialog;
                if (d == NULL)
                        continue;
                if (dialog_id(d) == dialog_id(dialog)) {
                        if (lastdl == DL_NULL)
                                client->cl_dqueue = dl->dl_next;
                        else
                                lastdl->dl_next = dl->dl_next;

                        dl_free(dl);
                        return 0;
                }

                lastdl = dl;
        }
        return -1;
}

/*
 * cl_addqueue:
 *      Add a dialog to a client's queue
 */

void
cl_addqueue(client, dialog, type)
CLIENT  *client;
DIALOG  dialog;
int     type;
{
        DLIST           *dl = client->cl_dqueue;
        static DLIST    *dl_create();
        DLIST           *odl = NULL;
        DIALOG          d;

        if ( dl == DL_NULL )
                client->cl_dqueue = dl_create(dialog,type);
        else {
                while (dl != DL_NULL) {
                        if ((dl->dl_status != DL_PENDING) &&
                           (dialog_get_flag(dialog,DI_SUPERCEDES))) {
                                /*
                                 * Replace any other supercedes dialogs
                                 * from this tool
                                 */
                                if (dialog_get_flag(d = dl->dl_dialog,
                                                        DI_SUPERCEDES)) {
                                  if ( (dialog_source(d) ==
                                          dialog_source(dialog)) &&
                                         (strcmp(dialog_name(d),
```

```
                                        dialog_name(dialog)) == 0)){
                                                odl = dl->dl_next;
                                                dl_free(dl);
                                                dl = dl_create(dialog,type)
                                                dl->dl_next = odl;
                                                return;
                                        }
                                }
                        }
                        odl = dl;
                        dl = dl->dl_next;
                }
                odl->dl_next = dl_create(dialog,type);
        }
        return;
}

/*
 * cl_writelock:
 *      Lock this client against writing.
 */

void
cl_writelock (client)
CLIENT  *client;
{
        client->cl_flags |= CL_WRLOCK;

        if (FD_ISSET(client->cl_fd,&sm_writefds)) {
                client->cl_flags |= CL_WRITESET;
                FD_CLR(client->cl_fd,&sm_writefds);
        }

        return;
}

/*
 * cl_wrunlock:
 *      Unlock this client and turn write select back on.
 */

void
cl_wrunlock (client)
CLIENT  *client;
{
        if (client->cl_flags & CL_WRITESET)
                FD_SET(client->cl_fd,&sm_writefds);
        client->cl_flags &= ~(CL_WRITESET|CL_WRLOCK);

        return;
}

/*
 * cl_notify:
 *      Send a notification of some event to the user.  This
 *      is used for internal error and informational messages.
 */

void
cl_notify(text,errflag)
char    *text;
int     errflag;
{
        DIALOG  d;
        CLIENT  *client = cl_find(clist, 0);            /* Get a pointer to the UIM */

        /*
         * Create the dialog
         */
        if (errflag)
                d = dialog_create(DI_NOTE,0,"Session Manager Error",text);
        else
                d = dialog_create(DI_NOTE,0,"Session Manager Message",text);

        /*
         * Set the dialog id
         */
        (void) dialog_set_id (d,++dilast);

        /*
         * Now, set the source to "-1", which is the session manager
         */
        (void) dialog_set_source(d,-1);
        (void) dialog_set_text(d,DI_V_TOOL,"Session Manager");

        /*
         * Queue it for delivery
         */
        cl_addqueue(client,d,DL_DIALOG);

        return;
}
```

sm/client.c

```
/*
 * dl_sendd:
 *      Send a dialog to a tool
 */

static  void
dl_sendd (client,dl)
CLIENT  *client;
DLIST   *dl;
{
        DIALOG          d;
        char            *dchar;
        int             dtype;
        static int      com_write(), com_write_end();

        d = dl->dl_dialog;
        dtype = dl->dl_type;

        while (d != NULL) {
                dchar = dialog_to_char(d);
                if (dchar != NULL) {
                        switch(dtype) {
                        case DL_DIALOG:
                                (void) sprintf(msg_buf, "DIALOG %s\n",
                                        stringize(dchar));
                                break;
                        case DL_REPLY:
                        case DL_OPTION:
                                (void) sprintf(msg_buf, "REPLY %s\n",
                                        stringize(dchar));
                                break;
                        }

                        if (com_write(client->cl_fd, msg_buf, strlen(msg_buf)) <
                                syslog (LOG_ERR, "DIALOG write failed %m"
                                break;
                        }
                        mmwb_stringfree(dchar);
                }
                d = dialog_next(d);
        }

        if (com_write_end(client->cl_fd) < 0)
                syslog (LOG_ERR, "DIALOG end failed %m");

}
```

```
/*
 * dl_respd:
 *      Send a dialog to a tool in the form of a response
 */

static  void
dl_respd (client,dl)
CLIENT  *client;
DLIST   *dl;
{
        DIALOG  d;
        char    *dchar;
        int     dtype;

        d = dl->dl_dialog;
        dtype = dl->dl_type;

        reply(client->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, "Okay");
        reply_begin_ASCII (client->cl_fd);

        while (d != NULL) {
                dchar = dialog_to_char(d);
                if (dchar != NULL) {
                        switch(dtype) {
                        case DL_DIALOG:
                                (void) sprintf(msg_buf, "DIALOG %s\n",
                                        stringize(dchar));
                        case DL_REPLY:
                        case DL_OPTION:
                                (void) sprintf(msg_buf, "REPLY %s\n",
                                        stringize(dchar));
                        }
                        reply_ASCII(client->cl_fd, msg_buf);
                        mmwb_stringfree(dchar);
                }
                d = dialog_next(d);
        }

        reply_end_ASCII(client->cl_fd);
        reply_end(client->cl_fd);
}

/*
 * com_write:
 *      buffer up the command for eventual write
 */
```

sm/client.c

```
static  char    my_buf[IO_SIZE*4];
static  int     my_buf_len = 0;

static int
com_write (fd, buf, len)
int     fd;
char    *buf;
int     len;
{
        static int              com_write_flush();

        if (my_buf_len+len >= IO_SIZE*4) {
                my_buf[my_buf_len] = '\0';
                return com_write_flush (fd, buf, len);
        }

        (void) strncpy (&my_buf[my_buf_len], buf, len);
        my_buf_len += len;
        return len;
}

/*
 * com_write_end:
 *      Add the obligatory "." and send the buffer
 */

static int
com_write_end (fd)
int     fd;
{
        static int              com_write_flush();

        return com_write_flush(fd, ".\n", 2);
}

/*
 * com_write_flush:
 *      write the data to the client
 */

static int
com_write_flush (fd, buf, len)
int     fd;
char    *buf;
int     len;
{
        struct  iovec   iov[2];
        int     err;

        if (my_buf_len + len < IO_SIZE*4) {
                (void) strncpy (&my_buf[my_buf_len], buf, len);
                my_buf_len += len;
                my_buf[my_buf_len] = '\0';
                err = write (fd, my_buf, my_buf_len);
                my_buf_len = 0;
                my_buf[0] = '\0';
                return err;
        } else {
                iov[0].iov_base = my_buf;
                iov[0].iov_len = my_buf_len;
                iov[1].iov_base = buf;
                iov[1].iov_len = len;
                my_buf_len = 0;
                return writev(fd, iov, 2);
        }
}

/*
 * dl_create:
 *      Create a dialog list structure
 */

static  DLIST *
dl_create (dialog,type)
DIALOG  dialog;
int     type;
{
        DLIST   *dl;

        dl = (DLIST *)emalloc(sizeof (DLIST));

        dl->dl_dialog = dialog;
        dl->dl_type = type;
        dl->dl_status = DL_WAITING;
        dl->dl_next = DL_NULL;
        return dl;
}

/*
 * dl_free:
 *      Free a dlist structure
 */

static  void
```

sm/client.c

```
dl_free (dl)
DLIST    *dl;
{
        dialog_free(dl->dl_dialog,0);
        (void) free ((char *)dl);
}
```

sm/error.c

```
/*
 * $Id: error.c,v 1.3 89/08/07 12:57:44 scooter Exp Locker: scooter $
 * $Log:                error.c,v $
 * Revision 1.3  89/08/07  12:57:44  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.2  88/09/19  17:10:45  conrad
 * Distinguish between hard and soft failures
 *
 * Revision 1.1  88/08/25  11:53:27  scooter
 * Initial revision
 *
 * Revision 1.1  88/07/25  13:15:25  conrad
 * Initial revision
 *
 */

#include "error.h"
#include "reply.h"
#include <syslog.h>

/*
 * error_internal:
 *      Internal server error.  Clear out all clients.
 */
void
error_internal(msg)
char    *msg;
{
        static int              die = FALSE;

        if (die) {
                /* Re-entered from reply().  All bets are off now. */
                abort();
        }
        die = TRUE;

        syslog (LOG_ERR,"internal error %s -- clearing all clients",msg);

        while (clist != NULL) {
                reply(CL_FILENO(clist), RS_FAILED, RT_OTHER, 3, "SERVER E
                reply(CL_FILENO(clist), RS_FAILED, RT_OTHER, 3, msg);
                reply(CL_FILENO(clist), RS_FAILED, RT_OTHER, 0, "Bye");
                reply_end(CL_FILENO(clist));
                (void) cl_free(clist);
        }
        abort();
        /* NOTREACHED */
}

/*
 * error_handshake:
 *      Protocol handshaking screwed up.  Terminate single client.
 */
void
error_handshake(clp)
CLIENT  *clp;
{
        syslog (LOG_ERR,"protocol error for client %s",clp->cl_name);

        reply(CL_FILENO(clp), RS_INFO, RT_OTHER, 4, "Handshake error");
        reply(CL_FILENO(clp), RS_FAILED, RT_OTHER, 0, "Bye");
        reply_end(CL_FILENO(clp));
        (void) cl_free(clp);
}

/*
 * error_reply:
 *      Non-fatal error - reply, log, and return.
 */
void
error_reply(clp,msg,err)
CLIENT  *clp;
char    *msg;
int      err;
{
        syslog (LOG_INFO,"%s, errno = %d, client = %s",msg,err,clp->cl_name);

        reply(CL_FILENO(clp), RS_INFO, RT_OTHER, err, msg);
        reply(CL_FILENO(clp), RS_FAILED, RT_OTHER, 0, "Command rejected");
        reply_end(CL_FILENO(clp));
}
```

sm/process.c

```
#ifndef lint
static	char *RCSid = "$Id: process.c,v 1.21 89/11/13 19:23:52 scooter Exp $";
#endif lint

/*
 * process.c:
 *		protocol processing routines for sm
 *
 * $Log:		process.c,v $
 * Revision 1.21  89/11/13  19:23:52  scooter
 * Debugging changes
 *
 * Revision 1.20  89/10/25  14:31:07  gregc
 * use mmwbmisc.h, ansi-C-ize.
 *
 * Revision 1.19  89/08/07  12:55:07  scooter
 * Added named session support and support for the TYPE command
 *
 * Revision 1.18  89/07/24  15:15:24  scooter
 * Fixed bugs in fork implementation
 *
 * Revision 1.17  89/07/19  10:54:59  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.16  89/04/03  18:02:55  scooter
 * Added current project to the return from a STATUS request
 *
 * Revision 1.15  89/03/08  15:46:57  scooter
 * More dialog changes
 *
 * Revision 1.14  89/03/01  18:00:24  pett
 * uses strchr instead of index (to take advantage of machdep.h)
 *
 * Revision 1.13  89/02/21  23:39:43  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.13  89/02/21  23:38:23  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.12  88/11/21  11:35:40  scooter
 * Modified sm and libsm to understand the differences between
 * processes and instruments
 *
 * Revision 1.11  88/10/13  11:34:00  scooter
 * Modified to use emalloc insted of malloc
 *
 * Revision 1.10  88/10/12  17:28:01  conrad
 * Don't let UIM respond to unack'ed dialogs
 *
 * Revision 1.9  88/10/04  17:10:53  conrad
 * KILL only takes one argument, not 2
 *
 * Revision 1.8  88/09/19  17:11:20  conrad
 * Use new dispatcher library interface
 *
 * Revision 1.7  88/09/16  23:56:46  scooter
 * Cleaned up reply code in tool_launch and readt so that error
 * replies get successfully returned.
 *
 * Revision 1.6  88/09/15  23:53:44  scooter
 * Added routine cl_quituim and cleaned up termination code
 * for UIMs to allow an interrupt or a close to act as if
 * the UIM had sent a QUIT.
 *
 * Revision 1.5  88/09/15  23:08:16  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
 * Revision 1.4  88/09/01  16:56:18  scooter
 * Cleaned up DIALOG protocol somewhat:  added READ_DIALOG and
 * READ_REPLY commands.  Also modified tool launching to include
 * the project IDs.
 *
 * Revision 1.3  88/08/29  20:29:20  scooter
 * Removed location from tool description
 *
 * Revision 1.2  88/08/29  20:23:56  scooter
 * Added write locking to clients and finished protocol
 * by adding deliv and external state switch to
 * SM_SENTD.
 *
 * Revision 1.1  88/08/25  11:53:34  scooter
 * Initial revision
 *
 *
 */

#include	"sm.h"
#include	"client.h"
#include	"reply.h"
#include	"error.h"
#include	"dialog.h"
#include	"mmwbmisc.h"
```

sm/process.c

```
#ifndef lint
static	char *RCSid = "$Id: process.c,v 1.21 89/11/13 19:23:52 scooter Exp $";
#endif lint

/*
 * process.c:
 *	protocol processing routines for sm
 *
 * $Log:	process.c,v $
 * Revision 1.21  89/11/13  19:23:52  scooter
 * Debugging changes
 *
 * Revision 1.20  89/10/25  14:31:07  gregc
 * use mmwbmisc.h, ansi-C-ize.
 *
 * Revision 1.19  89/08/07  12:55:07  scooter
 * Added named session support and support for the TYPE command
 *
 * Revision 1.18  89/07/24  15:15:24  scooter
 * Fixed bugs in fork implementation
 *
 * Revision 1.17  89/07/19  10:54:59  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.16  89/04/03  18:02:55  scooter
 * Added current project to the return from a STATUS request
 *
 * Revision 1.15  89/03/08  15:46:57  scooter
 * More dialog changes
 *
 * Revision 1.14  89/03/01  18:00:24  pett
 * uses strchr instead of index (to take advantage of machdep.h)
 *
 * Revision 1.13  89/02/21  23:39:43  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.13  89/02/21  23:38:23  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.12  88/11/21  11:35:40  scooter
 * Modified sm and libsm to understand the differences between
 * processes and instruments
 *
 * Revision 1.11  88/10/13  11:34:00  scooter
 * Modified to use emalloc insted of malloc
 *
 * Revision 1.10  88/10/12  17:28:01  conrad
 * Don't let UIM respond to unack'ed dialogs
 *
 * Revision 1.9  88/10/04  17:10:53  conrad
 * KILL only takes one argument, not 2
 *
 * Revision 1.8  88/09/19  17:11:20  conrad
 * Use new dispatcher library interface
 *
 * Revision 1.7  88/09/16  23:56:46  scooter
 * Cleaned up reply code in tool_launch and readt so that error
 * replies get successfully returned.
 *
 * Revision 1.6  88/09/15  23:53:44  scooter
 * Added routine cl_quituim and cleaned up termination code
 * for UIMs to allow an interrupt or a close to act as if
 * the UIM had sent a QUIT.
 *
 * Revision 1.5  88/09/15  23:08:16  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
 * Revision 1.4  88/09/01  16:56:18  scooter
 * Cleaned up DIALOG protocol somewhat:  added READ_DIALOG and
 * READ_REPLY commands.  Also modified tool launching to include
 * the project IDs.
 *
 * Revision 1.3  88/08/29  20:29:20  scooter
 * Removed location from tool description
 *
 * Revision 1.2  88/08/29  20:23:56  scooter
 * Added write locking to clients and finished protocol
 * by adding deliv and external state switch to
 * SM_SENTD.
 *
 * Revision 1.1  88/08/25  11:53:34  scooter
 * Initial revision
 *
 *
 */

#include	"sm.h"
#include	"client.h"
#include	"reply.h"
#include	"error.h"
#include	"dialog.h"
#include	"mmwbmisc.h"
```

```
static char msg_buf[REPLY_LENGTH];
int     dlast = 0;

#define HANDSHAKE_MIN 1
#define HANDSHAKE_MAX1
#define MIN_VERSION   1
#define MAX_VERSION   2

/*
 * The sm-uim protocol
 */

static int          uim_user();
static int          vers(), helo(), user(), null(), help();

static int          sendst(), readt(), readd(), queued(), quit(), killt(), deliv();
static int          sendr(), saveopt(), setreply(), sendproj(), quituim();
static int          tooltype(), verify_user();
static void         prhelp();


STATE   uim_prot[] = {

        /* Start state */
        {SM_VERS, {
                { "VERS", SM_HELO, vers },              /* Reply version
                {NULL}}},
        {SM_HELO, {
                { "HELO", SM_USER, helo },              /* Exchange hell
                {NULL}}},
        {SM_USER, {
                { "USER", SM_UIM, uim_user },           /* Get user nam
                {NULL}}},
        {SM_UIM, {
                { "HELP", SM_UIM, help },               /* Help */
                { "STATUS", SM_UIM, sendst },           /* Send Status *
                { "REPLY", SM_REPLY, readd },           /* Read reply */
                { "LAUNCH", SM_GETOPT, readt },         /* Read tool des
                { "KILL", SM_UIM, killt },              /* Kill a tool */
                { "QUIT", SM_UIM, quituim },            /* Close this clie
                {NULL}}},

        /* Sent dialog, awaiting acknowledgement */
        {SM_SENTD, {
                { "HELP", SM_SENTD, help },             /* Help */
                { "STATUS", SM_UIM, sendst },           /* Send Status */
                { "REPLY", SM_REPLY, readd },           /* Read reply */
                { "LAUNCH", SM_GETOPT, readt },         /* Read tool des */
                { "SUCCESS", SM_UIM, deliv },           /* Dialog delivered */
                { "FAILURE", SM_UIM, null },            /*   " not delivered */
                { "KILL", SM_UIM, killt },              /* Kill a tool */
                { "QUIT", SM_UIM, quituim },            /* Close this client */
                {NULL}}},
        {SM_REPLY, {
                { "REPLY", SM_REPLY, readd },           /* Dialog continuing */
                { ".", SM_UIM, sendr },                 /* Dialog complete */
                {NULL}}},
        {SM_GETOPT, {
                { "REPLY", SM_GETOPT, readd },          /* Dialog continuing */
                { ".", SM_UIM, saveopt },               /* Dialog complete */
                {NULL}}},
        ENDSTATE
};

STATE   tool_prot[] = {

        /* Start state */
        {SM_VERS, {
                { "VERS", SM_HELO, vers },              /* Reply version */
                {NULL}}},
        {SM_HELO, {
                { "HELO", SM_USER, helo },              /* Exchange hellos */
                {NULL}}},
        {SM_USER, {
                { "USER", SM_TYPE, user },              /* Get user name */
                {NULL}}},
        {SM_TYPE, {
                { "TYPE", SM_PROJ, tooltype },          /* Get the tool type */
                { "PROJECTS", SM_TOOL, sendproj },      /* Send the projects
                                                           (backwards compat */
                {NULL}}},
        {SM_PROJ, {
                { "PROJECTS", SM_TOOL, sendproj },      /* Send the projects */
                {NULL}}},
        {SM_TOOL, {
                { "HELP", SM_TOOL, help },              /* Help */
                { "STATUS", SM_TOOL, sendst },          /* Send Status */
                { "DIALOG", SM_DIALOG, readd },         /* Read dialog */
                { "READ_REPLY", SM_TOOL, setreply },    /* Send reply when
                                                           available */
                { "READ_DIALOG", SM_TOOL, setreply },   /* Send dialog when
                                                           available */
```

```
                { "QUIT", SM_TOOL, quit },              /* Close this clie
                {NULL}}},
        {SM_DIALOG, {
                { "DIALOG", SM_DIALOG, readd },         /* Dialog contin
                { ".", SM_TOOL, queued },               /* Dialog compl
                {NULL}}},
        ENDSTATE
};


static int
vers(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        int     hmin, hmax, min, max;
        int     hversion, version;

        if (nargs != 5) {
                reply(cl->cl_fd, RS_REJECT, RT_OTHER, RB_HANDSHAKE,
                        "VERS command syntax error");
                return -1;
        }

        hmin = atoi(args[1]);
        hmax = atoi(args[2]);
        min = atoi(args[3]);
        max = atoi(args[4]);

        if (HANDSHAKE_MIN > hmax || HANDSHAKE_MAX < hmin) {
                reply(cl->cl_fd, RS_FAILED, RT_OTHER, RB_HANDSHAKE,
                        "VERS protocol version mismatch");
                return -1;
        }

        if (MIN_VERSION > max || MAX_VERSION < min) {
                reply(cl->cl_fd, RS_FAILED, RT_OTHER, RB_HANDSHAKE,
                        "VERS version mismatch");
                return -1;
        }

        hversion = (HANDSHAKE_MAX > hmax) ? hmax : HANDSHAKE_MAX;
        version = (MAX_VERSION > max) ? max : MAX_VERSION;
        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, "Okay");
        (void) sprintf(msg_buf, "%d %d", hversion, version);
        reply_begin_ASCII(cl->cl_fd);
        reply_ASCII(cl->cl_fd, msg_buf);
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);

        return next_state;
}

static int
helo(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        if (nargs != 2) {
                reply(cl->cl_fd, RS_FAILED, RT_OTHER, RB_HANDSHAKE,
                        "HELO command missing conversation ID");
                return -1;
        }
        if (sm_cid != NULL || cl->cl_type != 0) {
                if (strcmp(sm_cid, args[1]) != 0) {
                        reply(cl->cl_fd, RS_FAILED, RT_OTHER, RB_HANDSHAKE,
                                "HELO command conversation ID mismatch");
                        return -1;
                }
        } else {
                sm_cid = mmwb_cpystring(args[1]);
        }

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, "Okay");
        reply_end(cl->cl_fd);
        return next_state;
}

static  int
uim_user(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        int     val;

        if ( (val = user(cl, nargs, args, next_state)) >= 0 ) {
                cl->cl_flags |= CL_WAITING;
                if (cl->cl_dqueue != DL_NULL)
```

```
                        CL_WRITEABLE(cl);
        }

        return val;
}

static int
user(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{

        if (nargs != 3) {
                reply(cl->cl_fd, RS_REJECT, RT_OTHER, RB_HANDSHAKE,
                        "USER command syntax error");
                return -1;
        }
        if (verify_user(cl, args[1], args[2]) < 0) {
                reply_noperm(cl->cl_fd);
                return -1;
        }

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, "Okay");
        reply_end(cl->cl_fd);
        return next_state;
}

#include <netdb.h>
#include <pwd.h>

/*
 * verify_user:
 *      Make sure that the remote account has access to the local account
 */
static int
verify_user(cl, local, remote)
CLIENT  *cl;
char    *local, *remote;
{
        register char       *host;
        register int        status;
        struct hostent      *hp;
        struct passwd       *pp;
        char    *strchr();

        host = strchr(remote, '@');
        if (host == NULL)
                return -1;
        pp = getpwnam(local);
        if (pp == NULL)
                return -1;
        if (chdir(pp->pw_dir) < 0)
                return -1;
        *host++ = '\0';
        hp = gethostbyname(host);
        if (hp == NULL)
                return -1;
        status = ruserok(hp->h_name, strcmp(local, "root") == 0, remote, local);

        cl->cl_uid = pp->pw_uid;
        cl->cl_gid = pp->pw_gid;                /* TODO: Should be an array */
        if (sm_user != NULL)
                free(sm_user);
        sm_user = mmwb_cpystring(local);
        return status;
}

static struct help {
        char    *command;
        char    *helptext;
} help_table [] =
{
 {"HELP", "Display the help information for a command"},
 {"STATUS", "Return the status of all tools" },
 {NULL,NULL}
};

static struct help help_commands =
{ NULL,
"Help is available for the following commands:\n\
  HELP STATUS " };


static int
help(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        struct  help    *h;
        void    prhelp();
```

```
        /*
         * Check for arguments
         */
        if (nargs < 2) {
                /*
                 * No arguments, print commands
                 */

                prhelp(cl, &help_commands);
                return next_state;
        }

        for ( h = help_table; h->command != NULL; h++) {
                if (!strcmp(args[1], h->command)) {
                        prhelp(cl, h);
                        return next_state;
                }
        }

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, 0, "No help is available");
        return next_state;
}

static void
prhelp (cl, ht)
CLIENT  *cl;
struct help *ht;
{
        reply(cl->cl_fd, RS_INFO, RT_OTHER, RG_OKAY, "Help follows");
        reply_begin_ASCII(cl->cl_fd);
        reply_ASCII(cl->cl_fd, ht->helptext);
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);
}


/* ARGSUSED */
static int
null(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        return next_state;
}
```

```
/*
 * sendst:
 *      send the status of all running tools back to the
 *      client.
 */

/* ARGSUSED */
static int
sendst(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        register CLIENT *clp;

        reply(cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, "Okay");
        reply_begin_ASCII(cl->cl_fd);

        for (clp = clist; clp != CL_NULL; clp = clp->cl_next) {
                (void) sprintf(msg_buf, "\"%s\" %d %d %d %s",
                        clp->cl_name, clp->cl_id, clp->cl_state,
                        clp->cl_type, stringize(clp->cl_curr));
                reply_ASCII(cl->cl_fd, msg_buf);
        }
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);

        return next_state;
}

/*
 * readt:
 *      args should be a tool description
 */

static int
readt(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        SM_TOOLDES      *tooldes;
        CLIENT          *newcl,*tool_launch();
        DIALOG          option,newdialog();
```

```
        char            *sys_proj,*home_proj,*curr_proj;

        if (nargs != 11) {
                reply( cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                        "LAUNCH -- illegal tool description");
                reply_end(cl->cl_fd);
                return -1;
        }

        /*
         * Get the tool description
         */
        tooldes = tooldes_create();

        tooldes->Name = mmwb_cpystring(args[1]);
        tooldes->Version = mmwb_cpystring(args[2]);
        tooldes->Subtools = mmwb_cpystring(args[3]);
        tooldes->OnError = atoi(args[4]);
        tooldes->Host = mmwb_cpystring(args[5]);
        tooldes->nodisplay = atoi(args[6]);
        tooldes->Type = atoi(args[7]);
        sys_proj = args[8];
        home_proj = args[9];
        curr_proj = args[10];

        /*
         * Launch the tool (if possible)
         */
        if ( (newcl = tool_launch(cl, tooldes,
                        sys_proj,home_proj,curr_proj)) == CL_NULL) {
                reply_end( cl->cl_fd );
                return 0;                       /* Don't change state */
        }

        /*
         * Create a blank dialog to dequeue later
         */
        option = dialog_create(0,0,(char *)NULL, (char *)NULL);
        (void) dialog_set_id(option,++dilast);
        cl_addqueue (cl, option, DL_DIALOG);

        /*
         * Reply to the UIM
         */
        (void) sprintf(msg_buf,"%d %d", newcl->cl_id, dilast);
        reply(cl->cl_fd, RS_MORE, RT_SM, RG_OKAY,
                "Send option sheet, tool and option IDs follow");
        reply_begin_ASCII(cl->cl_fd);
        reply_ASCII(cl->cl_fd, msg_buf);
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);

        cl_writelock(cl);               /* Lock the client against writing */

        return next_state;
}

/*
 * deliv:
 *      Successfully delivered dialog, mark it as such
 */

/* ARGSUSED */
static int
deliv(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        DLIST   *dl = cl->cl_dqueue;
        DIALOG  d;
        int     dlcount = 0;
        int     foundit = 0;

        /*
         * Find the pending dialog.
         */

        while (dl != NULL) {
                if (dl->dl_status == DL_PENDING) {
                        /*
                         * Found it, deal with it
                         */
                        d = dl->dl_dialog;
                        foundit++;
                        dl->dl_status = DL_DELIVERED;
                        if (dialog_type(d) == DI_NOTE)
                                (void) cl_dequeue(cl,d);

                }
                if (dl->dl_status != DL_DELIVERED)
                        dlcount++;
                dl = dl->dl_next;
```

```
        }

        cl_wrunlock(cl);
        if (dlcount == 0)
                FD_CLR(cl->cl_fd, &sm_writefds);

        if (foundit != 1)
                error_internal("delivered non-existant dialog");

        reply(cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, "Okay");
        reply_end(cl->cl_fd);

        return  next_state;
}

/*
 * readd:
 *      Read a dialog and add it to the client's dialog list.  Force
 *      the source to be right to avoid cheaters.
 */

/* ARGSUSED */
static int
readd(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        DIALOG  dp;

        /*
         * Get the dialog into internal form
         */

        dp = dialog_from_char(args[1]);

        if (dp == NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                        "DIALOG syntax error");
                reply_end(cl->cl_fd);
                return cl->cl_state;
        }

        /*
         * Add it to the client's current dialog
         */

        if (cl->cl_current == NULL)
                cl->cl_current = dp;
        else
                (void) dialog_append(cl->cl_current, dp);

        reply(cl->cl_fd, RS_MORE, RT_SM, RG_OKAY, "Send next DIALOG");
        reply_end(cl->cl_fd);

        cl_writelock(cl);               /* Lock the client against writing */

        return next_state;
}

/*
 * sendr:
 *      check the destination queue for the dialog and
 *      remove it, then schedule the dialog for delivery
 *      to the sender.
 */

/* ARGSUSED */
static int
sendr(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        CLIENT  *cl_find(),*c;
        DIALOG  dp = cl->cl_current;
        int     cl_dequeue();

        cl->cl_current = NULL;

        /*
         * Unlock the client so we can write to it.
         */

        cl_wrunlock(cl);

        /*
         * Dequeue the dialog from the client's queue
         */

        if (cl_dequeue(cl, dp) < 0) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH, "No such dialog");
```

```
                reply_end(cl->cl_fd);
                dialog_free(dp,0);
                return next_state;
        }

        /*
         * Force the dest.
         */

        (void) dialog_set_dest(dp, cl->cl_id);

        /*
         * Find the sender (now the destination)
         */

        c = cl_find(clist,dialog_source(dp));
        if (c == CL_NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH,
                        "Unknown destination");
                reply_end(cl->cl_fd);
                (void) dialog_free(dp,0);
                return next_state;
        }

        cl_addqueue(c, dp, DL_REPLY);        /* Schedule for delivery */

        if (c->cl_flags & CL_WAITING)
                CL_WRITEABLE(c);

        (void) sprintf(msg_buf,"Reply queued for delivery to %s (%d)",
                c->cl_name, c->cl_id);

        reply(cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, msg_buf);
        reply_end(cl->cl_fd);
        return next_state;
}

/*
 * saveopt:
 *      essentially the same code as above, but queue it as a
 *      DL_OPTION insted of a DL_REPLY and save a copy in the
 *      client structure.
 */

/* ARGSUSED */
static int
saveopt(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        CLIENT  *cl_find(),*c;
        DIALOG  dp = cl->cl_current;
        int     cl_dequeue();

        cl->cl_current = NULL;

        /*
         * Unlock the client so we can write to it.
         */
        cl_wrunlock(cl);

        /*
         * Dequeue the dialog from the client's queue
         */
        if (cl_dequeue(cl, dp) < 0) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH,
                        "No option request pending");
                reply_end(cl->cl_fd);
                dialog_free(dp,0);
                return next_state;
        }

        /*
         * Force the dest.
         */
        (void) dialog_set_dest(dp, cl->cl_id);

        /*
         * Find the sender (now the destination)
         */
        c = cl_find(clist,dialog_source(dp));
        if (c == CL_NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH,
                        "Unknown destination");
                reply_end(cl->cl_fd);
                (void) dialog_free(dp,0);
                return next_state;
        }

        (void) sprintf(msg_buf,"Option received for %s (%d)",
                c->cl_name, c->cl_id);
```

```
        /*
         * Schedule for delivery to all tools waiting for it
         */
        while (c != CL_NULL) {
                /*
                 * Save a copy in the option area of the tool description
                 */
                c->cl_tooldes->Options = dialog_copy(dp,0);

                cl_addqueue(c, tool_myoption(c, dp), DL_OPTION);

                if (c->cl_flags & CL_WAITING)
                        CL_WRITEABLE(c);

                /* Clear the option wait flag */
                c->cl_flags &= ~CL_OPWAIT;

                c = c->cl_child;
        }

        reply(cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, msg_buf);
        reply_end(cl->cl_fd);
        return next_state;
}


/*
 * queued:
 *      check the destination, then add the dialog to the
 *      destination client's dialog queue.  inform the source
 *      client that the dialog is queued for
 *      delivery, and add the destination client on the
 *      write pending list for the select.
 */

/* ARGSUSED */
static int
queued(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        CLIENT  *cl_find(),*c;
        DIALOG  dp = cl->cl_current;

        /*
         * Unlock the client so we can write to it.
         */

        cl_wrunlock(cl);

        /*
         * Find the destination client
         */

        c = cl_find(clist,dialog_dest(dp));
        if (c == CL_NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH,
                        "Unknown destination");
                reply_end(cl->cl_fd);
                return next_state;
        }

        (void) dialog_set_text(dp, DI_V_TOOL,
                        mmwb_cpystring(cl->cl_name));   /* Get the tool name */
        (void) dialog_set_id(dp,++dilast);
        (void) dialog_set_source(dp,cl->cl_id);         /* Force the source */

        cl_addqueue(c, dp, DL_DIALOG);

        cl->cl_current = NULL;

        if (c->cl_flags & CL_WAITING)
                CL_WRITEABLE(c);

        /*
         * OK, now send the tool the response and the dialog ID
         */

        (void) sprintf(msg_buf,"Dialog queued for delivery to %s (%d)",
                c->cl_name, c->cl_id);
        reply(cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, msg_buf);
        reply_begin_ASCII(cl->cl_fd);
        (void) sprintf(msg_buf,"%d",dilast);
        reply_ASCII(cl->cl_fd, msg_buf);
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);

        return next_state;
}

/*
```

sm/process.c

```
/*
 * quit:
 *      End the communication with a client.  This involves
 *      a close on the port, followed by releasing any possible
 *      resources, including any pending dialogs.
 */

/* ARGSUSED */
static int
quit(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, "Bye");
        reply_end(cl->cl_fd);
        cl_quit(cl,0);
        return 0;
}

/*
 * quituim:
 *      End the communication with the uim.  This involves
 *      a close on the port, followed by writing a STATE object
 *      into the user's home project.  Then the listen socket needs
 *      to be set up to accept the next UIM.
 */

/* ARGSUSED */
static int
quituim(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        extern  void    cl_quituim();

        if (clist == cl && clist->cl_next == CL_NULL) {
                (void) sprintf(msg_buf,"No clients, SM terminating");
                reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, msg_buf);
                reply_end(cl->cl_fd);
                (void) exit (0);
        }

        if (nargs == 2)
                (void) sprintf(msg_buf, "Session %s waiting on cid %s",
                                        args[1],sm_cid);
        else
                (void) sprintf(msg_buf, "Session waiting on cid %s",sm_cid);

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY2, msg_buf);
        reply_end(cl->cl_fd);

        if (nargs == 2)
                cl_quituim(cl,args[1]);
        else
                cl_quituim(cl,(char *)NULL);

        return  0;
}

/*
 * killt:
 *      Kill the tool.  Send a request to have the dispatcher
 *      kill the tool, then close off the socket and clear it.
 */

static int
killt(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        int     tool;
        CLIENT  *c;

        if (nargs != 2) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                        "KILL: Improper arg count");
                reply_end(cl->cl_fd);
                return next_state;
        }

        /*
         * Now find the tool to kill
         */

        tool = atoi(args[1]);
        c = cl_find(clist,tool);
        if (c == CL_NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH, "Unknown tool");
                reply_end(cl->cl_fd);
```

```
                return next_state;
        }

        /*
         * Now kill it
         */

        dispatcher_server_kill (c->cl_server);
        cl_quit(c,1);
        (void) sprintf (msg_buf, "Tool %d killed", tool);
        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, msg_buf);
        reply_end(cl->cl_fd);

        return next_state;
}

/*
 * setreply:
 *      indicate that the tool is ready to receive data.  If there
 *      is data pending set the client's bit in the write fd.
 */

/* ARGSUSED */
static int
setreply(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        cl->cl_flags |= CL_WAITING;
        if (cl->cl_dqueue != DL_NULL)
                CL_WRITEABLE(cl);
        /*
         * Note: no reply is given.  The reply will actually
         * come in the form of a REPLY series when data is
         * present
         */

        return next_state;
}

/*
 * tooltype:
 *      Get the tool type
 */

/* ARGSUSED */
static int
tooltype(cl, nargs, args, next_state)
CLIENT  *cl;
int     nargs;
char    *args[];
int     next_state;
{
        if (nargs != 2) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                        "TYPE: Improper arg count");
                reply_end(cl->cl_fd);
                return -1;
        }

        if (strcmp(args[1], "PROTOCOL") == 0) {
                cl->cl_type = CL_PROTOCOL;
        } else if (strcmp(args[1], "INSTRUMENT") == 0) {
                cl->cl_type = CL_INSTRUMENT;
                if (ui_fd == -1) {              /* Is there a UIM running? */
                        reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOUIM,
                                "TYPE: can't run an instrument without a UIM");
                        reply_end(cl->cl_fd);
                        return  -1;
                }
        } else {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_NOSUCH,
        "TYPE: Unknown type (known types are PROTOCOLs and INSTRUMENTS)");
                reply_end(cl->cl_fd);
                return -1;
        }

        reply(cl->cl_fd, RS_ACCEPT, RT_OTHER, RG_OKAY, "OK");
        reply_end(cl->cl_fd);

        return next_state;
}

/*
 * sendproj:
 *      Send the project information to the tool
 */

/* ARGSUSED */
static int
sendproj(cl, nargs, args, next_state)
CLIENT  *cl;
```

sm/process.c

```
int     nargs;
char    *args[];
int     next_state;
{
        reply (cl->cl_fd, RS_ACCEPT, RT_SM, RG_OKAY, "Project list follows");
        reply_begin_ASCII(cl->cl_fd);
        (void) sprintf(msg_buf, "SYSTEM %s\n",cl->cl_system);
        reply_ASCII(cl->cl_fd, msg_buf);
        (void) sprintf(msg_buf, "HOME %s\n",cl->cl_home);
        reply_ASCII(cl->cl_fd, msg_buf);
        (void) sprintf(msg_buf, "CURRENT %s\n",cl->cl_curr);
        reply_ASCII(cl->cl_fd, msg_buf);
        reply_end_ASCII(cl->cl_fd);
        reply_end(cl->cl_fd);

        return  next_state;
}
```

sm/reply.c

```
/*
 * $Id: reply.c,v 1.4 89/08/07 12:57:47 scooter Exp Locker: scooter $
 * $Log:                reply.c,v $
 * Revision 1.4  89/08/07  12:57:47  scooter
 * Changed $Header$ to $Id$
 *
 * Revision 1.3  88/08/28  20:38:54  scooter
 * Fixed typo in reply function def
 *
 * Revision 1.2  88/08/28  20:36:22  scooter
 * Efficiency modifications - buffered writes.
 *
 * Revision 1.4  88/08/08  09:46:41  conrad
 * Standardize buffer sizes
 * Add "DESCRIBE DATASET" command
 *
 * Revision 1.3  88/08/04  16:26:32  conrad
 * Support data type STRING and TABLE
 *
 * Revision 1.2  88/07/27  20:21:40  conrad
 * Use reply parameter instead of hardwired numbers
 *
 * Revision 1.1  88/07/25  13:15:30  conrad
 * Initial revision
 *
 */

#include "reply.h"
#include "error.h"

/*
 * The reply buffer is at most the length of the longest string plus
 * the status codes plus a space for separation plus a newline plus
 * a null.
 */
/*
 * reply:
 *          Send a reply to a file descriptor
 */
void
reply(fd, status, class, detail, string)
int         fd;
int         status, class, detail;
char        *string;
{
            register int            len;
            char                    reply_buff[BUFLEN];

            if (status < 0 || status > 15) {
                        error_internal("Reply status out of range");
                        /* NOTREACHED */
            }
            if (class < 0 || class > 15) {
                        error_internal("Reply class out of range");
                        /* NOTREACHED */
            }
            if (detail < 0 || detail > 15) {
                        error_internal("Reply detail out of range");
                        /* NOTREACHED */
            }
            len = 3 + 1 + strlen(string) + 1;
            if (len > IO_SIZE) {
                        error_internal("Reply too long");
                        /* NOTREACHED */
            }
            (void) sprintf(reply_buf, "%x%x%x %s\n", status, class, detail, string);
            reply__write(fd, reply_buf, len);
}

#define  REPLY_BUFSIZ     (BUFLEN * 4)
static char             reply__buf[REPLY_BUFSIZ];
static int              reply__cnt;

/*
 * reply__write:
 *          "write" a buffer.  Actually, just copy the data into a
 *          static buffer (if there is space)
 */
void
reply__write(fd, buf, len)
int         fd;
char        *buf;
int         len;
{
            struct iovec            iov[2];
            extern int              bcopy();

            if (REPLY_BUFSIZ - reply__cnt >= len) {
                        (void) bcopy(buf, &reply__buf[reply__cnt], len);
                        reply__cnt += len;
            }
            else {
                        iov[0].iov_base = reply__buf;
                        iov[0].iov_len = reply__cnt;
```

```
                iov[1].iov_base = buf;
                iov[1].iov_len = len;
                if (writev(fd, iov, 2) != reply__cnt + len) {
                        error_internal("reply__write failed");
                        /* NOTREACHED */
                }
                reply__cnt = 0;
        }
}

/*
 * reply__putchar:
 *      "write" a single character
 */
void
reply__putchar(fd, c)
int     fd;
char    c;
{
        if (reply__cnt >= REPLY_BUFSIZ) {
                if (write(fd, reply__buf, REPLY_BUFSIZ) != REPLY_BUFSIZ) {
                        error_internal("reply__putchar failed");
                        /* NOTREACHED */
                }
                reply__cnt = 0;
        }
        reply__buf[reply__cnt++] = c;
}

/*
 * reply_end:
 *      Send end-of-reply message to file descriptor
 */
void
reply_end(fd)
int     fd;
{
        struct iovec    iov[2];
        static char     eor[] = "600 End of reply\n";

        if (reply__cnt == 0) {
                if (write(fd, eor, sizeof eor - 1) != sizeof eor - 1) {
                        error_internal("Reply_end error");
                        /* NOTREACHED */
                }
        }
        else {
                iov[0].iov_base = reply__buf;
                iov[0].iov_len = reply__cnt;
                iov[1].iov_base = eor;
                iov[1].iov_len = sizeof eor - 1;
                if (writev(fd, iov, 2) != reply__cnt + sizeof eor - 1) {
                        error_internal("Reply_end error");
                        /* NOTREACHED */
                }
                reply__cnt = 0;
        }
}

/*
 * reply_begin_ASCII:
 *      Start sending ASCII data
 */
void
reply_begin_ASCII(fd)
int     fd;
{
        static char     boa[] = "202 ASCII data follows\n";

        reply__write(fd, boa, sizeof boa - 1);
}

/*
 * reply_ASCII:
 *      Send ASCII data
 */
void
reply_ASCII(fd, s)
int     fd;
char    *s;
{
        register int    len;
        static char     newline = '\n';

        len = strlen(s);
        reply__write(fd, s, len);
        if (s[len - 1] != '\n')
                reply__putchar(fd, '\n');
}

/*
 * reply_end_ASCII:
 *      Terminate ASCII data
 */
```

sm/reply.c

```
void
reply_end_ASCII(fd)
int	fd;
{
	reply__write(fd, ".\n", 2);
}

/*
 * reply_noperm:
 *	Send permission denied message
 */
void
reply_noperm(fd)
int	fd;
{
	static char	pd[] = "402 Permission denied\n";

	reply__write(fd, pd, sizeof pd - 1);
}
```

sm/sm.c

```
#ifndef lint
static char RCSid[] = "$Id: sm.c,v 1.8 89/10/25 14:30:37 gregc Exp $";
#endif lint

/*
 * sm.c - Mainline for MMWB Session Manager
 *
 * The Session Manager in MMWB manages all communications
 * between the tools and User Interface Managers.  In addition,
 * it is responsible for all tool invocation.  For more information,
 * see the paper "Session Management in MMWB".
 *
 * $Log:            sm.c,v $
 * Revision 1.8  89/10/25  14:30:37  gregc
 * ansi-C-ize.
 *
 * Revision 1.7  89/08/07  12:57:38  scooter
 * Changed $Header: /usr.MC68020/src/local/mmwb/src/platform/sm/RCS/sm.c,v 1.8
 *
 * Revision 1.6  89/07/19  10:55:04  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.5  88/10/21  11:00:21  conrad
 * Handle stop/cont signals only if they exist
 *
 * Revision 1.4  88/10/12  16:24:11  conrad
 * Log death by signals
 *
 * Revision 1.3  88/09/15  23:53:47  scooter
 * Added routine cl_quituim and cleaned up termination code
 * for UIMs to allow an interrupt or a close to act as if
 * the UIM had sent a QUIT.
 *
 * Revision 1.2  88/09/15  23:08:19  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
 * Revision 1.1  88/08/25  11:54:06  scooter
 * Initial revision
 *
 *
 */

#include        "sm.h"
#include        "client.h"
#include        <sys/types.h>
#include        <sys/time.h>
#include        <syslog.h>
#include        <signal.h>
#include        <sys/socket.h>
#include        <netinet/in.h>

static char     *usage_msg = "sm [-d] [-C cid]";

int     db_flag = 0;
fd_set  sm_readfds,sm_writefds;     /* Current bit arrays */
int     sm_count = 1;               /* Initially, just waiting */

/*
 * Global variables
 */

char    *sm_user = NULL;            /* The user name we're running under */
char    *sm_cid = NULL;             /* The current conversation id */
int     ui_fd;

main(argc,argv)
int     argc;
char    **argv;
{
        static void                 sm_prot();
        static SIGNALTYPE           bomb();
        extern  int                 optind;
        extern  char                *optarg;
        int                         c;

        /*
         * SM is only initiated by a User Interface Manager which
         * has started up with no running sessions.  Therefore,
         * we have no need of the tty, close all but the pipe to
         * the SM.
         */

        while ((c = getopt(argc,argv,"dC:")) != EOF)
        {
                switch(c)
                {
                        case 'd':
                                db_flag++;
                                break;

                        case 'C':
                                sm_cid = optarg;
```

```
                                        break;

                                default:
                                        printf("usage: %s\n", usage_msg);
                                        /* NOTREACHED */
                        }
                }

                if (!db_flag)
                        db_flag = isatty(0);

                /*
                 * the dispatcher will accept the initial connection
                 * and give us open file descriptors on 0, 1, and 2
                 */

                if (!db_flag)
                {
                        (void) close(1);
                        (void) close(2);
#ifdef LOG_LOCAL4
                        openlog ("mmwb-sm", LOG_PID, LOG_LOCAL4);
#else
                        openlog ("mmwb-sm", LOG_PID);
#endif
                        syslog (LOG_INFO, "started");
                }

                ui_fd = 0;

                clist = cl_newuim(ui_fd);
                cl_greet(clist);

                for (c = 1; c < NSIG; c++)
                        (void) signal(c, bomb);
                (void) signal(SIGPIPE, SIG_IGN);
#ifdef SIGTSTP
                (void) signal(SIGTSTP, SIG_DFL);
                (void) signal(SIGCONT, SIG_DFL);
#endif

                (void)sm_prot();                /* Main loop */
                /* NOTREACHED */
}

/*
 * bomb:
 *      Log a message about bombing out
 */
static
SIGNALTYPE
bomb(sig)
int     sig;
{
        switch (sig) {
         case SIGQUIT:
         case SIGILL:
         case SIGTRAP:
         case SIGIOT:
         case SIGEMT:
         case SIGFPE:
         case SIGBUS:
         case SIGSEGV:
         case SIGSYS:
                syslog(LOG_ERR, "signal %d, core dumped", sig);
                abort();
         default:
                syslog(LOG_ERR, "signal %d", sig);
                exit(1);
        }
}

/*
 * sm_prot is the main loop for sm.  It is basically nothing
 * more than a select loop which waits for activity on any
 * one of the open sockets to the UIM or to tools.  The
 * UIM is guaranteed to be on file descriptor 0, tools
 * are simply numbered based on the file descriptor of the
 * socket they are openned on.  There is a special case in
 * that there may not be anything on file descriptor 0 (i.e.
 * no UIM), in which case we will be awaiting connection on a socket.
 *
 */

static void
sm_prot()
{
        fd_set  w_readfds,w_writefds;           /* Working bit arrays */
        CLIENT  *c;

        /*
         * Main Loop
         */
```

```
        FD_ZERO (&sm_readfds);
        FD_ZERO (&sm_writefds);
        FD_SET  (ui_fd, &sm_readfds);

        while (clist != CL_NULL)
        {

                w_readfds = sm_readfds;
                w_writefds = sm_writefds;

                if (select (FD_SETSIZE, &w_readfds, &w_writefds,
                                (fd_set *) NULL, (struct timeval *) NULL) == -
                {
                        syslog(LOG_ERR, "select: %m");
                        exit (0);
                        /* NOTREACHED */
                }

                if (ui_fd == (-1) && FD_ISSET(0, &w_readfds)) {
                        accept_uim();
                        continue;
                }

                for (c = clist; c != CL_NULL; c = c->cl_next) {
                        if (FD_ISSET(c->cl_fd,&w_writefds))
                                cl_output(c);

                        if (FD_ISSET(c->cl_fd,&w_readfds))
                                cl_input(c);
                }

        }

}

accept_uim()
{
        CLIENT  *c;
        struct  sockaddr            sa;
        int     salen;

        /*
         * Got a new UIM, accept it
         */
        salen = sizeof sa;
        if ((ui_fd = accept(0, &sa, &salen)) < 0) {
                syslog(LOG_ERR, "accept: %m");
                return;
        }

        /*
         * Preserve fd 0 by making the
         * ui file descriptor 0
         */
        (void) close(0);
        (void) dup2(ui_fd,0);
        (void) close(ui_fd);
        ui_fd = 0;
        sm_count++;

        /*
         * Now delete the STATE object
         */

        sm_deletestate();

        /*
         * Find the UIM entry
         */
        for (c = clist; c != CL_NULL; c = c->cl_next) {
                if (c->cl_type == 0) {
                        c->cl_state = 0;
                        c->cl_indx = 0;
                        c->cl_flags = 0;
                        c->cl_cbuf[0] = '\0';
                        cl_greet(c);
                        return;
                }
        }

        syslog(LOG_ERR,"Couldn't find UIM entry");
        abort();
}
```

sm/smio.c

```
#ifndef lint
static char *RCSid = "$Id: smio.c,v 1.7 89/10/25 21:28:01 gregc Exp $";
#endif lint

/*
 * smio.c:
 *      These routines provide the interface between the
 *      session manager and the data manager.
 *
 * $Log:        smio.c,v $
 * Revision 1.7  89/10/25  21:28:01  gregc
 * remove lint.
 *
 * Revision 1.6  89/08/21  13:53:12  scooter
 * Added mmwbmisc include
 *
 * Revision 1.5  89/08/07  13:03:16  scooter
 * Made named sessions persistent
 *
 * Revision 1.4  89/08/07  12:53:18  scooter
 * Added named sessions -- the session name is determined by an
 * argument to the QUIT command from the UIM
 *
 * Revision 1.3  89/07/18  21:56:01  scooter
 * Removed includes for param.h and time.h, which are no longer used
 *
 * Revision 1.2  88/10/19  17:02:14  conrad
 * Use ofs.h and OFS_OBJECT rather than dm.h and DM_OBJECT
 *
 * Revision 1.1  88/09/19  21:41:08  scooter
 * Initial revision
 *
 *
 */

#include        "sm.h"
#include        "ofs.h"
#include        <mmwbmisc.h>
#include        <syslog.h>

static  OFS_OBJECT      smio_home_proj;
static  OFS_OBJECT      smio_state;
static  char            *ses_name = NULL;

/*
 * sm_writestate:
 *      This routine writes a Session Manager STATE object into
 *      the user's home project.  This object is then used by
 *      the UIM to determine the list of currently running
 *      session managers that it may want to connect to.
 */
void
sm_writestate(port,session_name)
u_short port;
char    *session_name;
{
        char            name_buf[100];
        char            *name;

        if (session_name == NULL) {
                if (ses_name == NULL)
                        name = sm_cid;
                else
                        name = ses_name;
        } else {
                name = session_name;
                ses_name = mmwb_cpystring(name);
        }

        /*
         * Begin by opening the Home project
         */
        if ( (smio_home_proj = ofs_proj_init((char *) NULL)) == NULL) {
                syslog(LOG_ERR, "unable to open user's home project: %s",
                        dm_errlist[dm_errno]);
                return;
        }

        if ( (smio_state = obj_create (smio_home_proj, name, OBJ_CL_STATE,
                        OBJ_T_SESSION)) == NULL) {
                syslog(LOG_ERR, "unable to create STATE object: %s",
                        dm_errlist[dm_errno]);
                return;
        }

        (void) sprintf (name_buf, "%d", (int) port);
        prop_set (smio_state, "CID", (void *) sm_cid );
        prop_set (smio_state, "Port", (void *) name_buf);
        prop_set (smio_state, "Pending", (void *) "0");

        return;
}

/*
```

sm/smio.c

```
 * sm_deletestate:
 *          delete the object created by sm_writestate after a
 *          connection has been made.
 */

void
sm_deletestate()
{

        if ( obj_delete (smio_state) != 0 )
                syslog(LOG_ERR, "unable to create STATE object: %s",
                        dm_errlist[dm_errno]);

        obj_close (smio_state);
        proj_close (smio_home_proj);

        return;
}
```

sm/tool.c

```
#ifndef lint
static char *RCSid = "$Id: tool.c,v 1.16 90/02/16 10:46:58 scooter Exp $";
#endif lint

/*
 * tool.c -             this module handles the interface to tools, in
 *                      particular, tool launching and options
 *
 * $Log:                tool.c,v $
 * Revision 1.16  90/02/16  10:46:58  scooter
 * Changed toolspec to allow "_","-",and "." in a toolspec
 *
 * Revision 1.15  89/10/25  14:31:11  gregc
 * use mmwbmisc.h, ansi-C-ize.
 *
 * Revision 1.14  89/08/07  12:57:34  scooter
 * Changed $Header: /local/src/mmwb/src/platform/sm/RCS/tool.c,v 1.16 90/02/16 10:
 *
 * Revision 1.13  89/07/24  23:46:06  scooter
 * Fixed problems with forked tools joining back prematurely
 *
 * Revision 1.12  89/07/24  15:15:27  scooter
 * Fixed bugs in fork implementation
 *
 * Revision 1.11  89/07/19  11:01:25  scooter
 * Replace cl_rdunlock with CL_READABLE to be more consistent with
 * CL_WRITEABLE.
 *
 * Revision 1.10  89/07/19  10:55:07  scooter
 * Added support for multi-step protocols, and internally generated
 * notification dialogs
 *
 * Revision 1.9  89/03/10  14:22:08  scooter
 * Various bug fixes to get everything working again
 *
 * Revision 1.8  89/03/02  22:25:58  pett
 * use strchr instead of index and declare emalloc to return void *
 *
 * Revision 1.7  88/10/13  11:34:02  scooter
 * Modified to use emalloc insted of malloc
 *
 * Revision 1.6  88/09/19  17:11:23  conrad
 * Use new dispatcher library interface
 *
 * Revision 1.5  88/09/17  00:28:07  scooter
 * Sanitize CID before call to dispatcher_call
 *
 * Revision 1.4  88/09/16  23:57:21  scooter
 * Cleaned up reply code in tool_launch and readt so that error
 * replies get successfully returned.
 *
 * Revision 1.3  88/09/15  23:08:20  scooter
 * Bug fixes, and addition of various routines for interface to libofs
 * and reestablishing connection to a new UIM
 *
 * Revision 1.2  88/08/29  20:29:26  scooter
 * Removed location from tool description
 *
 * Revision 1.1  88/08/25  11:54:15  scooter
 * Initial revision
 *
 *
 */

#include        <ctype.h>
#include        "sm.h"
#include        "client.h"
#include        "reply.h"
#include        "mmwbmisc.h"

extern  char    *strncpy();
extern  char    *strchr();
extern  char    *strcat();
extern  void    *emalloc();

/*
 * Private Internal variables
 */

static  int     tool_count;
static  int     t_error;
static  char    *t_errstr;

static  char    msg_buf[BUFSIZ];

/*
 * Private procedures
 */

SM_TOOLTREE *toollist(),*addnext(),*addfork(),*tool_buildtree(),*toolspec();
CLIENT  *tool_sublaunch();
char    *hostlist();
void    tool_error(),tool_free(),hostupdate(),forkupdate();
```

sm/tool.c

```
/*
 * tooldes_create:
 *      create a tool description structure
 */

SM_TOOLDES *
tooldes_create()
{
        SM_TOOLDES *td = (SM_TOOLDES *)emalloc(sizeof (SM_TOOLDES));

        td->Version = td->Subtools = td->Host = NULL;

        return td;
}

/*
 * tooldes_free:
 *      free a tool description
 */

void
tooldes_free(td)
SM_TOOLDES *td;
{
        if (td == (SM_TOOLDES *)NULL)
                return;

        mmwb_stringfree(td->Version);
        mmwb_stringfree(td->Subtools);
        mmwb_stringfree(td->Host);

        free((char *)td);
        return;
}

/*
 * tool_launch:
 *      launch a tool given the tool description
 *
 */

CLIENT  *
tool_launch(cl, tooldes,sys,home,curr)
CLIENT  *cl;
SM_TOOLDES      *tooldes;
char    *sys,*home,*curr;
{
        CLIENT          *client;
        SM_TOOLTREE     *tooltree;

        if (tooldes == (SM_TOOLDES *)NULL) {
                reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                        "No tool description");
                return CL_NULL;
        }

        /*
         * Build the tool tree
         */
        if ( (tooltree = tool_buildtree(tooldes)) == (SM_TOOLTREE *)NULL ) {
                if (t_error)
                        reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                                t_errstr);
                else
                        reply(cl->cl_fd, RS_FAILED, RT_SM, RB_SYNTAX,
                                "Unable to understand protocol specification");
                return CL_NULL;
        }

        /*
         * Now try to launch the first tool
         */
        client = tool_steplaunch(CL_NULL,tooltree,tooldes,1,sys,home,curr);
        if ( client == CL_NULL) {
                reply (cl->cl_fd, RS_FAILED, RT_SM, RB_UNKNOWN,
                        "unable to launch protocol");
                reply_end(cl->cl_fd);
                return CL_NULL;
        } else
                return  client;
}

/*
 * tool_myoption:
 *      search the list of options for this protocol and return
 *      the options specific to this tool.  By definition,
 *      this tools options consist of the dialogs in option
 *      sheet zero plus the dialogs in option sheet N, where
 *      N is this tool's 'tnum'.
 */

DIALOG
tool_myoption(client, option_list)
CLIENT          *client;
```

sm/tool.c

```
DIALOG              option_list;
{
        DIALOG  d = option_list;
        DIALOG  dp = NULL;
        int     optnum,tnum;
        int     err;

        if (client == (CLIENT *)NULL ||
                client->cl_tooltree == (SM_TOOLTREE *)NULL)
                        return  NULL;

        tnum = client->cl_tooltree->tt_tnum;

        while (d != NULL) {
                optnum = dialog_get_int(d,DI_V_OPTION,&err);
                if (optnum == 0 || optnum == tnum) {
                        /* Add it in */
                        dp = dialog_append(dp,dialog_copy(d,DI_THISDI));
                }
                d = dialog_next(d);
        }
        return  dp;
}

/*
 * tool_steplaunch:
 *      Launch a step.  This routine handles all launching of
 *      tools from the session manager.
 */

CLIENT *
tool_steplaunch(client, tooltree, tooldes, initial, sys, home, curr)
CLIENT          *client;
SM_TOOLTREE     *tooltree;
SM_TOOLDES      *tooldes;
int             initial;
char            *sys,*home,*curr;
{
        char    *host = mmwb_cpystring(tooltree->tt_host);
        char    *tool = tooltree->tt_tool;
        char    *p,*q;
        DISPATCHER_SERVER       server;

        q = host;

        do {
                p = strchr(q, ',');
                if (p != NULL)
                        *p++ = '\0';
                else {
                        /* Support space separated as well */
                        p = strchr(q, ' ');
                        if (p != NULL)
                                *p++ = '\0';
                }

                server = dispatcher_call(q, sm_user, sm_cid, tool);

                if (server != NULL) {
                        /*
                         * Successful open, create a client and return
                         * it.
                         */
                        if (client == CL_NULL) {
                                client = cl_newtool(server, tooldes);
                                client->cl_system = mmwb_cpystring(sys);
                                client->cl_home = mmwb_cpystring(home);
                                client->cl_curr = mmwb_cpystring(curr);
                        } else
                                client = cl_oldtool(client, server, tooldes);
                        /*
                         * Update the tooltree
                         */
                        tooltree->tt_client = client;
                        client->cl_tooltree = tooltree;
                        /*
                         * Send our banner
                         */
                        cl_greet (client);

                        if (!initial) {
                                /*
                                 * Queue up the tool's options
                                 */
                                cl_addqueue(client,
                                        tool_myoption(client,tooldes->Options),
                                        DL_OPTION);

                                if (client->cl_flags & CL_WAITING)
                                        CL_WRITEABLE(client);

                                (void) sprintf(msg_buf,
                                        "Started protocol %s on host %s",
                                        tooltree->tt_tool,q);
```

```
                        cl_notify(msg_buf,0);
                } else {
                        /*
                         * Mark the tool as awaiting options
                         */
                        client->cl_flags |= CL_OPWAIT;
                }

                CL_READABLE(client);

                mmwb_stringfree(host);
                if ((tooltree = tooltree->tt_fork) != NULL) {
                        CLIENT *cl;
                        /*
                         * We've got a fork here too.  Call ourselves
                         * recursively to get this one started
                         */
                        if ( (cl = tool_steplaunch(CL_NULL,tooltree,
                                toolcles,initial,sys,home,curr))
                                                        == CL
                                if (toolcles->OnError == SM_DIE) {
                                        cl_quit(client,1);
                                        return  CL_NULL;
                                }
                        }
                        client->cl_child = cl;
                }
                return client;
        } else
                syslog (LOG_INFO,
        "unable to launch tool %s on host %s with cid '%s' - error %d",
        tool,q,sm_cid,dispatcher_errno);

        q = p;
} while (q != NULL);

/*
 * We've failed!  If this is not the initial launch,
 * notify the user.
 */
if (!initial) {
        (void) sprintf(msg_buf,"Unable to start protocol %s",
                tooltree->tt_tool);
        cl_notify(msg_buf,1);
}

mmwb_stringfree(host);
        return  CL_NULL;
}

/*
 * tool_ismulti:
 *      Check to see if there is a next step of a tooltree, and
 *      if so, should it be executed.
 */

int
tool_ismulti(client,errflag)
CLIENT  *client;
int     errflag;
{
        SM_TOOLTREE     *smt;
        SM_TOOLCES      *smtd;

        if (client == CL_NULL)
                return  FALSE;

        smt = client->cl_tooltree;
        smtd = client->cl_toolcles;

        /*
         * Depending on whether there was an error or not,
         * handle multi-step protocols according to the
         * toolcles flags
         */

        if (smt->tt_next != (SM_TOOLTREE *)NULL) {
                /*
                 * Multi-step, check for error conditions
                 */
                if (!errflag || smtd->OnError == SM_IGNORE)
                        return  TRUE;
        }
        return  FALSE;
}

/*
 * tool_buildtree:
 *      Build a tool tree from the tool list and the list of
 *      hosts.  A tool list is a list of tools to be executed
 *      in series or in parrallel.
 */

SM_TOOLTREE *
```

sm/tool.c

```
tool_buildtree(tooldes)
SM_TOOLDES *toolde;
{
        SM_TOOLTREE     *tool_list;
        static void      tool_printtree();

        tool_count = 0;
        if (toolde->Subtools != NULL) {
                tool_list = toollist(&toolde->Subtools);
                hostupdate(tool_list,toolde->Host);
                tool_printtree(tool_list,0);
                return  tool_list;
        }

        return  (SM_TOOLTREE *)NULL;
}


static SM_TOOLTREE *
toollist(input)
char    **input;
{

#define  NEXT    01
#define  FORK    02

        char            *s = *input;
        SM_TOOLTREE     *listhead = NULL, *list = NULL;
        int             state = NEXT;

        /*
         * Get rid of leading white space
         */
        while (isspace(*s)) s++;

        while (*s) {
                /*
                 * Build the list by repeatedly calling toolspec
                 */
                if (state == NEXT) {
                        if (list == NULL)
                                listhead = list =
                                        addnext(list,toolspec(&s));
                        else
                                list = addnext(list,toolspec(&s));
                } else {
                        list = addfork(list,toolspec(&s));
                }

                if (t_error)
                        return (SM_TOOLTREE *)NULL;

                while (isspace(*s)) s++;

                switch (*s) {
                case    ',':
                        state = NEXT;
                        s++;
                        break;

                case    ']':
                        *input = s;
                        return  listhead;

                case    '|':
                        if (state == FORK) {
                                tool_error("Illegal protocol step syntax");
                                return (SM_TOOLTREE *)NULL;
                        }
                        state = FORK;
                        s++;
                        break;

                case    '\0':
                        break;

                default:
                        tool_error("Illegal protocol step syntax");
                        return (SM_TOOLTREE *)NULL;
                }
        }
        *input = s;
        return  listhead;
}

static SM_TOOLTREE *
toolspec(input)
char **input;
{
        char    *s = *input;
        char    *hosts;
        SM_TOOLTREE     *tool;

        /*
```

```
	 * Get rid of leading white space
	 */
	while (isspace(*s)) s++;

	/*
	 * Only two possibilities, either we have a '{', in
	 * which case we need to recurse, or we have a tool
	 * name.
	 */

	if (*s == '{') {
		s++;
		tool = toollist(&s);
		/* s should now point to a '}' */
		if (*s++ != '}' || t_error)
			return	NULL;
	} else {
		char *p = s;

		while (isalpha(*s) || isdigit(*s) ||
		    *s == '_' || *s == '.' || *s == '-') s++;
		tool = (SM_TOOLTREE *)emalloc(sizeof(SM_TOOLTREE));
		tool->tt_tool = (char *)emalloc((int)(s-p)+1);
		(void) strncpy(tool->tt_tool,p,(int)(s-p));
		*(tool->tt_tool+(s-p)) = '\0';		/* Make sure its null term
		tool->tt_next = (SM_TOOLTREE *)NULL;
		tool->tt_fork = (SM_TOOLTREE *)NULL;
		tool->tt_join = 0;
		tool->tt_tnum = ++tool_count;
		tool->tt_host = NULL;
	}

	*input = s;

	/*
	 * At this point, we should have a legitimate "tool".  Now
	 * check for a possible hostlist.  If we have one, update
	 * all empty "tt_hosts" fields in tool.
	 */

	if ((hosts = hostlist(input)) != NULL);
		hostupdate(tool,hosts);

	if (t_error) {
		tool_free(tool);
		return	(SM_TOOLTREE *)NULL;
	}
		return	tool;
}

static char *
hostlist(input)
char	**input;
{
		register char	*s = *input;
		register char	*t;

		/*
		 * Get rid of leading white space
		 */
		while (isspace(*s)) s++;

		/*
		 * If we've found a '{', we've found a host list, otherwise,
		 * don't update the string and just return
		 */

		if (*s != '{')
			return	NULL;

		s++;	/* Point past the '{' */
		t = strchr(s,'}');
		if (t == NULL) {
			tool_error("Illegal protocol host specification syntax");
			return NULL;
		}
		*t = '\0';
		/*
		 * Lets help a little by making sure that the 'host' string
		 * doesn't contain a '{','}', or a '|', which would indicate
		 * that we've got a syntax error
		 */
		if (strchr(s,'{') || strchr(s,'|') || strchr(s,'}')) {
			tool_error("Illegal protocol host specification syntax");
			return NULL;
		}
		*input = ++t;
		return	s;
}

static void
hostupdate(tool,hosts)
SM_TOOLTREE	*tool;
```

```
char    *hosts;
{
        while (tool != (SM_TOOLTREE *)NULL) {
                if (tool->tt_host == NULL)
                        tool->tt_host = mmwb_cpystring(hosts);
                if (tool->tt_fork != (SM_TOOLTREE *)NULL)
                        hostupdate(tool->tt_fork, hosts);
                tool = tool->tt_next;
        }
        return;
}

static  SM_TOOLTREE *
addnext(list,tool)
SM_TOOLTREE     *list;
SM_TOOLTREE     *tool;
{
        /*
         * Start with the easy part, add the tool on the list
         */
        if (list != (SM_TOOLTREE *)NULL) {
                /*
                 * Update the next pointers of the forked tools
                 */
                forkupdate(list,tool);

                /*
                 * OK, now update the list itself
                 */
                while (list->tt_next != (SM_TOOLTREE *)NULL)
                        list = list->tt_next;
                list->tt_next = tool;
        }
        return  tool;
}

static  SM_TOOLTREE *
addfork(list,tool)
SM_TOOLTREE     *list;
SM_TOOLTREE     *tool;
{
        SM_TOOLTREE     *l = list;

        /*
         * Add a fork to the current tool.  If a fork
         * already exists, then add a fork to the fork
         */
        while (list->tt_fork != (SM_TOOLTREE *)NULL)
                list = list->tt_fork;

        list->tt_fork = tool;

        return  l;
}

static void
forkupdate(list,tool)
SM_TOOLTREE     *list;
SM_TOOLTREE     *tool;
{
        SM_TOOLTREE     *l,*lf;

        l = list;
        while (l != (SM_TOOLTREE *)NULL) {
                if ((lf = l->tt_fork) != (SM_TOOLTREE *)NULL) {
                        if (lf->tt_next == (SM_TOOLTREE *)NULL)
                                lf->tt_next = tool;
                        tool->tt_join++;
                        forkupdate(lf,tool);
                }
                l = l->tt_next;
        }
        return;
}

void
tool_free(tool)
SM_TOOLTREE     *tool;
{
        SM_TOOLTREE     *t;

        while (tool != (SM_TOOLTREE *)NULL) {
                if (tool->tt_tool != NULL)
                        mmwb_stringfree(tool->tt_tool);
                if (tool->tt_host != NULL)
                        mmwb_stringfree(tool->tt_host);
                if (tool->tt_fork != (SM_TOOLTREE *)NULL)
                        tool_free(tool->tt_fork);
                t = tool->tt_next;
                (void)free((char *)tool);
                tool = t;
        }
        return;
}
```

sm/tool.c

```
static void
tool_error(errstr)
char	*errstr;
{
	t_errstr = errstr;
	t_error++;
	return;
}

/*
 * tool_printtree
 */

static void
tool_printtree(tree,level)
SM_TOOLTREE *tree;
int	level;
{
	char	tabs[180];
	int	i = level;

	tabs[0] = '\0';

	while (i--)
		(void) strcat(tabs,"    ");

	while (tree != (SM_TOOLTREE *)NULL) {
		syslog(LOG_DEBUG, "%sTool # %d",tabs,tree->tt_tnum);
		syslog(LOG_DEBUG, "%s  name: %s",tabs,tree->tt_tool);
		syslog(LOG_DEBUG, "%s  hosts: %s",tabs,tree->tt_host);
		syslog(LOG_DEBUG, "%s  join: %d",tabs,tree->tt_join);
		if (tree->tt_next != (SM_TOOLTREE *)NULL)
			syslog(LOG_DEBUG, "%s   next: %s",tabs,
				tree->tt_next->tt_tool);
		if (tree->tt_fork != (SM_TOOLTREE *)NULL)
			tool_printtree(tree->tt_fork,level+1);
		tree = tree->tt_next;
	}

	return;
}
```

libsm/sm.h

```
/*
 * $ID$
 *
 * sm.h - include file for libsm
 *
 * $Log:                sm.h,v $
 * Revision 1.21  89/08/07  12:42:43  scooter
 * Folded in session.h and changed session structure
 *
 * Revision 1.20  89/03/10  14:21:26  scooter
 * Various bug fixes to get everything working again
 *
 * Revision 1.19  89/03/09  14:09:59  scooter
 * Changed process to protocol
 *
 * Revision 1.18  89/03/02  22:10:28  scooter
 * Modified for new dialog semantics
 *
 * Revision 1.17  89/02/21  23:36:42  scooter
 * New dialog semantics
 *
 * Revision 1.16  88/11/10  16:51:56  gregc
 * sm_fetch_objs returns malloc'd NULL terminated list of objects opened.
 *
 * Revision 1.15  88/11/10  15:01:11  scooter
 * Add sm_fetch_objs to list of functions
 *
 * Revision 1.14  88/10/25  17:33:18  scooter
 * Changed sm_send_dialog to return the dialog_id.
 *
 * Revision 1.13  88/10/19  17:02:00  conrad
 * Use ofs.h and OFS_OBJECT rather than dm.h and DM_OBJECT
 *
 * Revision 1.12  88/10/14  13:02:03  gregc
 * add missing extern for sm_init().
 *
 * Revision 1.11  88/10/13  19:36:30  scooter
 * Added several new routines:
 *          sm_exit, sm_dettach_session, sm_notify, and sm_cancel
 * sm_perror was changed to sm_error.
 *
 * Revision 1.10  88/10/13  11:13:00  scooter
 * Added error E_SM_NOACK
 *
 * Revision 1.9  88/09/27  15:45:38  conrad
 * Added tool descriptions and sm_perror()
 *
 * Revision 1.8  88/09/24  20:40:07  scooter
 * Added routines to return the current triple of projects.
 *
 * Revision 1.7  88/09/23  14:48:55  conrad
 * Declare sm_set_file() properly
 *
 * Revision 1.6  88/09/20  14:12:55  conrad
 * Use file stream instead of file descriptor for connection to SM
 * Don't assume stdin is connection to SM
 *
 * Revision 1.5  88/09/19  22:13:05  scooter
 * Moved sm_send_reply into smui.c and defined a new
 * function: sm_send_reply_tool for tools which wish to
 * send replies.
 *
 * Revision 1.4  88/09/19  17:12:06  conrad
 * Give dispatcher errors their own error number
 *
 * Revision 1.3  88/09/15  23:02:56  scooter
 * Cleaned up proj interface
 *
 * Revision 1.2  88/09/01  15:25:28  conrad
 * Use machdep.h instead of #ifdef
 *
 * Revision 1.1  88/08/31  10:49:19  conrad
 * Initial revision
 *
 *
 */

#ifndef  INCLUDE_SM

#include        <stdio.h>
#include        <machdep.h>
#include        "dialog.h"
#include        "ofs.h"

/*
 * Version numbers
 */

#define  SM_MINVERS     1
#define  SM_MAXVERS     2

/*
 * The name of the Session Manager Service
 */
```

libsm/sm.h

```
#define  SM_SERVICE      "sm"

/*
 * Typedefs
 */

typedef  int        SM_TOOL;        /* A "tool" is just an accession number */

typedef  struct     sm_session_struct {
         char       *ss_name;       /* The name of the tool */
         SM_TOOL    ss_tool;        /* The ID of the tool */
         int        ss_state;       /* The state of the tool */
         int        ss_type;        /* The type of the tool */
         char       *ss_currp;      /* The current project */
         struct     sm_session_struct       *ss_next;       /* Next tool */
} SM_SESSION_STATUS;

typedef  struct     sm_ses_struct {
         char       *ses_name;      /* The name of the session */
         char       *ses_cid;       /* The Conversation ID of the session */
         char       *ses_user;      /* The user of the session */
         char       *ses_host;      /* The host the session is running on */
         char       *ses_date;      /* The date and time of the session */
         int        ses_sock;       /* The socket to rendezvous */
         int        ses_pending;    /* The count of pending dialogs */
         struct     sm_ses_struct   *ses_next;
} SM_SESSION;

typedef struct sm_tooldes {
         char       *Name;          /* Name of the tool */
         char       *Version;       /* Version number */
         int        nodisplay;      /* Display options ? */
         DIALOG     Options;        /* The option sheet */
         char       *Subtools;      /* Subtools (if any) */
         int        OnError;        /* If subtools, how to handle errors */
         char       *Host;          /* The hosts to execute */
         int        Type;           /* The tool type (1=process,2=instrument) */
         SM_TOOL    tool_id;        /* The resulting tool id */
         int        from_proj;      /* Which project tool was found */
         void       *Curr_Proj;     /* The currently selected project */
         void       *Home_Proj;     /* The home project */
         void       *Sys_Proj;      /* The system project */
         OFS_OBJECT obj;            /* Object containing this tool description */
         struct sm_tooldes          *next;
}        SM_TOOLDES;
```

```
#define  MAXSESSIONS    50      /* The maximum number of allowable sessions */

/*
 * Flags for Tool Descriptions - OnError
 */

#define  SM_IGNORE 1     /* Ignore errors */
#define  SM_DIE    2     /* Immediately die */
#define  SM_KILLB  3     /* Kill branches */

#define  SM_PROJ_CURR  1
#define  SM_PROJ_HOME  2
#define  SM_PROJ_SYS   3

/*
 * Functions
 */

extern   SM_SESSION              *sm_list_sessions();
extern   SM_SESSION_STATUS       *sm_session_status();
extern   FILE                    *sm_attach_session();
extern   int                     sm_dettach_session();
extern   int                     sm_register_tool();
extern   void                    sm_set_sys_proj();
extern   void                    sm_set_home_proj();
extern   void                    sm_set_curr_proj();
extern   OFS_OBJECT              sm_get_sys_proj();
extern   OFS_OBJECT              sm_get_home_proj();
extern   OFS_OBJECT              sm_get_curr_proj();
extern   void                    sm_set_file();

extern   SM_TOOLDES              *sm_read_protocol();
extern   SM_TOOLDES              *sm_read_instrument();
extern   SM_TOOLDES              *sm_list_protocol();
extern   SM_TOOLDES              *sm_list_instrument();
extern   int                     sm_write_protocol();
extern   int                     sm_write_instrument();
extern   void                    sm_free_tooldes();

extern   int                     sm_dispatch();

extern   DIALOG                  sm_read_reply();
extern   int                     sm_send_reply();
extern   int                     sm_send_reply_tool();
extern   int                     sm_send_dialog();
extern   SM_TOOLDES              *sm_invoke();
extern   int                     sm_launch();
```

libsm/sm.h

```
extern  int                     sm_cancel();
extern  int                     sm_kill();

extern  int                     sm_request_option();
extern  DIALOG                  sm_dialog();
extern  OFS_OBJECT              sm_init();
extern  OFS_OBJECT              sm_init_instrument();
extern  int                     sm_exit();
extern  int                     sm_notify();

extern  char                    *sm_error();

extern  OFS_OBJECT              *sm_fetch_objs();

/*
 * Globals
 */

extern  int                     sm_errno;
extern  char                    *sm_errlist[];
extern  FILE                    *sm_file;

/*
 * Errors
 */

#define  E_SM_NONE      0       /* No error */
#define  E_SM_IO        -1      /* I/O error */
#define  E_SM_OVERFLOW  -2      /* Internal buffer overflow */
#define  E_SM_PARAM     -3      /* Formal parameter error */
#define  E_SM_CONVERT   -4      /* (dialog) Conversion error */
#define  E_SM_PROT      -5      /* Protocol error */
#define  E_SM_NOOPTION  -6      /* Can't find requested option */
#define  E_SM_HANDSHAKE -7      /* Handshake error */
#define  E_SM_PROJINI   -8      /* Unable to initialize a project */
#define  E_SM_DISPATCH  -9      /* Dispatcher failed */
#define  E_SM_OFS       -10     /* Object File System failed */
#define  E_SM_BADTD     -11     /* Bad tool description */
#define  E_SM_NOACK     -12     /* Unable to ack message from SM */
#define E_SM_TYPEMIS    -13     /* Dialog type mismatch */

#define  INCLUDE_SM
#endif   INCLUDE_SM
```

libsm/fetchobjs.c

```
/*
 *      $Id: fetchobjs.c,v 1.14 89/10/25 14:26:56 gregc Exp $
 */

# include       "sm.h"
# include       "mmwbmisc.h"
# include       "ofs.h"

extern  char    *strchr(), *strcpy();
extern  void    *emalloc();

/*
 *      sm_fetch_objs:
 *
 *              Query user for objects, and return those which are selected.
 */

OFS_OBJECT      *
sm_fetch_objs(oh, name, class, type, min, max, prompt, help)
        OFS_OBJECT      oh;
        char            *name;
        int             class, type;
        int             min, max;
        char            *prompt, *help;
{
        DIALOG          d, d2;
        int             count, i;
        OFS_OBJECT      *result_objs;
        OFS_OBJ_DESC    *desc;
        int             len, err;
        char            *s, *sellist;
        int             upper, lower;

        /* 8 == fudge factor due to race between obj_count and obj_list */
        i = obj_count(oh, name, class, type) + 8;
        desc = (OFS_OBJ_DESC *) emalloc(i * sizeof (OFS_OBJ_DESC));
        count = obj_list(oh, name, class, type, desc, i);
        if (count == -1 && dm_errno == DM_ETOOSMALL)
                count = i;
        if (count <= 0) {
                free((char *) desc);
                sm_errno = E_SM_OFS;
                return NULL;
        }

        d = dialog_create(DI_SELECT,0,
                          mmwb_cpystring("object select"), (char *)NULL);
        (void) dialog_set_text(d, DI_V_HELP, mmwb_cpystring(help));
        (void) dialog_set_flag(d, DI_OPTIONAL, 0);

        if (max < 1)
                upper = count;
        else
                upper = max;

        (void) dialog_set_int(d, DI_V_UBOUND, upper);
        result_objs = (OFS_OBJECT *) emalloc((upper + 1) * sizeof (OFS_OBJECT));

        if (min < 0)
                lower = 1;
        else
                lower = min;

        (void) dialog_set_int(d, DI_V_LBOUND, lower);

        if (prompt != NULL)
                (void) dialog_set_text(d, DI_V_PROMPT, mmwb_cpystring(prompt));
        else {
                static  char    buf[64];

                (void) sprintf(buf, "Pick from %d to %d objects:",
                                lower, upper);
                (void) dialog_set_text(d, DI_V_PROMPT, mmwb_cpystring(buf));
        }

        sellist = NULL;
        for (i = 0, len = 0; i < count; i += 1) {
                int     slen;

                s = stringize(desc[i].name);
                slen = strlen(s);
                if (len == 0) {
                        sellist = (char *) emalloc(slen + 1);
                        (void) strcpy(sellist, s);
                        len = slen;
                } else {
                        sellist = (char *) erealloc(sellist, len + slen + 2);
                        sellist[len] = ' ';
                        (void) strcpy(sellist + len + 1, s);
                        len += slen + 1;
                }
        }

        (void) dialog_set_text(d, DI_V_SELLIST, sellist);
```

libsm/fetchobjs.c

```
try_again:
        if ((d2 = sm_dialog(d)) == NULL) {
                free((char *) result_objs);
                result_objs = NULL;
                goto done;
        }

        s = dialog_get_text(d2,DI_V_RESULT,&err);
        for (count = 0; s != NULL; count += 1) {
                i = atoi(s);
                result_objs[count] = obj_open_by_ident(oh, desc[i].ident);
                if (result_objs[count] == NULL) {
                        char    buf[128];

                        (void) sprintf(buf, "%.96s disappeared", desc[i].name);
                        for (i = 0; i < count; i += 1)
                                (void) obj_close(result_objs[i]);
                        dialog_free(d2,0);
                        (void) sm_notify(buf);
                        goto try_again;
                }
                if ((s = strchr(s, ' ')) != NULL)
                        s += 1;
        }
        result_objs[count] = NULL;              /* NULL terminate list */

done:
        dialog_free(d,0);
        dialog_free(d2,0);
        free((char *) desc);

        return result_objs;
}
```

libsm/sm.c

```
#ifndef  lint
static   char *RCSid = "$Id: sm.c,v 1.24 89/10/25 14:26:52 gregc Exp Locker: scoot
#endif   lint

/*
 * sm:
 *      general library routines for the interface to the
 *      session manager
 *
 */
#include         "sm.h"
#include         "mmwbmisc.h"
#include         "ofs.h"
#include         "dispatcher.h"
#include         <stdio.h>

#define  FAILURE -1
#define  SUCCESS 0

extern   void     *emalloc();

int      sm_errno = E_SM_NONE;          /* Global error variable */
FILE     *sm_file = NULL;               /* File stream to session manager */

static   DIALOG  sm_readd();
static   char    msg_buf[BUFSIZ];
static   char    resp_buf[BUFSIZ];
static   char    *toks[20];

/*
 * Storage for currently selected projects
 */
static   OFS_OBJECT     sm_curr_proj;
static   OFS_OBJECT     sm_home_proj;
static   OFS_OBJECT     sm_sys_proj;

/*
 * sm_read_reply:
 *      Read the reply to a dialog.  Each element of the dialog
 *      is sent preceded by the keyword "REPLY" and terminated
 *      by a newline.  A "." on a line by itself indicates the
 *      the end of the dialog.
 */
DIALOG
sm_read_reply(id)
int      id;
{
        (void) sprintf(resp_buf,"READ_REPLY %d\n",id);
        return   sm_readd(resp_buf,"REPLY");
}

/*
 * sm_read_dialog:
 *      Read a dialog.  Each element of the dialog
 *      is sent preceded by the keyword "DIALOG" and terminated
 *      by a newline.  A "." on a line by itself indicates the
 *      the end of the dialog.
 */
DIALOG
sm_read_dialog()
{
        (void) sprintf(resp_buf,"READ_DIALOG\n");
        return   sm_readd(resp_buf,"DIALOG");
}


/*
 * sm_readd:
 *      General dialog reader
 */
static   DIALOG
sm_readd(command,keyword)
char     *command,*keyword;
{
        DIALOG  dp,dq;
        extern   char *strchr();
        int      n;

        dq = NULL;

        sm_errno = E_SM_NONE;

        /*
         * Send the command
         */
        if ((n = write(fileno(sm_file), command, strlen(command)+1)) <= 0) {
                sm_errno = E_SM_IO;
                return   NULL;
        }

        /*
         * Get the response (better be a 230!)
         */
        if ( fgets (msg_buf, BUFSIZ, sm_file) == NULL) {
```

```
                sm_errno = E_SM_IO;
                return NULL;
        }

        if ( atoi(msg_buf) != 230 ) {
                sm_errno = E_SM_PROT;
                return NULL;
        }

        /*
         * ASCII Data ??
         */
        if ( fgets (msg_buf, BUFSIZ, sm_file) == NULL) {
                sm_errno = E_SM_IO;
                return NULL;
        }

        if ( atoi(msg_buf) != 202 ) {
                sm_errno = E_SM_PROT;
                return NULL;
        }

        /*
         * Great, read the dialog(s)
         */
        while (1) {

                if (fgets(msg_buf, BUFSIZ, sm_file) == NULL) {
                        sm_errno = E_SM_IO;
                        return NULL;
                }

                n = tokenize(msg_buf,toks,2);

                if (strcmp(toks[0],keyword) == 0) {
                        if (n != 2) {
                                sm_errno = E_SM_CONVERT;
                                /* Should clean up built list */
                                return NULL;
                        }
                        if (dq == NULL)
                                dq = dp = dialog_from_char(toks[1]);
                        else {
                                dp =
                                 dialog_append(dp,dialog_from_char(toks[1]))
                                dp = dialog_next(dp);
                        }
                } else if (strcmp(toks[0],".") == 0)
                        break;
        }

        /*
         * End of reply?
         */
        if ( fgets (msg_buf, BUFSIZ, sm_file) == NULL) {
                sm_errno = E_SM_IO;
                return NULL;
        }

        if ( atoi(msg_buf) != 600 ) {
                sm_errno = E_SM_PROT;
                return NULL;
        }
        return  dq;
}

/*
 * sm_send_dialog:
 *      Send a dialog to the session manager for delivery to the
 *      UIM.
 */
int
sm_send_dialog (dp)
DIALOG  dp;
{
        int                     result;
        static int              sm_sendd();

        result = sm_sendd(dp, "DIALOG");
        if (result)
                return  result;
        else
                return  dialog_id(dp);
}

/*
 * sm_send_reply_tool:
 *      Send a reply to the session manager for delivery to another
 *      tool. This differs from sm_send_reply in that no checking
 *      is done for option sheets.
 */
int
sm_send_reply_tool (dp)
DIALOG  dp;
```

```
{
        static int          sm_sendd();

        return    sm_sendd(dp, "REPLY");
}

/*
 * sm_sendd:
 *      The main dialog delivery routine
 */
static    int
sm_sendd (dp, command)
DIALOG  dp;
char    *command;
{
        DIALOG  d;
        char    *ds;

        sm_errno = E_SM_NONE;

        if (dp == NULL) {
                sm_errno = E_SM_PARAM;
                return sm_errno;
        }

        for (d = dp; d != NULL; d = dialog_next(d)) {

                if ( (ds = dialog_to_char(d)) == NULL) {
                        sm_errno = E_SM_CONVERT;
                        return sm_errno;
                }

                (void) sprintf(msg_buf,"%s %s\n",command,stringize(ds));
                mmwb_stringfree(ds);

                if ( (sm_errno = mmwb_command(sm_file, msg_buf, resp_buf, BU
                        return sm_errno;
        }

        (void) sprintf(msg_buf, ".\n");
        if ( (sm_errno = mmwb_command(sm_file, msg_buf, resp_buf, BUFSIZ) ) <
                return sm_errno;
        (void) dialog_set_id(dp, atoi(resp_buf));
        return    SUCCESS;
}

/*
 * sm_session_status:
 *      list the status and ID's of all running tools
 *
 */
SM_SESSION_STATUS *
sm_session_status()
{
        SM_SESSION_STATUS       *status,*next_status;
        char                    *s,*p;
        extern    char          *strchr();

        sm_errno = E_SM_NONE;

        if ((sm_errno = mmwb_command(sm_file, "STATUS\n", resp_buf, BUFSIZ)) < 0)
                return NULL;
        sm_errno = E_SM_NONE;    /* 0 or greater means no errors */

        s = p = &resp_buf[0];

        next_status = status = NULL;

        while ( (s = strchr(p,'\n')) != NULL) {
                *s++ = '\0';
                if (tokenize(p,toks,6) != 5) {
                        sm_errno = E_SM_PROT;
                        return NULL;
                }

                if (next_status == NULL) {
                  next_status = status =
                                (SM_SESSION_STATUS *)emalloc(sizeof (SM_SESSION_S
                } else {
                        next_status->ss_next =
                                (SM_SESSION_STATUS *)emalloc(sizeof (SM_SESSION_S
                        next_status = next_status->ss_next;
                }
                if (next_status == (SM_SESSION_STATUS *)NULL)
                        return (SM_SESSION_STATUS *)NULL;
                next_status->ss_name = mmwb_cpystring(toks[0]);
                next_status->ss_tool = (SM_TOOL) atoi(toks[1]);
                next_status->ss_state = atoi(toks[2]);
                next_status->ss_type = atoi(toks[3]);
                next_status->ss_currp = mmwb_cpystring(toks[4]);
                next_status->ss_next = NULL;
                p = s;
        }
        return status;
```

```
}

/*
 * sm_kill:
 *          send a kill request to the session manager
 */
int
sm_kill (tool)
SM_TOOLtool;
{

        (void) sprintf( msg_buf, "KILL %d\n", tool);

        if ( (sm_errno = mmwb_command(sm_file, msg_buf, resp_buf, BUFSIZ) ) <
                return  sm_errno;
        else
                return  SUCCESS;
}

/*
 * sm_launch:
 *          Send a request to the session manager to launch the
 *          specified tool.  If the launch succeeds, send the
 *          options as a reply.
 */
int
sm_launch (tooldes)
SM_TOOLDES      *tooldes;
{
        int         n,option_id;

        (void) sprintf( msg_buf,
        "LAUNCH \"%s\" \"%s\" \"%s\" %d \"%s\" %d %d \"%s\" \"%s\" \"%s\"\n",
                tooldes->Name,
                tooldes->Version,
                tooldes->Subtools,
                tooldes->OnError,
                tooldes->Host,
                tooldes->nodisplay,
                tooldes->Type,
                proj_id(sm_sys_proj),
                proj_id(sm_home_proj),
                proj_id(sm_curr_proj));

        if ( (sm_errno = mmwb_command(sm_file, msg_buf, resp_buf, BUFSIZ) ) <
                return  sm_errno;

        n = tokenize (resp_buf,toks,2);

        if (n != 2) {
                sm_errno = E_SM_PROT;
                return  sm_errno;
        }

        tooldes->tool_id = atoi(toks[0]);

        /*
         * OK, now the tool is launched, send it its options
         */
        if (tooldes->Options == NULL)
                tooldes->Options = dialog_create(DI_TEXT,0,
                                                 "Dummy","Ignore me");
        option_id = atoi(toks[1]);
        (void) dialog_set_id(tooldes->Options, option_id);
        (void) dialog_set_source(tooldes->Options, tooldes->tool_id);

        if (sm_send_reply_tool(tooldes->Options) < 0)
                return  sm_errno;
        else
                return  SUCCESS;
}

/*
 * sm_set_sys_proj:
 *          Set the system project.
 */
void
sm_set_sys_proj (proj)
OFS_OBJECT      proj;
{
        sm_sys_proj = proj;
        return;
}

/*
 * sm_get_sys_proj:
 *          Return the current system project.
 */
OFS_OBJECT
sm_get_sys_proj()
{
        return sm_sys_proj;
}
```

libsm/sm.c

```
/*
 * sm_set_home_proj:
 *          Set the home project.
 */
void
sm_set_home_proj (proj)
OFS_OBJECT      proj;
{
        sm_home_proj = proj;
        return;
}

/*
 * sm_get_home_proj:
 *          Return the current home project.
 */
OFS_OBJECT
sm_get_home_proj()
{
        return sm_home_proj;
}

/*
 * sm_set_curr_proj:
 *          Set the current project.
 */
void
sm_set_curr_proj (proj)
OFS_OBJECT      proj;
{
        sm_curr_proj = proj;
        return;
}

/*
 * sm_get_curr_proj:
 *          Return the current project.
 */
OFS_OBJECT
sm_get_curr_proj()
{
        return sm_curr_proj;
}

/*
 * sm_set_file:
 *          Set file stream to session manager
 */
void
sm_set_file (fd)
FILE    *fd;
{
        sm_file = fd;
        return;
}

/*
 * sm_errlist:
 *          List of printable error messages
 */
char    *sm_errlist[] = {
        "no error",
        "I/O error",
        "internal buffer overflow",
        "formal parameter error denied",
        "(dialog) conversion error",
        "protocol error",
        "cannot find requested option",
        "handshake error",
        "unable to initialize project",
        "dispatcher error",
        "object file system error",
        "bad tool description",
        "unable to acknowledge message from sm",
        "type mismatch in dialog"
};

#define  MAX_SMERR      (sizeof sm_errlist / sizeof sm_errlist[0])

/*
 * sm_perror:
 *          Print error message
 */
char *
sm_error()
{
        int         error_no;

        error_no = -sm_errno;
        if (error_no < 0 || error_no >= MAX_SMERR) {
                (void) sprintf(msg_buf,"Error %d\n",error_no);
                return      msg_buf;
        } else {
                switch (sm_errno) {
```

```
                case E_SM_DISPATCH:
                        return dispatcher_error();
                case E_SM_OFS:
                        return dm_error();
                default:
                        return sm_errlist[error_no];
                }
        }
}
```

libsm/smt.c

```
#ifndef lint
static char *RCSid = "$Id: smt.c,v 1.18 89/10/25 14:26:53 gregc Exp $";
#endif lint

/*
 * smt:
 *      library routines for the sm interface to tools
 *
 */

#include        <pwd.h>
#include        <sys/types.h>
#include        "sm.h"
#include        "dispatcher.h"
#include        "ofs.h"
#include        "mmwbmisc.h"

static  int     sm_version;
static  DIALOG  sm_options;

static  char    resp_buf[BUFSIZ];

#define SUCCESS 0
#define INSTRUMENT      1
#define PROTOCOL        2

static  OFS_OBJECT      sm_init_internal();

/*
 * sm_init:
 *      initializes the interface to the session manager.  this
 *      routine takes care of all of the handshaking, and waits
 *      for the option sheet.  The option sheet is kept in a cache
 *      for calls to sm_option.
 *
 *      NOTE: we assume we can use file descriptor 0 for all I/O
 */
OFS_OBJECT
sm_init (cid)
char    *cid;
{
        return  sm_init_internal(cid,(int *)NULL,PROTOCOL);
}

/*
 * sm_init_protocol
 */
OFS_OBJECT
sm_init_instrument(cid,semlist)
char    *cid;
int     *semlist;
{
        return  sm_init_internal(cid,semlist,INSTRUMENT);
}


static OFS_OBJECT
sm_init_internal (cid,semlist,type)
char    *cid;
int     *semlist;
int     type;
{
        struct  passwd  *pwd, *getpwuid();
        extern  UIDTYPE getuid();
        extern  char    *strchr();
        OFS_OBJECT      dmo, curr_proj = NULL;
        char            *s,*p;
        char            *toks[4];

        sm_errno = E_SM_NONE;
        pwd = getpwuid(getuid());

        /*
         * Take care of all of the handshaking
         */
        sm_set_file(stdin);
        if ((sm_errno = mmwb_handshake(sm_file, pwd->pw_name, cid, SM_MINVERS,
                SM_MAXVERS, &sm_version)) < 0) {
                        if (sm_errno == MMWB_HS_EPROTO)
                                sm_errno = E_SM_PROT;
                        else
                                sm_errno = E_SM_HANDSHAKE;
                        return NULL;
        }

        /*
         * Send our type
         */
        if (type == INSTRUMENT)
                sm_errno =
                        mmwb_command(sm_file, "TYPE INSTRUMENT\n", resp_buf, BUFSIZ
        else
                sm_errno =
                        mmwb_command(sm_file, "TYPE PROTOCOL\n", resp_buf, BUFSIZ);
```

libsm/smt.c

```
        if (sm_errno) {
                if (sm_errno == MMWB_HS_EPROTO)
                        sm_errno = E_SM_PROT;
                else
                        sm_errno = E_SM_HANDSHAKE;
                return  NULL;
        }

        /*
         * Last step in the handshaking, get our project information
         */
        if ((sm_errno = mmwb_command(sm_file, "PROJECTS\n", resp_buf, BUFSI
                if (sm_errno == MMWB_HS_EPROTO)
                        sm_errno = E_SM_PROT;
                else
                        sm_errno = E_SM_HANDSHAKE;
                return  NULL;
        }

        s = p = resp_buf;
        while ( (s = strchr(p,'\n')) != NULL) {
                *s = '\0';
                tokenize(p, toks, 2);
                if ( strcmp (toks[0], "SYSTEM") == 0) {
                        dmo = ofs_sys_init((char *)toks[1]);
                        if (dmo == NULL) {
                                sm_errno = E_SM_PROJINI;
                                return  NULL;
                        }
                        sm_set_sys_proj(dmo);
                } else if ( strcmp (toks[0], "HOME") == 0) {
                        dmo = ofs_proj_init((char *)toks[1]);
                        if (dmo == NULL) {
                                sm_errno = E_SM_PROJINI;
                                return  NULL;
                        }
                        sm_set_home_proj(dmo);
                } else if ( strcmp (toks[0], "CURRENT") == 0) {
                        dmo = ofs_proj_init((char *)toks[1]);
                        if (dmo == NULL) {
                                sm_errno = E_SM_PROJINI;
                                return  NULL;
                        }
                        sm_set_curr_proj(dmo);
                        curr_proj = dmo;
                } else if ( strcmp (toks[0], ".") == 0)
                        break;
                else {
                        sm_errno = E_SM_HANDSHAKE;
                        return  NULL;
                }
                p = ++s;
        }

        /*
         * Great, now read the option sheet
         */
        if ( (sm_options = sm_read_reply(-1)) == NULL)
                return NULL;

        /*
         * Now, if the semlist is non-null, send a semantic, notification
         * dialog telling the UIM what we're interested in
         */
#define D_SEM_SEMLIST 1
        if (semlist != NULL && *semlist != -1) {
                DIALOG  d = dialog_create(DI_NOTE, 0, "Semantic Dialog List",
                                (char *)NULL);
                int     result;
                char    buf[32];
                extern  char *strcat();

                (void) sprintf(resp_buf,"%d",*semlist++);
                while (*semlist != -1) {
                        (void) sprintf(buf,",%d",*semlist++);
                        (void) strcat(resp_buf,buf);
                }
                if (dialog_set_semantics(d,D_SEM_SEMLIST,resp_buf) < 0) {
                        sm_errno = E_SM_TYPEMIS;
                        return  NULL;
                }
                result = sm_send_dialog(d);
                dialog_free(d,0);
                if (result < 0)
                        return  NULL;
        }
        return curr_proj;
}

/*
 * sm_request_option:
 *      This routine returns the value associated with the
 *      requested (named) option.
 */
```

libsm/smt.c

```
int
sm_request_option (name,value,type)
char            *name;
void            *value;
int             type;
{
        DIALOG          dp = sm_options;
        static int      sm_get_opt();

        sm_errno = E_SM_NONE;
        while (dp != NULL) {
                if (strcmp(dialog_name(dp),name) == 0) {
                        if (sm_get_opt(dp, value, type) < 0) {
                                sm_errno = E_SM_TYPEMIS;
                                return  sm_errno;
                        } else
                                return SUCCESS;
                }
                dp = dialog_next(dp);
        }
        sm_errno = E_SM_NOOPTION;
        return  sm_errno;
}

static int
sm_get_opt(dp, value, type)
DIALOG  dp;
void    *value;
int     type;
{
        int     err;

        switch(type) {
        case    DI_IVAL:
        case    DI_SELECT:
                *((int *) value) = dialog_get_int(dp,DI_V_RESULT,&err);
                break;
        case    DI_FVAL:
                *((float *) value) = dialog_get_float(dp,DI_V_RESULT,&err);
                break;
        case    DI_TEXT:
                *((char **) value) = dialog_get_text(dp,DI_V_RESULT,&err);
                break;
        case    DI_YESNO:
        case    DI_BOOL:
                *((int *) value) = dialog_get_bool(dp,DI_V_RESULT,&err);
                break;
        case    DI_NOTE:
                err = 0;
                break;
        }

        if (err < 0)
                return E_SM_TYPEMIS;
        else
                return  SUCCESS;
}


/*
 * sm_dialog:
 *      This routine is called by a tool to send a dialog and
 *      then wait for the reply.
 */
DIALOG
sm_dialog (d)
DIALOG          d;
{
        int     did;
        sm_errno = E_SM_NONE;

        if (d == NULL) {
                sm_errno = E_SM_PARAM;
                return  NULL;
        }

        if ((did = sm_send_dialog(d)) < 0)
                return  NULL;

        if (dialog_type(d) == DI_NOTE)
                return  NULL;

        return (sm_read_reply(did));
}

/*
 * sm_exit:
 *      Processes call this to exit.  If the argument is non-null
 *      a notification dialog is sent to the user.
 */

int
sm_exit(msg)
char    *msg;
```

```
{
        if (msg != NULL && sm_notify(msg) < 0)
                return sm_errno;

        if ((sm_errno = mmwb_command(sm_file, "QUIT\n", resp_buf, BUFSIZ)) < 0
                sm_errno = E_SM_PROT;
                return   sm_errno;
        }

        return   SUCCESS;
}

/*
 * sm_notify:
 *      send a notification dialog to the user
 */
int
sm_notify(msg)
char    *msg;
{
        DIALOG  d;
        int     result;

        if (msg == NULL)
                return   SUCCESS;
        d = dialog_create(DI_NOTE, 0, (char *)NULL, msg);
        result = sm_send_dialog(d);
        dialog_free(d,0);
        if (result < 0)
                return   result;
        else
                return   SUCCESS;
}
```

libsm/smui.c

```
#ifndef  lint
static   char *RCSid = "$Id: smui.c,v 1.28 89/10/25 14:26:21 scooter Exp Locker: sc
#endif   lint

/*
 * smui:
 *      library routines for the sm interface to UIMs
 *
 */

#include        <stdio.h>
#include        <sys/types.h>
#include        <sys/socket.h>
#include        <sys/uio.h>
#include        <netinet/in.h>
#include        <arpa/inet.h>
#include        <netdb.h>
#include        <pwd.h>
#include        "sm.h"
#include        "ofs.h"
#include        "mmwbmisc.h"
#include        "dispatcher.h"

#define  SUCCESS 0

extern   void    *emalloc();
extern   char    *strcpy();

static   char    msg_buf[BUFSIZ];
#ifndef htons
extern   u_short htons();
#endif

static   DISPATCHER_SERVER      sm_server;

/*
 * sm_attach_session:
 *      Attach to a session.  If the argument is NULL, then
 *      start a new session.
 *
 */
FILE *
sm_attach_session (session, host)
SM_SESSION      *session;
char            *host;
{
        static FILE         *start_session();
        static FILE         *reattach_session();

        sm_errno = E_SM_NONE;
        /* Tell mmwb_command that we're a UIM */
        mmwb_uim();
        if (session == NULL)
                return  start_session(host);

        /*
         * Use the socket to rendezvous and handshake
         */
        return  reattach_session(session->ses_host, session->ses_sock,
                        session->ses_cid, session->ses_user);
}

/*
 * start_session:
 *      initiate a session.  Form a CID from the host and the
 *      response to ctime, then request a connection to a Session
 *      Manager.  Use the standard mmwb routines to handshake with
 *      the session manager.
 */
#include        <sys/time.h>

static   FILE *
start_session(thost)
char    *thost;
{
        char     cid[128];
        char     host[64];
        char    *time,*s,*strchr(),*getenv();
        struct   timeval    tv;
        struct   timezone tz;
        static FILE         *start_sm();
        FILE    *fd;
        char    user[20];
        int      version;
        struct   hostent   *h;
        UIDTYPE getuid();
        struct passwd *p = getpwuid((int) getuid());

        sm_errno = E_SM_NONE;
        if (thost == NULL && (thost = getenv("MMWB")) == NULL)
                (void) gethostname(host, 64);
        else
                (void) strcpy(host, thost);
```

```
        h = gethostbyname(host);
        if (h != NULL)
                (void) strcpy(host,h->h_name);
        (void) strcpy(user, p->pw_name);

        (void) gettimeofday(&tv, &tz);

        time = ctime(&tv.tv_sec);

        s = strchr(time,'\n');
        if (s != NULL) *s = '\0';

        (void) sprintf(cid, "%s:%s@%s",user,time,host);

        /*
         * OK, now start SM
         */
        if ( (fd = start_sm(host, user, cid, SM_SERVICE)) == NULL)
                return  NULL;

        /*
         * Now handshake
         */
        if ((sm_errno = mmwb_handshake(fd, user, cid, SM_MINVERS, SM_MAXV
                if (sm_errno == MMWB_HS_EPROTO)
                        sm_errno = E_SM_PROT;
                else
                        sm_errno = E_SM_HANDSHAKE;
                return  NULL;
        }

        sm_set_file(fd);
        return  fd;
}

static  FILE *
start_sm(host, account, cid, service)
char *host, *account, *cid, *service;
{
        int     fd;

        if ((sm_server = dispatcher_call(host, account, cid, service)) == NULL){
                sm_errno = E_SM_DISPATCH;
                return  NULL;
        }
        fd = dispatcher_server_fd(sm_server);
        return  fdopen(fd,"r+");
}

/*
 * sm_list_sessions:
 *      Returns a linked list of the active sessions in the user's
 *      home project.
 */
SM_SESSION *
sm_list_sessions()
{
        OFS_OBJECT      proj,session;
        OFS_OBJ_DESC    list[MAXSESSIONS];
        int             nsessions,port,pending;
        SM_SESSION      *ses_list = NULL, *ses_start;
        SM_SESSION      *ses;
        static SM_SESSION       *session_create();
        register        int     i;

    /*
     * Begin by openning the Home project
     */
        if ( (proj = sm_get_home_proj()) == NULL
    &&  (proj = ofs_proj_init((char *)NULL)) == NULL) {
                sm_errno = E_SM_IO;
        return          NULL;
    }

        if ( (nsessions = obj_list(proj, (char *) NULL, -1, OBJ_T_SESSION,
            list, MAXSESSIONS)) <= 0) {
                if (nsessions < 0)
                        sm_errno = E_SM_IO;
        return          NULL;
    }

        ses_start = NULL;
        for (i = 0; i < nsessions; i++) {
                /*
                 * Open the object
                 */
                if ( (session = obj_open_by_ident(proj,list[i].ident)) == NULL)
                        continue;

                /*
                 * Read the properties
                 */
                if ( prop_get(session, "Port", (void *) msg_buf) < 0 )
```

```
                continue;

        port = atoi(msg_buf);

        if ( prop_get(session, "Pending", (void *) msg_buf) < 0 )
                continue;

        pending = atoi(msg_buf);

        if ( prop_get(session, "CID", (void *) msg_buf) < 0 )
                continue;

        /*
         * Create the structure and link it in
         */
        ses = session_create(msg_buf,list[i].name,port,pending);
        if ( ses_list == NULL ) {
                ses_start = ses_list = ses;
        } else {
                ses_list->ses_next = ses;
                ses_list = ses;
        }

        /*
         * Close the object
         */
        (void) obj_close(session);
    }
    return  ses_start;
}

/*
 * session_create:
 *      Create a session structure and fill it with the arguments.
 */
static  SM_SESSION *
session_create (cid, name, port, pending)
char    *cid;
char    *name;
int     port,pending;
{
        SM_SESSION      *ses;
        char            *s,*strchr();

        ses = (SM_SESSION *)emalloc(sizeof (SM_SESSION));

        ses->ses_sock = port;
        ses->ses_pending = pending;
        ses->ses_cid = mmwb_cpystring(cid);
        ses->ses_name = mmwb_cpystring(name);
        if ( (s = strchr(cid,':')) != NULL) {
                *s = '\0';
                ses->ses_user = mmwb_cpystring(cid);
                cid = ++s;
        }
        if ( (s = strchr(cid,'@')) != NULL) {
                *s = '\0';
                ses->ses_date = mmwb_cpystring(cid);
                ses->ses_host = mmwb_cpystring(++s);
        }

        ses->ses_next = NULL;

        return  ses;
}

/*
 * sm_dispatch:
 *      This routine is called from the user interface manager
 *      when data is detected on the channel to the session manager.
 *      Its main function is to read incoming dialogs (the only thing
 *      currently that can be sent by the SM) and call uim_dialog
 *      to process them.  uim_dialog is assumed to be of type
 *      void and takes the dialog as its only argument.
 */
int
sm_dispatch ()
{
        DIALOG  dq,dp;
        char    *s;
        extern  char *strchr();
        extern  int     uim_dialog();
        char    *toks[20];
        int     n;

        dq = NULL;
        sm_errno = E_SM_NONE;

        for (;;) {
                if ( fgets (msg_buf, sizeof msg_buf, sm_file) == NULL) {
                        sm_errno = E_SM_IO;
                        return  sm_errno;
                }
```

```
                if ( (s = strchr(msg_buf,'\n')) != NULL)
                        *s = '\0';
                else {
                        sm_errno = E_SM_OVERFLOW;
                        return  sm_errno;
                }

                n = tokenize(msg_buf,toks,20);

                if (strcmp(toks[0],"DIALOG") == 0) {
                        if (n != 2) {
                                sm_errno = E_SM_CONVERT;
                                /* Should clean up built list */
                                return  sm_errno;
                        }
                        if (dq == NULL)
                                dq = dp = dialog_from_char(toks[1]);
                        else {
                                dp =
                                  dialog_append(dp,dialog_from_char(toks[1])
                                dp = dialog_next(dp);
                        }
                } else if (strcmp(toks[0],".") == 0) {
                        /* We've got the dialog, now ack it.  Note
                           that we must check to make sure that the
                           UIM didn't generate it first */
                        if ( (dialog_source(dq) != 0) &&
                             ((sm_errno = sm_ack_dialog(dq)) != E_SM_NONE) )
                                return  sm_errno;
                        return  uim_dialog(dq);
                } else {
                        sm_errno = E_SM_PROT;
                        return  sm_errno;
                }
        }
        /* NOTREACHED */
}

/*
 * sm_invoke:
 *      This routine is called by the UIM to request the invocation
 *      of a tool with full option sheet handling.
 */
#define  MAXPENDING     50      /* Exceedingly large */

static   SM_TOOLDES     *tool_queue[MAXPENDING];

SM_TOOLDES *
sm_invoke (td)
SM_TOOLDES      *td;
{
        static SM_TOOLDES       *cpytool();
        DIALOG  dp;
        int     i;

        if (td->nodisplay || td->Options == NULL) {
                if (sm_launch(td) < 0)
                        return  NULL;
                else
                        return  td;
        }

        /*
         * Find the first free slot in the table
         */
        for (i = 0; i < MAXPENDING; i++){
                if (tool_queue[i] == NULL) {
                        tool_queue[i] = cpytool(td);
                        dp = tool_queue[i]->Options;
                        (void) dialog_set_source(dp,0);
                        (void) dialog_set_dest(dp,i);
                        (void) dialog_set_text(dp,DI_V_TOOL,
                                mmwb_cpystring(tool_queue[i]->Name));
                        uim_dialog (tool_queue[i]->Options);
                        return  tool_queue[i];
                }
        }
        return  NULL;
}


/*
 * sm_send_reply:
 *      Send a reply to the session manager for delivery to a
 *      tool, and check for possible queued tool descriptions.
 */
int
sm_send_reply (dp)
DIALOG  dp;
{
        static SM_TOOLDES       *sm_checktq();
        SM_TOOLDES      *td;

        /*
```

```
		 * Handle special case of an Option sheet which has now been
		 * satisfied for a tool launch.
		 */
		if (dialog_source(dp) == 0) {
			if ((td = sm_checktq(dp)) != NULL)
				return	sm_launch (td);
		}
		return	sm_send_reply_tool (dp);
}

/*
 * sm_cancel:
 *	Cancel a pending tool invocation.
 */
int
sm_cancel(dp)
DIALOG	dp;
{
		static SM_TOOLDES		*sm_checktq();
		SM_TOOLDES	*td;

		if ((td = sm_checktq(dp)) != NULL)
			sm_free_tooldes(td);
		return	SUCCESS;
}

/*
 * sm_checktq:
 *	Check the tool description queue for existance of a
 *	specific Option
 */
static SM_TOOLDES *
sm_checktq (option)
DIALOG	option;
{
		SM_TOOLDES *td;

		if (dialog_source(option) != 0 || dialog_dest(option) > MAXPENDING)
			return	NULL;

		td = tool_queue[dialog_dest(option)];
		tool_queue[dialog_dest(option)] = NULL;
		return	td;
}

/*
 * cpytool:
 *	Make a copy of a tool description.
 */
static	SM_TOOLDES *
cpytool (td)
SM_TOOLDES	*td;
{
		SM_TOOLDES	*new_td;

		new_td = (SM_TOOLDES *)emalloc(sizeof (SM_TOOLDES));

		new_td->Name = mmwb_cpystring(td->Name);
		new_td->Version = mmwb_cpystring(td->Version);
		new_td->nodisplay = td->nodisplay;
		new_td->Options = dialog_copy (td->Options,0);
		new_td->Subtools = mmwb_cpystring(td->Subtools);
		new_td->OnError = td->OnError;
		new_td->Host = mmwb_cpystring(td->Host);
		new_td->tool_id = td->tool_id;
		new_td->Home_Proj = new_td->Curr_Proj = NULL;
		new_td->Type = td->Type;
		return	new_td;
}

/*
 * reattach_session:
 *	reattach to the session specified by the host, port pair and
 *	handshake with it using the cid and user.
 */
static FILE *
reattach_session(host,port,cid,user)
char	*host;
int	port;
char	*cid;
char	*user;
{
		struct	hostent	*h;
		struct	sockaddr_in		sin;
		int	s,version;
		FILE	*fd;

		if ( (h = gethostbyname(host)) == NULL) {
			sm_errno = E_SM_IO;
			return	NULL;
		}

		if ( (s = socket(PF_INET, SOCK_STREAM,0)) < 0) {
			sm_errno = E_SM_IO;
```

```
                return  NULL;
        }

        sin.sin_family = AF_INET;
        sin.sin_port = htons((u_short)port);
        bcopy(h->h_addr, (char *) &sin.sin_addr, h->h_length);
        if ( connect(s, (struct sockaddr *)(&sin),
            sizeof (struct sockaddr_in)) < 0) {
                sm_errno = E_SM_IO;
                return  NULL;
        }

        /*
         * Now handshake
         */
        fd = fdopen(s, "r+");
        if ( (sm_errno = mmwb_handshake(fd, user,cid,
            SM_MINVERS, SM_MAXVERS, &version )) < 0) {
                if (sm_errno == MMWB_HS_EPROTO)
                        sm_errno = E_SM_PROT;
                else
                        sm_errno = E_SM_HANDSHAKE;
                return  NULL;
        }
        sm_set_file(fd);
        return  fd;
}

/*
 * sm_dettach_session:
 *      Dettach from a session.
 */
int
sm_dettach_session(session_name)
char    *session_name;
{
        if (session_name != NULL)
                (void) sprintf(msg_buf,"QUIT %s\n",session_name);
        else
                (void) strcpy(msg_buf,"QUIT\n");
        return  mmwb_command(sm_file, msg_buf, (char *) NULL, 0);
}

/*
 * sm_ack_dialog:
 *      Acknowledge receipt of dialog
 */
/* ARGSUSED */
int
sm_ack_dialog(dp)
DIALOG  dp;
{
        return  mmwb_command(sm_file, "SUCCESS\n", (char *) NULL, 0);
}
```

libsm/tooldes.c

```
#ifndef  lint
static   char *RCSid = "$Id: tooldes.c,v 1.17 89/11/10 17:50:56 gregc Exp $";
#endif   lint

/*
 * tooldes:
 *      general library routines for handling tool descriptions
 *
 */

#include          "sm.h"
#include          "mmwbmisc.h"
#include          "ofs.h"
#include          <stdio.h>

#define  FAILURE -1
#define  SUCCESS 0

extern   void      *emalloc();

/*
 * sm_write_instrument:
 *      Write an instrument tool description out to a project
 */
int
sm_write_instrument(proj, td)
OFS_OBJECT     proj;
SM_TOOLDES     *td;
{
        static int          sm_write_tooldes();

        return sm_write_tooldes(OBJ_T_INSTRUMENT, proj, td);
}

/*
 * sm_write_protocol:
 *      Write a protocol tool description out to a project
 */
int
sm_write_protocol(proj, td)
OFS_OBJECT     proj;
SM_TOOLDES     *td;
{
        static int          sm_write_tooldes();

        return sm_write_tooldes(OBJ_T_PROTOCOL, proj, td);
}

/*
 * sm_write_tooldes:
 *      The real routine that writes a tool description out to the project
 */
static
int
sm_write_tooldes(type, proj, td)
int            type;
OFS_OBJECT     proj;
SM_TOOLDES     *td;
{
        register int        i, nopt;
        register DIALOG     dp;
        char                byte, *s;
        DM_TABLE            th;
        DM_ATTR             ah;

        if (proj != NULL) {
                td->obj = obj_create(proj, td->Name, OBJ_CL_TOOL, type);
                if (td->obj == NULL) {
                        sm_errno = E_SM_OFS;
                        return FAILURE;
                }
        }
        else {
                if (td->obj == NULL) {
                        sm_errno = E_SM_BADTD;
                        return FAILURE;
                }
        }

        /*
         * We now have an object handle.  Just dump the data back out
         * to the object.
         */
        (void) prop_set(td->obj, "version", (void *) td->Version);
        byte = td->nodisplay;
        (void) prop_set(td->obj, "nodisplay", (void *) &byte);
        (void) prop_set(td->obj, "subtools", (void *) td->Subtools);
        byte = td->OnError;
        (void) prop_set(td->obj, "on_error", (void *) &byte);
        (void) prop_set(td->obj, "host", (void *) td->Host);

        /*
         * Rewrite the option sheet as well
         */
```

```
        if (prop_get(td->obj, "options", (void *) &th) < 0) {
                sm_errno = E_SM_OFS;
                return FAILURE;
        }
        if ((ah = attr_open(th, "dialogs")) == NULL) {
                (void) tbl_close(th);
                sm_errno = E_SM_OFS;
                return FAILURE;
        }
        nopt = cell_count(ah, DM_I_MAX);
        for (i = 0, dp = td->Options; dp != NULL; dp = dialog_next(dp), i++) {
                s = dialog_to_char(dp);
                (void) cell_write(ah, i, (void *) s);
                mmwb_stringfree(s);
        }
        while (i < nopt)
                (void) cell_write(ah, i++, (void *) NULL);
        (void) attr_close(ah);
        (void) tbl_close(th);
        sm_errno = E_SM_NONE;
        return SUCCESS;
}

/*
 * sm_read_instrument:
 *      Read an instrument tool description from a project
 */
SM_TOOLDES *
sm_read_instrument(proj, name, find_first)
OFS_OBJECT      proj;
char            *name;
int             find_first;
{
        static SM_TOOLDES       *sm_read_tooldes();

        return sm_read_tooldes(OBJ_T_INSTRUMENT, proj, name, find_first);
}

/*
 * sm_read_protocol:
 *      Read a protocol tool description from a project
 */
SM_TOOLDES *
sm_read_protocol(proj, name, find_first)
OFS_OBJECT      proj;
char            *name;
int             find_first;
{
        static SM_TOOLDES       *sm_read_tooldes();

        return sm_read_tooldes(OBJ_T_PROTOCOL, proj, name, find_first);
}

/*
 * sm_read_tooldes:
 *      The real routine that reads a tool description from the project
 */
static
SM_TOOLDES *
sm_read_tooldes(type, proj, name, find_first)
int             type;
OFS_OBJECT      proj;
char            *name;
int             find_first;
{
        register SM_TOOLDES     *ntd, *otd;
        OFS_OBJECT              curr, home, sys;
        static SM_TOOLDES       *read_td(), *join_td(), *unique_td();

        if (proj != NULL)
                return read_td(type, proj, name, SM_PROJ_CURR);
        else {
                curr = sm_get_curr_proj();
                home = sm_get_home_proj();
                sys = sm_get_sys_proj();
                if (curr == home || curr == sys)
                        curr = NULL;
                if (home == sys)
                        home = NULL;
                if (curr != NULL) {
                        otd = read_td(type, curr, name, SM_PROJ_CURR);
                        if (otd != NULL && find_first && name != NULL)
                                return unique_td(otd);
                }
                else
                        otd = NULL;
                if (home != NULL) {
                        ntd = read_td(type, home, name, SM_PROJ_HOME);
                        if (ntd != NULL && find_first && name != NULL)
                                return unique_td(ntd);
                        otd = join_td(otd, ntd);
                }
                if (sys != NULL) {
                        ntd = read_td(type, sys, name, SM_PROJ_SYS);
```

```
                        if (ntd != NULL && find_first && name != NULL)
                                return unique_td(ntd);
                        otd = join_td(otd, ntd);
                }
                if (find_first)
                        otd = unique_td(otd);
                return otd;
        }
}

/*
 * read_td:
 *      Read all tool descriptions that match name from the given project
 */
static
SM_TOOLDES *
read_td(type, proj, name, which)
int             type;
OFS_OBJECT      proj;
char            *name;
int             which;
{
        register int                    i, nobj;
        OFS_OBJ_DESC                    *desc;
        OFS_OBJECT                      oh;
        SM_TOOLDES                      *td, *td_list, *last_td;
        static SM_TOOLDES               *get_td();
        int                             tooltype;

        if (type == OBJ_T_INSTRUMENT)
                tooltype = 2;
        else
                tooltype = 1;
        sm_errno = E_SM_NONE;
        nobj = obj_count(proj, name, -1, type);
        if (nobj <= 0) {
                if (nobj < 0)
                        sm_errno = E_SM_OFS;
                return NULL;
        }
        desc = (OFS_OBJ_DESC *) emalloc(nobj * sizeof (OFS_OBJ_DESC));
        if (obj_list(proj, name, -1, type, desc, nobj) != nobj) {
                (void) free((char *) desc);
                sm_errno = E_SM_OFS;
                return NULL;
        }
        td_list = last_td = NULL;
        for (i = 0; i < nobj; i++) {
                if ((oh = obj_open_by_ident(proj, desc[i].ident)) == NULL)
                        continue;
                if ((td = get_td(oh, which)) == NULL) {
                        (void) obj_close(oh);
                        continue;
                }
                td->Type = tooltype;
                if (last_td == NULL)
                        td_list = last_td = td;
                else {
                        last_td->next = td;
                        last_td = td;
                }
        }
        return td_list;
}

/*
 * get_td:
 *      Get a tool description out of an object
 */
static
SM_TOOLDES *
get_td(oh, which)
OFS_OBJECT      oh;
int             which;
{
        register SM_TOOLDES     *td;
        register DIALOG         dp, last_dp;
        register int            nopt, i;
        DM_TABLE                th;
        DM_ATTR                 ah;
        char                    byte, buf[BUFSIZ];

        td = (SM_TOOLDES *) emalloc(sizeof (SM_TOOLDES));
        bzero((char *) td, sizeof (SM_TOOLDES));
        td->Name = obj_name(oh);
        td->obj = oh;
        td->from_proj = which;
        dm_errno = DM_ENOERROR;

        if (prop_get(oh, "version", (void *) buf) < 0)
                goto failed;
        td->Version = mmwb_cpystring(buf);
```

```
        if (prop_get(oh, "nodisplay", (void *) &byte) < 0)
                goto failed;
        td->nodisplay = byte;

        if (prop_get(oh, "subtools", (void *) buf) < 0)
                goto failed;
        td->Subtools = mmwb_cpystring(buf);

        if (prop_get(oh, "host", (void *) buf) < 0)
                goto failed;
        td->Host = mmwb_cpystring(buf);

        if (prop_get(oh, "on_error", (void *) &byte) < 0)
                goto failed;
        td->OnError = byte;

        if (prop_get(oh, "options", (void *) &th) < 0)
                goto failed;
        ah = attr_open(th, "dialogs");
        if (ah == NULL) {
                (void) tbl_close(th);
                goto failed;
        }
        nopt = cell_count(ah, DM_I_MAX);
        last_dp = NULL;
        for (i = 0; i < nopt; i++) {
                if (cell_read(ah, i, (void *) buf) < 0)
                        continue;
                if ((dp = dialog_from_char(buf)) == NULL)
                        continue;
                if (last_dp == NULL)
                        td->Options = last_dp = dp;
                else {
                        last_dp = dialog_append(last_dp, dp);
                        last_dp = dp;
                }
        }

        (void) attr_close(ah);
        (void) tbl_close(th);

        sm_errno = E_SM_NONE;
        return td;

failed:
        sm_free_tooldes(td);
        if (dm_errno != DM_ENOERROR)
                sm_errno = E_SM_OFS;
        return NULL;
}

/*
 * join_td:
 *      Join two lists of tool descriptions and return the head
 */
static
SM_TOOLDES *
join_td(td1, td2)
SM_TOOLDES      *td1, *td2;
{
        register SM_TOOLDES     *td;

        if (td1 == NULL)
                return td2;
        if (td2 == NULL)
                return td1;
        for (td = td1; td->next != NULL; td = td->next)
                continue;
        td->next = td2;
        return td1;
}

/*
 * unique_td:
 *      Make sure all tool descriptions have unique names
 */
static
SM_TOOLDES *
unique_td(tdlist)
SM_TOOLDES      *tdlist;
{
        register SM_TOOLDES     *new, *ntd, *otd, *next;

        new = NULL;
        for (otd = tdlist; otd != NULL; otd = next) {
                next = otd->next;
                for (ntd = new; ntd != NULL; ntd = ntd->next)
                        if (strcmp(ntd->Name, otd->Name) == 0)
                                break;
                if (ntd != NULL) {
                        /* There was one already, just drop this one */
                        sm_free_tooldes(otd);
                }
                else {
```

```
                        otd->next = new;
                        new = otd;
                }
        }
        return new;
}

/*
 * sm_list_instrument:
 *      List instrument tool descriptions in a project
 */
SM_TOOLDES *
sm_list_instrument(proj)
OFS_OBJECT      proj;
{
        static SM_TOOLDES       *sm_read_tooldes();

        return sm_read_tooldes(OBJ_T_INSTRUMENT, proj, (char *) NULL, 1);
}

/*
 * sm_list_protocol:
 *      List protocol tool descriptions in a project
 */
SM_TOOLDES *
sm_list_protocol(proj)
OFS_OBJECT      proj;
{
        static SM_TOOLDES       *sm_read_tooldes();

        return sm_read_tooldes(OBJ_T_PROTOCOL, proj, (char *) NULL, 1);
}

/*
 * sm_free_tooldes:
 *      Free a tool description
 */
void
sm_free_tooldes(td)
SM_TOOLDES      *td;
{

        if (td->Curr_Proj != NULL)
                (void) free((char *) td->Curr_Proj);
        if (td->Home_Proj != NULL)
                (void) free((char *) td->Home_Proj);
        if (td->Sys_Proj != NULL)
                (void) free((char *) td->Sys_Proj);
        if (td->obj != NULL)
                (void) obj_close(td->obj);
        mmwb_stringfree(td->Version);
        mmwb_stringfree(td->Subtools);
        mmwb_stringfree(td->Host);
        dialog_free(td->Options, 0);
        (void) free((char *) td);
}
```

libdialog/dialog.h

```
#ifndef   DIALOG_INCLUDE
#ifndef NO_C_CODE
/*
 * $Id: dialog.h,v 1.13 89/10/30 17:06:38 scooter Exp Locker: pett $
 *
 * dialog.h:
 *          This header file contains all of the defines relevant
 *          to mmwb dialogs.
 *
 */

#include <machdep.h>

typedef  void *     DIALOG;

#define  DI_BUFLEN      1024

/*
 * Basic create, free, and copy
 */

extern   DIALOG  dialog_create(/*type,dest,name,prompt*/);
extern   DIALOG  dialog_copy(/*handle,flag*/);
extern   void    dialog_free(/*handle,flag*/);

/*
 * Dialogs can potentially be complex, which implies that
 * several dialogs are linked together.
 * The following routines manage these lists.
 */

extern   DIALOG  dialog_next(/*handle*/);
extern   DIALOG  dialog_append(/*handle1,handle2*/);
extern   DIALOG  dialog_delete(/*handle1,handle2*/); /* Delete 2 from list 1 */

/*
 * Routines to return certain special internal values
 */

extern   int     dialog_semtype(/*handle*/);
extern   int     dialog_type(/*handle*/);
extern   int     dialog_source(/*handle*/);          /* Return the source */
extern   int     dialog_dest(/*handle*/);            /* Return the destination */
extern   int     dialog_id(/*handle*/);              /* Return the dialog id */
extern   char    *dialog_name(/*handle*/);           /* Return the dialog name
extern   char    *dialog_prompt(/*handle*/);         /* Return the dialog promp

extern   int     dialog_set_semantics(/*handle,semtype,semopts*/);
extern   int     dialog_get_semantics(/*handle,semopts,success*/);

/* Routines to set the dialog source, destination(tool_id) and dialog id */

extern   int     dialog_set_dest(/*handle,dest*/);
extern   int     dialog_set_source(/*handle,source*/);
extern   int     dialog_set_id(/*handle,id*/);

/*
 * Dialogs contain a variety of values which would be very
 * difficult to set with a single call.  Also, the values area
 * often not used for every dialog.  These routines allow
 * the user to manipulate these values.
 */

         /* Note: string is NOT copied */
extern   char    *dialog_get_text(/*handle,VALUE,success*/);
         /* Note: string IS copied */
extern   int     dialog_set_text(/*handle,VALUE,string*/);
extern   int     dialog_get_int(/*handle,VALUE,success*/);
extern   int     dialog_set_int(/*handle,VALUE,int*/);
extern   double  dialog_get_float(/*handle,VALUE,success*/);
extern   int     dialog_set_float(/*handle,VALUE,float*/);
extern   int     dialog_get_bool(/*handle,VALUE,success*/);
extern   int     dialog_set_bool(/*handle,VALUE,bool*/);
extern   int     dialog_get_flag(/*handle,FLAG*/);
extern   int     dialog_set_flag(/*handle,FLAG,value*/);
extern   int     dialog_isnt_zero(/*handle,which*/);

/*
 * Sometimes its useful to have a text representation for dialogs.
 */

extern   char    *dialog_to_char(/*handle*/);
extern   DIALOG  dialog_from_char(/*string*/);

/*
 * Utility for forming selection lists
 */

extern   char    *dialog_form_sellist(/*count,string_array*/);

/*
 * Flags
 */
#endif NO_C_CODE
```

libdialog/dialog.h

```
#ifdef    comment
/*
 * The flags for free and copy are as follows:
 */
#endif    comment

#define   DI_THISDI          1        /* Free (or copy) just this dialog */


#undef comment               /* Defines for VALUE */

#define   DI_V_NAME          1        /* The dialog name */
#define   DI_V_PROMPT        2        /* The dialog prompt */
#define DI_V_LBOUND          3        /* The lower bounds */
#define DI_V_UBOUND          4        /* The upper bounds */
#define DI_V_RESULT          5        /* The result */
#define DI_V_SEMOPT          6        /* The semantic option */
#define   DI_V_HELP          7        /* The help text */
#define   DI_V_SELLIST       8        /* A selection list */
#define   DI_V_OPTION        9        /* The option sheet number */
#define   DI_V_TOOL          10       /* The tool name */
#define   DI_V_SOURCE        11       /* The source of the dialog */
#define   DI_V_TYPE          12       /* The dialog type */
#define   DI_V_SEMTYPE       13       /* The dialog semantic type */

#undef comment               /* FLAGS for dialog */

#define   DI_OPTIONAL        0x0001   /* Default result is valid */
#define   DI_NODISPLAY       0x0002   /* Don't display this */
#define   DI_SUPERCEDES      0x0004   /* Replace all previous dialogs of this name */

#undef comment               /* Dialog Types */

#define DI_IVAL 1       /* Integer evaluator */
#define DI_FVAL 2       /* Float */
#define DI_TEXT 3       /* Text string */
#define DI_SELECT 4     /* Selection list, return value is list of indexes */
#define DI_BOOL 5       /* A true or false */
#define DI_YESNO 6      /* Same as above, but yes/no */
#define DI_NOTE 7       /* No return value */


#define   DIALOG_INCLUDE
#endif    DIALOG_INCLUDE
```

libdialog/dialog_int.h

```
/*
 * $Id: dialog_int.h,v 1.4 89/10/25 21:22:15 gregc Exp $
 *
 * dialog_int.h:
 *      Internal structures and defines for the dialog library
 *
 */

#include        <machdep.h>
#include        "dialog.h"

typedef union   {
        int     integer;
        double  real;
        char    *string;
        int     yesno;
} DI_VALUE;

typedef struct dialog_st {
        int         di_version;     /* Dialog structure version */
        int         di_id;          /* The dialog ID (assigned by sm) */
        char        *di_name;       /* The dialog name */
        int         di_type;        /* The dialog type */
        int         di_stype;       /* The semantic type (if any) */
        char        *di_semopt;     /* String for use by semantics */
        char        *di_prompt;     /* The prompt */
        char        *di_help;       /* Any help text */
        char        *di_sellist;    /* A possible selection list */
        int         di_flags;       /* Any possible flags */
        int         di_source;      /* The source of the dialog */
        int         di_dest;        /* The destination of the dialog */
        char        *di_tool;       /* The name of the tool (for namestripe) */
        int         di_option;      /* The option sheet number */
        struct      dialog_st *di_next;
        /*** all unions must be at end of structure ***/
        DI_VALUE di_result;         /* The answer */
        DI_VALUE di_upper;          /* Upper bounds */
        DI_VALUE di_lower;          /* Lower bounds */
}       DIALOG_INT;

#define DI_NULL (DIALOG_INT *)NULL
#define DI_VERSION      1
#define DI_NTOKENS      17
```

libdialog/dialog_char.c

```
#ifndef lint
static char *RCSid = "$Id: dialog_char.c,v 1.7 89/11/13 19:24:22 conrad Exp $";
#endif lint

/*
 * dialog_char.c:
 *     Routines to convert dialogs into and out of a character
 *          representaion.
 *
 */

#include <stdio.h>
#include "dialog_int.h"
#include "mmwbmisc.h"

static     char      conv_buf[BUFSIZ];

/*
 * dialog_from_char:
 *          convert a dialog from a character representation.
 */

DIALOG
dialog_from_char(text)
char      *text;
{
          char                *tokens[DI_NTOKENS+2];
          static DIALOG       d_from_tokv();

          /*
           * Tokenize the strings
           */

          if (tokenize(text, tokens, DI_NTOKENS) != DI_NTOKENS)
                    return    (DIALOG)NULL;

          /*
           * Let d_from_tokv do its thing.
           */

          return    d_from_tokv(DI_NTOKENS,tokens);
}

/*
 * dialog_from_tokv:
 *          convert a dialog from a token list.
 */

static DIALOG
d_from_tokv(argc, args)
int       argc;
char      **args;
{
          DIALOG_INT          *dp;
          static int          gethex();
          static DI_VALUE     getval(),getbounds();

          if (argc != DI_NTOKENS)
                    return (DIALOG)NULL;

          dp = (DIALOG_INT *)dialog_create(0,0,(char *)0,(char *)0);

          dp->di_version = atoi(*(++args));
          /*
           * Check version number here (someday)
           */
          dp->di_name = mmwb_cpystring(*args);
          dp->di_type = atoi(*(++args));
          dp->di_stype = atoi(*(++args));
          dp->di_semopt = mmwb_cpystring(*(++args));
          dp->di_prompt = mmwb_cpystring(*(++args));
          dp->di_help = mmwb_cpystring(*(++args));
          dp->di_sellist = mmwb_cpystring(*(++args));
          dp->di_result = getval(*(++args),dp->di_type);
          dp->di_upper = getbounds(*(++args),dp->di_type);
          dp->di_lower = getbounds(*(++args),dp->di_type);
          dp->di_flags = gethex(*(++args));
          dp->di_source = atoi(*(++args));
          dp->di_dest = atoi(*(++args));
          dp->di_tool = mmwb_cpystring(*(++args));
          dp->di_option = atoi(*(++args));
          dp->di_id = atoi(*(++args));

          return (DIALOG)dp;
}


/*
 * dialog_to_char:
 *          convert a dialog to a character representation
 */

char *
dialog_to_char(dialog)
```

libdialog/dialog_char.c

```
DIALOG  dialog;
{
        static char         *itoa(), *xtoa(), *sm_value2char(), *bounds2char();
        char                buf[DI_BUFLEN];
        register char       *cp;
        DIALOG_INT          *dp = (DIALOG_INT *)dialog;

        cp = buf;
        cp = mmwb_strend(cp, itoa(dp->di_version));
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_name));           /* Name */
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_type));                /* Dialog type */
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_stype));               /* Semantic typ
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_semopt));         /* Semantic opti
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_prompt));         /* Prompt string
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_help));           /* Help text */
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_sellist)); /* Selection list */
        *cp++ = ' ';
        cp = mmwb_strend(cp,
            sm_value2char(dp->di_result,dp->di_type)); /* Return value */
        *cp++ = ' ';
        cp = mmwb_strend(cp,
                bounds2char(dp->di_upper,dp->di_type)); /* Upper bounds */
        *cp++ = ' ';
        cp = mmwb_strend(cp,
                bounds2char(dp->di_lower,dp->di_type)); /* Lower bounds */
        *cp++ = ' ';
        cp = mmwb_strend(cp, xtoa(dp->di_flags));               /* Flags */
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_source));              /* Source toolID
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_dest));                /* Destination to
        *cp++ = ' ';
        cp = mmwb_strend(cp, stringize(dp->di_tool));           /* Tool name */
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_option));              /* Option numbe
        *cp++ = ' ';
        cp = mmwb_strend(cp, itoa(dp->di_id));                  /* Dialog ID */
        *cp++ = ' ';
        *cp = '\0';

        if (cp >= &buf[DI_BUFLEN]) {
                (void) fprintf(stderr, "overflowed buffer in dialog_to_char\n");
                exit(1);
        }
        return mmwb_cpystring(buf);
}

/*
 * getval:
 *      return the value given the text representation and the
 *      value type.
 */

static DI_VALUE
getval ( text, type )
char    *text;
int     type;
{
        DI_VALUE        val;

        switch  (type)
        {
        case    DI_IVAL:
                (void) sscanf ( text, "%d", &val.integer );
                break;

        case    DI_FVAL:
                (void) sscanf ( text, "%lf", &val.real );
                break;

        case    DI_TEXT:
        case    DI_SELECT:
                val.string = mmwb_cpystring( text );
                break;

        case    DI_NOTE:
                val.integer = 0;
                break;

        case    DI_BOOL:
                if ( strncmp (text, "TRUE", 4) == 0 )
                        val.yesno = 1;
                else
                        val.yesno = 0;
                break;

        case    DI_YESNO:
```

```
                if ( strncmp (text, "YES", 3) == 0 )
                        val.yesno = 1;
                else
                        val.yesno = 0;
                break;
        }

        return val;
}

/*
 * getbounds:
 *      return the bounds given the text and the value type.
 */

static DI_VALUE
getbounds ( text, type )
char    *text;
int     type;
{
        DI_VALUE        val;

        val.integer = 0;

        switch  (type)
        {

        case    DI_IVAL:
        case    DI_TEXT:
        case    DI_SELECT:
                return  getval(text, DI_IVAL);

        case    DI_FVAL:
                return  getval(text, DI_FVAL);

        case    DI_BOOL:
        case    DI_YESNO:
        case    DI_NOTE:
                /*
                 * Bounds are not supported for these types
                 */
                return  val;
        }

        return  val;
}

/*
 * gethex:
 *      return the next value given the string representation
 */

static int
gethex ( text )
char    *text;
{
        int     val;
        (void) sscanf( text, "%x", &val );
        return val;
}


/*
 * sm_value2char:
 *      return the text representation of the value.
 */

static char *
sm_value2char (val, type)
DI_VALUE        val;
int             type;
{
        char    *strcpy();

        switch(type)
        {
        case    DI_IVAL:
                (void) sprintf(conv_buf,"%d",val.integer);
                break;

        case    DI_FVAL:
                (void) sprintf(conv_buf,"%g",val.real);
                break;

        case    DI_YESNO:
                if (val.yesno)
                        (void) strcpy(conv_buf, "YES");
                else
                        (void) strcpy(conv_buf, "NO");
                break;

        case    DI_BOOL:
                if (val.yesno)
                        (void) strcpy(conv_buf, "TRUE");
```

```
                else
                        (void) strcpy(conv_buf, "FALSE");
                break;

        case    DI_TEXT:
        case    DI_SELECT:
                return stringize(val.string);
                break;

        case    DI_NOTE:
                (void) sprintf(conv_buf, "\"\"");
                break;

        }

        return conv_buf;
}


/*
 * bounds2char:
 *      return the text representation of the bounds.
 */

static char *
bounds2char (val, type)
DI_VALUE        val;
int             type;
{
        switch  (type)
        {
        case    DI_IVAL:
        case    DI_TEXT:
        case    DI_SELECT:
                return  sm_value2char(val, DI_IVAL);

        case    DI_FVAL:
                return  sm_value2char(val, DI_FVAL);

        default:
                break;
                /* Fall through */
        }

        return  sm_value2char(val, DI_NOTE);
}
```

```
/*
 * itoa:
 *      return the text value of the input integer
 */

static  char    *
itoa (val)
int     val;
{
        (void) sprintf (conv_buf,"%d",val);

        return  conv_buf;
}

/*
 * xtoa:
 *      return the text value of the input hex value
 */

static  char    *
xtoa (val)
int     val;
{
        (void) sprintf (conv_buf,"%x",val);

        return  conv_buf;
}
```

libdialog/dialog_manip.c

```
#ifndef lint
static char *RCSid = "$Id: dialog_manip.c,v 1.7 89/10/25 21:22:26 gregc Exp $";
#endif lint

/*
 * dialog_manip.c:
 *      Routines for the manipulation of dialogs
 *
 */

#include <stdio.h>
#include "dialog_int.h"
#include "mmwbmisc.h"

/*
 * dialog_create:
 *      create a dialog of the requested type and semantic type.
 */

DIALOG
dialog_create(type,dest,name,prompt)
int     type,dest;
char    *name,*prompt;
{
        DIALOG_INT      *dialog;

        dialog = (DIALOG_INT *)ecalloc(1, sizeof (DIALOG_INT));

        dialog->di_type = type;
        dialog->di_dest = dest;
        dialog->di_name = mmwb_cpystring(name);
        dialog->di_prompt = mmwb_cpystring(prompt);
        dialog->di_version = DI_VERSION;

        return  (DIALOG)dialog;
}

/*
 * dialog_free:
 *      Free an existing dialog structure
 */

void
dialog_free(dialog,flag)
DIALOG  dialog;
int     flag;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d == DI_NULL)
                return;

        mmwb_stringfree(d->di_name);
        mmwb_stringfree(d->di_prompt);
        mmwb_stringfree(d->di_help);
        mmwb_stringfree(d->di_semopt);
        mmwb_stringfree(d->di_sellist);
        if (d->di_type == DI_TEXT)
                mmwb_stringfree(d->di_result.string);

        if ( (flag & DI_THISDI) == 0 )
                dialog_free((DIALOG)d->di_next,flag);

        free((char *)d);

        return;
}

/*
 * dialog_copy:
 *      return a copy of the requested dialog
 */

DIALOG
dialog_copy(dialog,flag)
DIALOG  dialog;
int     flag;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;
        DIALOG_INT      *nd;

        /*
         * Make sure there is something to copy
         */

        if (d == DI_NULL)
                return  (DIALOG)DI_NULL;

        nd = (DIALOG_INT *)dialog_create(d->di_type,d->di_dest,
                                         d->di_name,d->di_prompt);

        if (nd == DI_NULL)
                return  (DIALOG)DI_NULL;
```

```
        nd->di_stype = d->di_stype;
        nd->di_semopt = mmwb_cpystring(d->di_semopt);
        nd->di_help = mmwb_cpystring(d->di_help);
        nd->di_sellist = mmwb_cpystring(d->di_sellist);
        nd->di_flags = d->di_flags;
        nd->di_source = d->di_source;
        nd->di_tool = mmwb_cpystring(d->di_tool);
        nd->di_option = d->di_option;
        nd->di_upper = d->di_upper;
        nd->di_lower = d->di_lower;
        if (nd->di_type == DI_TEXT)
                nd->di_result.string = mmwb_cpystring(d->di_result.string);
        else
                nd->di_result = d->di_result;

        if ( (flag & DI_THISDI) == 0 )
                if (d->di_next != DI_NULL)
                        nd->di_next = (DIALOG_INT *)dialog_copy((DIALOG)d-

        return  (DIALOG)nd;
}

/*
 * dialog_next:
 *      return the next dialog in the list.  returns NULL if the
 *      argument is the last dialog.
 */

DIALOG
dialog_next(dialog)
DIALOG  dialog;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d == DI_NULL)
                return  NULL;

        return  (DIALOG)d->di_next;
}

/*
 * dialog_append:
 *      append a dialog to a list.  Returns the pointer to the
 *      head of the list.
 */

DIALOG
dialog_append(head,tail)
DIALOG  head,tail;
{
        DIALOG_INT      *d;
        DIALOG          dialog, dtmp;

        if (head == NULL)
                return  tail;

        dialog = head;
        while ( (dtmp = dialog_next(dialog)) != NULL )
                dialog = dtmp;

        d = (DIALOG_INT *)dialog;

        d->di_next = (DIALOG_INT *)tail;

        return  head;
}

/*
 * dialog_delete:
 *      delete a dialog from a list.  the dialog to delete is chosen
 *      by a match between EITHER the dialog id or the dialog name (if
 *      the id is 0).
 *
 *      NOTE: if the argument and the dialog to be deleted are at the
 *      same address, then the caller obviously did not make a copy, and
 *      the dialog is not freed. Otherwise, the dialog is freed.
 */

DIALOG
dialog_delete(list,del)
DIALOG  list,del;
{
        DIALOG          dtmp = list;
        DIALOG          last = NULL;
        static DIALOG   d_delete();
        char            *name;
        int             id;

        /*
         * First search for a pointer match
         */

        do {
                if (dtmp == del) {
```

```
                        DIALOG_INT *d = (DIALOG_INT *)dtmp;
                        return dialog_append(last,(DIALOG)(d->di_next));
                }

                last = dtmp;
        } while ( (dtmp = dialog_next(dtmp)) != NULL );

        /*
         * Now search for an id match
         */

        dtmp = list;
        last = NULL;

        if ((id = dialog_id(del)) != 0)
                do {
                        if (dialog_id(dtmp) == id)
                                return  d_delete(last,dtmp);

                        last = dtmp;
                } while ( (dtmp = dialog_next(dtmp)) != NULL );

        /*
         * OK, finally try a name match
         */

        dtmp = list;
        last = NULL;

        if ((name = dialog_name(del)) != NULL)
                do {
                        if (!strcmp(dialog_name(dtmp),name))
                                return  d_delete(last,dtmp);

                        last = dtmp;
                } while ( (dtmp = dialog_next(dtmp)) != NULL );

        /*
         * No match, return a NULL
         */

        return  NULL;
}

static DIALOG
d_delete(first,skip)
DIALOG  first,skip;
{
        DIALOG  newlist;
        DIALOG_INT *d = (DIALOG_INT *)skip;

        newlist = dialog_append(first,(DIALOG)(d->di_next));

        dialog_free(skip,DI_THISDI);

        return  newlist;
}
```

libdialog/dialog_util.c

```
#ifndef lint
static char *RCSid = "$Id: dialog_util.c,v 1.3 89/10/25 21:22:30 gregc Exp $";
#endif lint

#include <string.h>

/*
 * dialog_util.c:
 *     Routines to aid in construction/interpretation of dialogs.
 *
 */

#include <stdio.h>
#include "mmwbmisc.h"

/*
 * dialog_form_sellist:
 *          Given an array of strings, form a string that can be used as the
 * selection list field of a dialog.
 */

char *
dialog_form_sellist(num_choices, choices)
int         num_choices;
char        *choices[];
{
            char        *sellist, *cur;
            int         tlen, clen;

            if (num_choices < 1)
                        return (char *) NULL;
            sellist = mmwb_cpystring(stringize(choices[0]));
            tlen = strlen(sellist);
            for (num_choices--, choices++; num_choices > 0;
              num_choices--, choices++) {
                        cur = stringize(choices[0]);
                        clen = strlen(cur);
                        sellist = (char *) erealloc(sellist,
                          (tlen + clen + 2) * sizeof(char));
                        (void) strcpy(sellist + tlen, " ");
                        (void) strcpy(sellist + tlen + 1, cur);
                        tlen += clen + 1;
            }
            return sellist;
}
```

libdialog/dialog_vals.c

```
#ifndef lint
static char *RCSid = "$Id: dialog_vals.c,v 1.11 90/02/16 10:44:39 scooter Exp $";
#endif lint

/*
 * dialog_vals.c:
 *      Routines for setting and returning dialog values
 *
 */

#include <stdio.h>
#include "dialog_int.h"
#include "mmwbmisc.h"

DI_VALUE        d_get_value();
int             d_put_value();

/*
 * dialog_type:
 *      return the dialog type.
 */

int
dialog_type(dialog)
DIALOG  dialog;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d != DI_NULL)
                return  d->di_type;
        else
                return  -1;
}

/*
 * dialog_semtype:
 *      return the dialog semantic type.
 */

int
dialog_semtype(dialog)
DIALOG  dialog;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d != DI_NULL)
                return  d->di_stype;
        else
                return  -1;
}

/*
 * dialog_source:
 *      return the dialog source.
 */

int
dialog_source(dialog)
DIALOG  dialog;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d != DI_NULL)
                return  d->di_source;
        else
                return  -1;
}

/*
 * dialog_dest:
 *      return the dialog destination.
 */

int
dialog_dest(dialog)
DIALOG  dialog;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d != DI_NULL)
                return  d->di_dest;
        else
                return  -1;
}

/*
 * dialog_id:
 *      return the dialog id.
 */

int
dialog_id(dialog)
DIALOG  dialog;
{
```

libdialog/dialog_vals.c

```
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d != DI_NULL)
		return	d->di_id;
	else
		return	-1;
}

/*
 * dialog_name:
 *	return the dialog name.
 */

char *
dialog_name(dialog)
DIALOG	dialog;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d != DI_NULL)
		return	d->di_name;
	else
		return	NULL;
}

/*
 * dialog_prompt:
 *	return the dialog prompt.
 */

char *
dialog_prompt(dialog)
DIALOG	dialog;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d != DI_NULL)
		return	d->di_prompt;
	else
		return	NULL;
}

/*
 * dialog_set_dest:
 *	set the dialog destination
 */

int
dialog_set_dest(dialog,dest)
DIALOG	dialog;
int	dest;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d == DI_NULL)
		return	-1;

	d->di_dest = dest;
	return	0;
}

/*
 * dialog_set_source:
 *	set the dialog source
 */

int
dialog_set_source(dialog,source)
DIALOG	dialog;
int	source;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d == DI_NULL)
		return	-1;

	d->di_source = source;
	return	0;
}

/*
 * dialog_set_id:
 *	set the dialog id
 */

int
dialog_set_id(dialog,id)
DIALOG	dialog;
int	id;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d == DI_NULL)
		return	-1;
```

```
	d->di_id = id;
	return	0;
}

/*
 * dialog_set_semantics:
 *	set the dialog id
 */

int
dialog_set_semantics(dialog,stype,semopt)
DIALOG	dialog;
int	stype;
char	*semopt;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	if (d == DI_NULL)
		return	-1;

	d->di_stype = stype;
	if (semopt != NULL)
		d->di_semopt = mmwb_cpystring(semopt);
	else
		d->di_semopt = NULL;
	return	0;
}

/*
 * dialog_get_semantics:
 *	get the dialog semantics
 */

int
dialog_get_semantics(dialog,semopt,success)
DIALOG	dialog;
char	**semopt;
int	*success;
{
	DIALOG_INT	*d = (DIALOG_INT *)dialog;

	*success = 0;

	if (d == DI_NULL) {
		*success = -1;
		return	0;
	}

	if (semopt != NULL)
		*semopt = d->di_semopt;

	return	d->di_stype;
}

/*
 * dialog_get_text:
 *	return a text value.
 */

char *
dialog_get_text(dialog,which,success)
DIALOG	dialog;
int	which;
int	*success;
{
	int		type = 0;
	DI_VALUE	v = d_get_value(dialog,which,&type);

	if (type != DI_TEXT || dialog == NULL) {
		*success = -1;
		return (char *) NULL;
	} else
		*success = 0;

	return v.string;
}

/*
 * dialog_get_int:
 *	return a integer value.
 */

int
dialog_get_int(dialog,which,success)
DIALOG	dialog;
int	which;
int	*success;
{
	int		type = 0;
	DI_VALUE	v = d_get_value(dialog,which,&type);

	if (type != DI_IVAL || dialog == NULL) {
		*success = -1;
```

```
                return 0;
        } else
                *success = 0;

        return v.integer;
}

/*
 * dialog_get_float:
 *      return a floating point value.
 */

double
dialog_get_float(dialog,which,success)
DIALOG  dialog;
int     which;
int     *success;
{
        int                     type = 0;
        DI_VALUE        v = d_get_value(dialog,which,&type);

        if (type != DI_FVAL || dialog == NULL) {
                *success = -1;
                return 0.0;
        } else
                *success = 0;

        return v.real;
}

/*
 * dialog_get_bool:
 *      return a boolean value.
 */

int
dialog_get_bool(dialog,which,success)
DIALOG  dialog;
int     which;
int     *success;
{
        int                     type = 0;
        DI_VALUE        v = d_get_value(dialog,which,&type);

        if (type != DI_IVAL || dialog == NULL) {
                *success = -1;
                return -1;
        } else
                *success = 0;

        return v.yesno;
}

/*
 * dialog_set_text:
 *      set a text value in a dialog
 */

int
dialog_set_text(dialog,which,text)
DIALOG  dialog;
int     which;
char    *text;
{
        DI_VALUE        v;

        if (dialog == NULL)
                return  -1;

        v.string = mmwb_cpystring(text);
        return  d_put_value(dialog,which,v,DI_TEXT);
}

/*
 * dialog_set_int:
 *      set an integer value in a dialog
 */

int
dialog_set_int(dialog,which,value)
DIALOG  dialog;
int     which;
int     value;
{
        DI_VALUE        v;

        if (dialog == NULL)
                return  -1;

        v.integer = value;
        return  d_put_value(dialog,which,v,DI_IVAL);
}

/*
```

libdialog/dialog_vals.c

```
 * dialog_set_float:
 *      set a floating point value in a dialog
 */

int
dialog_set_float(dialog,which,value)
DIALOG  dialog;
int     which;
double  value;
{
        DI_VALUE        v;

        if (dialog == NULL)
                return  -1;

        v.real = value;
        return  d_put_value(dialog,which,v,DI_FVAL);
}

/*
 * dialog_set_bool:
 *      set a boolean value in a dialog
 */

int
dialog_set_bool(dialog,which,value)
DIALOG  dialog;
int     which;
int     value;
{
        DI_VALUE        v;

        if (dialog == NULL)
                return  -1;

        v.yesno = value;
        return  d_put_value(dialog,which,v,DI_IVAL);
}

/*
 * dialog_set_flag:
 *      Set a flag in a dialog
 */

int
dialog_set_flag(dialog,flag,value)
DIALOG  dialog;
int     flag;
int     value;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d == DI_NULL)
                return  -1;

        if (value)
                d->di_flags |= flag;
        else
                d->di_flags &= ~flag;

        return  0;
}

/*
 * dialog_get_flag:
 *      Return TRUE if the flag is set, FALSE otherwise.
 */

int
dialog_get_flag(dialog,flag)
DIALOG  dialog;
int     flag;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;

        if (d == DI_NULL)
                return  -1;

        if (d->di_flags & flag)
                return  1;
        else
                return  0;
}

/*
 * dialog_isnt_zero:
 *      Returns TRUE if an entry is non zero.
 */

int
dialog_isnt_zero(dialog, which)
DIALOG  dialog;
int     which;
{
```

```
        int             type;
        DI_VALUE        v;

        v = d_get_value(dialog, which, &type);
        switch (type) {
        case DI_IVAL:
                return v.integer != 0;
        case DI_FVAL:
                return v.real != 0.0;
        case DI_TEXT:
        case DI_SELECT:
                if (v.string == NULL)
                        return 0;
                return 1;
        case DI_BOOL:
        case DI_YESNO:
                return v.yesno != 0;
        case DI_NOTE:
                return 0;
        }
        return 0;
}


/*
 * Local routines for handling the set and put routines
 */


/*
 * d_get_value:
 *      returns an DI_VALUE from a dialog
 */

static DI_VALUE
d_get_value(dialog,which,type)
DIALOG  dialog;
int     which;
int     *type;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;
        DI_VALUE        v;

        *type = DI_NOTE;
        v.integer = -1;

        if (d == DI_NULL)
                return  v;

switch (which) {
case    DI_V_TYPE:
        *type = DI_IVAL;
        v.integer = d->di_type;
        break;
case    DI_V_SEMTYPE:
        *type = DI_IVAL;
        v.integer = d->di_stype;
        break;
case    DI_V_OPTION:
        *type = DI_IVAL;
        v.integer = d->di_option;
        break;
case    DI_V_NAME:
        *type = DI_TEXT;
        v.string = d->di_name;
        break;
case    DI_V_PROMPT:
        *type = DI_TEXT;
        v.string = d->di_prompt;
        break;
case    DI_V_SEMOPT:
        *type = DI_TEXT;
        v.string = d->di_semopt;
        break;
case    DI_V_HELP:
        *type = DI_TEXT;
        v.string = d->di_help;
        break;
case    DI_V_SELLIST:
        *type = DI_TEXT;
        v.string = d->di_sellist;
        break;
case    DI_V_TOOL:
        *type = DI_TEXT;
        v.string = d->di_tool;
        break;
case    DI_V_LBOUND:
        if (d->di_type == DI_FVAL)
                *type = DI_FVAL;
        else
                *type = DI_IVAL;
        v = d->di_lower;
        break;
case    DI_V_UBOUND:
```

```
                if (d->di_type == DI_FVAL)
                        *type = DI_FVAL;
                else
                        *type = DI_IVAL;
                v = d->di_upper;
                break;
        case    DI_V_RESULT:
                if (d->di_type == DI_TEXT || d->di_type == DI_SELECT)
                        *type = DI_TEXT;
                else if (d->di_type == DI_FVAL)
                        *type = DI_FVAL;
                else
                        *type = DI_IVAL;
                v = d->di_result;
                break;
        }

        return  v;
}

/*
 * d_put_value:
 *      sets an DI_VALUE into a dialog
 */

static int
d_put_value(dialog,which,v,type)
DIALOG          dialog;
int             which;
DI_VALUE        v;
int             type;
{
        DIALOG_INT      *d = (DIALOG_INT *)dialog;
        static int      d_bounds_type_ok(), d_result_type_ok();

        if (d == DI_NULL)
                return  -1;

        switch (which) {
        case    DI_V_TYPE:
                if (type != DI_IVAL)
                        return  -1;
                d->di_type = v.integer;
                break;
        case    DI_V_SEMTYPE:
                if (type != DI_IVAL)
                        return  -1;
                d->di_stype = v.integer;
                break;
        case    DI_V_OPTION:
                if (type != DI_IVAL)
                        return  -1;
                d->di_option = v.integer;
                break;
        case    DI_V_NAME:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_name != NULL)
                        mmwb_stringfree(d->di_name);
                d->di_name = v.string;
                break;
        case    DI_V_PROMPT:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_prompt != NULL)
                        mmwb_stringfree(d->di_prompt);
                d->di_prompt = v.string;
                break;
        case    DI_V_SEMOPT:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_semopt != NULL)
                        mmwb_stringfree(d->di_semopt);
                d->di_semopt = v.string;
                break;
        case    DI_V_HELP:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_help != NULL)
                        mmwb_stringfree(d->di_help);
                d->di_help = v.string;
                break;
        case    DI_V_SELLIST:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_sellist != NULL)
                        mmwb_stringfree(d->di_sellist);
                d->di_sellist = v.string;
                break;
        case    DI_V_TOOL:
                if (type != DI_TEXT)
                        return  -1;
                if (d->di_tool != NULL)
                        mmwb_stringfree(d->di_tool);
```

```
                d->di_tool = v.string;
                break;
        case    DI_V_LBOUND:
                if (!d_bounds_type_ok(d->di_type,type))
                        return  -1;
                d->di_lower = v;
                break;
        case    DI_V_UBOUND:
                if (!d_bounds_type_ok(d->di_type,type))
                        return  -1;
                d->di_upper = v;
                break;
        case    DI_V_RESULT:
                if (!d_result_type_ok(d->di_type,type))
                        return  -1;
                if (d->di_type == DI_TEXT && d->di_result.string != NULL)
                        mmwb_stringfree(d->di_result.string);
                d->di_result = v;
                break;
        }

        return  0;
}

static int
d_bounds_type_ok(vtype,type)
int     vtype,type;
{
        switch(vtype) {
        case    DI_FVAL:
                if (type != DI_FVAL)
                        return  0;
                break;
        case    DI_IVAL:
        case    DI_BOOL:
        case    DI_YESNO:
        case    DI_TEXT:
        case    DI_SELECT:
                if (type != DI_IVAL)
                        return  0;
                break;
        }

        return  -1;
}

static int
d_result_type_ok(vtype,type)
int     vtype,type;
{
        switch(vtype) {
        case    DI_IVAL:
        case    DI_BOOL:
        case    DI_YESNO:
                if (type != DI_IVAL)
                        return  0;
                break;
        case    DI_FVAL:
                if (type != DI_FVAL)
                        return  0;
                break;
        case    DI_TEXT:
        case    DI_SELECT:
                if (type != DI_TEXT)
                        return  0;
                break;
        }

        return  -1;
}
```

twb/browse.h

```
/*
 * $Id: browse.h,v 1.3 90/04/16 19:19:36 conrad Exp $
 *
 * browse.h - browser interface defines
 *
 */

#ifndef   BROWSE_INCLUDE

#include <machdep.h>

        /* Types */
typedef  void *     BROWSER;
typedef  void * FIELD;                  /* This may not be needed */

        /* Defines */

/* Browser Types */
#define  BR_FORM                        /* Form browser */
#define  BR_LIST 2                      /* List browser */
#define  BR_COL 3                       /* Row/Column browser */

/* Field Types */
#define  BR_TYPEIN       1              /* Type-in field */
#define  BR_BUTTON       2              /* Button (selection) */
#define  BR_LABEL        3              /* Label field (inactive) */
#define  BR_PROMPT       4              /* A prompt field */
#define  BR_TITLE        5              /* A title field */
#define  BR_TOGGLE       6              /* A toggle */
#define  BR_SELLIST      7              /* A list of values */

/* Format Types */
#define  BR_CENTERED     1              /* A centered field */
#define  BR_RJUST        2              /* Right-justified field */
#define  BR_LJUST        3              /* Left-justified field */

/* Display information */
#define  BR_INVERSE      0x100          /* Inverse video */
#define  BR_BLINKING     0x200          /* Blinking */
#define  BR_UNDERLINED   0x400          /* Underlined */
#define  BR_BOLD         0x800          /* Bold */
#define  BR_SCROLLING    0x1000         /* Scrolling field */
#define  BR_NOINVERSE    0x2000         /* Inhibit inverse */

/* Flags for br_field */
#define  BR_STUFF_FIELD 1               /* Stuff this field into the
                                           previous "private" */
#define  BR_IS_FIELD     2              /* This is a field, not an other */

/* Flags for br_display */
#define  BR_NOCLEAR      1              /* Don't clear the window */
#define  BR_OVERLAY      2              /* Overlay this browser */

/* Button tokens */
#define  BR_UP           (char)255
#define  BR_DOWN         (char)254
#define  BR_RIGHT        (char)253
#define  BR_LEFT         (char)252
#define  BR_RETURN       (char)251
#define  BR_SELECT       (char)250
#define  BR_ESCAPE       (char)249
#define  BR_CLEAR        (char)248

/*
 * Some defines to save typing
 */
#define  VNULL           (void *)NULL
#define  FVNULL          (void *(*)())NULL
#define  FNULL           (void (*)())NULL

        /* Functions */
extern   BROWSER  br_create();          /* Create a browser */
extern   int      br_format();          /* Format the browser */
extern   int      br_title();           /* Add a title */
extern   int      br_prompt();          /* Add a prompt */
extern   int      br_button();          /* Add a button */
extern   int      br_toggle();          /* Add a toggle input */
extern   int      br_select_list();              /* Add a selection list */
extern   int      br_pair();            /* Add a prompt-input pair
                                           (or a selection-data pair) */
extern   int      br_typein();          /* Add a typein field */
extern   int      br_label();           /* Add a label */
extern   int      br_display();         /* Display a browser */
extern   void *   br_activate();        /* Activate the browser */
extern   void     br_free();            /* Free a browser */
extern   void     br_clear();           /* Erase a browser display */
extern   int      br_field();           /* Add a field (not recommended) */
extern   int      br_nscreen();         /* Return the number of screens */
extern   void *   br_private();         /* Return private data of current selection */
extern   char *   br_string();          /* Return string (label) data of
                                           current selection */
extern   void     br_clear_field();             /* Clear a field */
extern   void     br_reset();           /* Reset the browser */
extern   char *   br_prompt_input();
```

twb/browse.h

```
/* Intrisic functions */
extern  int     br_browse();        /* Call another browser recursively */
extern  void *  br_get_typein();            /* Input textual data */
extern  void *  br_get_toggle();            /* Input a toggle */
extern  void *  br_get_sellist();           /* Input a sellist */

#define BROWSE_INCLUDE
#endif  BROWSE_INCLUDE
```

twb/cmdtab.h

```
/*
 * $Id: cmdtab.h,v 2.0 90/01/02 22:37:41 scooter Exp $
 *
 * cmdtab.h:
 *      Command table definitions
 *
 * $Log:        cmdtab.h,v $
 * Revision 2.0  90/01/02  22:37:41  scooter
 * Libbrowse release
 *
 * Revision 1.1  89/09/18  14:13:16  scooter
 * Initial revision
 *
 *
 */

/*
 * Command table
 *
 * The following structure contains the information necessary to
 * process a command.  Actually a command is processed by a possible
 * series of tables.  If cmd_args or cmd_oargs is (-1), then any
 * possible number of arguments are possible.
 */
typedef struct cmd_st {
        char    *cmd_name;      /* Command name */
        char    *cmd_desc;      /* Short description */
        char    *cmd_help;      /* Long description */
        int      cmd_args;      /* Number of mandatory arguments */
        int      cmd_oargs;     /* Number of optional arguments */
        int     (*cmd_func)();  /* Function to call */
        int      cmd_parg;      /* A private argument (passed to cmd_func) */
        struct   cmd_st *cmd_next; /* Next table to branch (if args) */
} COMMAND;

#define  MAXTOKS 25
```

twb/disp.h

```
/*
 * $Id: disp.h,v 2.0 90/01/02 22:37:55 scooter Exp $
 *
 * disp.h:
 *          include file for twb display functions
 *
 * $Log:            disp.h,v $
 * Revision 2.0  90/01/02  22:37:55  scooter
 * Libbrowse release
 *
 * Revision 1.2  89/11/14  21:41:04  scooter
 * delint and added NAME, COPY, and DELETE commands
 *
 * Revision 1.1  89/11/01  23:45:50  scooter
 * Initial revision
 *
 *
 */

#ifndef  DISP_INCLUDE
#define  DISP_INCLUDE

#include          <signal.h>
#include          <sys/time.h>
#include          <sys/types.h>

/*
 * Externals
 */
extern   void      disp_init();
extern   int       disp_header();
extern   void      disp_error();
extern   void      disp_message();
extern   void      disp_errclear();

extern   void      *Header;
extern   void      *Command;
extern   void      *Dialog;
extern   void      *Display;
extern   void      *Error;

#endif   DISP_INCLUDE
```

twb/io.h

```
/*
 * $Id: io.h,v 1.1 90/01/02 22:30:27 scooter Exp $
 *
 * io.h - I/O interface for browsers
 *
 */

#include        "browse.h"

        /* Externals */
extern  void    *io_curwindow;          /* The current window */

        /* Input-Output functions */
extern  void    io_init();              /* Initialization */
extern  void    io_end();               /* Termination */
extern  void    *io_create_window();    /* Create a window */
extern  void    *io_create_overlay();   /* Create an overlay window */
extern  void    io_destroy_window();    /* Delete a window */
extern  char    io_get_button();        /* Input a button press */
extern  char    *io_get_text();         /* Text input */
extern  int     io_put_text();          /* Output some text */
extern  int     io_win_clear();         /* Clear this window */
extern  int     io_scroll();            /* Scroll a window */
extern  int     io_set_flags();         /* Set display flags */
extern  int     io_clear_flags();       /* Clear display flags */
extern  int     io_get_dims();          /* Get the window dimensions */

        /* User-provided functions */
extern  char    *user_input();          /* User input function */
```

twb/twb.h

```
/*
 * $Id: twb.h,v 2.1 90/04/16 18:22:12 scooter Exp $
 *
 * twb.h:
 *          include file for twb
 *
 * $Log:            twb.h,v $
 * Revision 2.1  90/04/16  18:22:12  scooter
 * Include string.h
 *
 * Revision 2.0  90/01/02  22:37:57  scooter
 * Libbrowse release
 *
 * Revision 1.5  89/11/14  21:38:17  scooter
 * delint and added NAME, COPY, and DELETE commands
 *
 * Revision 1.4  89/11/01  23:44:12  scooter
 * Cleaned up display and browser interface
 *
 * Revision 1.3  89/10/30  17:25:22  scooter
 * lint clean-up
 *
 * Revision 1.2  89/10/23  16:07:55  scooter
 * Working version - genie upload
 *
 * Revision 1.1  89/09/18  14:13:24  scooter
 * Initial revision
 *
 *
 */

#include <mmwbmisc.h>
#include <ofs.h>
#include <sm.h>
#include <dispatcher.h>
#include <curses.h>
#include <syslog.h>
#include "machdep.h"
#include <sys/types.h>
#include <string.h>
#ifdef    __STDC__
#include <stdarg.h>
#else
#include <varargs.h>
#endif

#ifndef   FALSE
#define   FALSE 0
#endif

#ifndef   TRUE
#define TRUE 1
#endif

extern    char      *strcpy();

/*
 * Globals
 */
extern    int       verbose_flag;
extern    int       log_flag;

extern    OFS_OBJECT      fs_visible;
```

twb/commands.c

```
#ifndef lint
static char RCSid[] = "$Id: commands.c,v 2.3 90/04/16 22:00:50 scooter Exp $";
#endif lint

/*
 * commands.c:
 *          command processing for twb
 *
 */

#include        "twb.h"
#include        "cmdtab.h"
#include        "browse.h"
#include        "disp.h"
#include        <stdio.h>

/*
 * Command tables
 */
static  int     help(),launch(),quit(),select(),answer(),set();
static  int     status(),kill();
static  int     list_fs(),list_dialogs();
static  int     browse(),createproj();
static  int     launch_tool();
static  int     delete(),name(),proj_select();
static  int     design_protocol();

static COMMAND  cmd_list[] = {
        { "PROTOCOLS","List known protocols",NULL,0,2,
                                        list_fs,OBJ_T_PROTOCOL,NULL },
        { "INSTRUMENTS","List known instruments",NULL,0,2,
                                        list_fs,OBJ_T_INSTRUMENT,NULL },
        { "PROJECTS","List projects",NULL,0,2,list_fs,OBJ_T_PROJECT,NULL },
        { "OBJECTS","List objects within a project",NULL,0,2,list_fs,-1,NULL },
        { "DIALOGS","List unanswered dialogs",NULL,0,2,list_dialogs,0,NULL },
        { NULL }
};

static COMMAND  cmd_tab[] = {
        { "QUIT","Leave twb",NULL,0,0,quit,0,NULL },
        { "STATUS","Display the status of the session",NULL,0,0,status,0,NULL },
        { "BEGIN","Begin a protocol",NULL,0,1,launch,OBJ_T_PROTOCOL,NULL },
        { "END","End a protocol",NULL,0,1,kill,OBJ_T_PROTOCOL,NULL },
#if     0
        { "START","Start an instrument",NULL,0,1,launch,OBJ_T_INSTRUMENT,N
        { "STOP","Stop an instrument",NULL,0,1,kill,OBJ_T_INSTRUMENT,NULL },
#endif
        { "LIST","List various things",NULL,0,2,NULL,0,cmd_list },
        { "HELP","Display information about commands",NULL,0,1,help,0,NULL },
        { "ANSWER","Repond to a dialog",NULL,0,1,answer,0,NULL },
        { "CREATE","Create a new project",NULL,0,1,createproj,0,NULL },
        { "BROWSE","Browse the project tree",NULL,0,0,browse,0,NULL },
        { "DELETE","Delete an object",NULL,0,1,delete,0,NULL },
        { "NAME","Name an object",NULL,0,2,name,0,NULL },
        { "SELECT","Select a project",NULL,0,1,proj_select,0,NULL },
        { "MODIFY","Modify an option sheet",NULL,0,1,design_protocol,OBJ_T_PROTOCOL,NULL
#if     0
        { "MENU","Switch to MENU mode",NULL,0,0,NULL,0,cmd_tab },
        /* NOTE: Set has its own tables, so we let it do its own processing */
        { "SET","Set various internal values",NULL,0,-1,set,0,NULL },
        { "COPY","Make a copy of an object",NULL,0,2,copy,0,NULL },
#endif
        { NULL }
};

/*
 * Return the starting command table
 */
COMMAND
cmd_get_tab()
{
        return  &cmd_tab[0];
}

/* ARGSUSED */
static int
quit(cmd,arg,ntoks,tokens)
COMMAND cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern void session_close();

        if (ntoks > 0)
                session_close(tokens[1]);
        else
                session_close((char *)NULL);
        return  -1;
}

/* ARGSUSED */
static int
status(cmd,arg,ntoks,tokens)
```

```
COMMAND *cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern void session_disp_status();

        session_disp_status();
        return  0;
}

/*
 * launch:
 *      Execute a tool
 */

/* ARGSUSED */
static int
launch(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
    register SM_TOOLDES   *protocols,*instruments;
        char                    *tool_name,*tool_browse();

        protocols = sm_list_protocol((OFS_OBJECT) NULL);
        instruments = sm_list_instrument((OFS_OBJECT) NULL);

        if (ntoks == 0) {
                if (arg == OBJ_T_PROTOCOL) {
                        tool_name = tool_browse("Protocols",
                                "Select the desired protocol", protocols,1);
                        if (tool_name == NULL)
                                return;
                        if (launch_tool(cmd,tool_name,protocols))
                                return;
                } else {
                        tool_name = tool_browse("Instruments",
                                "Select the desired instrument", instruments,1)
                        if (tool_name == NULL)
                                return;
                        if (launch_tool(cmd,tool_name,instruments))
                                return;
                }
        } else {
                tool_name = tokens[0];
                if (arg == OBJ_T_PROTOCOL) {
                        if (launch_tool(cmd,tool_name,protocols))
                                return;
                        if (launch_tool(cmd,tool_name,instruments))
                                return;
                } else {
                        if (launch_tool(cmd,tool_name,instruments))
                                return;
                        if (launch_tool(cmd,tool_name,protocols))
                                return;
                }
                disp_error("No such tool: %s", tool_name);
        }
        return;
}

static int
launch_tool(cmd,name,tlist)
COMMAND *cmd;
char    *name;
SM_TOOLDES *tlist;
{
        register SM_TOOLDES     *td;
        for (td = tlist; td != NULL; td = td->next) {
                if (strcmp(td->Name, name) != 0)
                        continue;
                disp_message(0,"Launching %s...",td->Name);
                if (sm_invoke(td) == NULL)
                        disp_error("Unable to %s %s: %s",
                                cmd->cmd_name,name,sm_error());
                return  TRUE;
        }
        return  FALSE;
}

/*
 * kill:
 *      Kill a protocol or an instrument
 */

/* ARGSUSED */
static int
kill(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int     arg;
```

twb/commands.c

```
int	ntoks;
char	*tokens[];
{
	extern	int	session_browse_status();
	int	tnum;

	if (ntoks < 1) {
		tnum = session_browse_status(arg);
		if (tnum < 1)
			return	0;
	} else
		tnum = atoi(tokens[0]);
	if (sm_kill(tnum)){
		if (arg == OBJ_T_PROTOCOL)
			disp_error("Unable to END protocol - %s",sm_error());
		else
			disp_error("Unable to STOP instrument - %s",sm_error(
	} else {
		if (arg == OBJ_T_PROTOCOL)
			disp_message(TRUE,"Protocol ENDed");
		else
			disp_message(TRUE,"Instrument STOPed");
	}

	return	0;
}

/*
 * list_fs:
 *	list OFS objects
 */

/* ARGSUSED */
static int
list_fs(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int	arg;
int	ntoks;
char	*tokens[];
{
	int	nobj;
	OFS_OBJ_DESC	*desc;
	char	buf[180];
	extern	OFS_OBJ_DESC *obj_browse();

	nobj = obj_count(sm_get_curr_proj(),(char *)NULL,-1,arg);
	desc = (OFS_OBJ_DESC *)emalloc(sizeof(OFS_OBJ_DESC)*(nobj+1));

	nobj = obj_list(sm_get_curr_proj(),(char *)NULL,-1,arg,desc,nobj);
	desc[nobj].name = NULL;
	(void) sprintf(buf,"%s List",cmd->cmd_name);
	(void) obj_browse(buf,(char *)NULL,desc,0);
	(void) free((char *)desc);
	return	0;
}

/*
 * help:
 *	Display help text
 */

/* ARGSUSED */
static int
help(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int	arg;
int	ntoks;
char	*tokens[];
{
	extern	int	help_browse();
	(void) help_browse(cmd_tab,"HELP",(char *)NULL);
	return	0;
}

/*
 * answer:
 *	Answer a dialog
 */

/* ARGSUSED */
static int
answer(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int	arg;
int	ntoks;
char	*tokens[];
{
	int	dnum;
	extern	int	di_list();
	extern	void	di_answer();

	if (ntoks < 1) {
		dnum = di_list(TRUE,"Pending Dialogs",
```

```
                        "Select the dialog to answer");
                if (dnum == -1)
                        return  0;
                di_answer(dnum);
        } else
                di_answer(atoi(tokens[0]));
        return  0;
}

/*
 * list_dialogs:
 *      List all pending dialogs
 */

/* ARGSUSED */
static int
list_dialogs(cmd,arg,ntoks,tokens)
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern  int     di_list();

        (void) di_list(FALSE,"Pending Dialogs", (char *)NULL);
        return  0;
}

/*
 * browse:
 *      Browse the project tree
 */

/* ARGSUSED */
static int
browse(cmd,arg,ntoks,tokens)
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern int obj_ofs_browse();
        (void) obj_ofs_browse((OFS_OBJECT)NULL,-999,(char *)NULL,0);
        return  0;
}

/*
 * createproj:
 *      create a new project in the current project
 */

/* ARGSUSED */
static int
createproj(cmd,arg,ntoks,tokens)
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        OFS_OBJECT      newproj;
        char            *projname;
        extern  void    *Error;

        if (ntoks < 1) {
                /*
                 * The user didn't give us a name, get one
                 */
                projname = br_prompt_input(Error,
                                "Enter the name of the new project:",
                                -1,-1,-1);
                disp_errclear();
                if (projname == NULL)
                        return  0;

                newproj = proj_create(sm_get_curr_proj(),projname,
                        (char *)NULL,(char *)NULL);
        } else
                newproj = proj_create(sm_get_curr_proj(),tokens[0],
                        (char *)NULL,(char *)NULL);

        if (newproj == NULL)
                disp_error("Unable to create %s: %s",tokens[0],dm_error());
        return  0;
}


/*
 * delete:
 *      Delete an object
 */

/* ARGSUSED */
static int
delete(cmd,arg,ntoks,tokens)
```

```
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern int obj_ofs_browse();
        extern void obj_ofs_delete();
        int                     nobj;
        OFS_OBJ_DESC    desc;

        if (ntoks < 1) {
                /*
                 * Invoke the browser
                 */
                (void) obj_ofs_browse((OFS_OBJECT)NULL,-999,
                        "Type 'd' to delete the desired object",'d');
        } else {
                nobj = obj_count(sm_get_curr_proj(),tokens[0],-1,-1);
                if (nobj > 1) {
                        disp_error("Too many objects by that name");
                        return 0;
                }
                (void) obj_list(sm_get_curr_proj(),tokens[0],-1,-1,&desc,1);
                obj_ofs_delete(sm_get_curr_proj(),&desc);
        }
        return  0;
}

/*
 * name:
 *      Name an object
 */

/* ARGSUSED */
static int
name(cmd,arg,ntoks,tokens)
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        extern int obj_ofs_browse();
        extern void obj_ofs_rename();
        int                     nobj;
        OFS_OBJ_DESC    desc;

        if (ntoks < 1) {
                /*
                 * Invoke the browser
                 */
                (void) obj_ofs_browse((OFS_OBJECT)NULL,-999,
                        "Type 'n' to rename the desired object",'n');
        } else {
                nobj = obj_count(sm_get_curr_proj(),tokens[0],-1,-1);
                if (nobj > 1) {
                        disp_error("Too many objects by that name");
                        return 0;
                }
                (void) obj_list(sm_get_curr_proj(),tokens[0],-1,-1,&desc,1);
                if (ntoks > 1)
                        obj_ofs_rename(sm_get_curr_proj(),&desc,tokens[1]);
                else
                        obj_ofs_rename(sm_get_curr_proj(),&desc,NULL);
        }
        return  0;
}

/*
 * proj_select:
 *      select a new default project
 */

/* ARGSUSED */
static int
proj_select(cmd,arg,ntoks,tokens)
COMMAND2 cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
        OFS_OBJECT      obj;
        extern int obj_ofs_browse();
        OFS_OBJ_DESC    desc;
        int                     nobj;

        if (ntoks < 1) {
                /*
                 * Invoke the browser
                 */
                (void) obj_ofs_browse((OFS_OBJECT)NULL,-999,
                        "Type '+' to select the desired project",'+');
        } else {
                nobj = obj_count(sm_get_curr_proj(),tokens[0],-1,OBJ_T_PROJECT);
                if (nobj > 1) {
```

```
                        disp_error("Too many projects by that name");
                        return 0;
                }
                (void) obj_list(sm_get_curr_proj(),tokens[0],-1,-1,&desc,1);
                obj = obj_open_by_ident(sm_get_curr_proj(),desc.ident);
                sm_set_curr_proj(proj_open(obj));
                disp_header();
        }
        return  0;
}

/*
 * edit_option:
 *      Allow the user to edit an option sheet.
 */

/* ARGSUSED */
static int
design_protocol(cmd,arg,ntoks,tokens)
COMMAND *cmd;
int     arg;
int     ntoks;
char    *tokens[];
{
    register SM_TOOLDES    *protocols,*instruments;
        char                    *tool_name,*tool_browse();
        int                     edit_options();

        protocols = sm_list_protocol((OFS_OBJECT) NULL);
        instruments = sm_list_instrument((OFS_OBJECT) NULL);

        if (ntoks == 0) {
                if (arg == OBJ_T_PROTOCOL) {
                        tool_name = tool_browse("Protocols",
                                "Select the desired protocol", protocols,1);
                        if (tool_name == NULL)
                                return;
                        if (edit_options(cmd,tool_name,protocols))
                                return;
                } else {
                        tool_name = tool_browse("Instruments",
                                "Select the desired instrument", instruments,1)
                        if (tool_name == NULL)
                                return;
                        if (edit_options(cmd,tool_name,instruments))
                                return;
                }
        } else {
                tool_name = tokens[0];
                if (arg == OBJ_T_PROTOCOL) {
                        if (edit_options(cmd,tool_name,protocols))
                                return;
                        if (edit_options(cmd,tool_name,instruments))
                                return;
                } else {
                        if (edit_options(cmd,tool_name,instruments))
                                return;
                        if (edit_options(cmd,tool_name,protocols))
                                return;
                }
                disp_error("No such tool: %s", tool_name);
        }
        return;
}

static int
edit_options(cmd,name,tlist)
COMMAND *cmd;
char    *name;
SM_TOOLDES *tlist;
{
        register SM_TOOLDES     *td;
        extern   void       di_edit_option();

        for (td = tlist; td != NULL; td = td->next) {
                if (strcmp(td->Name, name) != 0)
                        continue;
                disp_message(0,"Editting %s...",td->Name);
                di_edit_option(cmd->cmd_parg,td);
                return  TRUE;
        }
        return  FALSE;
}
```

twb/disp.c

```
#ifndef lint
static char RCSid[] = "$Id: disp.c,v 2.0 90/01/02 22:38:48 scooter Exp $";
#endif lint

/*
 * disp.c:
 *          curses interface for twb
 *
 */
#include          "twb.h"
#include          "disp.h"
#include          "browse.h"
#include          "io.h"

/*
 * Globals
 */
void      *Header = NULL;          /* header window */
static int          Header_lines;  /* # of lines in the header window */
#define   DEFHEADER_L   1

void      *Dialog = NULL;          /* dialog notification window */
static int          Dialog_lines;  /* # of lines in the dialog window */
#define   DEFDIALOG_L   2

void      *Command = NULL;         /* command input window */
static int          Command_lines; /* # of lines in the command window */
#define   DEFCOMMAND_L 2

void      *Display = NULL;         /* information display (main) window */
static int          Display_lines; /* LINES-Command_lines-Dialogs_lines-Head

void      *Error = NULL;                    /* Error/Message display */
static    int       Error_lines = 2;        /* No discussion ! */
static    int       Error_disp = FALSE;     /* Error or message on screen */

static    char      *disp_progname;
static    void      *disp_curwindow = NULL;

/*
 * disp_init:
 *          Initialize the display
 */
void
disp_init(progname)
char      *progname;
{
          int       err,maxx,maxy,cury;

          /*
           * Start by setting the sizes
           */
          Dialog_lines = set_get_int("dialog lines",&err);
          if (err)
                    Dialog_lines = DEFDIALOG_L;

          Command_lines = set_get_int("command lines",&err);
          if (err)
                    Command_lines = DEFCOMMAND_L;

          Header_lines = set_get_int("header lines",&err);
          if (err)
                    Header_lines = DEFHEADER_L;

          disp_progname = progname;

          /*
           * Now create the windows
           */
          io_init(stdin);
          (void) io_get_dims((void *)NULL,&maxx,&maxy);

          Header = io_create_window(0,0,maxx,Header_lines);
          cury = Header_lines;

          Display_lines = maxy-Dialog_lines-Command_lines-
                                        Header_lines-Error_lines;
          Display = io_create_window(0,cury,maxx,cury+Display_lines);
          cury += Display_lines;

          Error = io_create_window(0,cury,maxx,cury+Error_lines);
          cury += Error_lines;

          Command = io_create_window(0,cury,maxx,cury+Command_lines);
          cury += Command_lines;

          Dialog = io_create_window(0,cury,maxx,maxy);

          (void) disp_header();          /* Display the header */

          return;
}

/*
```

twb/disp.c

```
 * disp_close:
 *      return to normal
 */
void
disp_close()
{
        io_end();
        return;
}


/*
 * disp_header:
 *      display the current header information.  The
 *      header contains the current project, the title, and the date and
 *      time.
 */
int
disp_header()
{
        extern  char    *proj_name();
        time_t  clock, time();
        void    *lastwindow = io_curwindow;
        char    *curr = proj_name(sm_get_curr_proj());
        char    *t,*strrchr(),*ctime();
        int     maxx,maxy;

        /*
         * Set the clock
         */
#ifndef DEBUG
        (void) alarm(0);
        (void) signal(SIGALRM,disp_header);
        (void) alarm(60);
#endif  DEBUG

        (void) io_get_dims(Header,&maxx,&maxy);

        /*
         * Clear the header
         */
        (void) io_win_clear(Header);

        /*
         * Output the object name
         */
        (void) io_put_text(Header,0,0,curr);

        /*
         * Output our name
         */
        (void) io_put_text(Header,(maxx-strlen(disp_progname))/2,0,disp_progname);

        /*
         * Output the date/time
         */
        clock = time((time_t *)0);
        t = ctime(&clock);
        t[16] = '\0';
        (void) io_put_text(Header,maxx-strlen(t)-1,0,t);

        (void) io_display(Header);
        (void) io_display(lastwindow);
        return;
}

/*
 * disp_dialog:
 *      display a line in the dialog window
 */
void
disp_dialog(string)
char    *string;
{
        void    *lastwindow = io_curwindow;
        int     maxx,maxy;

        if (string == NULL)
                return;

        (void) io_get_dims(Dialog,&maxx,&maxy);

        (void) io_scroll(Dialog);
        (void) io_set_flags(Dialog,BR_INVERSE);
        (void) io_put_text(Dialog,(maxx-strlen(string))/2,Dialog_lines-1,string);
        (void) io_clear_flags(Dialog,BR_INVERSE);
        (void) io_display(Dialog);
        (void) io_display(lastwindow);
        return;
}

/*
 * disp_error:
 *      display an error on the screen
```

twb/disp.c

```
 */
/* VARARGS1 */
void
disp_error(format, va_alist)
char *format;
#ifdef    __STDC__
va_list   va_alist;
#else
va_dcl
#endif
{
        va_list   args;
        void      *lastwindow = io_curwindow;
        char      buf[BUFSIZ];

#ifdef    __STDC__
        va_start(args, va_alist);
#else
        va_start(args);
#endif
        buf[0] = '\007';
        (void) vsprintf(&buf[1],format,args);
        va_end(args);

        if (Error == NULL) {
                fputs(buf,stderr);
                fprintf(stderr,"\n");
        } else {
                (void) io_scroll(Error);
        /* Do a refresh to something always happens */
                (void) io_display(Error);
                (void) io_set_flags(Error,BR_INVERSE);
                (void) io_put_text(Error,-1,-1,buf);
                (void) io_clear_flags(Error,BR_INVERSE);
                (void) io_display(Error);
                (void) io_display(lastwindow);
                Error_disp = TRUE;
        }
        return;
}

/*
 * disp_message:
 *        display a message on the screen
 */
/* VARARGS1 */
void
disp_message(rst_cursor,va_alist)
int       rst_cursor;
#ifdef    __STDC__
va_list   va_alist;
#else
va_dcl
#endif
{
        void      *lastwindow = io_curwindow;
        va_list   args;
        char      *format;
        char      buf[BUFSIZ];

#ifdef    __STDC__
        va_start(args, va_alist);
#else
        va_start(args);
#endif
        format = va_arg(args, char *);
        (void) vsprintf(buf,format,args);
        va_end(args);

        if (Error == NULL) {
                fputs(buf, stdout);
                fprintf(stdout,"\n");
        } else {
                (void) io_scroll(Error);
        /* Do a refresh to something always happens */
                (void) io_display(Error);
                (void) io_put_text(Error,-1,-1,buf);
                (void) io_display(Error);
                if (rst_cursor)
                        (void) io_display(lastwindow);
                Error_disp = TRUE;
        }
        return;
}

/*
 * disp_errclear:
 *        Clear the errr/message area
 */
void
disp_errclear()
{
        void      *lastwindow = io_curwindow;
```

twb/disp.c

```
        if (Error != NULL) {
                (void) io_win_clear(Error);
                (void) io_display(Error);
                (void) io_display(Display);
                (void) io_display(lastwindow);
                Error_disp = FALSE;
        }
        return;
}
```

twb/fs.c

```
#ifndef lint
static char RCSid[] = "$Id: fs.c,v 2.0 90/01/02 22:38:44 scooter Exp $";
#endif lint

/*
 * fs.c:
 *          ofs interface for twb
 *
 */

#include          "twb.h"
#include          <syslog.h>

static OFS_OBJECT          fs_sys = NULL;      /* System project */
static OFS_OBJECT          fs_home = NULL;     /* Home project */
static OFS_OBJECT          fs_curr = NULL;     /* Currently selected project */
static OFS_OBJECT          fs_visible = NULL; /* Currently visible project */

/*
 * fs_init - initialize the default projects
 */

void
fs_init()
{
          /*
           * Open up project directories
           */
          if (verbose_flag)
                    fprintf(stdout,"Initializing default projects\n");

          if (log_flag)
                    syslog(LOG_INFO,"calling ofs_sys_init");

          if ((fs_sys = ofs_sys_init((char *) NULL)) == NULL) {
                    fprintf(stderr, "ofs_sys_init: %s\n", dm_error());
                    if (dm_errno == DM_EDISPATCHER)
                              fprintf(stderr, "dispatcher: %s\n", dispatcher_error());
                    exit(1);
          }
          if (log_flag)
                    syslog(LOG_INFO,"ofs_sys_init returns");

          sm_set_sys_proj(fs_sys);
          if (log_flag)
                    syslog(LOG_INFO,"calling ofs_proj_init");

          if ((fs_home = ofs_proj_init((char *) NULL)) == NULL) {
                    fprintf(stderr, "ofs_proj_init: %s\n", dm_error());
                    if (dm_errno == DM_EDISPATCHER)
                              fprintf(stderr, "dispatcher: %s\n", dispatcher_error());
                    exit(1);
          }
          if (log_flag)
                    syslog(LOG_INFO,"ofs_proj_init returns");

          sm_set_home_proj(fs_home);
          sm_set_curr_proj(fs_home);

          if (verbose_flag)
                    fprintf(stdout,"Default projects initialized\n");

          return;
}

/*
 * fs_close:
 *          Close the open projects
 */

void
fs_close()
{
          (void) proj_close(sm_get_curr_proj());
          (void) proj_close(sm_get_home_proj());
          (void) proj_close(sm_get_sys_proj());
          return;
}
```

twb/help.c

```
#ifndef lint
static char *RCSid = "$Id: help.c,v 2.0 90/01/02 22:38:50 scooter Exp $";
#endif lint

/*
 * help.c - help routines
 */

#include        "twb.h"
#include        "browse.h"
#include        "cmdtab.h"
#include        "disp.h"

/*
 * Help browser
 */

int
help_browse(com_table, title, prompt)
COMMAND com_table[];
char    *title;
char    *prompt;
{
        BROWSER browser;
        int             i = 0;
        COMMAND         *cmd;
        char            lasttok;

        browser = br_create(Display,BR_FORM,title,prompt);

        /*
         * Now, put together the browser list
         */

        while (com_table[i].cmd_name != NULL) {
                if (br_pair(browser,mmwb_cpystring(com_table[i].cmd_name),
                        BR_LABEL,mmwb_cpystring(com_table[i].cmd_desc),
                        0,BR_LJUST,strlen(com_table[i].cmd_desc),(char *)NU
                        (void *)&com_table[i],FVNULL,FNULL)) {
                                disp_error("br_pair failed!");
                                return  -1;
                        }
                i++;
        }

        /*
         * OK, now format it
         */
        if (br_format(browser)) {
                disp_error("Unable to format help browser");
                exit(1);
        }

        if (br_display(browser,-1,0)) {
                disp_error("Unable to display help browser");
                exit(1);
        }

        /*
         * Now let the user select a command for further information
         */
        while ((cmd = (COMMAND *)br_activate(browser,&lasttok)) != NULL) {
                if (cmd->cmd_next != NULL) {
                        if (help_browse(cmd->cmd_next,title,prompt)) {
                                br_free(browser);
                                return  1;
                        } else
                                (void) br_display(browser,-1,0);
                } else if (cmd->cmd_help != NULL)
                        disp_message (TRUE, "Long help not implemented yet");
                else
                        disp_error ("No further information available");
        }
        br_free(browser);
        if (lasttok == BR_ESCAPE)
                return  1;

        return 0;
}
```

twb/input.c

```
#ifndef lint
static char RCSid[] = "$Id: input.c,v 2.2 90/04/16 18:26:43 scooter Exp $";
#endif lint

/*
 * input.c:
 *      input handling
 *
 */

#include        "twb.h"
#include        "browse.h"
#include        "cmdtab.h"
#include        "disp.h"
#include        <errno.h>

void
command_input(stream,init)
FILE    *stream;
int     init;
{
        char    *result;
        char    *toks[MAXTOKS];
        char    buffer[BUFSIZ];
        int     ntoks;
        extern  COMMAND *cmd_get_tab();

        while(1) {
                if (init) {
                        result = fgets(buffer,BUFSIZ,stream);
                } else {
                        (void) io_scroll(Command); /* Scroll the window */
                        result = br_prompt_input(Command,"> ",0,-1,-1);
                }

                disp_errclear();

                if (result != NULL && *result != '\n') {
                        ntoks = tokenize(result,toks,MAXTOKS);
                        if (cmd_process(ntoks,toks,cmd_get_tab()) < 0)
                                return;
                }
        }
}

static int
cmd_process(ntoks,tokens,table)
int     ntoks;
char    *tokens[];
COMMAND table[];
{
        COMMAND *command,*cmd_find();

        command = cmd_find(tokens[0],table);
        if (command == NULL) {
                disp_error("Illegal command: %s",tokens[0]);
                return 0;
        }

        if (ntoks-1 < command->cmd_args) {
                disp_error("Command requires %d arguments",
                        command->cmd_args);
                return 0;
        }

        if (command->cmd_next != NULL)
                if (ntoks > 1 )
                        return
                          cmd_process(ntoks-1,&tokens[1],command->cmd_next);
                else
                        return
                          cmd_menu(command->cmd_next,command->cmd_name);
        else
                return (*command->cmd_func)
                        (command,command->cmd_parg,ntoks-1,&tokens[1]);
}

static int
cmd_menu(cmd_table,command)
COMMAND cmd_table[];
char    *command;
{
        BROWSER browser;
        int             i = 0,result;
        COMMAND         *cmd;
        char            lasttok;
        char            buffer[BUFSIZ];
        char            *title,*prompt,*tokens[2];

        (void) sprintf(buffer,"%s Command Subcommands",command);
        title = mmwb_cpystring(buffer);

        (void) sprintf(buffer,"Select the desired %s subcommand", command);
        prompt = mmwb_cpystring(buffer);
```

```
	tokens[0] = command;
	tokens[1] = NULL;

	browser = br_create(Display,BR_FORM,title,prompt);

	/*
	 * Now, put together the browser list
	 */

	while (cmd_table[i].cmd_name != NULL) {
		if (br_pair(browser,mmwb_cpystring(cmd_table[i].cmd_name),
			BR_LABEL,mmwb_cpystring(cmd_table[i].cmd_desc),
			0,BR_LJUST,strlen(cmd_table[i].cmd_desc),(char *)NU
			(void *)&cmd_table[i],FVNULL,FNULL)) {
				disp_error("br_pair failed!");
				return	-1;
			}
		i++;
	}

	/*
	 * OK, now format it
	 */
	if (br_format(browser)) {
		disp_error("Unable to format menu browser");
		exit(1);
	}

	if (br_display(browser,-1,0)) {
		disp_error("Unable to display menu browser");
		exit(1);
	}

	/*
	 * Now let the user select a command for further information
	 */
	if ((cmd = (COMMAND *)br_activate(browser,&lasttok)) != NULL) {
		if (cmd->cmd_next != NULL)
			result = cmd_menu(cmd->cmd_next, cmd->cmd_name);
		else
			result = (*cmd->cmd_func)
				(cmd,cmd->cmd_parg,1,&tokens[1]);
	}
	br_free(browser);
	if (lasttok == BR_ESCAPE)
		return	1;
	return result;
}

static COMMAND *
cmd_find(token,table)
char *token;
COMMAND table[];
{
	int	i = 0;

	while (table[i].cmd_name != NULL) {
		if (strcasecmp(token,table[i].cmd_name) == 0)
			return	&table[i];
		i++;
	}

	return	NULL;
}

/*
 * user_input:
 *	user input routine
 */
char *
user_input(callback,window,stream,token)
char	*(*callback)();
void	*window;
FILE	*stream;
char	*token;
{
	extern	int errno;
	int	max;
	fd_set	mask;
	char	*result,c;

	/*
	 * Set up the mask
	 */
	for (;;) {
		FD_ZERO(&mask);
		FD_SET(fileno(stream),&mask);
		if (sm_file != NULL) {
			FD_SET(fileno(sm_file),&mask);
			if (fileno(stream) > fileno(sm_file))
				max = fileno(stream)+1;
			else
```

```
                        max = fileno(sm_file)+1;
                } else
                        max = fileno(stream)+1;

                if (select(max,&mask,(fd_set *)NULL,(fd_set *)NULL,
                        (struct timeval *)NULL) < 0) {
                        if (errno == EINTR)
                                continue;
                        perror("select");
                        (void) exit(1);
                }

                /*
                 * Check for input from the session manager
                 */
                if ((sm_file != NULL) && FD_ISSET(fileno(sm_file),&mask)) {
                        if (sm_dispatch() < 0) {
                                disp_error("sm_dispatch failed: %s",
                                        sm_error());
                                (void) exit(1);
                        }
                        if (feof(sm_file))
                                return  NULL;
                }

                /*
                 * Check for input from the user
                 */
                if (FD_ISSET(fileno(stream),&mask)) {
                        if (stream == stdin)
                                c = getchar();
                        else
                                c = fgetc(stream);

                        if ((result = callback(window,c,token)) != NULL)
                                return  result;
                }
        }
}
```

twb/object.c

```
#ifndef lint
static char RCSid[] = "$Id: object.c,v 2.2 90/04/16 18:27:05 scooter Exp $";
#endif lint

/*
 * object.c:
 *          object support routines
 *
 */

#include          "twb.h"
#include          "browse.h"
#include          "disp.h"

static    void      obj_info(),obj_help();
void      obj_ofs_copy(),obj_ofs_rename(),obj_ofs_delete();
static    char      *obj_ofs_getprop();
static    OFS_OBJ_TYPE   obj_gettype();
static    OFS_OBJ_CLASS  obj_getclass();

/*
 * obj_browse:
 *          object browser
 */

OFS_OBJ_DESC *
obj_browse(title,prompt,objlist,select)
char      *title,*prompt;
OFS_OBJ_DESC   objlist[];
int       select;
{
          BROWSER   browser;
          OFS_OBJ_DESC   *result;
          int       i = 0;
          char      lasttok;

          /*
           * Now, put together the browser list
           */
          browser = br_create(Display,BR_COL,title,prompt);

          while (objlist[i].name != NULL) {
                    if (br_button(browser,mmwb_cpystring(objlist[i].name),
                              -1,-1,-1,0,(void *)&objlist[i],FVNULL,FNULL))
                              disp_error("br_button failed!");
                    i++;
          }

          /*
           * Initialize the browser
           */
          if (br_display(browser,-1,0))
                    disp_error("unable to display browser");

          /*
           * Now let the user select the object
           */
          if (select)
                    result = (OFS_OBJ_DESC *)
                              br_activate(browser,&lasttok);
          else
                    result = NULL;

          br_free(browser);
          return    result;
}

/*
 * obj_ofs_browse:
 *          The interactive project browser.  This routine allows the user to
 *          interactively browse up and down the project tree, including project
 *          selection, and object information
 */
int
obj_ofs_browse(start,prev_ident,prompt,command)
OFS_OBJECT start;
int       prev_ident;
char      *prompt;
int       command;
{
   extern         char      *proj_name();
   OFS_OBJECT     curproj,parent;
   int            nobj;
   char           buffer[BUFSIZ],lasttok;
   OFS_OBJECT     proj_open_parent(),object;
   OFS_OBJ_DESC *desc,*rdesc;
   BROWSER        browser;
   int            i,istart;

   /*
    * Begin by finding the current project.  This will be the starting
    * point of our browsing.
    */
   if (start != NULL)
```

```
        curproj = start;
else
        curproj = sm_get_curr_proj();

for(;;) {
        /*
         * Get all objects in this project
         */
        nobj = obj_count(curproj, (char *)NULL, -1, -1);
        desc = (OFS_OBJ_DESC *)emalloc(sizeof(OFS_OBJ_DESC)*(nobj+2));
        nobj = obj_list(curproj, (char *)NULL, -1, -1, &desc[1], nobj+1);
        desc[nobj+1].name = NULL;
        parent = proj_open_parent(curproj);
        if (parent != NULL) {
                istart = 0;
                desc[0].name = proj_name(parent);
                desc[0].ident = obj_ident(parent);
                desc[0].class = obj_class(parent);
                desc[0].type = obj_type(parent);
                (void) obj_close(parent);
                nobj++;
        } else
                istart = 1;

        /*
         * OK, now we have the list, create a browser, COLUMN type
         */
        (void) sprintf(buffer, "Current project: %s",proj_name(curproj));
        browser = br_create(Display,BR_COL,mmwb_cpystring(buffer),prompt);

        for (i = istart; i < nobj; i++)
                (void) br_button(browser,mmwb_cpystring(desc[i].name),
                        -1,-1,-1,BR_LJUST,(void *)&desc[i],FVNULL,FNULL);

        (void) br_display(browser,1,0);

        /*
         * OK, now let the user browse.  The commands are as follows:
         *
         *      SELECT  - select this object
         *      '='     - display the type of this object
         *      '?'     - display information about this object
         *      '+'     - make this the current project
         *      'd'     - delete this object
         *      'n'     - (re)name this object
         *      'c'     - copy this object
         *      ESCAPE - bail all the way out
         *      RETURN - if the object is a project, move into that project
         */
        for(;;) {
        (void) br_activate (browser,&lasttok);
        switch (lasttok) {
        case    BR_RETURN:
                        br_free(browser);
                        free((char *)desc);
                return 0;
        case    BR_ESCAPE:
                        br_free(browser);
                        free((char *)desc);
                return 1;
        case    BR_SELECT:
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                        if (rdesc->class == OBJ_CL_PROJECT) {
                                if (rdesc->ident == prev_ident)
                                        return 0;

                                if (rdesc->ident != -1)
                                        object = proj_open(
                                          obj_open_by_ident(curproj,rdesc->ident));
                                else
                                        object = proj_open_parent(curproj);

                                if (obj_ofs_browse(object,obj_ident(curproj),
                                                prompt, command)) {
                                        br_free(browser);
                                        free((char *)desc);
                                        (void) obj_close(object);
                                        return 1;
                                } else {
                                        (void) obj_close(object);
                                        (void) br_display(browser,1,0);
                                }
                        }
                continue;
        case    '?':
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                obj_help(curproj,rdesc);
                        break;
        case    '=':
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                obj_info(rdesc);
                continue;
                    case        '+':
                        if (command != '+' && command != '\0')
```

```
                        continue;
                sm_set_curr_proj(curproj);
                disp_header();
                continue;
        case    'd':
                if (command != 'd' && command != '\0')
                        continue;
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                obj_ofs_delete(curproj,rdesc);
                break;
        case    'n':
                if (command != 'n' && command != '\0')
                        continue;
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                obj_ofs_rename(curproj,rdesc);
                break;
        case    'c':
                if (command != 'c' && command != '\0')
                        continue;
                rdesc = (OFS_OBJ_DESC *)br_private(browser);
                obj_ofs_copy(curproj,rdesc);
                break;
        default:
                disp_error(
"Browse commands: ? help, = info, + select, d delete, n name, c copy");
                continue;
        }
                br_free(browser);
                free((char *)desc);
                break;
        }
    }
}

/*
 * obj_ofs_delete
 */
void
obj_ofs_delete(project,desc)
OFS_OBJECT    project;
OFS_OBJ_DESC *desc;
{
        char    *response;
        char    prompt[BUFSIZ];
        OFS_OBJECT      obj;
        extern  void    *Error;

        (void) sprintf(prompt,"Are you sure you want to delete object %s? ",
                desc->name);

        if ( (response = br_prompt_input(Error,prompt,5,-1,-1)) == NULL) {
                disp_errclear();
                return;
        }

        disp_errclear();

        if (strncasecmp("yes",response,strlen(response)) == 0) {
                if ((obj = obj_open_by_ident(project,desc->ident)) == NULL) {
                        (void) sprintf(prompt,"Unable to open %s: %s",
                                desc->name,ofs_error());
                        disp_message(TRUE,prompt);
                        return;
                }
                if (desc->class == OBJ_CL_PROJECT) {
                        OFS_OBJECT      proj;

                        proj = proj_open(obj);
                        if (proj_delete(proj) >= 0) {
                                (void) sprintf(prompt,
                                        "Project %s has been deleted",
                                        desc->name);
                                disp_message(TRUE,prompt);
                        }
                        (void) proj_close(proj);
                } else if (obj_delete(obj) >= 0) {
                        (void) sprintf(prompt,"Object %s has been deleted",
                                desc->name);
                        disp_message(TRUE,prompt);
                }
                (void) obj_close(obj);
        }
        return;
}

/*
 * obj_help:
 *      paint a form describing the object:
 *
 *      Name:
 *      Class:
 *      Type:
 *      Owner:
 *      Last Accessed:
```

twb/object.c

```
 *      Last Updated:
 *      Properties:
 *
 */
static  void
obj_help(project,desc)
OFS_OBJECT   project;
OFS_OBJ_DESC *desc;
{
        BROWSER         browser;
        OFS_OBJ_TYPE    typedesc;
        OFS_OBJ_CLASS   classdesc;
        OFS_OBJECT      object;
        OFS_STAT        stat;
        char            buffer[BUFSIZ];
        int             nprop,i;
        char            *propbuf;
        char            **proplist;
        extern  char    *ctime();

        object = obj_open_by_ident(project,desc->ident);
        if (obj_stat(object,&stat) < 0) {
                disp_error("Unable to get status on %s",desc->name);
                (void) obj_close(object);
                return;
        }

        /*
         * Now, put together the browser list
         */
        (void) sprintf(buffer,"Detailed information for object %s",
                desc->name);
        browser = br_create(Display,BR_FORM,mmwb_cpystring(buffer),
                        (char *)NULL);

        (void) sprintf(buffer,"Name:\t%s",desc->name);
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        classdesc = obj_getclass(desc);

        (void) sprintf(buffer,"Class:\t%s",classdesc.name);
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        typedesc = obj_gettype(desc);

        (void) sprintf(buffer,"Type:\t%s",typedesc.name);
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        (void) sprintf(buffer,"Owner:\t%s",stat.owner);
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        (void) sprintf(buffer,"Last accessed:\t%s",
                                        ctime(&(stat.atime)));
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        (void) sprintf(buffer,"Last modified:\t%s",
                                        ctime(&(stat.mtime)));
        (void) br_label(browser,mmwb_cpystring(buffer),5,-1,-1,BR_LJUST,0);

        /*
         * O.K, now cycle through the properties
         */

        (void) br_label(browser,mmwb_cpystring(" "),5,-1,-1,BR_LJUST,0);
        (void) sprintf(buffer,"Property list");
        (void) br_label(browser,mmwb_cpystring(buffer),0,-1,-1,BR_LJUST,0);

        nprop = prop_count(object);
        propbuf = (char *)emalloc(nprop*OFS_PROP_NAMELEN);
        proplist = (char **)emalloc(nprop * sizeof(char *));
        for (i = 0; i < nprop; i++)
                proplist[i] = propbuf + (i * OFS_PROP_NAMELEN);

        nprop = prop_list(object,proplist,nprop);

        for (i = 0; i < nprop; i++) {
                (void) sprintf(buffer,"%s:",proplist[i]);
                if (br_pair(browser,mmwb_cpystring(buffer),BR_LABEL,
                        mmwb_cpystring(obj_ofs_getprop(object,proplist[i])),
                        0,BR_LJUST,strlen(obj_ofs_getprop(object,proplist[i])),
                        (char *)NULL,VNULL,FVNULL,FNULL)) {
                                disp_error("unable to construct reply!");
                                return;
                        }
        }

        /*
         * Initialize the browser
         */
        if (br_display(browser,1,0))
                disp_error("unable to display reply");

        /******* THIS NEEDS HELP!! *******/
        if (br_nscreen(browser) > 1) {
```

```
                char    *ans;
                int     screen = 1;
                for(;;) {
                        ans = br_prompt_input(Error,
                            "Press return for next screen, escape to end",
                            5,-1,-1);
                        if (ans == NULL)
                                break;
                        if (++screen > br_nscreen(browser) )
                                screen = 1;
                        (void) br_display(browser,screen,0);
                }
        } else {
                (void) br_prompt_input(Error,"Press return when ready",5,-1,-1);
        }
        disp_errclear();

        free((char *)proplist);
        free(propbuf);
        br_free(browser);
        (void) obj_close(object);
        return;
}

/*
 * obj_info:
 *      describe the object on the message line:
 *              "%s is a %s of type %s"
 */
static  void
obj_info(object)
OFS_OBJ_DESC *object;
{
        OFS_OBJ_TYPE    typelist;
        char            *typename = "unknown";
        char            *typedesc = NULL;
        char            buffer[180];

        typelist = obj_gettype(object);

        if (typelist.name != NULL) {
                typename = typelist.name;
                typedesc = typelist.description;
        }

        if (*typename == 'a' || *typename == 'i' || *typename == 'o' ||
                *typename == 'e' || *typename == 'u')
                (void) sprintf(buffer,"'%s' is an %s (%s)",
                                object->name,typename,typedesc);
        else
                (void) sprintf(buffer,"'%s' is a %s (%s)",
                                object->name,typename,typedesc);

        disp_message(TRUE, buffer);

        return;
}

/*
 * obj_gettype:
 *      Return the type description for an object
 */
static  OFS_OBJ_TYPE
obj_gettype(desc)
OFS_OBJ_DESC  *desc;
{
        int         type,ntype,i;
        OFS_OBJ_TYPE    *typelist;
        OFS_OBJ_TYPE    result;
        OFS_OBJECT      sysproj = sm_get_sys_proj();

        type = desc->type;

        ntype = ofs_type_count(sysproj);

        typelist = (OFS_OBJ_TYPE *)emalloc(sizeof(OFS_OBJ_TYPE)*(ntype+2));
        ntype = ofs_type_list(sysproj,typelist,ntype+1);

        result.name = NULL;
        for (i = 0; i < ntype; i++) {
                if (typelist[i].type == type) {
                        result = typelist[i];
                        break;
                }
        }
        free((char *)typelist);
        return  result;
}

/*
 * obj_getclass:
 *      Return the class description for an object
 */
static  OFS_OBJ_CLASS
```

twb/object.c

```
obj_getclass(desc)
OFS_OBJ_DESC  *desc;
{
        int         class,nclass,i;
        OFS_OBJ_CLASS *classlist;
        OFS_OBJ_CLASS  result;
        OFS_OBJECT     sysproj = sm_get_sys_proj();

        class = desc->class;

        nclass = ofs_class_count(sysproj);

        classlist = (OFS_OBJ_CLASS *)emalloc(sizeof(OFS_OBJ_CLASS)*(nclass
        nclass = ofs_class_list(sysproj,classlist,nclass+1);

        result.name = NULL;
        for (i = 0; i < nclass; i++) {
                if (classlist[i].class == class) {
                        result = classlist[i];
                        break;
                }
        }

        free((char *)classlist);
        return    result;
}

/*
 * obj_ofs_getprop:
 *      Return a character property value
 */

static char *
obj_ofs_getprop(object,propname)
OFS_OBJECT      object;
char            *propname;
{
        OFS_PROP        ph;
        int             intval;
        short           shortval;
        unsigned        char    byteval;
        float           floatval;
        double          dblval;
        char            charval;
        DM_TABLE        thval;
        DM_STYPE        stype;
        static  char    buffer[DM_MAX_STRING];

        ph = prop_open(object,propname);
        if (ph == NULL)
                return    NULL;

        stype = prop_stype(ph);
        switch (stype.base_type) {
                case DM_S_INT1:
                        if (stype.length > 1) {
                                (void) sprintf(buffer,
                                        "array of %d 1 byte integers",
                                        stype.length);
                        } else {
                                if (prop_read(ph,(void *)&byteval) < 0)
                                        return    NULL;
                                (void) sprintf(buffer,"%d",byteval);
                        }
                        break;
                case DM_S_INT2:
                        if (stype.length > 1) {
                                (void) sprintf(buffer,"array of %d shorts",
                                        stype.length);
                        } else {
                                if (prop_read(ph,(void *)&shortval) < 0)
                                        return    NULL;
                                (void) sprintf(buffer,"%d",shortval);
                        }
                        break;
                case DM_S_INT4:
                        if (stype.length > 1) {
                                (void) sprintf(buffer,"array of %d integers",
                                        stype.length);
                        } else {
                                if (prop_read(ph,(void *)&intval) < 0)
                                        return    NULL;
                                (void) sprintf(buffer,"%d",intval);
                        }
                        break;
#ifdef  someday
                case DM_S_INT8:
#endif
                case DM_S_FLOAT4:
                        if (stype.length > 1) {
                                (void) sprintf(buffer,"array of %d floats",
                                        stype.length);
                        } else {
```

```
                        if (prop_read(ph,(void *)&floatval) < 0)
                                return  NULL;
                        (void) sprintf(buffer,"%f",floatval);
                }
                break;
        case DM_S_FLOAT8:
                if (stype.length > 1) {
                        (void) sprintf(buffer,"array of %d doubles",
                                stype.length);
                } else {
                        if (prop_read(ph,(void *)&dblval) < 0)
                                return  NULL;
                        (void) sprintf(buffer,"%f",dblval);
                }
                break;
#ifdef  someday
        case DM_S_FLOAT16:
#endif
        case DM_S_CHAR:
                if (stype.length > 1) {
                        (void) sprintf(buffer,"array of %d characters",
                                stype.length);
                } else {
                        if (prop_read(ph,(void *)&charval) < 0)
                                return  NULL;
                        (void) sprintf(buffer,"%c",charval);
                }
                break;
        case DM_S_BYTE:
                if (stype.length > 1) {
                        (void) sprintf(buffer,"array of %d bytes",
                                stype.length);
                } else {
                        if (prop_read(ph,(void *)&charval) < 0)
                                return  NULL;
                        (void) sprintf(buffer,"0x%x",charval);
                }
                break;
        case DM_S_STRING:
                if (stype.length > 1) {
                        (void) sprintf(buffer,"array of %d strings",
                                stype.length);
                } else {
                        if (prop_read(ph,(void *)buffer) < 0)
                                return  NULL;
                }
                break;
        case DM_S_TABLE:
                if (stype.length > 1) {
                        (void) sprintf(buffer,"array of %d tables",
                                stype.length);
                } else {
                        if (prop_read(ph,(void *)&thval) < 0)
                                return  NULL;
                        intval = attr_count(thval,DM_A_EXIST);
                        (void) sprintf(buffer,
                                "table with %d attribute%s", intval,
                                        intval == 1?"":"s");
                        (void) tbl_close(thval);
                }
                break;
        }
        (void) prop_close(ph);
        return  &buffer[0];
}

/*
 * obj_ofs_rename:
 *      rename an object
 */

void
obj_ofs_rename(project,desc,newname)
OFS_OBJECT      project;
OFS_OBJ_DESC    *desc;
char            *newname;
{
        char    buffer[BUFSIZ];
        OFS_OBJECT      oh;

        if (newname == NULL) {
                (void) sprintf(buffer,"Enter the new name for %s: ",desc->name);
                if( (newname = br_prompt_input(Error,buffer,5,-1,-1)) == NULL)
                        return;
        }

        oh = obj_open_by_ident(project,desc->ident);
        if (obj_rename(oh,newname) >= 0) {
                (void) sprintf(buffer,"Object %s has been renamed to %s",
                        desc->name,newname);
                disp_message(TRUE,buffer);
        }
        (void) obj_close(oh);
        return;
```

```
}

/*
 * obj_ofs_copy:
 *          copy an object
 */

/* ARGSUSED */
void
obj_ofs_copy(project,desc,newname)
OFS_OBJECT      project;
OFS_OBJ_DESC    *desc;
char            *newname;
{
        disp_error("Object copy is not implemented yet");
        return;
}
```

twb/session.c

```
#ifndef lint
static char RCSid[] = "$Id: session.c,v 2.1 90/02/16 10:48:35 scooter Exp $";
#endif lint

/*
 * session.c:
 *      session manager interface for twb
 *
 */

#include        "twb.h"
#include        "browse.h"
#include        "disp.h"

static  SM_SESSION      *session_select();

/*
 * session_init - initialize the session
 */

int
session_init(host,session_name,new_session)
char    *host,*session_name;
int     new_session;
{
        register SM_SESSION     *sess;

        if (new_session) {
                if (log_flag)
                        syslog(LOG_INFO,"calling sm_attach_session");
                if (sm_attach_session((SM_SESSION *)NULL, host) == NULL) {
                        disp_error("Unable to create new session: %s",
                                                                        sm_er
                        exit(1);
                }
                if (log_flag)
                        syslog(LOG_INFO,"sm_attach_session returns");

                return FALSE;
        }

        if (session_name == NULL) {
                sess = session_select();
                if (sess == (SM_SESSION *)(-1))
                        return  FALSE;
        } else {
                sess = sm_list_sessions();
                while (sess != NULL) {
                        if (strcmp(sess->ses_name,session_name) == 0)
                                break;
                        sess = sess->ses_next;
                }
                if (sess == NULL) {
                        disp_error("Unable to connect to session: %s",
                                                        session_name);
                        return  FALSE;
                }
        }

        if (sm_attach_session(sess, host) == NULL) {
                disp_error("Unable to attach to %s: %s",
                                                session_name,sm_error());
                return  FALSE;
        }
        return TRUE;
}

/*
 * session_close:
 *      close the session
 */

void
session_close(session_name)
char    *session_name;
{
        if (sm_dettach_session(session_name))
                disp_error("Unable to dettach from session: %s", sm_error());
        return;
}

/*
 * session_disp_status:
 *      Display the status of the session
 */
void
session_disp_status()
{
        SM_SESSION_STATUS       *slist,*sp;
        BROWSER                 browser;
        char                    buffer[80];
        int                     y;

#define DISP_NAME       5
```

twb/session.c

```
#define DISP_ID         DISP_NAME+15
#define DISP_STATE      DISP_ID+5
#define DISP_TYPE       DISP_STATE+7
#define DISP_PROJECT    DISP_TYPE+10

        if ( (slist = sm_session_status()) == NULL) {
                disp_error("No session manager!");
                return;
        }

        browser = br_create(Display,BR_FORM,
                        mmwb_cpystring("Session Status"), (char *)NULL);

        y = 3;
        (void) br_label(browser,mmwb_cpystring("Name"),
                        DISP_NAME,y,0,BR_CENTERED,0);
        (void) br_label(browser,mmwb_cpystring("ID"),
                        DISP_ID,y,0,BR_CENTERED,0);
        (void) br_label(browser,mmwb_cpystring("State"),
                        DISP_STATE,y,0,BR_CENTERED,0);
        (void) br_label(browser,mmwb_cpystring("Type"),
                        DISP_TYPE+3,y,0,BR_CENTERED,0);
        (void) br_label(browser,mmwb_cpystring("Project"),
                        DISP_PROJECT+5,y,0,BR_CENTERED,0);

        y++;
        while (slist != NULL) {
                sp = slist->ss_next;
                (void) br_label(browser,mmwb_cpystring(slist->ss_name),
                        DISP_NAME,++y,0,BR_LJUST,0);
                (void) sprintf(buffer,"%3d",slist->ss_tool);
                (void) br_label(browser,mmwb_cpystring(buffer),DISP_ID,
                        y,0,BR_RJUST,0);
                (void) sprintf(buffer,"%5d",slist->ss_state);
                (void) br_label(browser,mmwb_cpystring(buffer),DISP_STATE,
                        y,0,BR_RJUST,0);

                switch(slist->ss_type) {
                case    0:
                        (void) strcpy(buffer,"UIM");
                        break;
                case    1:
                        (void) strcpy(buffer,"Protocol");
                        break;
                case    2:
                        (void) strcpy(buffer,"Instrument");
                        break;
                default:
                        (void) sprintf(buffer,"%d",slist->ss_type);
                }
                (void) br_label(browser,mmwb_cpystring(buffer),DISP_TYPE,
                        y,0,BR_LJUST,0);
                (void) br_label(browser,mmwb_cpystring(slist->ss_currp),
                        DISP_PROJECT,y,0,BR_LJUST,0);
                (void) free((char *) slist);
                slist = sp;
        }
        if (br_display(browser,-1,0) )
                disp_error("Unable to display browser!");

        br_free(browser);
        return;
}
/*
 * session_browse_status:
 *      return the tool number of a selected running tool
 */
int
session_browse_status(type)
int     type;
{
        SM_SESSION_STATUS       *slist,*s;
        BROWSER                 browser;
        char    buffer[180];
        char    *title,*prompt;
        char    lasttok;

        if ( (slist = sm_session_status()) == NULL) {
                disp_error("No session manager!");
                return -1;
        }

        s = slist;

        /*
         * Now, put together the browser list
         */
        if (type == OBJ_T_PROTOCOL) {
                (void) strcpy(buffer,"Active Protocols");
                title = mmwb_cpystring(buffer);
                (void) strcpy(buffer,"Select the desired protocol");
                prompt = mmwb_cpystring(buffer);
        } else {
                (void)strcpy(buffer,"Active Instruments");
```

```
                title = mmwb_cpystring(buffer);
                (void) strcpy(buffer,"Select the desired instrument");
                prompt = mmwb_cpystring(buffer);
        }

        browser = br_create(Display,BR_COL,title,prompt);

        while (s != NULL) {
                if ((type == OBJ_T_PROTOCOL && s->ss_type == 1) ||
                   (type == OBJ_T_INSTRUMENT && s->ss_type == 2))
                        (void) br_button(browser,mmwb_cpystring(s->ss_name)
                                -1,-1,-1,BR_LJUST,(void *)s,FVNULL,FNULL)
                s = s->ss_next;
        }

        /*
         * Initialize the browser
         */
        if (br_display(browser,-1,0)) {
                disp_error("Unable to display browser!");
                br_free(browser);
                return -1;
        }

        /*
         * Now let the user select the session
         */
        s = (SM_SESSION_STATUS *)br_activate(browser,&lasttok);
        br_free(browser);
        if (s == NULL)
                return    -1;
        return    s->ss_tool;
}

/*
 * uim_dialog:
 *        callback from the session manager
 */
uim_dialog(dp)
DIALOG  dp;
{
        char      buffer[BUFSIZ];
        char      *p;
        int       err,nfields,semtype;
        DIALOG    d;
        extern    void      di_add(),disp_dialog();

        semtype = dialog_get_semantics(dp,NULL,&err);

        if (semtype != 0) {
                if (semtype == 2) {
                        /* Its an echo dialog */
                        sm_send_reply(dp);
                        dialog_free(dp);
                        return    0;
                }
        }

        di_add(dp);                    /* Add this to the list of pending dialogs */

        if (set_get_bool("dialog prompt",&err))
                p = dialog_get_text(dp,DI_V_PROMPT,&err);
        else
                p = dialog_name(dp);

        for (nfields = 0, d = dp; d != NULL; d = dialog_next(d)) nfields++;

        (void) sprintf(buffer,"Dialog %d from %s (%d fields): %s",dialog_id(dp),
                dialog_get_text(dp,DI_V_TOOL,&err), nfields, p);

        disp_dialog(buffer);
        return 0;
}

/*
 * session_select:
 *        Select a session from a possible list
 */

static SM_SESSION *
session_select()
{
        SM_SESSION      *sess,*s;
        BROWSER         browser;
        char            lasttok;

        /*
         * Get the session list
         */
        sess = sm_list_sessions();
        if (sess == NULL)
                return    NULL;

        /*
```

```
        * Now, put together the browser list
        */
       browser = br_create(Display,BR_FORM,
                       mmwb_cpystring("Current Active Sessions"),(char *)NU

       (void)br_button(browser,mmwb_cpystring("New session"),12,-1,-1,BR_LJUS
                       (void *)NULL,FVNULL,FNULL);

       for (s = sess; s != NULL; s = s->ses_next) {
               (void) br_button(browser,mmwb_cpystring(s->ses_name),12,-1,-1,
                       BR_LJUST,(void *)s,FVNULL,FNULL);
       }

       /*
        * Initialize the browser
        */
       (void) br_display(browser,-1,0);

       /*
        * Now let the user select the session
        */
       s = (SM_SESSION *)br_activate(browser,&lasttok);
       br_free(browser);
       return s;
}

/*
 * tool_browse:
 */
char *
tool_browse(title,prompt,tool_list,select)
char *title,*prompt;
SM_TOOLDES *tool_list;
int     select;
{
       BROWSER         browser;
       SM_TOOLDES      *td;
       char            lasttok;
       char            *result;

       /*
        * Now, put together the browser list
        */
       browser = br_create(Display,BR_COL,mmwb_cpystring(title),
                       mmwb_cpystring(prompt));

       for (td = tool_list; td != NULL; td = td->next) {
               (void) br_button(browser,mmwb_cpystring(td->Name),-1,
                       -1,-1,BR_LJUST,(void *)td->Name,FVNULL,FNULL);
       }

       /*
        * Initialize the browser
        */
       (void) br_display(browser,-1,0);

       /*
        * Now let the user select the session
        */
       if (select)
               result = (char *)br_activate(browser,&lasttok);
       else
               result = NULL;

       br_free(browser);
       return  result;
}
```

twb/set.c

```
#ifndef lint
static char RCSid[] = "$Id: set.c,v 2.0 90/01/02 22:38:32 scooter Exp $";
#endif lint

/*
 * set.c:
 *          interface to set-able defaults
 *
 */

#include          "twb.h"
#include          <stdio.h>

/*
 * Types
 */
#define  SET_INT          1
#define  SET_FLOAT        2
#define  SET_TEXT         3
#define  SET_BOOLEAN      4

/*
 * The set structure
 */
typedef struct set_st {
          char      *set_name;          /* Value name (e.g. dialog_rows) */
          char      *set_abbrev;        /* Abbreviation (e.g. dr) */
          int        set_type;          /* Type (see above) */
          char      *set_default;       /* The default value */
          union     set_union {
                    int         set_int;
                    float       set_float;
                    char       *set_text;
          }         set_value;          /* The actual value */
} SET;

/*
 * The set table
 */
static SET          set_table[] =
{
          {"dialog lines","dl",SET_INT,"2"},
          {"command lines","cl",SET_INT,"2"},
          {"header lines","hl",SET_INT,"1"},
          {"dialog prompt","dp",SET_BOOLEAN,"FALSE"},
          {NULL}
};


/*
 * local functions
 */
static    SET       *set_find();

/*
 * set_init - initialize the defaults
 */
void
set_init()
{
          int       i = 0;

          while (set_table[i].set_name != NULL) {
                    switch(set_table[i].set_type) {
                    case      SET_INT:
                              set_table[i].set_value.set_int =
                                        atoi(set_table[i].set_default);
                              break;

                    case      SET_BOOLEAN:
                              if (strcasecmp(set_table[i].set_default,"TRUE") == 0)
                                        set_table[i].set_value.set_int = TRUE;
                              else
                                        set_table[i].set_value.set_int = FALSE;
                              break;

                    case      SET_FLOAT:
                              (void)sscanf(set_table[i].set_default,"%f",
                                        &set_table[i].set_value.set_float);
                              break;

                    case      SET_TEXT:
                              set_table[i].set_value.set_text =
                                        mmwb_cpystring(set_table[i].set_default);
                              break;
                    }
                    i++;
          }
          return;
}

/*
 * set_get_int:
```

twb/set.c

```
 *          Get the value of an integer variable
 */
int
set_get_int(var,err)
char        *var;
int         *err;
{
            SET         *sp;

            *err = TRUE;

            if (var == NULL)
                        return      -1;

            if ( (sp = set_find(var)) == NULL)
                        return      -1;

            if (sp->set_type != SET_INT)
                        return      -1;

            *err = FALSE;
            return      sp->set_value.set_int;
}

/*
 * set_get_bool:
 *          Get the value of a boolean variable
 */
int
set_get_bool(var,err)
char        *var;
int         *err;
{
            SET         *sp;

            *err = TRUE;

            if (var == NULL)
                        return      -1;

            if ( (sp = set_find(var)) == NULL)
                        return      -1;

            if (sp->set_type != SET_BOOLEAN)
                        return      -1;

            *err = FALSE;
            return      sp->set_value.set_int;
}

static SET *
set_find(var)
char *var;
{
            int         i = 0;

            while (set_table[i].set_name != NULL) {
                        if (strcasecmp(set_table[i].set_name,var) == 0)
                                    return &set_table[i];
                        i++;
            }
            return      NULL;
}
```

twb/twb.c

```
#ifndef lint
static char RCSid[] = "$Id: twb.c,v 2.0 90/01/02 22:38:36 scooter Exp $";
#endif lint

/*
 * twb:
 *          a curses-based user interface manager for MMWB
 *
 */

#include          <stdio.h>
#include          <pwd.h>
#include          <syslog.h>
#include          "twb.h"
#include          "dm.h"
#include          "sm.h"

#define  USAGE    "[-n] [-h session-host] [session-name]"
#define  VTDEFAULTS       ".twbrc"
#define  VERSMAJOR        0
#define  VERSMINOR        1

/*
 * Globals
 */
int      verbose_flag = 0;
int      log_flag = 0;

main(argc,argv)
int      argc;
char     **argv;
{
         register int         c;
         int                  new_session;
         char                 *host = NULL;
         char                 *session_name = NULL;
         FILE                 *init;
         char                  progname[BUFSIZ];
         extern int           optind;
         extern char          *optarg;
         extern void          fs_init(),parse(),disp_close(),fs_close();
         extern void          disp_init(),set_init(),command_input();
         UIDTYPE getuid();

         /*
          * Process arguments
          */
         new_session = FALSE;

         (void) sprintf(progname,"%s V%d.%d",argv[0],VERSMAJOR,VERSMINOR);

         while ((c = getopt(argc, argv, "vnlh:")) != EOF)
                  switch (c) {
                   case 'v':
                            verbose_flag++;
                            break;

                   case 'l':
                            log_flag++;
#ifdef   LOG_LOCAL4
                            openlog("mmwb-twb", LOG_PID, LOG_LOCAL4);
#else
                            openlog("mmwb-twb", LOG_PID);
#endif
                            break;

                   case 'n':
                            new_session = TRUE;
                            break;

                   case 'h':
                            host = optarg;
                            break;
                  }

         if (optind+1 == argc)
                  session_name = argv[optind++];

         if (optind != argc) {
                  (void) fprintf(stderr, "Usage: %s %s\n", argv[0], USAGE);
                  (void) exit(1);
         }

         fs_init();                       /* Initialize the ofs */
         set_init();                      /* Initialize the variables to default */

         if ( (init = fopen(VTDEFAULTS,"r")) != NULL) {
                  if (verbose_flag)
                           (void) fprintf(stdout,"Reading init from %s\n",
                                                      VTDEFAULTS);
                  command_input(init,1);           /* Set any defaults */
                  (void) fclose(init);
         } else {
                  /*
```

```
                * Check for ~/.twbrc
                */
               char     buf[BUFSIZ];
               struct   passwd *pwd;

               if ( (pwd = getpwuid((int)getuid())) == NULL ) {
                        fprintf(stderr,
                 "Warning: can't get user information, ~/%s not read\n",
                 VTDEFAULTS);
               } else {
                        (void) sprintf(buf,"%s/%s",pwd->pw_dir,VTDEFAULTS);
                        if ( (init = fopen(buf,"r")) != NULL) {
                                 if (verbose_flag)
                                          (void) fprintf(stdout,
                                          "Reading defaults from %s\n",buf);
                                 command_input(init,1);          /* Set any defau
                                 (void) fclose(init);
                        }
               }
        }

        disp_init(progname);                    /* Initialize the display */

        /* Initialize the session */
        if (log_flag)
                syslog(LOG_INFO, "initializing session");
        if (session_init(host,session_name,new_session))
                command_input(stdin,0);                         /* Handle input *

        disp_close();
        fs_close();
}
```

twb/twbdialog.c

```
#ifndef lint
static char *RCSid = "$Id: twbdialog.c,v 2.2 90/04/16 18:21:12 scooter Exp $";
#endif lint

/*
 * twbdialog.c - twb dialog support
 */

#include        "twb.h"
#include        "browse.h"
#include        "disp.h"

typedef struct dqueue_st {
        DIALOG  dq_dialog;
        struct dqueue_st *dq_next;
} DQUEUE;

static  DQUEUE *dq_head = (DQUEUE *)NULL;
static  DQUEUE *dq_tail = (DQUEUE *)NULL;
static  char    *di_get_data();

/*
 * di_add:
 *      add a dialog to the list of pending dialogs
 */
void
di_add(dp)
DIALOG dp;
{
        DQUEUE *dq = (DQUEUE *)emalloc(sizeof(DQUEUE));

        dq->dq_dialog = dp;
        dq->dq_next = (DQUEUE *)NULL;

        if (dq_head == (DQUEUE *)NULL) {
                dq_head = dq_tail = dq;
        } else {
                dq_tail->dq_next = dq;
                dq_tail = dq;
        }
}

/*
 * di_list:
 *      list all pending dialogs -- provide for selection if
 *      requested
 */
int
di_list(select,title,prompt)
int     select;
char    *title;
char    *prompt;
{
        BROWSER browser;
        DQUEUE *dq = dq_head;
        DIALOG  d;
        char    buffer[180];
        int     err;
        char    lasttok;

        /*
         * Put together the browser list
         */
        browser = br_create(Display,BR_FORM,mmwb_cpystring(title),
                        mmwb_cpystring(prompt));

        while (dq != NULL) {
                char *button,*label;
                d = dq->dq_dialog;
                (void) sprintf(buffer,"%d",dialog_id(d));
                button = mmwb_cpystring(buffer);
                (void) sprintf(buffer,"from %s --> %s",
                        dialog_get_text(d,DI_V_TOOL,&err),dialog_name(d));
                label = mmwb_cpystring(buffer);
                (void) br_pair(browser, button, BR_LABEL, label, BR_LJUST,0,
                        strlen(label),(char *)NULL,(void *)dq,FVNULL,FNULL);
                dq = dq->dq_next;
        }

        /*
         * Initialize the browser
         */
        if (br_display(browser,-1,0)) {
                disp_error("Unable to display browser!");
                return -1;
        }

        if (select == FALSE)
                return  -1;

        /*
         * Now let the user select a dialog
         */
        dq = (DQUEUE *)br_activate(browser,&lasttok);
```

```
        br_free(browser);
        if (dq != (DQUEUE *)NULL)
                return  dialog_id(dq->dq_dialog);
        else
                return -1;
}

/*
 * di_answer:
 *      answer a dialog
 */
void
di_answer(dnum)
int     dnum;
{
        BROWSER browser;
        DQUEUE *dq = dq_head;
        DQUEUE *qentry;
        DIALOG  d,dt,selected_dialog;
        char    buffer[180];
        char    *title,lasttok;
        int     err;
        void    di_help(),di_bounds(),di_add_field();
        void    *answer;

        while (dq != NULL) {
                d = dq->dq_dialog;
                if (dialog_id(d) == dnum)
                        break;
                dq = dq->dq_next;
        }

        if (dq == NULL) {
                disp_error("No unanswered dialog number %d",dnum);
                return;
        }

        qentry = dq;

        /*
         * OK, d now contains the dialog we want to answer, begin
         * by creating the form.
         */
        (void) sprintf(buffer,"Dialog %d: %s from %s", dialog_id(d),
                        dialog_name(d), dialog_get_text(d,DI_V_TOOL,&err));
        title = mmwb_cpystring(buffer);
        browser = br_create(Display,BR_FORM,title,(char *)NULL);

        dt = d;
        while (d != NULL) {
                di_add_field(browser,d);
                d = dialog_next(d);
        }

        /*
         * Initialize the browser
         */
        if (br_display(browser,-1,0)) {
                disp_error("Unable to display browser!");
                return;
        }

        /*
         * Get the input until the user is done
         */

        for(;;) {
                answer = br_activate(browser,&lasttok);
                switch (lasttok) {
                case    BR_ESCAPE:
                        br_free(browser);
                        return;
                case    BR_RETURN:
                        selected_dialog = (DIALOG) br_private(browser);
                        if (answer != NULL) {
                                char *s;
                                if ((s = strchr(answer,'\n')) != NULL)
                                        *s = '\0';
                                di_put_data(selected_dialog,(char *)answer);
                                continue;
                        } else
                                break;
                case    '?':
                        selected_dialog = (DIALOG) br_private(browser);
                        di_help(selected_dialog);
                        continue;
                case    '=':
                        selected_dialog = (DIALOG) br_private(browser);
                        di_bounds(selected_dialog);
                        continue;
                case    BR_SELECT:
                        continue;
                }
                break;
```

```
	}

	/*
	 * OK, now send the dialog (unless its a note)
	 */
	if (dialog_type(dt) != DI_NOTE) {
		if (sm_send_reply(dt)) {
			disp_error("Dialog reply failed: %s",sm_error());
			br_free(browser);
			return;
		}
	}

	/*
	 * Its sent, free it
	 */
	dq = dq_head;
	if (dq == qentry)
		dq_head = qentry->dq_next;
	else
		while (dq != NULL) {
			if (dq->dq_next == qentry) {
				dq->dq_next = qentry->dq_next;
				break;
			}
			dq = dq->dq_next;
		}

	dialog_free(qentry->dq_dialog,0);
	free((char *)qentry);
	br_free(browser);

	return;
}

/*
 * Edit an option sheet
 */
void
di_edit_option(type,td)
int		type;
SM_TOOLDES	*td;
{
	BROWSER	browser;
	DIALOG	d,dt,selected_dialog;
	char	buffer[180];
	char	*title,lasttok,*ans;
	void	di_help(),di_bounds(),di_add_field();
	void	*answer;

	(void) sprintf(buffer,"Options sheet for %s, version %s",
			td->Name, td->Version);
	title = mmwb_cpystring(buffer);
	browser = br_create(Display,BR_FORM,title,(char *)NULL);

	(void) sprintf(buffer,"Subtools: %s",td->Subtools);
	br_label(browser,mmwb_cpystring(buffer),5,-1,-1,0,0);
	(void) sprintf(buffer,"Hosts: %s",td->Host);
	br_label(browser,mmwb_cpystring(buffer),5,-1,-1,0,0);
	switch(td->OnError) {
		case	SM_IGNORE:
			ans = "Ignore";
			break;
		case	SM_DIE:
			ans = "Die";
			break;
		case	SM_KILLB:
			ans = "Kill branch";
			break;
		default:
			ans = "Unknown";
			break;
	}
	(void) sprintf(buffer,"Action on error: %s",ans);
	br_label(browser,mmwb_cpystring(buffer),5,-1,-1,0,0);

	if (td->nodisplay)
		(void) sprintf(buffer,"Display option sheet: No");
	else
		(void) sprintf(buffer,"Display option sheet: Yes");
	br_label(browser,mmwb_cpystring(buffer),5,-1,-1,0,0);
	br_label(browser,mmwb_cpystring(" "),5,-1,-1,0,0);

	if (td->Options == NULL) {
		(void) br_display(browser,-1,0);
		disp_message(TRUE,"There are no options for %s",td->Name);
		sm_free_tooldes(td);
		br_free(browser);
		return;
	}

	d = td->Options;
	while (d != NULL) {
		di_add_field(browser,d);
```

```
                d = dialog_next(d);
        }

        /*
         * Initialize the browser
         */
        if (br_display(browser,-1,0)) {
                disp_error("Unable to display browser!");
                return;
        }

        /*
         * Get the input until the user is done
         */
        for(;;) {
                answer = br_activate(browser,&lasttok);
                switch (lasttok) {
                case    BR_ESCAPE:
                        br_free(browser);
                        sm_free_tooldes(td);
                        return;
                case    BR_RETURN:
                        selected_dialog = (DIALOG) br_private(browser);
                        if (answer != NULL) {
                                char *s;
                                if ((s = strchr(answer,'\n')) != NULL)
                                        *s = '\0';
                                di_put_data(selected_dialog,(char *)answer);
                                continue;
                        } else
                                break;
                case    '?':
                        selected_dialog = (DIALOG) br_private(browser);
                        di_help(selected_dialog);
                        continue;
                case    '=':
                        selected_dialog = (DIALOG) br_private(browser);
                        di_bounds(selected_dialog);
                        continue;
                case    BR_SELECT:
                        continue;
                }
                break;
        }

        ans = br_prompt_input(Error,
                "Do you wish to update this option sheet?",5,-1,-1);
        if (ans != NULL && strncasecmp("yes",ans,strlen(ans)) == 0) {
                int     err;
                if (type == OBJ_T_PROTOCOL)
                        err = sm_write_protocol(sm_get_curr_proj(),td);
                else
                        err = sm_write_instrument(sm_get_curr_proj(),td);
                if (err)
                        disp_error(
                                "unable to update %s in current project: %s",
                                td->Name, sm_error());
        }
        sm_free_tooldes(td);
        br_free(browser);

        return;
}


/*
 * This routine builds all of the fields and browsers necessary
 * for filling out a dialog form
 */

static void
di_add_field(browser,dialog)
BROWSER browser;
DIALOG  dialog;
{
        char    *data,*sellist;
        int     length,err;

        data = di_get_data(dialog,&length);

        switch(dialog_type(dialog)) {
        case    DI_IVAL:
        case    DI_FVAL:
        case    DI_TEXT:
                (void) br_pair(browser,mmwb_cpystring(dialog_prompt(dialog)),
                        BR_TYPEIN,mmwb_cpystring(data),BR_INVERSE,
                        BR_LJUST,length,(char *)NULL,(void *)dialog,
                        FVNULL,FNULL);
                break;

        case    DI_SELECT:
                (void) br_pair(browser,mmwb_cpystring(dialog_prompt(dialog)),
                        BR_SELLIST,mmwb_cpystring(data),BR_INVERSE,
                        BR_LJUST,length,
```

```
                        dialog_get_text(dialog,DI_V_SELLIST,&err),
                        (void *)dialog,FVNULL,FNULL);
                break;
        case    DI_BOOL:
                sellist = "True False";
                (void) br_pair(browser,mmwb_cpystring(dialog_prompt(dialog)),
                        BR_TOGGLE,mmwb_cpystring(data),BR_INVERSE,
                        BR_LJUST,length,sellist,
                        (void *)dialog,FVNULL,FNULL);
                break;
        case    DI_YESNO:
                sellist = "Yes No";
                (void) br_pair(browser,mmwb_cpystring(dialog_prompt(dialog)),
                        BR_TOGGLE,mmwb_cpystring(data),BR_INVERSE,
                        BR_LJUST,length,sellist,
                        (void *)dialog,FVNULL,FNULL);
                break;
        case    DI_NOTE:
                (void) br_label(browser,mmwb_cpystring(dialog_prompt(dialog)),
                        -1,-1,-1,BR_LJUST,0);
                break;
        }
        return;

}


/*
 * The following routines deal with the "results" field of
 * dialogs
 */
static char *
di_get_data(d,length)
DIALOG  d;
int     *length;
{
        /*
         * Convert the results (if any) to text and set an appropriate
         * field length
         */
        static char buffer[180];
        int     intval,err;
        double  floatval;
        char    *textval,*s,*strchr();
        char    *sel_list[80],*selected[80];

        if (d == NULL)
                return  NULL;

switch (dialog_type(d)) {
case    DI_IVAL:
        *length = 11;
        intval = dialog_get_int(d,DI_V_RESULT,&err);
        (void) sprintf(buffer,"%-10d",intval);
        break;
case    DI_FVAL:
        *length = 11;
        floatval = dialog_get_float(d,DI_V_RESULT,&err);
        (void) sprintf(buffer,"%-10f",floatval);
        break;
case    DI_TEXT:
        *length = 41;
        textval = dialog_get_text(d,DI_V_RESULT,&err);
        (void) sprintf(buffer,"%-40s",textval);
        break;
case    DI_SELECT:
        /*
         * Since multi-selections are possible, just display
         * the first selection in a list.  If there are more,
         * indicate it with an asterisk in column 40
         */
        *length = 41;
        textval = mmwb_cpystring(dialog_get_text(d,DI_V_RESULT,&err));
        s = mmwb_cpystring(dialog_get_text(d,DI_V_SELLIST,&err));
        intval = tokenize(textval,selected,80);
        (void) tokenize(s,sel_list,80);
        (void) sprintf(buffer,"%-40s",mmwb_cpystring(sel_list[atoi(selected[0])]));
        if (intval > 1)
                buffer[40] = '*';
        mmwb_stringfree(s);
        mmwb_stringfree(textval);
        break;
case    DI_BOOL:
        *length = 7;
        intval = dialog_get_bool(d,DI_V_RESULT,&err);
        if (intval)
                (void) sprintf(buffer,"%-6s","True");
        else
                (void) sprintf(buffer,"%-6s","False");
        break;
case    DI_YESNO:
        *length = 7;
        intval = dialog_get_bool(d,DI_V_RESULT,&err);
        if (intval)
```

```
                        (void) sprintf(buffer,"%-6s","Yes");
                else
                        (void) sprintf(buffer,"%-6s","No");
                break;
        default:
                *length = 0;
                return  NULL;
        }
        if (err != 0) {
                disp_error("Dialog conversion error -- %s",sm_error());
                return  NULL;
        }
        return  mmwb_cpystring(buffer);
}

static int
di_put_data(d,data)
DIALOG  d;
char    *data;
{
        int     intval;
        double  floatval;

        if (data == NULL || d == NULL)
                return  1;

        switch (dialog_type(d)) {
        case    DI_IVAL:
                if( sscanf(data,"%d",&intval) != 1)
                        return  1;
                return  dialog_set_int(d,DI_V_RESULT,intval);

        case    DI_FVAL:
                if( sscanf(data,"%lf",&floatval) != 1)
                        return  1;
                return dialog_set_float(d,DI_V_RESULT,floatval);

        case    DI_TEXT:
        case    DI_SELECT:
                return  dialog_set_text(d,DI_V_RESULT,data);

        case    DI_BOOL:
                if(strncasecmp(data,"true",4) == 0)
                        intval = 1;
                else if(strncasecmp(data,"false",5) == 0)
                        intval = 0;
                else
                        return 1;
                return dialog_set_bool(d,DI_V_RESULT,intval);

        case    DI_YESNO:
                if(strncasecmp(data,"yes",3) == 0)
                        intval = 1;
                else if(strncasecmp(data,"no",2) == 0)
                        intval = 0;
                else
                        return 1;
                return dialog_set_bool(d,DI_V_RESULT,intval);

        default:
                return  1;
        }
}

/*
 * di_help:
 *      display the dialog help message
 */
static void
di_help(d)
DIALOG  d;
{
        char    *help;
        int     err;

        if (d == NULL)
                return;

        help = dialog_get_text(d,DI_V_HELP,&err);
        if (err)
                disp_error("Dialog conversion error - %s", sm_error());
        else
                if (help && strlen(help) > 0)
                        disp_message(TRUE, help);
                else
                        disp_message(TRUE, "No help available");
        return;
}

/*
 * di_bounds:
 *      display the dialog bounds message
 */
static void
```

twb/twbdialog.c

```
di_bounds(d)
DIALOG  d;
{
        int     err = 0,upperint,lowerint;
        double  upperfloat,lowerfloat;
        char    buffer[180];

        if (d == NULL)
                return;

        switch( dialog_type(d) ) {
        case    DI_IVAL:
                upperint = dialog_get_int(d,DI_V_UBOUND,&err);
                lowerint = dialog_get_int(d,DI_V_LBOUND,&err);
                (void) sprintf(buffer,
                        "an integer value between %d and %d is expected",
                        lowerint,upperint);
                break;
        case    DI_FVAL:
                upperfloat = dialog_get_float(d,DI_V_UBOUND,&err);
                lowerfloat = dialog_get_float(d,DI_V_LBOUND,&err);
                (void) sprintf(buffer,
                        "a floating point value between %f and %f is expected",
                        lowerfloat,upperfloat);
                break;
        case    DI_TEXT:
                upperint = dialog_get_int(d,DI_V_UBOUND,&err);
                lowerint = dialog_get_int(d,DI_V_LBOUND,&err);
                (void) sprintf(buffer,
                   "a text string between %d and %d characters is expected",
                        lowerint,upperint);
                break;
        case    DI_SELECT:
                upperint = dialog_get_int(d,DI_V_UBOUND,&err);
                lowerint = dialog_get_int(d,DI_V_LBOUND,&err);
                (void) sprintf(buffer,
                  "Field is a selection list where up to %d items may be selected",
                        upperint);
                break;
        case    DI_BOOL:
                (void) sprintf(buffer,"a TRUE or FALSE is expected");
                break;
        case    DI_YESNO:
                (void) sprintf(buffer,"a YES or NO is expected");
                break;
        }

        if (err)
                return;

        disp_message(TRUE,buffer);
        return;
}
```

libbrowse/browse.h

```
/*
 * $Id: browse.h,v 1.3 90/04/16 19:19:36 conrad Exp $
 *
 * browse.h - browser interface defines
 *
 */

#ifndef  BROWSE_INCLUDE

#include <machdep.h>

        /* Types */
typedef  void *     BROWSER;
typedef  void * FIELD;                  /* This may not be needed */

        /* Defines */

/* Browser Types */
#define  BR_FORM                        /* Form browser */
#define  BR_LIST 2                      /* List browser */
#define  BR_COL 3                       /* Row/Column browser */

/* Field Types */
#define  BR_TYPEIN       1       /* Type-in field */
#define  BR_BUTTON       2       /* Button (selection) */
#define  BR_LABEL        3       /* Label field (inactive) */
#define  BR_PROMPT       4       /* A prompt field */
#define  BR_TITLE        5       /* A title field */
#define  BR_TOGGLE       6       /* A toggle */
#define  BR_SELLIST      7       /* A list of values */

/* Format Types */
#define  BR_CENTERED     1       /* A centered field */
#define  BR_RJUST        2       /* Right-justified field */
#define  BR_LJUST        3       /* Left-justified field */

/* Display information */
#define  BR_INVERSE      0x100   /* Inverse video */
#define  BR_BLINKING     0x200   /* Blinking */
#define  BR_UNDERLINED   0x400   /* Underlined */
#define  BR_BOLD         0x800   /* Bold */
#define  BR_SCROLLING    0x1000  /* Scrolling field */
#define  BR_NOINVERSE    0x2000  /* Inhibit inverse */

/* Flags for br_field */
#define  BR_STUFF_FIELD 1        /* Stuff this field into the
                                    previous "private" */
#define  BR_IS_FIELD     2       /* This is a field, not an other */

/* Flags for br_display */
#define  BR_NOCLEAR      1       /* Don't clear the window */
#define  BR_OVERLAY      2       /* Overlay this browser */

/* Button tokens */
#define  BR_UP           (char)255
#define  BR_DOWN         (char)254
#define  BR_RIGHT        (char)253
#define  BR_LEFT         (char)252
#define  BR_RETURN       (char)251
#define  BR_SELECT       (char)250
#define  BR_ESCAPE       (char)249
#define  BR_CLEAR        (char)248

/*
 * Some defines to save typing
 */
#define  VNULL           (void *)NULL
#define  FVNULL          (void *(*)())NULL
#define FNULL            (void (*)())NULL

        /* Functions */
extern   BROWSER br_create();           /* Create a browser */
extern   int     br_format();           /* Format the browser */
extern   int     br_title();            /* Add a title */
extern   int     br_prompt();           /* Add a prompt */
extern   int     br_button();           /* Add a button */
extern   int     br_toggle();           /* Add a toggle input */
extern   int     br_select_list();              /* Add a selection list */
extern   int     br_pair();             /* Add a prompt-input pair
                                           (or a selection-data pair) */
extern   int     br_typein();           /* Add a typein field */
extern   int     br_label();            /* Add a label */
extern   int     br_display();          /* Display a browser */
extern   void *  br_activate();         /* Activate the browser */
extern   void    br_free();             /* Free a browser */
extern   void    br_clear();            /* Erase a browser display */
extern   int     br_field();            /* Add a field (not recommended) */
extern   int     br_nscreen();          /* Return the number of screens */
extern   void *  br_private();          /* Return private data of current selection */
extern   char *  br_string();           /* Return string (label) data of
                                           current selection */
extern   void    br_clear_field();              /* Clear a field */
extern   void    br_reset();            /* Reset the browser */
extern   char *  br_prompt_input();
```

libbrowse/browse.h

```
/* Intrisic functions */
extern  int     br_browse();        /* Call another browser recursively */
extern  void *  br_get_typein();            /* Input textual data */
extern  void *  br_get_toggle();            /* Input a toggle */
extern  void *  br_get_sellist();           /* Input a sellist */

#define BROWSE_INCLUDE
#endif  BROWSE_INCLUDE
```

libbrowse/browse_int.h

```
/*
 * $Id: browse_int.h,v 1.1 90/01/02 22:29:59 scooter Exp $
 *
 * browse_int.h - browser internal interface
 *
 */

#include        "browse.h"
#include        "io.h"

        /* Browser flags */
#define BR_NOFORMAT   1               /* Not formatted yet */
#define BR_FORMAT     2               /* Formatting complete */

        /* Structures */
typedef struct    field_st {
        int       fi_type;            /* Field type */
        int       fi_format;          /* Format flags */
        int       fi_display;         /* Display flags */
        char      *fi_string;         /* Label string */
        char      *fi_sellist;        /* (possible) selection list */
        int       fi_length;          /* Length of field */
        int       fi_y;               /* start of field */
        int       fi_x;
        int       fi_screen;          /* Screen # */
        int       fi_flags;           /* flags */
        void      *fi_private;        /* Private data */
        void      *(*fi_func)();      /* User-input function */
        void      (*fi_free)();       /* Call back for free */
        struct    field_st *fi_next;
} FIELD_INT;

typedef struct    browse_st {
        int       br_type;            /* Browser type */
        int       br_flags;           /* Browser flags */
        int       br_ncols;           /* # of columns */
        int       br_nrows;           /* # of rows */
        int       br_curscr;          /* Current screen # */
        int       br_nscreen;         /* # of screens */
        int       br_nfields;         /* # of fields */
        int       br_nother;          /* # of others on the screen */
        void      *br_window;         /* Window of this browser */
        FIELD_INT         *br_flist;  /* Field list */
        FIELD_INT         **br_fields;/* Input fields */
        FIELD_INT         **br_others;/* non-input fields */
        FIELD_INT         **br_cursel;/* Current selection */
} BROWSER_INT;
```

libbrowse/cursesio.h

```
/*
 * $Id: cursesio.h,v 1.1 90/01/02 22:30:09 scooter Exp $
 *
 * cursesio.h - curses I/O interface for browsers
 *
 */

#include        "io.h"
#include        <curses.h>
```

libbrowse/io.h

```
/*
 * $Id: io.h,v 1.1 90/01/02 22:30:27 scooter Exp $
 *
 * io.h - I/O interface for browsers
 *
 */

#include        "browse.h"

        /* Externals */
extern  void    *io_curwindow;          /* The current window */

        /* Input-Output functions */
extern  void    io_init();              /* Initialization */
extern  void    io_end();               /* Termination */
extern  void    *io_create_window();    /* Create a window */
extern  void    *io_create_overlay();   /* Create an overlay window */
extern  void    io_destroy_window();    /* Delete a window */
extern  char    io_get_button();        /* Input a button press */
extern  char    *io_get_text();         /* Text input */
extern  int     io_put_text();          /* Output some text */
extern  int     io_win_clear();         /* Clear this window */
extern  int     io_scroll();            /* Scroll a window */
extern  int     io_set_flags();         /* Set display flags */
extern  int     io_clear_flags();       /* Clear display flags */
extern  int     io_get_dims();          /* Get the window dimensions */

        /* User-provided functions */
extern  char    *user_input();          /* User input function */
```

libbrowse/browse.c

```
#ifndef lint
static  char *RCSid = "$Id: browse.c,v 1.2 90/01/08 16:29:51 scooter Exp $";
#endif  lint

/*
 * browse.c:
 *      A (sort of) general browser interface
 */

#include        <stdio.h>
#include        "browse_int.h"

extern  char    *malloc(),*strcpy(),*strchr();

/*
 * br_create:
 *      create a BROWSER structure
 */
BROWSER
br_create(window,type,title,prompt)
void    *window;
int     type;
char    *title,*prompt;
{
        char    *br_cpystring();
        BROWSER_INT     *B = (BROWSER_INT *)malloc(sizeof(BROWSER_INT)
        B->br_type = type;
        B->br_window = window;
        B->br_flags = BR_NOFORMAT;
        B->br_flist = NULL;
        B->br_fields = NULL;
        B->br_others = NULL;
        B->br_cursel = NULL;
        B->br_nscreen = 0;
        B->br_curscr = 1;
        B->br_ncols = 0;
        B->br_nrows = 0;
        B->br_nother = 0;
        B->br_nfields = 0;

        if (title != NULL)
                br_title((BROWSER)B,0,br_cpystring(title));

        if (prompt != NULL)
                br_prompt((BROWSER)B,0,br_cpystring(prompt));

        return  (BROWSER)B;
}

/*
 * br_free:
 *      Free a browser.
 */
void
br_free(browser)
BROWSER browser;
{
        BROWSER_INT *B = (BROWSER_INT *)browser;
        FIELD_INT *F,*F2;

        if (B == NULL)
                return;

        F = B->br_flist;

        /*
         * Free all of the fields
         */
        while (F != NULL) {
                F2 = F->fi_next;
                if (F->fi_string != NULL)
                        (void) free((char *)F->fi_string);
                if (F->fi_sellist != NULL)
                        (void) free((char *)F->fi_sellist);

                if (F->fi_free != NULL)
                        (F->fi_free)((char *)F->fi_private);

                (void) free((char *)F);
                F = F2;
        }

        (void) free( (char *)B->br_fields );
        (void) free( (char *)B->br_others );
        (void) free( (char *)B );
        return;
}

/*
 * br_title:
 *      Add a title to a browser.  This routine puts the title on
 *      the second line, centered.
 */
int
```

libbrowse/browse.c

```
br_title(browser,screen,title)
BROWSER	*browser;
int	screen;
char	*title;
{
	int	maxx,maxy;

	(void) io_get_dims(((BROWSER_INT *)browser)->br_window,&maxx,&maxy

	return	br_field(browser,BR_TITLE,title,(char *)NULL,BR_CENTERED,0,
			maxx,-1,1,screen,VNULL,FVNULL,FNULL,0);
}

/*
 * br_prompt:
 *	Add a prompt to a browser.  This routine puts the prompt on
 *	the bottom line at x = 5.
 */
int
br_prompt(browser,screen,prompt)
BROWSER	*browser;
int	screen;
char	*prompt;
{
	int	maxx,maxy;

	(void) io_get_dims(((BROWSER_INT *)browser)->br_window,&maxx,&maxy

	return	br_field(browser,BR_PROMPT,prompt,(char *)NULL,BR_LJUST,
			maxx-6,5,maxy-1,screen,VNULL,FVNULL,FNULL,0);
}

/*
 * br_button:
 *	Add a button to a browser.
 */
int
br_button(browser,label,x,y,screen,format,data,function,ufree)
BROWSER	*browser;
char	*label;
int	x,y,screen,format;
void	*data;
void	*(*function)();
void	(*ufree)();
{
	int	length;
	if (label != NULL)
		length = strlen(label);
	else
		length = 1;

	if (format == 0)
		format = BR_LJUST;
	return	br_field (browser,BR_BUTTON,label,(char *)NULL,format,0,
			length,x,y,screen,data,function,ufree,0);
}

/*
 * br_label:
 *	Add a label to a browser.
 */
int
br_label(browser,label,x,y,screen,format,display)
BROWSER	*browser;
char	*label;
int	x,y,format,display,screen;
{
	int	length;
	if (label != NULL)
		length = strlen(label);
	else
		length = 1;

	if (format == 0)
		format = BR_LJUST;

	return	br_field (browser,BR_LABEL,label,(char *)NULL,format,display,
			length,x,y,screen,VNULL,FVNULL,FNULL,0);
}

/*
 * br_typein:
 *	Add a typein field to a browser (with a possible label)
 */
int
br_typein(browser,label,curval,x,y,screen,display,length)
BROWSER	*browser;
char	*label,*curval;
int	x,y,screen,display,length;
{
	if (label != NULL) {
		int	len = strlen(label);
		br_field (browser,BR_LABEL,label,(char *)NULL,0,display,
			len,x,y,screen,VNULL,FVNULL,FNULL,0);
```

```
                length -= len;
                x += len;
        }
        return  br_field (browser,BR_TYPEIN,curval,(char *)NULL,0,
                        display,length, x,y,screen,VNULL,FVNULL,
                        FNULL,BR_IS_FIELD);
}

/*
 * br_toggle:
 *      Add a toggle field to a browser (with a possible label)
 */
int
br_toggle(browser,label,curval,sellist,x,y,screen,display,length,
                data,function,ufree)
BROWSER browser;
char    *label,*curval,*sellist;
int     x,y,screen,display,length;
void    *data;
void    *(*function)();
void    (*ufree)();
{
        if (label != NULL) {
                int     len = strlen(label);
                br_field (browser,BR_LABEL,label,(char *)NULL,0,display,
                        len,x,y,screen,VNULL,FVNULL,FNULL,0);
                length -= len;
                x += len;
        }
        return  br_field (browser,BR_TOGGLE,curval,sellist,0,
                        display,length, x,y,screen,data,function,
                        ufree,BR_IS_FIELD);
}

/*
 * br_select_list:
 *      Add a selection list field to a browser (with a possible label)
 */
int
br_select_list(browser,label,curval,sellist,x,y,screen,display,length,
                data,function,ufree)
BROWSER browser;
char    *label,*curval,*sellist;
int     x,y,screen,display,length;
void    *data;
void    *(*function)();
void    (*ufree)();
{
        if (label != NULL) {
                int     len = strlen(label);
                br_field (browser,BR_LABEL,label,(char *)NULL,0,display,
                        len,x,y,screen,VNULL,FVNULL,FNULL,0);
                length -= len;
                x += len;
        }
        return  br_field (browser,BR_TOGGLE,curval,sellist,0,
                        display,length, x,y,screen,data,function,
                        ufree,BR_IS_FIELD);
}


/*
 * br_pair:
 *      Add a BUTTON-TYPEIN, BUTTON-LABEL, BUTTON-TOGGLE, or BUTTON-LIST pair.
 *      These differ from other types because the data elements
 *      are associated.
 */
int
br_pair(browser,button_label,t_type,t_label,t_display,t_format,t_length,
        t_sellist,data,function,ufree)
BROWSER browser;
int     t_type,t_display,t_format,t_length;
char    *button_label,*t_label,*t_sellist;
void    *data;
void    *(*function)();
void    (*ufree)();
{
        int     length;

        if (button_label == NULL)
                length = 1;
        else
                length = strlen(button_label);

        switch(t_type) {
        case    BR_TYPEIN:
                if ( br_field(browser,BR_BUTTON,button_label,(char *)NULL,
                        BR_LJUST,0,length,-1,-1,-1,
                        VNULL,br_get_typein,FNULL,0) )
                                return  -1;
                if ( br_field(browser,BR_TYPEIN,t_label,(char *)NULL,
                        t_format,t_display,t_length,-1,-2,-2,data,
                        function,ufree,BR_STUFF_FIELD) )
                                return  -1;
```

```
                break;

        case    BR_LABEL:
                if ( br_field(browser,BR_BUTTON,button_label,(char *)NULL,
                        BR_LJUST,0,length,-1,-1,-1,data,function,ufree,0) )
                                return  -1;
                if ( br_field(browser,BR_LABEL,t_label,(char *)NULL,t_format,
                        t_display, t_length,-1,-2,-2,VNULL,FVNULL,FNULL,0) )
                                return  -1;
                break;

        case    BR_SELLIST:
                if ( br_field(browser,BR_BUTTON,button_label,(char *)NULL,
                        BR_LJUST,0,length,-1,-1,-1,
                        VNULL,br_get_sellist,FNULL,0) )
                                return  -1;
                if ( br_field(browser,BR_SELLIST,t_label,(char *)t_sellist,
                        t_format,t_display,t_length,-1,-2,-2,
                        data,function,ufree,BR_STUFF_FIELD) )
                                return  -1;
                break;

        case    BR_TOGGLE:
                if ( br_field(browser,BR_BUTTON,button_label,(char *)NULL,
                        BR_LJUST,0,length,-1,-1,-1,
                        VNULL,br_get_toggle,FNULL,0) )
                                return  -1;
                if ( br_field(browser,BR_SELLIST,t_label,(char *)t_sellist,
                        t_format,t_display,t_length,-1,-2,-2,data,function,
                        ufree,BR_STUFF_FIELD) )
                                return  -1;
                break;
        }
        return  0;
}

/*
 * br_field:
 *      Add a field to a browser.  This is the most general interface,
 *      but not recommended because of the complexity.  br_title, br_prompt,
 *      br_pair, and br_select are much easier to use and eventually call
 *      this routine anyway.
 */

int
br_field(browser,type,string,sellist,format,display,
        length,x,y,screen,private,function,ufree,flag)
BROWSER browser;
int     type,format,length,display,y,x,screen,flag;
char    *string;
char    *sellist;
void    *private;
void    *(*function)();
void    (*ufree)();
{
        FIELD_INT       *F, *f;
        BROWSER_INT     *B = (BROWSER_INT *)browser;

        if (browser == NULL)
                return  -1;

        F = (FIELD_INT *)malloc(sizeof(FIELD_INT));
        F->fi_type = type;
        F->fi_string = string;
        F->fi_sellist = sellist;
        F->fi_format = format;
        F->fi_display = display;
        F->fi_length = length;
        F->fi_y = y;
        F->fi_x = x;
        F->fi_screen = screen;
        F->fi_private = private;
        F->fi_func = function;
        F->fi_free = ufree;
        F->fi_next = (FIELD_INT *)NULL;
        F->fi_flags = flag;

        /*
         * Add it into the browser list
         */
        f = B->br_flist;
        if (f == (FIELD_INT *)NULL)
                B->br_flist = F;
        else {
                while (f->fi_next != (FIELD_INT *)NULL)
                        f = f->fi_next;

                f->fi_next = F;
                if (flag == BR_STUFF_FIELD)
                        if (f->fi_private == NULL) {
                                f->fi_private = (void *)F;
                                f->fi_flags |= BR_STUFF_FIELD;
                        }
}
```

```
	return	0;
}

/*
 * br_nscreen:
 *	return the number of screens in this browser
 */
int
br_nscreen(browser)
BROWSER browser;
{
	BROWSER_INT	*B = (BROWSER_INT *)browser;

	if (browser == NULL)
		return	-1;

	return	B->br_nscreen;
}

/*
 * br_private:
 *	return the private data to the current selection.  NOTE:
 *	this may point indirectly to a stuffed field.
 */
void *
br_private(browser)
BROWSER browser;
{
	BROWSER_INT	*B = (BROWSER_INT *)browser;
	FIELD_INT	*target;

	if (browser == NULL)
		return	NULL;

	if (B->br_cursel == NULL)
		return	NULL;

	target = *B->br_cursel;
	if (target->fi_flags & BR_STUFF_FIELD)
		target = (FIELD_INT *)target->fi_private;

	return	target->fi_private;
}

/*
 * br_string:
 *	return the string (label) to the current selection
 */
char *
br_string(browser)
BROWSER browser;
{
	BROWSER_INT	*B = (BROWSER_INT *)browser;
	FIELD_INT	*target;

	if (browser == NULL)
		return	NULL;

	if (B->br_cursel == NULL)
		return	NULL;

	target = *B->br_cursel;
	if (target->fi_flags & BR_STUFF_FIELD)
		target = (FIELD_INT *)target->fi_private;

	return	target->fi_string;
};

/*
 * br_reset:
 *	reset a browser
 */
void
br_reset(browser)
BROWSER browser;
{
	BROWSER_INT	*B = (BROWSER_INT *)browser;

	if (browser != NULL)
		B->br_cursel = NULL;

	return;
}

/*
 * br_prompt_input:
 *	short-hand to create a field and return the input
 */
char *
br_prompt_input(window,prompt,x,y,length)
void	*window;
char	*prompt;
int	x,y,length;
{
```

libbrowse/browse.c

```
        int         maxx,maxy;
        char        *result,*s,token;
        BROWSER     browser;
        char        *br_cpystring();
        int         plen = 0;

        if (prompt != NULL)
                plen = strlen(prompt);

        io_get_dims(window,&maxx,&maxy);
        browser = br_create(window,BR_FORM,(char *)NULL,(char *)NULL);
        if (x == -1)
                x = 0;
        if (y == -1)
                y = maxy-1;
        if (length = -1)
                length = maxx-1;
        (void) br_label(browser,prompt,x,y,0,0,0);
        (void) br_typein(browser,(char *)NULL,(char *)NULL,x+plen+1,y,0,
                        BR_NOINVERSE,length-plen-1);
        br_display(browser,0,BR_NOCLEAR);
        result = (char *)br_activate(browser,&token);
        if (token == BR_ESCAPE)
                return  NULL;
        if (result == NULL)
                return  NULL;
        if ( (s = strchr(result,'\n')) != NULL )
                *s = '\0';
        return br_cpystring(result);
}


/*
 * br_cpystring:
 *      make a copy of a string.
 */
char *
br_cpystring(string)
char    *string;
{
        char    *newstring;

        if (string == NULL)
                return  NULL;

        newstring = (char *)malloc((unsigned)(strlen(string)+1));
        if (newstring == NULL)
                return  NULL;

        (void) strcpy(newstring,string);
        return  newstring;
}
```

libbrowse/cursesio.c

```
/*
 * $Id: cursesio.c,v 1.3 90/04/16 19:20:07 conrad Exp $
 *
 * cursesio.c - curses I/O interface for browsers
 *
 */

#include        "cursesio.h"
#include        <signal.h>

#undef CTRL
#define  CTRL(x)  ((x)&037)

/*
 * Package variables
 */
void        *io_curwindow = NULL;
static      FILE        *io_channel = NULL;
static      int         io_exit();
extern      char        *strcpy(),*malloc();

/*
 * io_init:
 *          Initialize the io system
 */
void
io_init(channel)
FILE        *channel;
{
#ifndef __TERMIO_H__
            int         tstp();
#else
            struct termio           tiobuf;
#endif

            io_channel = channel;
            initscr();
            cbreak();
            noecho();

#ifndef __TERMIO_H__
            signal(SIGTSTP,tstp);
#else
            if (ioctl(0, TCGETA, &tiobuf) >= 0) {
                        tiobuf.c_iflag |= ICRNL;
                        (void) ioctl(0, TCSETA, &tiobuf);
            }
#endif
            signal(SIGINT,io_exit);
            return;
};

/*
 * io_end:
 *          clean up.
 */
void
io_end()
{
            endwin();
}

/*
 * io_create_window:
 *          create a window
 */
void *
io_create_window (x_start,y_start,x_end,y_end)
int         x_start,y_start,x_end,y_end;
{
            WINDOW*win;

            win = newwin(y_end-y_start,x_end-x_start,y_start,x_start);
            return      (void *)win;
}

/*
 * io_create_overlay:
 *          create an overlay window
 */
void *
io_create_overlay (window,x_start,y_start,x_end,y_end)
void        *window;
int         x_start,y_start,x_end,y_end;
{
            WINDOW*W = (WINDOW *)window,*win;
            int         ys,xs,len,wid;

            ys = y_start+W->_begy;
            xs = x_start+W->_begx;
            len = y_end-y_start;
            wid = x_end-x_start;
            win = newwin(len,wid,ys,xs);
            overwrite(win,W);
```

```
	return	(void *)win;
}


/*
 * io_destroy_window:
 *	destroy a window:
 */
void
io_destroy_window(window)
void	*window;
{
	delwin((WINDOW *)window);
	return;
}

/*
 * io_get_dims:
 *	return window dimensions
 */
int
io_get_dims(window,maxx,maxy)
void	*window;
int	*maxx,*maxy;
{
	WINDOW*curwin = (WINDOW *)window;

	if (window == NULL) {
		if (curscr != NULL) {
			*maxx = curscr->_maxx;
			*maxy = curscr->_maxy;
			return	0;
		}

		*maxx = *maxy = 0;
		return -1;
	}

	*maxx = curwin->_maxx;
	*maxy = curwin->_maxy;
	return 0;
}

/*
 * io_put_text:
 *	Put text up on a window
 */
int
io_put_text(window,x,y,text)
void	*window;
int	x,y;
char	*text;
{
	int	len;
	char	*t;
	WINDOW*W = (WINDOW *)window;

	if (window == NULL)
		return	-1;

	if (text == NULL)
		return	0;

	if (x != -1 && y != -1)
		wmove(W,y,x);
	if (text == NULL)
		return	0;
	len = strlen(text);
	if (len+x <= W->_maxx)
		waddstr(W,text);
	else {
		t = (char *)malloc(strlen(text));
		strncpy(t,text,W->_maxx-x-1);
		waddstr(W,t);
		(void) free(t);
	}
	return	0;
}

/*
 * io_win_clear:
 *	clear a window
 */
int
io_win_clear(window)
void	*window;
{
	if (window == NULL)
		return	-1;

	wclear((WINDOW *)window);
	return	0;
}
```

libbrowse/cursesio.c

```
/*
 * io_scroll:
 *      scroll a window
 */
int
io_scroll(window)
void    *window;
{
        WINDOW*curwindow = (WINDOW *)window;
        int     x,y;

        if (window == NULL)
                return  -1;

        scrollok(curwindow,TRUE);
        wmove(curwindow,0,0);
        wdeleteln(curwindow);
        io_get_dims(window,&x,&y);
        wmove(curwindow,y-1,0);
        wclrtoeol(curwindow);
        wmove(curwindow,y-1,0);
        io_display(curwindow);
        return  0;
}

/*
 * io_display:
 *      display a window
 */
int
io_display(window)
void    *window;
{
        int     mask,oldmask;

        if (window == NULL)
                return  -1;

#ifdef SIGBLOCK
        mask = sigmask(SIGALRM);
        oldmask = sigblock(mask);
#else
        (void) sighold(SIGALRM);
#endif
        wrefresh((WINDOW *)window);
        io_curwindow = window;
#ifdef SIGBLOCK
        sigsetmask(oldmask);
#else
        (void) sigrelse(SIGALRM);
#endif
        return  0;
}

/*
 * io_set_flags:
 *      Set the various display flags
 */
int
io_set_flags(window,flag)
void    *window;
int     flag;
{
        if (window == NULL)
                return  -1;

        if (flag & BR_INVERSE)
                wstandout((WINDOW *)window);

        /* Nothing else is supported (yet) */
        return  0;
}

/*
 * io-clear_flags:
 *      Clear the various display flags
 */
int
io_clear_flags(window,flag)
void    *window;
int     flag;
{
        if (window == NULL)
                return  -1;

        if (flag & BR_INVERSE)
                wstandend((WINDOW *)window);

        /* Nothing else is supported (yet) */
        return  0;
}

/*
 * io_exit:
```

```
 *          just exit.
 */
static int
io_exit()
{
        io_end();
        exit(0);
        /* NOTREACHED */
}

/*
 * input routines: io_get_button & io_get_text
 */

/*
 * io_get_button:
 *          Return the button pressed by the user
 */
char
io_get_button(window)
void    *window;
{
        static  char    *io_parse_button();
        char    lasttoken;

        (void) user_input(io_parse_button,window,io_channel,&lasttoken);
        return  lasttoken;
}

/*
 * io_get_text:
 *          Return the text input by the user
 */
char *
io_get_text(window,x,y,length,lasttoken)
void    *window;
int     x,y,length;
char    *lasttoken;
{
        static  char    *io_parse_text();
        char    *text,*val;
        int     i;

        if (window != NULL) {
                wmove((WINDOW *)window,y,x);
                io_display(window);
        }

        for (;;) {
                text = user_input(io_parse_text,window,io_channel,lasttoken);
                switch (*lasttoken) {
                case    BR_CLEAR:
                        if (window != NULL) {
                                wmove((WINDOW *)window,y,x);
                                for (i = length;i > 0; i--)
                                        waddch((WINDOW *)window,' ');
                                wmove((WINDOW *)window,y,x);
                                io_display(window);
                        }
                        continue;
                case    BR_ESCAPE:
                        return  NULL;
                case    BR_RETURN:
                        val = (char *)malloc((unsigned)(strlen(text)+1));
                        if (val == NULL)
                                return  NULL;
                        (void) strcpy(val,text);
                        return  val;
                }
        }
        /* NOTREACHED */
}

/*
 * io_parse_button:
 */
/* ARGSUSED */
static
char *
io_parse_button(window,input,lasttoken)
void    *window;
char    input;
char    *lasttoken;
{
        *lasttoken = input;

        switch (input) {

        case    'h':
                *lasttoken = BR_LEFT;
                return  lasttoken;
        case    'j':
                *lasttoken = BR_DOWN;
                return  lasttoken;
```

```
		case	'k':
			*lasttoken = BR_UP;
			return	lasttoken;
		case	'l':
			*lasttoken = BR_RIGHT;
			return	lasttoken;
		case	'\033':
			*lasttoken = BR_ESCAPE;
			return	lasttoken;
		case	' ':
			*lasttoken = BR_SELECT;
			return	lasttoken;
		case	'\n':
			*lasttoken = BR_RETURN;
			return	lasttoken;
		case	CTRL('L'):
			wrefresh(curscr);
			return	NULL;
		default:
			return	lasttoken;
		}
}

/*
 * io_parse_text:
 */
static
char *
io_parse_text(window,input,lasttoken)
void	*window;
char	input;
char	*lasttoken;
{
		static	char	buffer[BUFSIZ];
		static	int	bufptr = 0;
		int	x,y;

		x = ( (WINDOW *)window)->_curx;
		y = ( (WINDOW *)window)->_cury;

		*lasttoken = input;

#ifdef __TERMIO_H__
		if (input == erasechar()) {
#else
		if (input == _tty.sg_erase) {
#endif
			buffer[--bufptr] = '\0';
			wmove ((WINDOW *)window,y,--x);
			waddch((WINDOW *)window,' ');
			wmove ((WINDOW *)window,y,x);
			io_display(window);
#ifdef __TERMIO_H__
		} else if (input == killchar()) {
#else
		} else if (input == _tty.sg_kill) {
#endif
			bufptr = 0;
			*lasttoken = BR_CLEAR;
			return	&buffer[0];
		} else if (input == '\n') {
			buffer[bufptr++] = input;
			buffer[bufptr] = '\0';
			bufptr = 0;
			*lasttoken = BR_RETURN;
			return	&buffer[0];
		} else if (input == '\033') {
			*lasttoken = BR_ESCAPE;
			bufptr = 0;
			return	&buffer[0];
		} else if (input == CTRL('L')) {
			wrefresh(curscr);
		} else {
			waddch((WINDOW *)window,input);
			io_display(window);
			buffer[bufptr++] = input;
			buffer[bufptr] = '\0';
		}
		return	NULL;
}
```

libbrowse/display.c

```
#ifndef lint
static  char *RCSid = "$Id: display.c,v 1.2 90/01/08 16:30:19 scooter Exp $";
#endif  lint

/*
 * display.c:
 *      display routines for libbrowse
 */

#include        <stdio.h>
#include        "browse_int.h"
static void     *br_get_button();

/*
 * br_display:
 *      Routine to display a browser
 */

int
br_display(browser,screen,flags)
BROWSER browser;
int     screen,flags;
{
        static  int     br_disp_field();
        int     err = 0;
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        FIELD_INT       *f;

        if (browser == NULL)
                return  -1;

        if ((B->br_flags & BR_FORMAT) == 0)
                if (br_format(browser) < 0)
                        return  -1;

        if (!(flags & BR_OVERLAY))
                br_clear(browser);

        if (screen > 0)
                B->br_curscr = screen;
        else if (B->br_curscr == 0)
                B->br_curscr = 1;

        for (f = B->br_flist;f != NULL; f = f->fi_next)
                if (f->fi_screen == 0 || f->fi_screen == B->br_curscr)
                        if (br_disp_field(B->br_window,f,0))
                                err = -1;
        io_display(B->br_window);
        return  err;
}

/*
 * br_clear:
 *      clear a browser's area
 */
void
br_clear(browser)
BROWSER browser;
{
        BROWSER_INT     *B = (BROWSER_INT *)browser;

        if (browser == NULL)
                return;

        io_win_clear(B->br_window);
        return;
}

/*
 * br_activate:
 *      activate the browser
 */
void    *
br_activate(browser,lastchar)
BROWSER browser;
char    *lastchar;
{
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        int             retflag;
        void            *result;

        *lastchar = -1;

        if (browser == NULL)
                return  NULL;

        if (B->br_nfields == 0)
                return  NULL;

        if (B->br_cursel == NULL)
                B->br_cursel = B->br_fields;

        for (;;) {
                (void) br_disp_field(B->br_window,*B->br_cursel,BR_INVERSE);
```

```
                io_display(B->br_window);

                switch ((*B->br_cursel)->fi_type) {

                case    BR_TYPEIN:
                        return  br_get_typein((BROWSER)B,lastchar);

                case    BR_BUTTON:
                        result = br_get_button(B,lastchar,&retflag);
                        if (retflag)
                                return  result;
                        break;

                case    BR_TOGGLE:
                        return br_get_toggle((BROWSER)B,lastchar);

                case    BR_SELLIST:
                        return br_get_sellist((BROWSER)B,lastchar);

                default:
                        continue;
                }

                if ((*B->br_cursel)->fi_screen != B->br_curscr) {
                        B->br_curscr = (*B->br_cursel)->fi_screen;
                        br_display((BROWSER)B,0,0);
                }
        }
        /* NOTREACHED */
}

/*
 * br_get_button:
 *      input a button and return the correct token
 */
static void *
br_get_button(B,lastchar,retflag)
BROWSER_INT     *B;
char    *lastchar;
int     *retflag;
{
        char    token;
        FIELD_INT       **last_field = B->br_fields+B->br_nfields-1;

        *retflag = -1;          /* Return */
        *lastchar = token = io_get_button(B->br_window);
        switch (token) {
case    BR_UP:
        (void) br_disp_field(B->br_window,*B->br_cursel,0);
        B->br_cursel -= B->br_ncols;
        if (B->br_cursel < B->br_fields) {
                while (B->br_cursel <= last_field)
                        B->br_cursel += B->br_ncols;
                B->br_cursel -= B->br_ncols;
        }
        break;
case    BR_DOWN:
        (void) br_disp_field(B->br_window,*B->br_cursel,0);
        B->br_cursel += B->br_ncols;
        if (B->br_cursel > last_field) {
                while (B->br_cursel >= B->br_fields)
                        B->br_cursel -= B->br_ncols;
                B->br_cursel += B->br_ncols;
        }
        break;
case    BR_LEFT:
        (void) br_disp_field(B->br_window,*B->br_cursel,0);
        if (B->br_cursel == B->br_fields)
                B->br_cursel = last_field;
        else
                B->br_cursel--;
        break;
case    BR_RIGHT:
        (void) br_disp_field(B->br_window,*B->br_cursel,0);
        if (B->br_cursel == last_field)
                B->br_cursel = B->br_fields;
        else
                B->br_cursel++;
        break;
case    BR_SELECT:
        if ( (*B->br_cursel)->fi_func != NULL )
                return
                  (*(*B->br_cursel)->fi_func)((BROWSER)B,lastchar);
        return  (*B->br_cursel)->fi_private;
case    BR_RETURN:
case    BR_ESCAPE:
default:
        return  NULL;
        }
        *retflag = 0;
        return  NULL;
}

/*
```

```
 * br_get_toggle:
 *          Input a toggle field
 */
void *
br_get_toggle(browser,lastchar)
BROWSER         browser;
char            *lastchar;
{
        FIELD_INT       *target;
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        char            *label,*sellist,*list[3];
        extern          char *br_cpystring();
        int             selval;

        if (browser == NULL)
                return  NULL;

        target = *B->br_cursel;

        if (!(target->fi_flags & BR_IS_FIELD))
                target = (FIELD_INT *) target->fi_private;

        label = target->fi_string;

        /*
         * OK, first we need to get the two toggle values
         */
        if (target->fi_sellist == NULL)
                return  NULL;

        sellist = br_cpystring(target->fi_sellist);

        /* Tokenize it */
        (void) tokenize(sellist,list,2);

        /*
         * find out which one is currently set, if none is, then
         * set the first one
         */
        if (label == NULL)
                selval = 0;
        else if (strncasecmp(label,list[0],strlen(list[0])) == 0)
                selval = 0;
        else
                selval = 1;

        target->fi_string = list[selval];

        for(;;) {
                br_disp_field(B->br_window,target,BR_INVERSE);
                br_display((BROWSER)B);

                /* Get the input */
                *lastchar = io_get_button(B->br_window);
                switch (*lastchar) {
                case    BR_UP:
                case    BR_DOWN:
                case    BR_LEFT:
                case    BR_RIGHT:
                case    BR_SELECT:
                        if (selval)
                                selval = 0;
                        else
                                selval = 1;
                        target->fi_string = list[selval];
                        break;
                case    BR_RETURN:
                        target->fi_string = br_cpystring(list[selval]);
                        (void) free(label);
                        (void) free(sellist);
                        if (target->fi_func != NULL)
                                return  (*target->fi_func)((BROWSER)B,lastchar);
                        else
                                return  (void *)target->fi_string;
                case    BR_ESCAPE:
                        target->fi_string = label;
                        return  NULL;
                }
        }
        /* NOTREACHED */
}

/*
 * br_get_sellist:
 *          Input a selection list field.  This is actually a mini-browser
 *          which overlays the current browser.  We get the current x,y
 *          position of the field, and build an overlay window centered
 *          around that position.
 */
void *
br_get_sellist(browser,lastchar)
BROWSER         browser;
char            *lastchar;
{
```

```
        FIELD_INT       *target;
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        char            *label,*sellist,*list[80],*ans;
        extern          char *br_cpystring();
        int             length,selval,nvals,xstart,ystart,i;
        void            *newwin;
        BROWSER         newbrowser;

        if (browser == NULL)
                return  NULL;

        target = *B->br_cursel;

        if (!(target->fi_flags & BR_IS_FIELD))
                target = (FIELD_INT *) target->fi_private;

        label = target->fi_string;
        length = target->fi_length;

        /*
         * OK, first we need to get the selection values
         */
        if (target->fi_sellist == NULL)
                return  NULL;

        sellist = br_cpystring(target->fi_sellist);

        /* Tokenize it */
        nvals = tokenize(sellist,list,80);

        /*
         * find out which one is currently set, if none is, then
         * set the first one
         */
        selval = 0;

        for (i = 0; i < nvals; i ++) {
                if (strcasecmp(label,list[i]) == 0) {
                        selval = i;
                        break;
                }
        }

        /*
         * Now create a new window centered on the current window
         */
        xstart = target->fi_x;
        ystart = target->fi_y - nvals/2;
        if (ystart < 2) ystart = 2;
        newwin = io_create_overlay(B->br_window,xstart,ystart,length,nvals);

        newbrowser = br_create(newwin,BR_FORM,(char *)NULL,(char *)NULL);
        for (i = 0; i < nvals; i++) {
                br_button(newbrowser,br_cpystring(list[i]),0,-1,-1,0,
                                (void *)&list[i],FVNULL,FNULL);
        }

        br_display(newbrowser,0,BR_OVERLAY);

        ans = (char *)br_activate(newbrowser,lastchar);

        (void) br_free(newbrowser);
        (void) io_destroy_window(newwin);
        io_display(B->br_window);
        if (ans == NULL)
                target->fi_string = label;
        else {
                target->fi_string = br_cpystring(ans);
                br_disp_field(B->br_window,target,0);
        }

        (void) free(label);
        (void) free(sellist);
        if (target->fi_func != NULL)
                return  (*target->fi_func)((BROWSER)B,lastchar);
        else
                return  (void *)target->fi_string;
}

/*
 * br_get_typein:
 *      input data into a typein field
 */
void *
br_get_typein(browser,lastchar)
BROWSER browser;
char    *lastchar;
{
        FIELD_INT       *target;
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        char            *text;
        extern          char *br_cpystring();

        if (browser == NULL)
```

```
                return  NULL;

        target = *B->br_cursel;
        if ( target->fi_flags & BR_IS_FIELD) {
                io_set_flags(B->br_window,target->fi_display);
                br_clear_field(B->br_window,target);
        } else {
                target = (FIELD_INT *) target->fi_private;
                io_set_flags(B->br_window,BR_INVERSE|target->fi_display);
                br_clear_field(B->br_window,target);
        }
        text = io_get_text(B->br_window,target->fi_x,target->fi_y,
                                target->fi_length,lastchar);
        io_clear_flags(B->br_window,BR_INVERSE);
        return  (void *)br_cpystring(text);
}

static int
br_disp_field(window,field,disp_flags)
void    *window;
FIELD_INT       *field;
int             disp_flags;
{
        int     start,end,x,y,length,err;

        start = field->fi_x;
        end = start+field->fi_length;
        y = field->fi_y;

        if (field->fi_display & BR_NOINVERSE)
                disp_flags = disp_flags & ~BR_INVERSE;

        io_set_flags(window,field->fi_display|disp_flags);
        br_clear_field(window,field);

        if (field->fi_string == NULL) {
                io_clear_flags(window,field->fi_display|disp_flags);
                return 0;
        }

        length = strlen(field->fi_string);
        if (length > field->fi_length)
                length = field->fi_length;

        switch(field->fi_format) {

        case BR_CENTERED:
                x = start + (field->fi_length-length)/2;
                break;
        case BR_LJUST:
                x = start;
                break;
        case BR_RJUST:
                x = end - length;
                break;

        }

        err =  io_put_text(window,x,y,field->fi_string);
        io_clear_flags(window,field->fi_display|disp_flags);
        return  err;
}

/*
 * br_clear_field:
 *      clear a field
 */

void
br_clear_field(window,field)
void    *window;
FIELD_INT       *field;
{
        char    buffer[BUFSIZ];
        int     i = field->fi_length;

        buffer[i] = '\0';

        while (i--)
                buffer[i] = ' ';

        (void) io_put_text(window,field->fi_x,field->fi_y,buffer);
        return;
}
```

libbrowse/format.c

```
#ifndef lint
static  char *RCSid = "$Id: format.c,v 1.2 90/01/08 16:30:20 scooter Exp $";
#endif  lint

/*
 * format.c:
 *      format and layout routines for libbrowse
 */

#include        <stdio.h>
#include        "browse_int.h"

static  void    br_layout();
extern  void    *emalloc();

/*
 * br_format:
 *      Format a browser
 */
int
br_format(browser)
BROWSER browser;
{
        BROWSER_INT     *B = (BROWSER_INT *)browser;
        int             err;

        if (browser == NULL)
                return  -1;

        switch (B->br_type) {
        case    BR_FORM:
                err = br_form_layout(B);
                break;
/*
        case    BR_LIST:
                err = br_list_layout(B);
                break;
*/
        case    BR_COL:
                err = br_column_layout(B);
                break;
        }
        if (err)
                return  -1;

        B->br_flags |= BR_FORMAT;
        return  0;
}

/*
 * The actual layout routines
 */

static  int
br_column_layout(B)
BROWSER_INT     *B;
{
        int     maxx,maxy,ncols = 0,nfields = 0,nother = 0;
        int     maxlen = 0;
        int     curcol = 0,currow = 0;
        FIELD_INT       *f,**others,**flds;
        extern  char    *malloc();

        (void) io_get_dims(B->br_window,&maxx,&maxy);

        for (f = B->br_flist; f != NULL; f = f->fi_next) {
                switch(f->fi_type) {
                case    BR_TYPEIN:
                case    BR_SELLIST:
                case    BR_TOGGLE:
                        return  -1;     /* Can't have a typein! */
                case    BR_TITLE:
                case    BR_PROMPT:
                        nother++;
                        break;
                case    BR_BUTTON:
                        if (f->fi_length > maxlen)
                                maxlen = f->fi_length;
                        nfields++;
                        break;
                case    BR_LABEL:
                        if (f->fi_y == -1)
                                nother++;
                        break;
                }
        }

        ncols = maxx/(maxlen+5);        /* # of columns in the list */
        curcol = 0;
        currow = 2;

        /* Update the browser structure */
        B->br_ncols = ncols;
        B->br_nfields = nfields;
```

```
	B->br_nother = nother;
	B->br_nscreen = 1;


	/*
	 * Allocate space
	 */
	flds = B->br_fields =
			(FIELD_INT **)malloc(sizeof(FIELD_INT *)*(nfields+1));
	others = B->br_others =
			(FIELD_INT **)malloc(sizeof(FIELD_INT *)*(nother+1));

	/*
	 * Now lay it out
	 */
	for (f = B->br_flist; f != NULL; f = f->fi_next) {
		switch (f->fi_type) {

		case BR_TITLE:
			br_layout(f,0,0,1,BR_CENTERED);
			currow++;
			*others = f;
			others++;
			break;
		case BR_PROMPT:
			br_layout(f,0,5,maxy-2,BR_LJUST);
			*others = f;
			others++;
			break;
		case BR_BUTTON:
			br_layout(f,1,curcol*(maxlen+5),currow,BR_LJUST);
			if (++curcol >= ncols) {
				currow++;
				curcol = 0;
			}
			*flds = f;
			flds++;
			break;
		case BR_LABEL:
			if (f->fi_y == -1) {
				if (curcol != 0) {
					currow++;curcol = 0;
				}
				br_layout(f,1,0,currow++,BR_LJUST);
			} else
				br_layout(f,1,0,currow,BR_LJUST);
			*others = f;
			others++;
			break;
		}
		if (f->fi_next != NULL) {
			if (f->fi_next->fi_y == -2)
				f->fi_next->fi_y = f->fi_y;
			if (f->fi_next->fi_x == -2)
				f->fi_next->fi_x = f->fi_x;
			if (f->fi_next->fi_screen == -2)
				f->fi_next->fi_screen = f->fi_screen;
		}
	}
	return	0;
}

static	int
br_form_layout(B)
BROWSER_INT	*B;
{
	int		maxx,maxy,nfields = 0,nother = 0;
	int		maxlen = 0,currow = 2,curscr = 1;
	FIELD_INT	*f,**others,**flds;

	(void) io_get_dims(B->br_window,&maxx,&maxy);

	for (f = B->br_flist; f != NULL; f = f->fi_next) {
		switch(f->fi_type) {
		case	BR_TYPEIN:
		case	BR_SELLIST:
		case	BR_TOGGLE:
			if (f->fi_flags & BR_IS_FIELD) {
				if (f->fi_length > maxlen)
					maxlen = f->fi_length;
				nfields++;
			} else
				nother++;
			break;
		case	BR_TITLE:
		case	BR_PROMPT:
			nother++;
			break;
		case	BR_BUTTON:
			if (f->fi_length > maxlen)
				maxlen = f->fi_length;
			nfields++;
			break;
		case	BR_LABEL:
```

```
                        nother++;
                        break;
                }
        }

        B->br_ncols = 1;
        B->br_nfields = nfields;
        B->br_nother = nother;

        /*
         * Allocate space
         */
        flds = B->br_fields =
                        (FIELD_INT **)emalloc(sizeof(FIELD_INT *)*(nfields+1)
        others = B->br_others =
                        (FIELD_INT **)emalloc(sizeof(FIELD_INT *)*(nother+1)

        /*
         * Now lay it out
         */
        for (f = B->br_flist; f != NULL; f = f->fi_next) {
                switch (f->fi_type) {

                case BR_TITLE:
                        br_layout(f,0,0,1,BR_CENTERED);
                        currow++;
                        *others = f;
                        others++;
                        break;
                case BR_PROMPT:
                        br_layout(f,0,5,maxy-2,BR_LJUST);
                        *others = f;
                        others++;
                        break;
                case BR_BUTTON:
                        br_layout(f,curscr,5,currow,BR_LJUST);
                        *flds = f;
                        flds++;
                        break;
                case BR_TYPEIN:
                case BR_TOGGLE:
                case BR_SELLIST:
                        if (f->fi_flags & BR_IS_FIELD) {
                                br_layout(f,curscr,0,currow,BR_LJUST);
                                *flds = f;
                                flds++;
                                break;
                        }
                case BR_LABEL:
                        br_layout(f,curscr,maxlen+7,currow,BR_LJUST);
                        *others = f;
                        others++;
                        break;
                }

                if (f->fi_y == currow) {
                        currow++;
                        if (currow > maxy-2) {
                                curscr++;
                                currow = 2;
                        }
                }

                if (f->fi_next != NULL) {
                        if (f->fi_next->fi_y == -2)
                                f->fi_next->fi_y = f->fi_y;
                        if (f->fi_next->fi_x == -2)
                                f->fi_next->fi_x = f->fi_x;
                        if (f->fi_next->fi_screen == -2)
                                f->fi_next->fi_screen = f->fi_screen;
                }
        }

        B->br_nscreen = curscr;
        return  0;
}


/*
 * br_layout:
 *      set the locations if they're not set already
 */
static void
br_layout(field,screen,x,y,format)
FIELD_INT       *field;
int             screen,x,y,format;
{
        if (field->fi_screen == -1)
                field->fi_screen = screen;

        if (field->fi_x == -1)
                field->fi_x = x;

        if (field->fi_y == -1)
```

```
                field->fi_y = y;

        If (field->fi_format == 0)
                field->fi_format = format;

        return;
}
```

ftd/ftd.c

```
#ifndef lint
char *RCSid = "$Id: ftd.c,v 1.13 90/04/23 15:47:46 conrad Exp $";
#endif lint

/*
 * Foreign Tool Driver
 */

# include <ftd.h>
# include <signal.h>
# include <setjmp.h>
# include <syslog.h>
# include <sys/time.h>
# ifdef RLIMIT
# include <sys/resource.h>
# endif
# include <errno.h>

# define TIMEOUT 1
# define USAGE ""

int         timeout_flag;

char        *progname = NULL;
char        *progpath = NULL;
char        *progargs = NULL;
char        *ftdrules = NULL;
char        *cddir = NULL;

int         interactive = 0;
jmp_buf     timer;

main(argc,argv)
char **argv;
{
            extern      char *optarg;
            extern      int  optind,opterr;
            extern      char *strrchr();
            int         i,readfd,writefd;
            SIGNALTYPE          timeout();
            char        buffer[BUFSIZ];
            char        *cid,*result,c;
            static int          bufcnt;
            extern      int         errno;
            extern      char        *sys_errlist[];
#ifdef RLIMIT
            struct      rlimit      rlp;
#endif

            cid = NULL;
            while ((i = getopt(argc,argv,"C:r:p:a:c:")) != EOF) {
                        switch (i) {
                                    case 'C':
                                                cid = optarg;
                                                break;

                                    case 'r':
                                                ftdrules = optarg;
                                                interactive++;
                                                break;

                                    case 'p':
                                                progpath = optarg;
                                                interactive++;
                                                break;

                                    case 'a':
                                                progargs = optarg;
                                                interactive++;
                                                break;

                                    case 'c':
                                                cddir = optarg;
                                                break;

                                    default:
                                                (void) fprintf(stderr,USAGE);
                                                exit(2);
                        }
            }

            if (interactive > 0 && interactive < 3) {
                        (void) fprintf(stderr,
                                    "Must specify rules, program, and args\n");
                        exit(2);
            }

            if (interactive == 0) {

#ifdef      LOG_LOCAL4
                        openlog("mmwb_ftd", LOG_PID, LOG_LOCAL4);
#else
                        openlog("mmwb_ftd", LOG_PID);
#endif
```

```
        /* Handshake and establish the SM connection */

        if (sm_init(cid) == NULL) {
                syslog(LOG_ERR,"sm_init failed: %s",sm_error());
                (void) exit(1);
        }

        /*
          Get the mandatory options: progpath,progargs,ftrules
        */

        if (sm_request_option("program", (void *) &progpath, DI_TEXT)

                (void) sprintf(buffer,"Can't get program name - %s",
                                                        sm_error());
                (void) sm_exit(buffer);
        }

        if (sm_request_option("args", (void *) &progargs, DI_TEXT) < 0)
                (void) sm_exit("Can't get arguments");

        if (sm_request_option("rules", (void *) &ftdrules, DI_TEXT) < 0)
                (void) sm_exit("Can't get rule file");

        (void) sm_request_option("working_directory", (void *) &cddir, DI_


}

if (cddir != NULL) {
        char *s,*index();
        if ( (s = index(cddir,'\n')) != NULL )
                *s = '\0';
        if ( chdir(cddir) < 0) {
                (void) sprintf(buffer,
                        "Unable to change to directory %s", cddir);
                if (interactive)
                        (void) sprintf(stderr,"%s\n",buffer);
                else
                        (void) sm_notify(buffer);
        }
}


progname = strrchr(progpath,'/');
if (progname != NULL)
        progname++;
else
        progname = progpath;


/*
  Open ftdrules and parse
*/

if ((result = ftd_parserules(ftdrules, (FTD_CMDTAB *) NULL)) != NULL) {
        if(interactive) {
                (void) fprintf(stderr,"%s\n",result);
                exit(1);
        } else {
                syslog(LOG_ERR,"ftd_parserules fails - %s",result);
                (void) sm_exit(result);
        }
}

#ifdef RLIMIT
        /*
         * Now up the memory limits
         */
        (void) getrlimit(RLIMIT_DATA, &rlp);
        rlp.rlim_cur = rlp.rlim_max;            /* Get the max */

        if (setrlimit(RLIMIT_DATA, &rlp) < 0) {
                (void) sprintf(buffer,"setrlimit failed: %s",sys_errlist[errno]);
                if (interactive) {
                        (void) fprintf(stderr,"%s\n",buffer);
                        exit(1);
                } else
                        (void) sm_exit(buffer);
        }
#endif

        if ((result = ftd_launch(progpath,progname,progargs,&readfd,&writefd))
                                                                != NULL) {
                if(interactive) {
                        (void) fprintf(stderr,"%s\n",result);
                        exit(1);
                } else {
                        syslog(LOG_ERR,"ftd_launch fails - %s",result);
                        (void) sm_exit(result);
                }
        }
```

ftd/ftd.c

```
        /*
          The program is now running, loop waiting for input
          and handle it appropriately.  There is somewhat
          of a problem in the general case.  Some foreign
          tools may prompt the user without a trailing newline.
          Against this eventuality, we need to set an alarm.
        */

        /* Make sure SIGALRM is not blocked */
#ifdef SIGBLOCK
        i = sigsetmask(0);
        (void) sigsetmask(i & ~sigmask(SIGALRM));
#else
        (void) sigrelse(SIGALRM);
#endif
        (void) signal(SIGALRM,timeout);
        bufcnt = 0;
        while (setjmp(timer) == 1 || read(readfd, &c, 1) == 1) {
                (void) alarm(0);
                if (timeout_flag == 1 || (buffer[bufcnt++] = c) == '\n') {
                        buffer[bufcnt] = '\0';
                        result = ftd_process(writefd,&buffer[0]);
                        if (result != NULL) {
                                (void) ftd_term(readfd,writefd);
                                if(interactive) {
                                        (void) fprintf(stderr,"%s\n",result);
                                        exit(1);
                                } else {
                                        syslog(LOG_ERR,
                                          "ftd_proecess fails - %s",result);
                                        (void) sm_exit(result);
                                }
                        }
                        buffer[0] = '\0';
                        bufcnt = 0;
                }
                timeout_flag = 0;
                if (bufcnt > 0)
                        (void) alarm(TIMEOUT);
        }

        if (interactive) {
                (void) fprintf(stdout, "%s complete\n", progname);
                exit(0);
        }
        (void) sprintf(buffer,"%s complete",progname);
        (void) sm_exit(buffer);
}

SIGNALTYPE
timeout()
{
        timeout_flag = 1;
        longjmp(timer,1);
}
```

ftd_skel/ftd_skel.c

```
#ifndef  lint
char *RCSid = "$Header: ftd_skel.c,v 1.2 89/06/26 11:58:12 scooter Exp $";
#endif lint

/*
 * Skeleton Foreign Tool Driver
 */

# include <ftd.h>
# include <signal.h>
# include <setjmp.h>
# include <syslog.h>

# define TIMEOUT 1
# define USAGE ""

int         timeout_flag;

char        *progname = NULL;
char        *progpath = NULL;
char        *progargs = NULL;
char        *ftdrules = NULL;

int         interactive = 0;
jmp_buf     timer;


/*
 * Define any locally desired functions
 */

FTD_CMDTAB mytable[] = {
        { NULL /* function name */, NULL /* function pointer */}
};


main(argc,argv)
char **argv;
{
        extern  char *optarg;
        extern  int  optind,opterr;
        extern  char *strrchr();
        int         i,readfd,writefd;
        int         timeout();
        char        buffer[BUFSIZ];
        char        *cid,*result,c;

while ((i = getopt(argc,argv,"C:r:p:a:")) != EOF) {
        switch (i) {
                case 'C':
                        cid = optarg;
                        break;

                case 'r':
                        ftdrules = optarg;
                        interactive++;
                        break;

                case 'p':
                        progpath = optarg;
                        interactive++;
                        break;

                case 'a':
                        progargs = optarg;
                        interactive++;
                        break;

                default:
                        (void) fprintf(stderr,USAGE);
                        exit(2);
        }
}

if (interactive > 0 && interactive < 2) {
        (void) fprintf(stderr,
                "Must specify rules, program, and args\n");
        exit(2);
}

if (interactive == 0) {

        openlog("mmwb_ftd", LOG_PID, LOG_LOCAL4);

        /* Handshake and establish the SM connection */

        if (sm_init(cid) == NULL)
                (void) exit(1);

        /*
          Get the mandatory options: progpath,progargs,ftrules
        */

        if (sm_request_option("program", &progpath, DI_TEXT) < 0) {
```

```
                sprintf(buffer,"Can't get program name - %s",
                                                        sm_error());
                sm_exit(buffer);
        }

        if (sm_request_option("args", &progargs, DI_TEXT) < 0)
                sm_exit("Can't get arguments");

        if (sm_request_option("rules", &ftdrules, DI_TEXT) < 0)
                sm_exit("Can't get rule file");


}
progname = strrchr(progpath,'/');
if (progname != NULL)
        progname++;
else
        progname = progpath;


/*
  Add any locally desired options here
*/

/*
  Open ftdrules and parse
*/

if ((result = ftd_parserules(ftdrules,NULL)) != NULL) {
        if(interactive) {
                fprintf(stderr,"%s\n",result);
                exit(1);
        } else {
                syslog(LOG_ERR,"ftd_parserules fails - %s",result);
                sm_exit(result);
        }
}

if ((result = ftd_launch(progpath,progname,progargs,&readfd,&writefd))
                                                                != NU
        if(interactive) {
                fprintf(stderr,"%s\n",result);
                exit(1);
        } else {
                syslog(LOG_ERR,"ftd_launch fails - %s",result);
                sm_exit(result);
        }
}

/*
  The program is now running, loop waiting for input
  and handle it appropriately.  There is somewhat
  of a problem in the general case.  Some foreign
  tools may prompt the user without a trailing newline.
  Against this eventuality, we need to set an alarm.
*/

/* Make sure SIGALRM is not blocked */
i = sigsetmask(0);
(void) sigsetmask(i & ~sigmask(SIGALRM));
(void) signal(SIGALRM,timeout);
i = 0;
while (setjmp(timer) == 1 || read(readfd, &c, 1) == 1) {
        (void) alarm(0);
        if (timeout_flag == 1 || (buffer[i++] = c) == '\n') {
                buffer[i] = '\0';
                result = ftd_process(writefd,&buffer[0]);
                if (result != NULL) {
                        (void) ftd_term(&readfd,&writefd);
                        if(interactive) {
                                fprintf(stderr,"%s\n",result);
                                exit(1);
                        } else {
                                syslog(LOG_ERR,
                                  "ftd_proecess fails - %s",result);
                                sm_exit(result);
                        }
                }
                buffer[0] = '\0';
                i = 0;
        }
        timeout_flag = 0;
        (void) alarm(TIMEOUT);
}

if (interactive) {
        (void) fprintf(stdout, "%s complete\n", progname);
        exit(0);
}
(void) sprintf(buffer,"%s complete",progname);
(void) sm_exit(buffer);
}

int
```

ftd_skel/ftd_skel.c

```
timeout()
{
        timeout_flag = 1;
        longjmp(timer,1);
}
```

libftd/ftd.h

```
/*
 * $Id: ftd.h,v 1.7 89/11/06 13:01:22 gregc Exp $
 *
 * Include file for the Foreign Tool Driver library
 *
 * $Log:		ftd.h,v $
 * Revision 1.7  89/11/06  13:01:22  gregc
 * fix ftd_process prototype.
 *
 * Revision 1.6  89/10/26  15:11:17  gregc
 * oops.
 *
 * Revision 1.5  89/10/26  13:28:30  gregc
 * add prototypes.
 *
 * Revision 1.4  89/10/25  15:18:18  gregc
 * use mmwbmisc.h
 *
 * Revision 1.3  89/06/26  10:39:53  scooter
 * Working version - interactive + SM
 *
 * Revision 1.2  89/05/15  15:41:25  scooter
 * Added ftd_term
 *
 * Revision 1.1  89/05/15  15:24:21  scooter
 * Initial revision
 *
 *
 */

#ifndef	FTD_INCLUDE

#include		<stdio.h>
#include		<dialog.h>
#include		<mmwbmisc.h>
#include		<sm.h>

#ifdef __STDC__
#define	_P_(x)	x
#else
#define	_P_(x)	()
#endif

typedef	struct	ftd_cmdtab_s {
		char	*command;
		char	*(*function)();
} FTD_CMDTAB;

extern	char	*ftd_parserules _P_((char *, FTD_CMDTAB *));
extern	char	*ftd_process _P_((int, char *));
extern	char	*ftd_launch _P_((char *, char *, char *, int *, int *));
extern	char	*ftd_term _P_((int, int));

/*
 * Intrinsic functions
 */
extern	char	*ftd_notify _P_((int, char *, int, DIALOG));
extern	char	*ftd_dialog _P_((int, char *, int, DIALOG));
extern	char	*ftd_print _P_((int, char *, int, DIALOG));
extern	char	*ftd_input _P_((int, char *, int, DIALOG));
extern	char	*ftd_sink _P_((int, char *, int, DIALOG));

/*
 * Flag values
 */

#define	FTD_QUIT		1	/* Quit when done */
#define	FTD_SEND		2	/* Write the result to the program */
#define	FTD_OPTION		4	/* Try to get the data from the option sheet */
#define	FTD_COPY_PROMPT		/* Copy the prompt from the tool */

#undef _P_

#define	FTD_INCLUDE
#endif	FTD_INCLUDE
```

libftd/ftd_internal.h

```
/*
 * $Id: ftd_internal.h,v 1.4 89/11/06 13:01:10 gregc Exp $
 *
 * libftd internal include
 *
 * $Log:            ftd_internal.h,v $
 * Revision 1.4  89/11/06  13:01:10  gregc
 * add prototypes
 *
 * Revision 1.3  89/10/26  15:16:38  gregc
 * use ftd_yy* instead of yy* so other programs that use this library
 * can use yacc.
 *
 * Revision 1.2  89/06/26  10:39:29  scooter
 * Working version - interactive + SM
 *
 * Revision 1.1  89/05/15  15:24:45  scooter
 * Initial revision
 *
 *
 */

#ifndef  FTD_I_INCLUDE

#include          "ftd.h"

#define  TRUE     1
#define  FALSE    0

#ifdef __STDC__
#define  _P_(x)   x
#else
#define  _P_(x)   ()
#endif

/*
 * Parsed version of a rule
 */

typedef  struct ftd_s {
         char *name;              /* Name of the rule */
         char *precond;           /* Preconditions */
         char *pattern;           /* Pattern to match */
         int  flags;              /* Flags */
         char *action;            /* Character representation of action */
         char *(*function)();     /* Function to call */
         DIALOG dialog;           /* The dialog to pass to function */
         char *postcond;          /* Postconditions */
} FTD_TAB;

/*
 * Some tunable constants
 */

#define  MAXVARS        50       /* Maximum number of variables */
#define  MAXRULES       100      /* Maximum number of rule entries */

extern   int      parsing_error;
extern   char     *ftd_yyfilename;
extern   int      ftd_yylineno;
extern   char     *ftd_setcond _P_((char *));
extern   int      ftd_chkcond _P_((char *));

extern   FTD_TAB*ftd_table _P_((void));
extern   FTD_TAB*ftd_next _P_((FTD_TAB *));

#undef _P_

#define  FTD_I_INCLUDE
#endif   FTD_I_INCLUDE
```

libftd/grammar.h

```
typedef union {
        int             integer;
        double          real;
        char            *charval;
        FTD_TAB         ftival;
        struct  {
                union   {
                        int     integer;
                        double  real;
                }       lower, upper;
        }               range;
} YYSTYPE;
extern YYSTYPE ftd_yylval;
# define PERIOD 257
# define COLON 258
# define SEMICOLON 259
# define COMMA 260
# define QUESTION 261
# define IMPLIES 262
# define EQUALS 263
# define LEFTBRACE 264
# define RIGHTBRACE 265
# define LEFTPAREN 266
# define RIGHTPAREN 267
# define LEFTBRACKET 268
# define RIGHTBRACKET 269
# define DOTDOT 270
# define K_INTEGER 271
# define K_REAL 272
# define K_TEXT 273
# define K_SELECT 274
# define K_FROM 275
# define K_MULTI 276
# define K_BOOLEAN 277
# define K_YESNO 278
# define K_NOTIFIER 279
# define K_LABEL 280
# define K_PROMPT 281
# define K_HELP 282
# define K_SEMANTIC 283
# define K_FLAGS 284
# define K_OPTIONS 285
# define K_DEFAULT 286
# define K_YES 287
# define K_NO 288
# define K_TRUE 289
# define K_FALSE 290
# define K_DIALOG 291
# define K_OPTION 292
# define K_SHEET 293
# define F_QUIT 294
# define F_OPTION 295
# define F_SEND 296
# define F_CPYPRMPT 297
# define IDENTIFIER 298
# define QUOTED_TEXT 299
# define INTEGER 300
# define REAL 301
# define B_AND 302
# define B_OR 303
# define AND 304
# define OR 305
# define LT 306
# define LE 307
# define GT 308
# define GE 309
# define R_EQUAL 310
# define NOT_EQUAL 311
# define NOT 312
# define ADD 313
# define SUBTRACT 314
# define MULTIPLY 315
# define DEVIDE 316
# define UMINUS 317
```

libftd/ftd_cond.c

```
#ifndef lint
static char *RCSid = "$Id: ftd_cond.c,v 1.6 89/11/13 17:13:15 scooter Exp $";
#endif    lint

/*
 * ftd_cond:
 *          routines to handle condition expressions and
 *          variables
 */

#include "ftd_internal.h"
#include <ctype.h>

static struct var_st {
          char      *var_name;
          int       var_val;
} var_table[MAXVARS];

typedef   struct    ex_st {
          int       value;
          char      *variable;
          char      op;
} EVEX;

/* Token values */
#define   ERROR   -1
#define   OPERATOR          1
#define   CONSTANT          2
#define   VARIABLE          3

static int lastvar = 0;
static char cond_buffer[1024];

static int          eval_expr();
static EVEX         eval_op();

/*
 * ftd_chkcond:
 *          return 0 if a condition has not been met,
 *          return 1 otherwise.
 *
 */

int
ftd_chkcond(condition)
char      *condition;
{
          EVEX              expr;
          int               tok;
          char              *cond;

          if (condition == NULL)
                    return 1;

          cond = mmwb_cpystring(condition);

          tok = eval_expr(&cond,&expr);

          mmwb_stringfree(cond);

          if (tok == ERROR)
                    return    0;

          return    expr.value;
}

/*
 * ftd_setcond:
 *          set a condition
 */

char *
ftd_setcond(condition)
char      *condition;
{
          EVEX      expr;
          char      *cond,*next_cond,*p,*strchr();

          if (condition == NULL)
                    return NULL;

          next_cond = cond = mmwb_cpystring(condition);

          /* Now loop through all the requested conditions */

          while (next_cond != NULL) {

                    /* Get a pointer to the next condition */
                    p = strchr(next_cond,',');
                    if (p != NULL)
                              *p++ = '\0';

                    if (eval_expr(&next_cond,&expr) == ERROR) {
                              (void) sprintf(cond_buffer, "Illegal assignment: %s",
```

```
                        return(cond_buffer);
                }
                next_cond = p;
        }

        mmwb_stringfree(cond);

        return NULL;
}

/*
 * Set the value of a variable
 */

static void
set_var(name,value)
char *name;
int  value;
{
        int     i;

        for (i = 0; i < lastvar; i++) {
                if (strcmp(name,var_table[i].var_name) == 0) {
                        var_table[i].var_val = value;
                        return;
                }
        }

        var_table[lastvar].var_name = mmwb_cpystring(name);
        var_table[lastvar++].var_val = value;
        return;
}

/*
 * Return the value of a variable
 */

static int
get_var(name,err)
char *name;
int *err;
{
        int     i;

        *err = 0;
```



```
condition);

        for (i = 0; i < lastvar; i++) {
                if (strcmp(name,var_table[i].var_name) == 0) {
                        return var_table[i].var_val;
                }
        }

        *err = -1;
        return 0;
}

/*
 * Evaluate an expression
 */

#define  PLUS     '+'
#define  MINUS    '-'
#define  TIMES    '*'
#define  DIVIDE   '/'
#define  GT       '>'
#define  LT       '<'
#define  LE       '.'
#define  GE       ':'
#define  NOT      '~'
#define  NOTEQUAL '#'
#define  AND      '&'
#define  OR       '|'
#define  B_AND    '$'
#define  B_OR     '%'
#define  ASSIGNMENT '_'
#define  EQUAL    '='


static int
eval_expr(expr,val)
char **expr;
EVEX  *val;
{
        char    *p = *expr;
        char    *s,c;
        int     err;
        EVEX    lhs,op,rhs;
        EVEX    eval_op();

        /* Clear out val */

        val->op = '\0';
        val->value = 0;
```

```
        val->variable = NULL;

        while (isspace(*p))
                p++;

        switch (c = *p++) {

        /* Operators */
        case PLUS:
        case MINUS:
        case DIVIDE:
        case TIMES:
                val->op = c;
                break;

        case '>':
                if ((c = *p++) == '>') {
                        val->op = GT;
                        break;
                } else if (c == '=') {
                        val->op = GE;
                        break;
                }
                return  ERROR;

        case '<':
                if (*p++ == '<') {
                        val->op = LT;
                        break;
                } else if (c == '=') {
                        val->op = LE;
                        break;
                }
                return  ERROR;

        case '!':
                if (*p == '=') {
                        val->op = NOTEQUAL;
                        p++;
                        break;
                }
                return  ERROR;

        case '~':
                val->op = NOT;
                break;

        case '=':
                if (*p != '=') {
                        val->op = ASSIGNMENT;
                        break;
                } else {
                        val->op = EQUAL;
                        p++;
                        break;
                }

        case '&':
                if (*p != '&') {
                        val->op = B_AND;
                        break;
                } else {
                        val->op = AND;
                        p++;
                        break;
                }

        case '|':
                if (*p != '|') {
                        val->op = B_OR;
                        break;
                } else {
                        val->op = OR;
                        p++;
                        break;
                }

        /* Expression */
        case '(':
        {
                int     unary = 0;
                int     tok;

                tok = eval_expr(&p,&lhs);               /* Get the LHS */
                if (tok != OPERATOR) {
                        tok = eval_expr(&p,&op);/* Get the operator */
                        if (tok != OPERATOR)
                                return  ERROR;
                } else {
                        op = lhs;                       /* Oops, Unary operator */
                        unary++;
                }

                tok = eval_expr(&p,&rhs);               /* Get the RHS */
```

```
        if (tok != CONSTANT)
                if (tok != VARIABLE)
                        return  ERROR;
                else {
                        rhs.value = get_var(rhs.variable,&err);
                        if (err == -1)
                                return  ERROR;
                }

        if (unary != 0)
                *val = eval_op((EVEX *)NULL,&op,&rhs);          /* Eva
        else
                *val = eval_op(&lhs,&op,&rhs);          /* Evaluate */

        while (isspace(*p))
                p++;

        if (*p != ')')
                return  ERROR;

        *expr = ++p;
        return  CONSTANT;
}

/* Constant or Variable */
default:
        s = (--p);
        if (isdigit(c)) {                       /* Constant */
                while (isdigit(*s))
                        s++;
                *s = '\0';
                (void) sscanf(p,"%d",&val->value);
                *expr = ++s;
                return  CONSTANT;
        } else {                                /* Better be a var */
                while (!isspace(*s))
                        s++;
                *s = '\0';
                val->variable = p;
                *expr = ++s;
                return  VARIABLE;
        }
}

*expr = p;
return  OPERATOR;
}

/*
 * eval_op
 */

static EVEX
eval_op(lhs,op,rhs)
EVEX *lhs,*op,*rhs;
{
        int     left,right,err;
        EVEX    result;

        /* clear result */

        result.value = 0;
        result.variable = NULL;
        result.op = '\0';

        right = rhs->value;

        /*
         * Special case assignments
         */

        if (op->op == ASSIGNMENT) {
                /* Assume lhs is a variable, rhs is a constant */
                set_var(lhs->variable,rhs->value);
                result = *rhs;
                return  result;
        }

        /*
         * Check for unary operators
         */

        if (lhs == (EVEX *)NULL) {
                switch (op->op) {
                case    NOT:
                        result.value = !right;
                        break;
                case    '-':
                        result.value = -right;
                        break;
                }
                return  result;
        }
```

```
/*
 * If the lhs is a variable, resolve it
 */

if (lhs->variable != NULL)
        left = get_var(lhs->variable,&err);
else
        left = lhs->value;

switch (op->op) {

case    PLUS:
        result.value =  left + right;
        break;

case    MINUS:
        result.value =  left - right;
        break;

case    TIMES:
        result.value =  left * right;
        break;

case    DIVIDE:
        result.value =  left / right;
        break;

case    EQUAL:
        result.value = (left==right);
        break;

case    GT:
        result.value = (left>>right);
        break;

case    LT:
        result.value = (left<<right);
        break;

case    NOTEQUAL:
        result.value = (left!=right);
        break;

case    LE:
        result.value = (left<=right);
        break;

case    GE:
        result.value = (left>=right);
        break;

case    AND:
        result.value = (left&&right);
        break;

case    OR:
        result.value = (left||right);
        break;

case    B_AND:
        result.value = (left&right);
        break;

case    B_OR:
        result.value = (left|right);
        break;
}

return  result;
}
```

libftd/ftd_debug.c

```
#ifndef lint
static char *RCSid = "$Header: ftd_debug.c,v 1.1 89/06/26 10:38:23 scooter Exp $";
#endif lint

/*
 * This module contains some intrinsics used for debugging
 * purposes only
 *
 */

#include        "ftd.h"

/* ARGSUSED */
char *
ftd_print(fd, prompt, flags, template)
int         fd;
char        *prompt;
int         flags;
DIALOG      template;
{
        (void) fprintf(stdout,"%s", prompt);
        return      NULL;
}

/* ARGSUSED */
char *
ftd_input(fd, prompt, flags, template)
int         fd;
char        *prompt;
int         flags;
DIALOG      template;
{
        int         err;
        static      char        buf[BUFSIZ];

        if (flags & FTD_COPY_PROMPT)
                (void) fprintf(stdout,"%s", prompt);
        else
                (void) fprintf(stdout,"%s",
                        dialog_get_text(template, DI_V_PROMPT, &err));

        (void) fgets(buf,BUFSIZ,stdin);

        if (flags & FTD_SEND)
                write(fd,buf,strlen(buf));

        return      NULL;
}

/*
 * ftd_sink:
 *          just throw it away!
 */

/* ARGSUSED */
char *
ftd_sink(fd, prompt, flags, template)
int         fd;
char        *prompt;
int         flags;
DIALOG      template;
{
        return      NULL;
}
```

libftd/ftd_dialog.c

```
#ifndef lint
static char     *RCSid = "$Header: /local/src/mmwb/src/ftd/libftd/RCS/ftd_dialog.
#endif

# include <strings.h>
# include "ftd.h"
# include <sm.h>
# include <syslog.h>

static  char    buf[BUFSIZ];

/*
 * ftd_dialog:
 *      Function called to handle input from foreign tool.  First we
 *      check whether the request may be filled in by an option value.
 *      If so, we get the value and send it to the tool.  If not, we
 *      construct a dialog from the given template, send it off, and
 *      wait for a reply.  When we get the reply, we send it to the
 *      foreign tool.
 */
char *
ftd_dialog(fd, prompt, flags, template)
int     fd;
char    *prompt;
int     flags;
DIALOG  template;
{
        int     type;
        char    *name;
        int     err;
        void    *value;
        int     value_i;
        float   value_f;
        char    *value_s;
        DIALOG  answer;

        /*
         * Get the type of the return value
         */
        type = dialog_type(template);
#ifdef  DEBUG
        syslog(LOG_DEBUG,"ftd_dialog: %s",prompt);
#endif  DEBUG

        /*
         * Check for option or dialog
         */
        if (flags & FTD_OPTION) {
                name = dialog_name(template);
#ifdef  DEBUG
                syslog(LOG_DEBUG,"ftd_dialog: looking for option %s",name);
#endif  DEBUG
                switch (type) {
                 case DI_IVAL:
                 case DI_YESNO:
                 case DI_BOOL:
                        value = (void *)&value_i;
                        break;
                 case DI_FVAL:
                        value = (void *)&value_f;
                        break;
                 case DI_TEXT:
                        value = (void *)&value_s;
                        break;
                 default:
                        (void) sprintf(buf,"Unknown dialog type %d",type);
                        return buf;
                }
                if (sm_request_option(name, value, type) < 0) {
                        (void) sprintf(buf,"Option %s is not available",name);
                        return buf;
                }
        }
        else {
                if (flags & FTD_COPY_PROMPT)
                        if (dialog_set_text(template, DI_V_PROMPT, prompt) < 0){
                                (void) sprintf(buf,"Unable to copy prompt");
                                return buf;
                        }
#ifdef  DEBUG
                syslog(LOG_DEBUG,"ftd_dialog: sending dialog %s",prompt);
#endif  DEBUG
                answer = sm_dialog(template);
                if (answer == NULL) {
                        (void) sprintf(buf,"Dialog failed");
                        return buf;
                }
                switch (type) {
                 case DI_YESNO:
                 case DI_BOOL:
                        value_i = dialog_get_bool(answer, DI_V_RESULT, &err);
                        break;
                 case DI_IVAL:
                        value_i = dialog_get_int(answer, DI_V_RESULT, &err);
```

```
                        break;
                case DI_FVAL:
                        value_f = dialog_get_float(answer, DI_V_RESULT, &err
                        break;
                case DI_TEXT:
                        value_s = dialog_get_text(answer, DI_V_RESULT, &err)
                        break;
                }
                if (err < 0) {
                        if ((flags & FTD_OPTION) == 0)
                                dialog_free(answer);
                        (void) sprintf(buf,"Type mismatch in dialog");
                        return  buf;
                }
        }

        /*
         * Check whether we should send
         */
        if (flags & FTD_SEND) {
#ifdef  DEBUG
                syslog(LOG_DEBUG,"ftd_dialog: sending response");
#endif  DEBUG
                switch (type) {
                case DI_YESNO:
                        value_s = value_i ? "y\n" : "n\n";
                        break;
                case DI_BOOL:
                        value_s = value_i ? "t\n" : "f\n";
                        break;
                case DI_IVAL:
                        (void) sprintf(buf, "%d\n", value_i);
                        break;
                case DI_FVAL:
                        (void) sprintf(buf, "%g\n", value_f);
                        break;
                case DI_TEXT:
                        (void) sprintf(buf, "%s\n", value_s);
                        break;
                }
#ifdef  DEBUG
        syslog(LOG_DEBUG,"Sending to foreign tool: (%d) %s",strlen(buf),buf);
#endif  DEBUG
                if (write(fd, buf, strlen(buf)) < 0)
                        goto failed;
        }
        /*
         * Done.  Dispose of the reply dialog and return success
         */
        return NULL;

failed:
        if ((flags & FTD_OPTION) == 0)
                dialog_free(answer);
        (void) sprintf(buf, "write to program failed");
        return buf;
}

/*
 * ftd_notify:
 *      Function called to send notification to user.
 */
/* ARGSUSED */
char *
ftd_notify(fd, prompt, flags, template)
int     fd;
char    *prompt;
int     flags;
DIALOG  template;
{
        if (flags & FTD_COPY_PROMPT)
                if (dialog_set_text(template, DI_V_PROMPT, prompt) < 0){
                        (void) sprintf(buf,"Unable to copy prompt");
                        return buf;
                }
        (void) sm_dialog(template);
        return NULL;
}
```

libftd/ftd_launch.c

```
#ifndef lint
static char *RCSid = "$Header: /usr.MC68020/src/local/mmwb/src/ftd/libftd/RCS/ftd_l
#endif lint

/*
 * ftd_launch:
 *      launch a foreign tool
 */

# include "ftd_internal.h"

char *
ftd_launch(progpath,progname,progargs,readfd,writefd)
char    *progpath,*progname,*progargs;
int     *readfd,*writefd;
{
        static char             buffer[BUFSIZ];

        if (popentty2(progpath,progname,progargs,readfd,writefd) < 0) {
                (void) sprintf(buffer,"Unable to execute %s",progpath);
                return  buffer;
        }
        return  NULL;
}

char *
ftd_term(readfd,writefd)
int     readfd,writefd;
{
        static  char buf[BUFSIZ];

        if (pclosetty2(readfd,writefd)) {
                (void) sprintf(buf,"Unable to close process");
                return  buf;
        }

        return  NULL;
}
```

libftd/ftd_parserules.c

```
#ifndef  lint
static char *RCSid = "$Id: ftd_parserules.c,v 1.6 89/10/26 13:28:55 gregc Exp $";
#endif lint

/*
 * ftd_parserules:
 *          parse a rule file
 */

# include "ftd_internal.h"

/*
 * Define the "intrinsics table"
 */

static    FTD_CMDTAB itable[] = {
          {"ftd_dialog",          ftd_dialog},
          {"ftd_notify",  ftd_notify},
          {"ftd_print", ftd_print},
          {"ftd_input", ftd_input},
          {"ftd_sink", ftd_sink},
          { NULL, NULL}
};

extern    FILE      *ftd_yyin, *ftd_yyout;
extern    char      *ftd_yyfilename;
extern    int       ftd_yylineno;

char *
ftd_parserules(rulefile, cmdtab)
char *rulefile;
FTD_CMDTAB *cmdtab;
{
          static char          buffer[BUFSIZ];
          static char          *ftd_convert();

          /*
            Open ftdrules and process
          */

          if ((ftd_yyin = fopen(rulefile, "r")) == NULL) {
                    (void) sprintf(buffer,"unable to open %s for reading\n",
                                                                        rulefil
                    return    buffer;
          }

          ftd_yyfilename = rulefile;
          ftd_yylineno = 1;
          (void) ftd_yyparse();
          return ftd_convert(cmdtab);
}

/*
 * ftd_convert:
 *          convert function names to function pointers
 *
 */

static char *
ftd_convert(ctable)
FTD_CMDTAB      *ctable;
{
          FTD_TAB             *rules = ftd_table();
          static    char      buffer[BUFSIZ];
          static int            chkaction();

          /*
           * For each entry in "rules" find a match in
           * itable or ctable.  If no match exists,
           * return an error
           */

          while (rules->action != NULL) {
                    if (chkaction(ctable,rules) &&
                        chkaction(itable,rules)) {
                              (void) sprintf(buffer,"Unable to find a match for %s",
                                        rules->action);
                              return    buffer;
                    }
                    rules = ftd_next(rules);
          }
          return NULL;
}

static int
chkaction(table,rule)
FTD_CMDTAB *table;
FTD_TAB   *rule;
{
          if (table == NULL)
                    return    -1;

          while (table->command != NULL) {
                    if (strcmp(table->command,rule->action) == 0) {
```

libftd/ftd_parserules.c

```
                        rule->function = table->function;
                        return    0;
                }
                table++;
        }
        return -1;
}
```

libftd/ftd_process.c

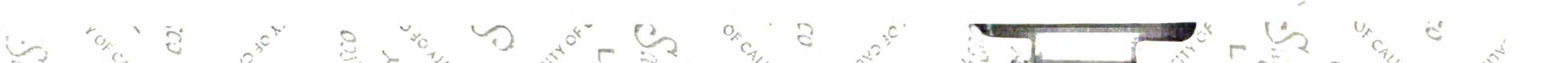
```
#ifndef lint
static char *RCSid = "$Header: ftd_process.c,v 1.2 89/06/26 10:38:58 scooter Exp $";
#endif lint

# include "ftd_internal.h"

char *
ftd_process(writefd, buffer)
int         writefd;
char        *buffer;
{
        char        *result;
        static      char        errbuf[BUFSIZ];
        char        *err,*re_comp();
        FTD_TAB*table = ftd_table();

        while (table->name != NULL) {
                if (ftd_chkcond(table->precond)) {

                        /*
                         * preconditions are met!
                         */
                        if ( (err = re_comp(table->pattern)) != NULL ) {
                          (void) sprintf(errbuf,"Illegal pattern: %s (%s)",
                                        table->pattern, err);
                          return errbuf;
                        }

                        if (re_exec(buffer) == 1) {
                                /*
                                 * The pattern matches!
                                 */
                                result = (table->function)(writefd,buffer,
                                        table->flags,table->dialog);
                                if (result == NULL)
                                        result = ftd_setcond(table->postcon
                                return result;
                        }
                }
                table = ftd_next(table);
        }
        return NULL;
}
```

libftd/popen2.c

```
/*
 * Copyright (c) 1980 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>

extern	char *malloc();

static	int *popen_pid;
static	int nfiles;

int
popen2(path,program,args,readfd,writefd)
	char *path;
	char *program;
	char *args;
	int  *readfd,*writefd;
{
	int p1[2];
	int p2[2];
	int rfd, wfd, pid;

	if (nfiles <= 0)
		nfiles = getdtablesize();
	if (popen_pid == NULL) {
		popen_pid = (int *)
			malloc((unsigned)(nfiles * sizeof *popen_pid));
		if (popen_pid == NULL)
			return -1;
		for (pid = 0; pid < nfiles; pid++)
			popen_pid[pid] = -1;
	}
	if (pipe(p1) < 0)
		return -1;
	if (pipe(p2) < 0)
		return -1;

	if ((pid = vfork()) == 0) {
		/* read and write reverse roles in child */
		(void) dup2(p2[0], 0);		/* standard in */
		(void) close(p2[0]);
		(void) close(p1[0]);
		(void) dup2(p1[1], 1);		/* standard out */
		(void) dup2(p1[1], 2);		/* standard error */
		(void) close(p1[1]);
		(void) close(p1[0]);
		execl(path, program, args, (char *)NULL);
		_exit(127);
	}

	rfd = p1[0];
	(void) close(p1[1]);
	wfd = p2[1];
	(void) close(p2[0]);

	if (pid == -1) {
		(void) close(rfd);
		(void) close(wfd);
		return -1;
	}

	popen_pid[rfd] = pid;
	*readfd = rfd;
	*writefd = wfd;
	return 0;
}

pclose2(readfd,writefd)
int	readfd,writefd;
{
	int child, pid, omask;
	union wait status;

	child = popen_pid[readfd];
	popen_pid[readfd] = -1;
	(void) close(readfd);
	(void) close(writefd);
	if (child == -1)
		return -1;

	omask = sigblock(sigmask(SIGINT)|sigmask(SIGQUIT)|sigmask(SIGHUP));
	while ((pid = wait(&status)) != child && pid != -1)
		;
	(void) sigsetmask(omask);
	return (pid == -1 ? -1 : status.w_status);
}
```

libftd/popentty2.c

```
/*
 * Copyright (c) 1980 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#include <stdio.h>
#include <signal.h>
#include <sys/wait.h>
#include <sys/file.h>
#include <sys/types.h>
#include <sys/stat.h>
#ifndef BSDPTY
#include <fcntl.h>
#include <sys/sysmacros.h>
#endif

static char *line = "/dev/ptyXX";

int
popentty2(path,program,args,readfd,writefd)
        char *path;
        char *program;
        char *args;
        int  *readfd,*writefd;
{
        int     master,slave,pid;
        static  int getptys();

        if (getptys(&master,&slave) < 0)
                return  -1;

        if ((pid = vfork()) < 0) {
                (void) close(master);
                (void) close(slave);
                return  -1;
        }

        if (pid == 0) {
                (void) close(master);
                (void) dup2(slave,0);
                (void) dup2(slave,1);
                (void) dup2(slave,2);
                if (slave > 2)
                        (void) close(slave);
                execl(path, program, args, (char *)NULL);
                _exit(127);
        }
        (void) close(slave);
        *readfd = dup(master);
        *writefd = master;
        return  0;
}

pclosetty2(readfd,writefd)
int     readfd,writefd;
{
        (void) close(readfd);
        (void) close(writefd);
        return  0;
}

static int
getptys(masterfd,slavefd)
int     *masterfd,*slavefd;
{
#ifdef BSDPTY
        char *pty, *bank, *cp;
        struct stat stb;

        pty = &line[strlen("/dev/ptyp")];
        for (bank = "pqrs"; *bank; bank++) {
                line[strlen("/dev/pty")] = *bank;
                *pty = '0';
                if (stat(line, &stb) < 0)
                        break;
                for (cp = "0123456789abcdef"; *cp; cp++) {
                        *pty = *cp;
                        *masterfd = open(line, O_RDWR);
                        if (*masterfd >= 0) {
                                char *tp = &line[strlen("/dev/")];
                                int ok;

                                /* verify slave side is usable */
                                *tp = 't';
                                *slavefd = open(line, O_RDWR);
                                *tp = 'p';
                                if (*slavefd >= 0)
                                        return 0;
                                (void) close(*masterfd);
                        }
                }
        }
        return  -1;
```

libftd/popentty2.c

```
#else
        int             fd;
        struct stat     sb;
        char            name[20];

        fd = open("/dev/ptc", O_RDWR|O_NDELAY);
        if (fd < 0)
                return -1;
        if (fstat(fd, &sb) < 0) {
                (void) close(fd);
                return -1;
        }
        (void) sprintf(name, "/dev/ttyq%d", minor(sb.st_rdev));
        *slavefd = open(name, O_RDWR|O_NDELAY);
        if (*slavefd < 0) {
                (void) close(fd);
                return -1;
        }
        *masterfd = fd;
        return 0;
#endif
}
```

libftd/scanner.l

```
%{
/*
 * $Id: scanner.l,v 1.4 89/10/25 15:17:37 gregc Exp $
 *
 * FTD scanner
 *
 */

#include "ftd_internal.h"
#include "grammar.h"
#include <errno.h>

extern int          atoi();
extern double       atof();


    /* should use hash table so non-keywords can be detected faster */

struct keytab {
    char    *keyword;
    int     token;
};

static struct keytab keywords[]     = {
            { "FTD_COPY_PROMPT", F_CPYPRMPT },
            { "FTD_OPTION", F_OPTION },
            { "FTD_QUIT", F_QUIT },
            { "FTD_SEND", F_SEND },
    { "boolean", K_BOOLEAN },
    { "default", K_DEFAULT },
    { "dialog", K_DIALOG },
    { "false", K_FALSE },
    { "flags", K_FLAGS },
    { "from", K_FROM },
    { "help", K_HELP },
    { "integer", K_INTEGER },
    { "label", K_LABEL },
    { "multi", K_MULTI },
    { "no", K_NO },
    { "notifier", K_NOTIFIER },
    { "options", K_OPTIONS },
    { "prompt", K_PROMPT },
    { "real", K_REAL },
    { "select", K_SELECT },
    { "semantic", K_SEMANTIC },
    { "text", K_TEXT },
    { "true", K_TRUE },
    { "yes", K_YES },
    { "yesno", K_YESNO },
};

static struct keytab *
get_keyword()
{
    register int    low, mid, high, diff;

    low = 0;
    high = (sizeof keywords / sizeof keywords[0]) - 1;
    while (low <= high) {
        mid = (low + high) / 2;
        diff = strcmp(yytext, keywords[mid].keyword);
        if (diff == 0)
            return &keywords[mid];
        if (diff < 0)
            high = mid - 1;
        else
            low = mid + 1;
    }
    return NULL;
}

# undef     YYLMAX
# define    YYLMAX      BUFSIZ

char    *yyfilename;
extern  char *strchr();


%}

integer             [0-9][0-9]*
real                [0-9][0-9]*\.[0-9]*
comm                \#[^\n]*\n
identifier          [a-zA-Z/][a-zA-Z0-9\_\.\-]*
quoted_string       \"([^"]|\\\")*\"

%%

{integer}                       {yylval.integer =
                                        atoi(yytext);
                                 return(INTEGER);}

{real}                          {yylval.real =
                                        atof(yytext);
```


```
                        return(REAL);}

{quoted_string}         {yylval.charval = strchr(yytext+1,\"');
                        *yylval.charval = '\0';
                        yylval.charval =
                                mmwb_cpystring(yytext+1);
                        return(QUOTED_TEXT);}

'('                     {return(LEFTPAREN);}
')'                     {return(RIGHTPAREN);}
'{'                     {return(LEFTBRACE);}
'}'                     {return(RIGHTBRACE);}
'['                     {return(LEFTBRACKET);}
']'                     {return(RIGHTBRACKET);}
','                     {return(COMMA);}
':'                     {return(COLON);}
';'                     {return(SEMICOLON);}
'.'                     {return(PERIOD);}
'?'                     {return(QUESTION);}
'='"="                  {return(IMPLIES);}
'='                     {return(EQUALS);}
'*'                     {return(MULTIPLY);}
'/'                     {return(DEVIDE);}
'+'                     {return(ADD);}
'-'                     {return(SUBTRACT);}

'&&'                    {return(AND);}
'||'                    {return(OR);}
'>='                    {return(GE);}
'<='                    {return(LE);}
'=='                    {return(R_EQUAL);}
'!='                    {return(NOT_EQUAL);}
'~'                     {return(NOT);}
'&'                     {return(B_AND);}
'|'                     {return(B_OR);}

'..'                    {return(DOTDOT);}

{identifier}            {
                                struct  keytab  *k;

                                if ((k = get_keyword()) != NULL) {
                                        yylval.charval = k->keywo
                                        return k->token;
                                }
                                yylval.charval =
                                        mmwb_cpystring(yytext);
                                return(IDENTIFIER);
                        }

{comm} ;

[ \t\n]      ;
```


libftd/grammar.y

```
%{
static char *RCSid = "$Header: /local/src/mmwb/src/ftd/libftd/RCS/grammar.y,v 1.7 9

#include <stdio.h>
#include <mmwbmisc.h>
#include <assert.h>
#include "ftd_internal.h"

extern  char *re_comp();

static  FTD_TAB ftd_list[MAXRULES];

int     rule_count = 0;
int     parsing_error = 0;

DIALOG  cur_dialog;

# define        IS_SET(e)       dialog_isnt_zero(cur_dialog, e)
# define        SET_INT(e, i)   assert(dialog_set_int(cur_dialog, e, i) == 0)
# define        SET_FLOAT(e, f) assert(dialog_set_float(cur_dialog, e, f) == 0)
# define        SET_TEXT(e, t)  assert(dialog_set_text(cur_dialog, e, t) == 0)
# define        SET_BOOL(e, b)  assert(dialog_set_bool(cur_dialog, e, b) == 0)
# define        SET_FLAG(e, f)  assert(dialog_set_flag(cur_dialog, e, f) == 0)

%}


%start  fti

%union {
        int             integer;
        double          real;
        char            *charval;
        FTD_TAB         ftival;
        struct  {
                union   {
                        int     integer;
                        double  real;
                }       lower, upper;
        }               range;
}

%token  PERIOD COLON SEMICOLON COMMA
%token  QUESTION IMPLIES EQUALS
%token  LEFTBRACE RIGHTBRACE LEFTPAREN RIGHTPAREN LEFTBRACKET
%token  DOTDOT

%token  <charval> K_INTEGER K_REAL K_TEXT K_SELECT K_FROM
%token  <charval> K_MULTI K_BOOLEAN K_YESNO K_NOTIFIER K_LABEL
%token  <charval> K_PROMPT K_HELP K_SEMANTIC K_FLAGS K_OPTIONS
%token  <charval> K_DEFAULT K_YES K_NO K_TRUE K_FALSE
%token  <charval> K_DIALOG K_OPTION K_SHEET
%token  <charval> F_QUIT F_OPTION F_SEND F_CPYPRMPT

%token  <charval> IDENTIFIER QUOTED_TEXT

%token  <integer> INTEGER
%token  <real>    REAL

%left   B_AND B_OR
%left   AND OR
%left   LT LE GT GE
%left   R_EQUAL NOT_EQUAL
%left   NOT
%left   ADD SUBTRACT
%left   MULTIPLY DEVIDE
%left   UMINUS

%type   <charval> fti_name var_name func_name
%type   <charval> bool_expr rel_expr arith_expr expr
%type   <charval> precondition postcondition assignment
%type   <charval> pattern name
%type   <charval> opt_option_string opt_default_text select_from_head
%type   <charval> opt_default_int_list int_list ident_or_string ident

%type   <ftival> conditions execution rule dialog dialog_list action

%type   <integer> opt_default_integer opt_default_boolean opt_default_yesno
%type   <integer> rule_flag flag_expr
%type   <real>    opt_default_real real
%type   <range>   opt_integer_range opt_real_range

%%


fti:    rules
                ;

rules:  rules rule
                {
                        ftd_list[rule_count++] = $2;
                }
        | rule
                {
```

```
                        ftd_list[rule_count++] = $1;
                }
                ;

rule:   name conditions execution postcondition SEMICOLON
                {
                        $$ = $3;
                        $$.name = $1;
                        $$.precond = $2.precond;
                        $$.pattern = $2.pattern;
                        $$.postcond = $4;
                        (void) dialog_set_text($$.dialog,DI_V_NAME,$$.name);
                }
        | name conditions execution SEMICOLON
                {
                        $$ = $3;
                        $$.name = $1;
                        $$.precond = $2.precond;
                        $$.pattern = $2.pattern;
                        $$.postcond = NULL;
                        (void) dialog_set_text($$.dialog,DI_V_NAME,$$.name);
                }
        ;

name:   fti_name COLON
                {
                        $$ = $1;
                }
        ;

conditions
        : precondition pattern
                {
                        $$.precond = $1;
                        $$.pattern = $2;
                }
        | pattern
                {
                        $$.precond = NULL;
                        $$.pattern =$1;
                }
        ;

execution
        : action dialog
                {
                        $$ = $1;
                        $$.dialog = $2.dialog;
                }
        ;

precondition
        : expr QUESTION
                { $$ = $1; }
        ;

pattern: QUOTED_TEXT IMPLIES
                {
                        char *s;

                        If( (s = re_comp($1)) != NULL) {
                                fprintf(stderr,"Pattern error: %s\n",s);
                                YYERROR;
                        }
                        $$ = $1;
                }
        ;

action: func_name COMMA
                {
                        $$.action = $1;
                        $$.flags = 0;
                }
        | func_name LEFTBRACKET flag_expr RIGHTBRACKET COMMA
                {
                        $$.action = $1;
                        $$.flags = $3;
                }
        | LEFTBRACKET flag_expr RIGHTBRACKET COMMA
                {
                        $$.action = mmwb_cpystring("ftd_dialog");
                        $$.flags = $2;
                }
        |
                {
                        $$.action = mmwb_cpystring("ftd_dialog");
                        $$.flags = 0;
                }
        ;

postcondition
        : postcondition COMMA assignment
                {
                        $$ = fti_post($1,$3);
```

```
                |
        | assignment
                { $$ = $1; }
        ;

assignment
        : var_name EQUALS expr
                {
                        $$ = make_expr($1,"=",$3);
                }
        ;

var_name: IDENTIFIER
                { $$ = $1; }
        ;

fti_name: IDENTIFIER
                { $$ = $1; }
        ;

func_name: IDENTIFIER
                { $$ = $1; }
        ;

expr:   LEFTPAREN expr RIGHTPAREN
                { $$ = make_expr($2,NULL,NULL); }
        | INTEGER
                { $$ = itoa($1); }
        | var_name
                { $$ = $1; }
        | bool_expr         %prec NOT
                { $$ = $1; }
        | arith_expr        %prec UMINUS
                { $$ = $1; }
        | rel_expr          %prec AND
                { $$ = $1; }
        ;

bool_expr: expr B_AND expr
                {
                        $$ = make_expr($1, "&", $3);
                }
        | expr B_OR expr
                {
                        $$ = make_expr($1, "|", $3);
                }
        | NOT expr
                {
                        $$ = make_expr(NULL, "~", $2);
                }
        ;

rel_expr
        : expr AND expr
                {
                        $$ = make_expr($1, "&&", $3);
                }
        | expr OR expr
                {
                        $$ = make_expr($1, "||", $3);
                }
        | expr GT expr
                {
                        $$ = make_expr($1, ">>", $3);
                }
        | expr LT expr
                {
                        $$ = make_expr($1, "<<", $3);
                }
        | expr GE expr
                {
                        $$ = make_expr($1, ">=", $3);
                }
        | expr LE expr
                {
                        $$ = make_expr($1, "<=", $3);
                }
        | expr R_EQUAL expr
                {
                        $$ = make_expr($1, "==", $3);
                }
        | expr NOT_EQUAL expr
                {
                        $$ = make_expr($1, "!=", $3);
                }
        ;

arith_expr
        : expr ADD expr
                {
                        $$ = make_expr($1, "+", $3);
                }
        | expr SUBTRACT expr
                {
```

```
                        $$ = make_expr($1, "-", $3);
                }
        | expr MULTIPLY expr
                {
                        $$ = make_expr($1, "*", $3);
                }
        | expr DEVIDE expr
                {
                        $$ = make_expr($1, "/", $3);
                }
        | SUBTRACT expr         %prec UMINUS
                {
                        $$ = make_expr(NULL, "~", $2);
                }
        ;

rule_flag: F_QUIT
                {
                        $$ = FTD_QUIT;
                }
        | F_OPTION
                {
                        $$ = FTD_OPTION;
                }
        | F_SEND
                {
                        $$ = FTD_SEND;
                }
        | F_CPYPRMPT
                {
                        $$ = FTD_COPY_PROMPT;
                }
        ;

flag_expr:      rule_flag
                {
                        $$ = $1;
                }
        | flag_expr B_AND flag_expr
                {
                        $$ = $1 & $3;
                }
        | flag_expr B_OR flag_expr
                {
                        $$ = $1 | $3;
                }
        | NOT flag_expr
                {
                        $$ = ~$2;
                }
        ;

dialog  : LEFTBRACE
                {
                        cur_dialog = dialog_create(0,0,(char *)NULL,
                                                   (char *)NULL);
                }
          dialog_statement_list RIGHTBRACE
                {
                        $$.dialog = cur_dialog;
                }
        | LEFTBRACE dialog_list RIGHTBRACE
                {
                        $$.dialog = $2.dialog;
                }
        ;

dialog_list
        : LEFTBRACE
                {
                        cur_dialog = dialog_create(0,0,(char *)NULL,
                                                   (char *)NULL);
                }
          dialog_statement_list RIGHTBRACE
                {
                        $$.dialog = cur_dialog;
                }
        | dialog_list LEFTBRACE
                {
                        cur_dialog = dialog_create(0,0,(char *)NULL,
                                                   (char *)NULL);
                        $$.dialog = dialog_append($1.dialog,cur_dialog);
                }
          dialog_statement_list RIGHTBRACE
        ;

dialog_statement_list
        : dialog_statement_list dialog_statement SEMICOLON
        | dialog_statement_list error SEMICOLON
                { yyerror("malformed dialog statement"); }
        |
        ;
```

```
dialog_statement
        : K_INTEGER opt_integer_range opt_default_integer
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_IVAL);
                        SET_INT(DI_V_RESULT, $3);
                        SET_INT(DI_V_UBOUND, $2.upper.integer);
                        SET_INT(DI_V_LBOUND, $2.lower.integer);
                }
        | K_REAL opt_real_range opt_default_real
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_FVAL);
                        SET_FLOAT(DI_V_RESULT, $3);
                        SET_FLOAT(DI_V_UBOUND, $2.upper.integer);
                        SET_FLOAT(DI_V_LBOUND, $2.lower.integer);
                }
        | K_TEXT opt_integer_range opt_default_text
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_TEXT);
                        SET_TEXT(DI_V_RESULT, $3);
                        SET_INT(DI_V_UBOUND, $2.upper.integer);
                        SET_INT(DI_V_LBOUND, $2.lower.integer);
                }
        | select_from_head RIGHTBRACE opt_default_integer
                {
                        char    buf[64];

                        if (dialog_type(cur_dialog) != DI_SELECT)
                                break;

                        (void) sprintf(buf, "%d", $3);
                        SET_TEXT(DI_V_RESULT, mmwb_cpystring(buf));
                        SET_INT(DI_V_UBOUND, 1);
                        SET_INT(DI_V_LBOUND, 1);
                        SET_TEXT(DI_V_SELLIST, $1);
                }
        | K_MULTI select_from_head RIGHTBRACE opt_integer_range opt_default_int_list
                {
                        if (dialog_type(cur_dialog) != DI_SELECT)
                                break;

                        SET_TEXT(DI_V_RESULT, $5);
                        SET_INT(DI_V_UBOUND, $4.upper.integer);
                        SET_INT(DI_V_LBOUND, $4.lower.integer);
                        SET_TEXT(DI_V_SELLIST, $2);
                }
        | K_BOOLEAN opt_default_boolean
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_BOOL);
                        SET_BOOL(DI_V_RESULT, $2);
                }
        | K_YESNO opt_default_yesno
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_YESNO);
                        SET_BOOL(DI_V_RESULT, $2);
                }
        | K_NOTIFIER
                {
                        if (IS_SET(DI_V_TYPE))
                                yyerror("dialog type already given");
                        else
                                SET_INT(DI_V_TYPE, DI_NOTE);
                }
        | K_LABEL ident_or_string
                {
                        if (IS_SET(DI_V_NAME))
                                yyerror("dialog name already given");
                        else
                                SET_TEXT(DI_V_NAME, $2);
                }
        | K_PROMPT QUOTED_TEXT
                {
                        if (IS_SET(DI_V_PROMPT))
                                yyerror("dialog prompt already given");
                        else
```

```
                        SET_TEXT(DI_V_PROMPT, $2);
                }
        | K_HELP QUOTED_TEXT
                {
                        if (IS_SET(DI_V_HELP))
                                yyerror("dialog help already given");
                        else
                                SET_TEXT(DI_V_HELP, $2);
                }
        | K_SEMANTIC ident_or_string opt_option_string
                {
                        int     semtype;

                        if (IS_SET(DI_V_SEMTYPE)) {
                                yyerror("dialog semantic type already given");
                                break;
                        }

                        if ((semtype = semantic_lookup($2)) == 0)
                                yyerror("unknown semantic type");
                        else {
                                SET_INT(DI_V_SEMTYPE, semtype);
                                SET_TEXT(DI_V_SEMOPT, $3);
                        }
                }
        | flag_list
        ;

select_from_head
        : K_SELECT K_FROM LEFTBRACE QUOTED_TEXT
                {
                        if (IS_SET(DI_V_TYPE)) {
                                yyerror("dialog type already given");
                                break;
                        }
                        SET_INT(DI_V_TYPE, DI_SELECT);
                        $$ = mmwb_cpystring(stringize($4));
                }
        | select_from_head COMMA QUOTED_TEXT
                {
                        char    *s;
                        int     len;

                        if (dialog_type(cur_dialog) != DI_SELECT)
                                break;

                        s = stringize($3);
                        len = strlen($1);
                        $$ = (char *)erealloc($1, len + strlen(s) + 1);
                        (void) strcpy($$ + len, s);
                }
        ;

flag_list
        : K_FLAGS ident
                {
                        int     flag;

                        if ((flag = lookup_flag($2)) == 0)
                                yyerror("unknown flag");
                        else
                                SET_FLAG(flag, 1);
                }
        | flag_list COMMA ident
                {
                        if (IS_SET(DI_V_TYPE))
                                yyerror("dialog type already given");
                        else
                                SET_INT(DI_V_TYPE, DI_NOTE);
                }
        ;

opt_integer_range
        : INTEGER DOTDOT INTEGER
                {
                        $$.lower.integer = $1;
                        $$.upper.integer = $3;
                }
        |
                {
                        $$.lower.integer = 0;
                        $$.upper.integer = 0;
                }
        ;

opt_default_integer
        : K_DEFAULT INTEGER
                { $$ = $2; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = 0; }
        ;

opt_real_range
        : real DOTDOT real
```

```
                {
                        $$.lower.real = $1;
                        $$.upper.real = $3;
                }
        |
                {
                        $$.lower.real = 0.0;
                        $$.upper.real = 0.0;
                }
        ;

opt_default_real
        : K_DEFAULT real
                { $$ = $2; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = 0.0; }
        ;

opt_default_text
        : K_DEFAULT QUOTED_TEXT
                { $$ = $2; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = NULL; }
        ;

opt_default_boolean
        : K_DEFAULT K_TRUE
                { $$ = TRUE; SET_FLAG(DI_OPTIONAL, 1); }
        | K_DEFAULT K_FALSE
                { $$ = FALSE; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = FALSE; }
        ;

opt_default_yesno
        : K_DEFAULT K_YES
                { $$ = TRUE; SET_FLAG(DI_OPTIONAL, 1); }
        | K_DEFAULT K_NO
                { $$ = FALSE; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = FALSE; }
        ;

opt_option_string
        : K_OPTIONS QUOTED_TEXT
                { $$ = $2; SET_FLAG(DI_OPTIONAL, 1); }
        |
                { $$ = NULL; }
        ;

opt_default_int_list
        :       K_DEFAULT int_list
                { $$ = $2; }
        |
                { $$ = ""; }
        ;

int_list
        :       INTEGER
                {
                        char    buf[32];

                        (void) sprintf(buf, "%d", $1);
                        $$ = mmwb_cpystring(buf);
                }
        |       int_list COMMA INTEGER
                {
                        char    buf[32];
                        int     len;

                        (void) sprintf(buf, "%d", $3);
                        len = strlen($1);
                        $$ = (char *)erealloc($1, len + strlen(buf) + 2);
                        $$[len] = ' ';
                        (void) strcpy($$ + len + 1, buf);
                }
        ;

real    : REAL
                { $$ = $1; }
        | INTEGER
                { $<real>$ = $1; }
        ;

ident_or_string
        : ident
                { $$ = $1; }
        | QUOTED_TEXT
                { $$ = $1; }
        ;

ident   : IDENTIFIER
                { $$ = $1; }
        | K_INTEGER
```

```
                { $$ = $1; }
        | K_REAL
                { $$ = $1; }
        | K_TEXT
                { $$ = $1; }
        | K_MULTI
                { $$ = $1; }
        | K_SELECT
                { $$ = $1; }
        | K_FROM
                { $$ = $1; }
        | K_BOOLEAN
                { $$ = $1; }
        | K_YESNO
                { $$ = $1; }
        | K_NOTIFIER
                { $$ = $1; }
        | K_LABEL
                { $$ = $1; }
        | K_FLAGS
                { $$ = $1; }
        | K_PROMPT
                { $$ = $1; }
        | K_HELP
                { $$ = $1; }
        | K_SEMANTIC
                { $$ = $1; }
        | K_DEFAULT
                { $$ = $1; }
        | K_OPTIONS
                { $$ = $1; }
        | K_TRUE
                { $$ = $1; }
        | K_FALSE
                { $$ = $1; }
        | K_YES
                { $$ = $1; }
        | K_NO
                { $$ = $1; }
        ;

%%

/*
 * Yacc mandated routines
 */

yyerror(s)
char    *s;
{
        if (yyfilename == NULL)
                fprintf(stderr, "%s on line %d\n", s, yylineno);
        else
                fprintf(stderr, "%s on line %d of %s\n", s, yylineno,
                                                        yyfilename);
        parsing_error = TRUE;
}

yywrap()
{
        return 1;
}


/*
 * make_expr - return an expression
 */

static char *
make_expr(first,op,second)
char *first,*op,*second;
{
        char    buf[BUFSIZ];


        buf[0] = '(';
        buf[1] = ' ';
        buf[2] = '\0';

        if (first != NULL) {
                (void) strcat(buf, first);
                mmwb_stringfree(first);
        }

        if (op != NULL) {
                (void) strcat(buf, " ");
                (void) strcat(buf, op);
                (void) strcat(buf, " ");
        }

        if (second != NULL) {
                (void) strcat(buf, second);
                mmwb_stringfree(second);
        }
```

```
        (void) strcat(buf, " ");
        (void) strcat(buf,")");
        return  mmwb_cpystring(buf);
}

/*
 * fti_post
 */

static char *
fti_post (post, assign)
char *post,*assign;
{
        char    *result;

        result = (char *)emalloc(strlen(post)+strlen(assign)+2);

        (void) sprintf(result,"%s,%s",post,assign);

        (void) free(post);
        (void) free(assign);
        return  result;
}

/*
 * ftd_table:
 *      Return a pointer to the top of the rule list
 */

FTD_TAB *
ftd_table()
{
        return  &ftd_list[0];
}

/*
 * ftd_next:
 *      Return a pointer to the next table entry
 */

FTD_TAB *
ftd_next(tpointer)
FTD_TAB *tpointer;
{
        return  ++tpointer;
}
```

```
/*
 * ftd_printrules
 */

ftd_printrules(ofile)
FILE    *ofile;
{
        int     i;

        fprintf(ofile, "static FTD_TAB rule_list[] = {\n");

        for (i = 0; i < rule_count; i++) {
                fprintf(ofile,"{\n\"%s\",\"%s\",%s,%s,%s,",
                        ftd_list[i].name, ftd_list[i].precond,
                        ftd_list[i].pattern, ftd_list[i].flags,
                        ftd_list[i].action);

                if (ftd_list[i].dialog != NULL) {
                        DIALOG  d = ftd_list[i].dialog;
                        fprintf(ofile,"\n{");
                        while (d != NULL) {
                                fprintf(ofile,"\t%s,\n",
                                        stringize(dialog_to_char(d)));
                                d = dialog_next(d);
                        }
                        fprintf(ofile,"},");
                } else
                        fprintf(ofile,"{},");

                fprintf(ofile,"\"%s\"},\n",ftd_list[i].postcond);
        }
        fprintf(ofile,"{NULL}\n};\n");
}

/*
 * itoa - integer to ascii conversion
 */

static char *
itoa(i)
int i;
{
        char    buf[80];

        (void) sprintf(buf,"%d",i);
```

```
        return  mmwb_cpystring(buf);
}

/*
 *      lookup_flag:
 *
 *              Return flag that corresponds name given.  Returns zero
 *      if the name is not a flag name.
 */

static int
lookup_flag(flag_name)
        char    *flag_name;
{
        if (strcmp(flag_name, "optional") == 0)
                return DI_OPTIONAL;
        if (strcmp(flag_name, "nodisplay") == 0)
                return DI_NODISPLAY;
        if (strcmp(flag_name, "supercedes") == 0)
                return DI_SUPERCEDES;
        return 0;
}


/*
 *      semantic_lookup:
 *
 *              Return semantic type that corresponds name given.  Returns zero
 *      if the name is not a known semantic type.  Eventually, this should be
 *      replaced by a Data Manager query.
 */

static int
semantic_lookup(name)
        char    *name;
{
        if (strcmp(name, "object") == 0)
                return 1;
        if (strcmp(name, "echo") == 0)
                return 2;
        return 0;
}
```

UC San Francisco LIBRARY
UNIVERSITY OF CALIFORNIA

564043

3 1378 00564 0431

UC San Francisco LIBRARY

FOR REFERENCE

NOT TO BE TAKEN FROM THE ROOM

BRO DART CAT. NO. 23 012 PRINTED IN U.S.A.